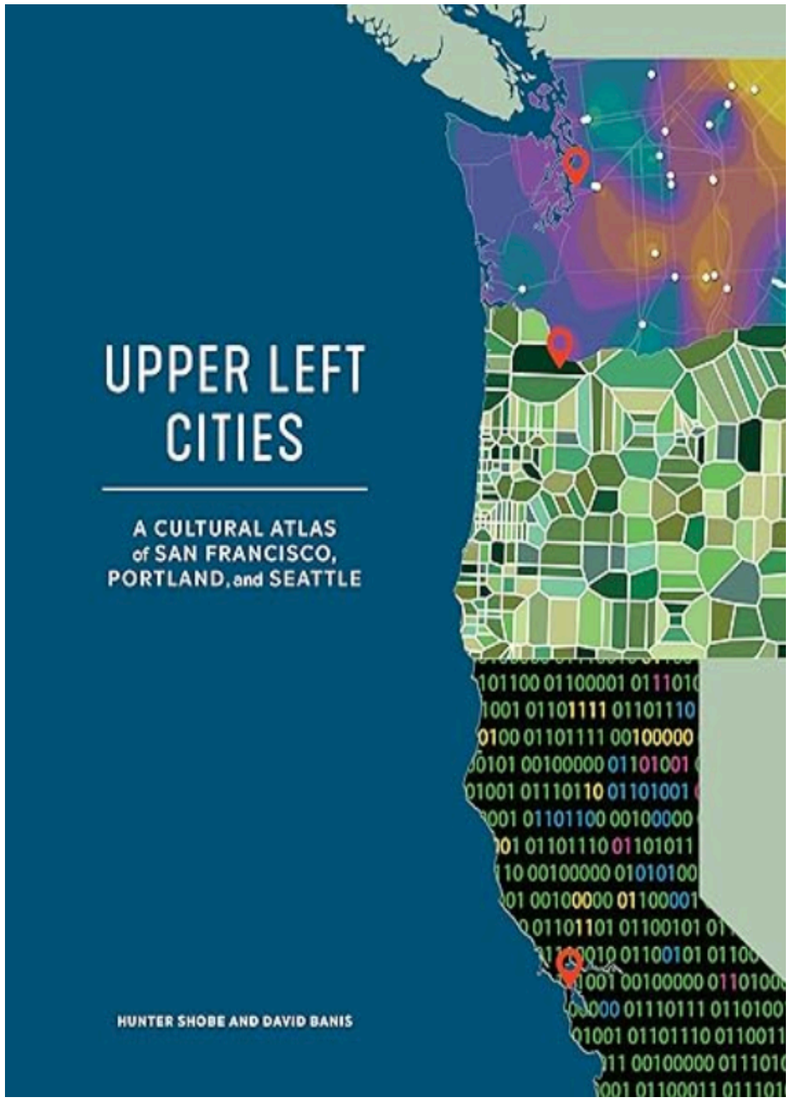
Cartographic Tour of R and {ggplot2}

Justin Sherrill

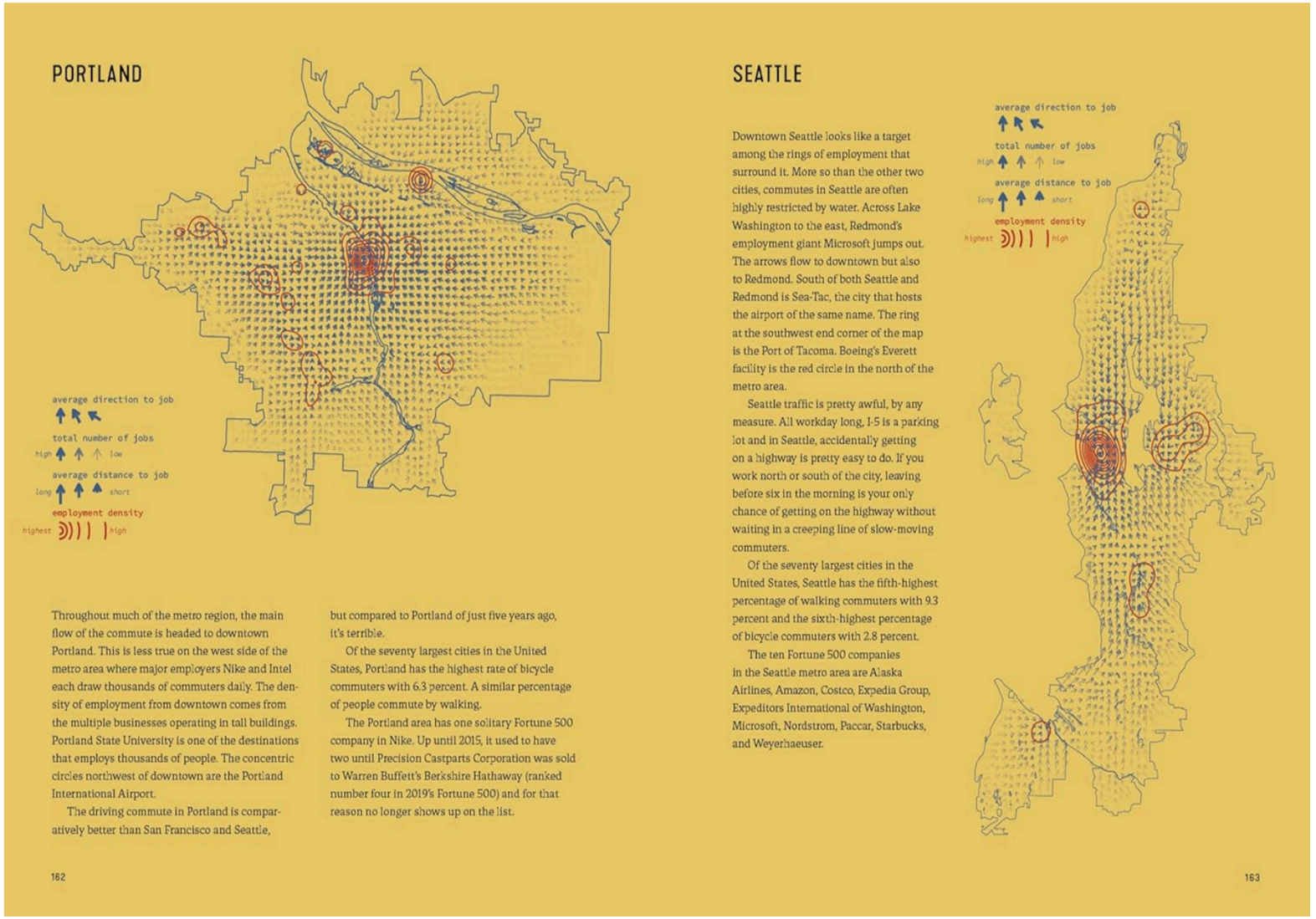
Who am I?

- 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

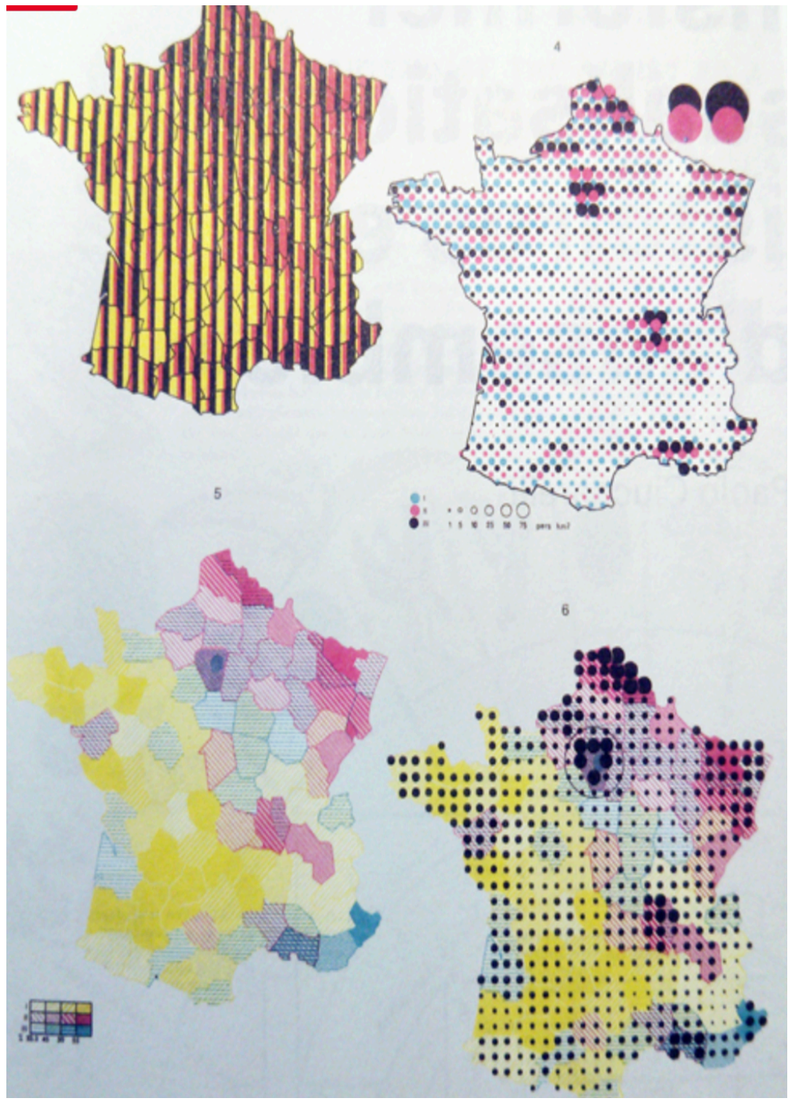
- Sometimes you need a custom package to accomplish the task at hand
- Other times, {sf} and {ggplot2} had the solution all along

What is “good” cartography?

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

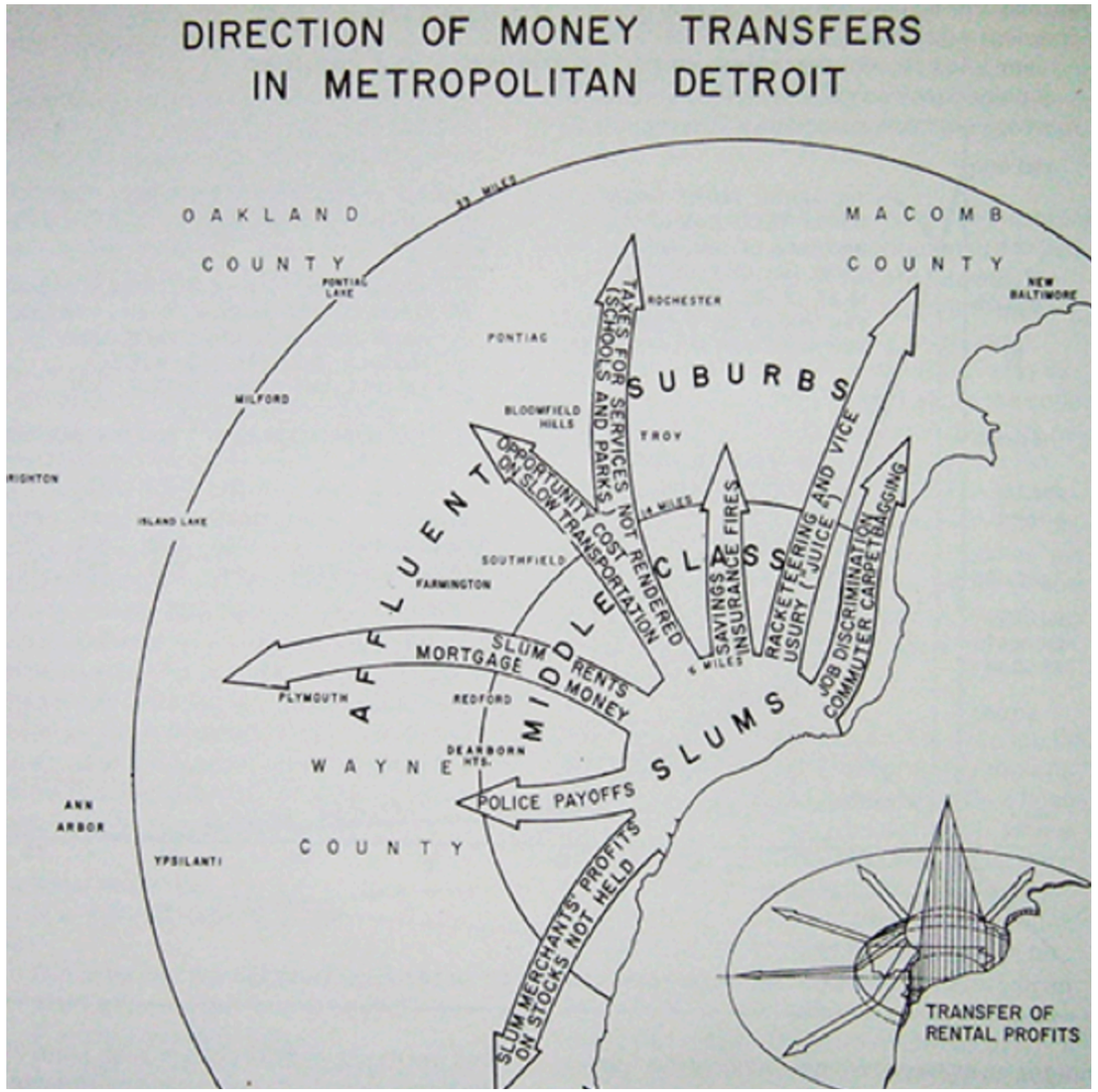
How to make a good map

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



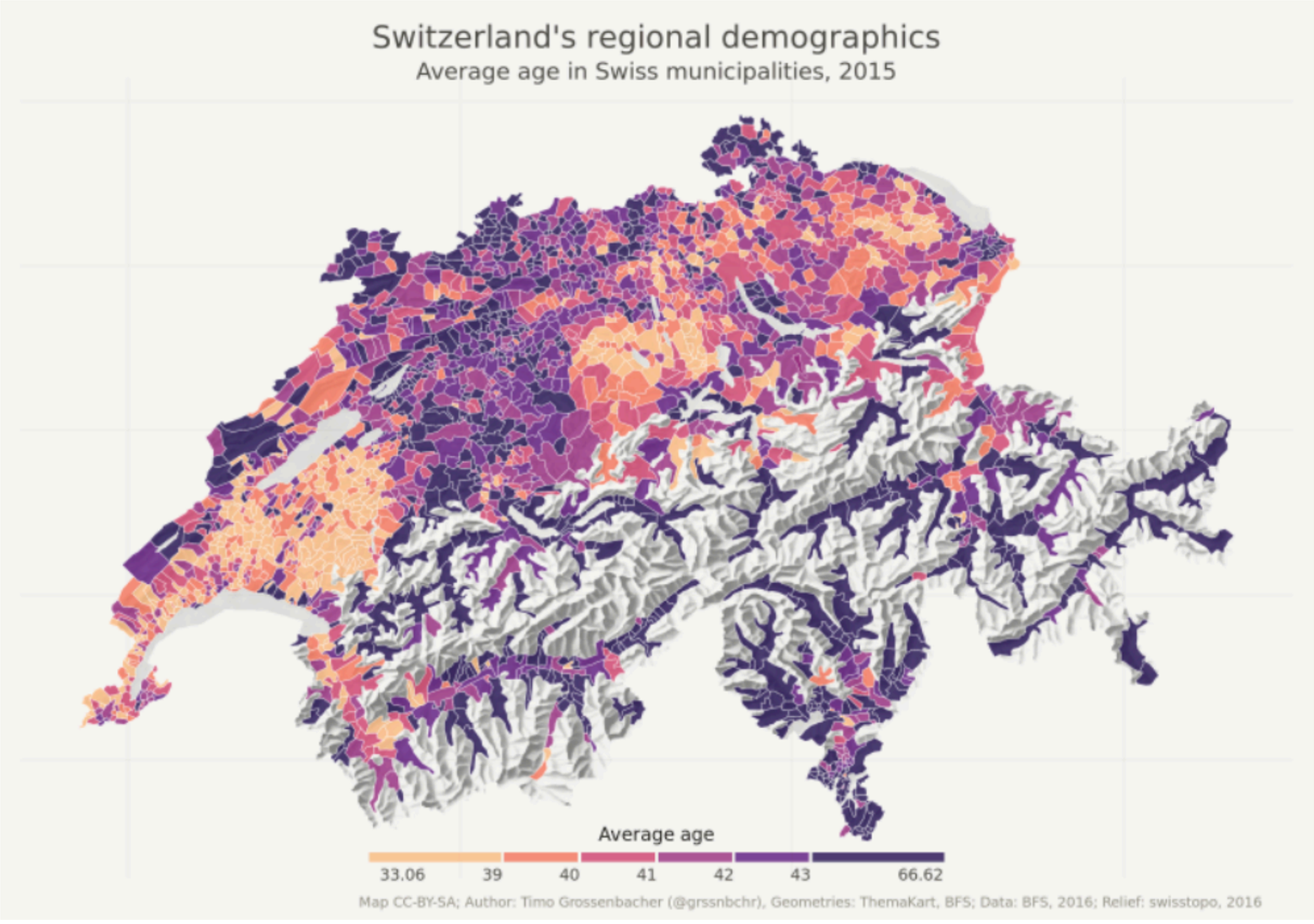
How to make a good map

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



How to make a good map

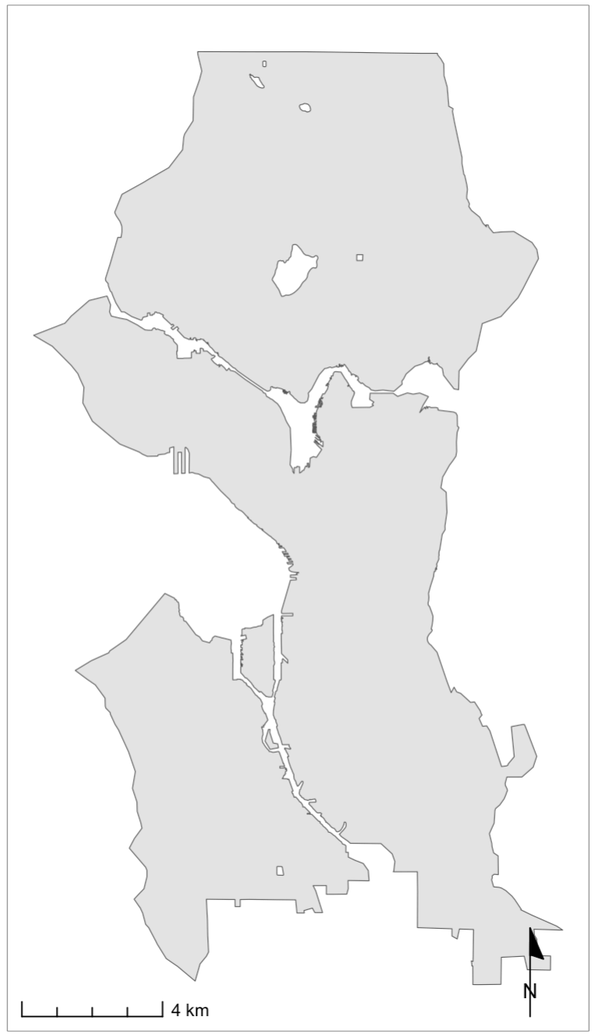
- Copy a good map
- `Jacques Bertin`
- `William Bunge`
- `Timo Grossenbacher`



Marginalia

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

```
sea <- places("wa",
              cb = TRUE) %>%
  filter(NAME == "Seattle") %>%
  st_transform(3857) %>%
  erase_water(area_threshold = .25)

ggplot() +
  geom_sf(data = sea) +
  ggspatial::annotation_north_arrow(location = "br",
                                    which_north = "true",
                                    style = north_arrow_minimal) +
  annotation_scale(location = "bl",
                   style = "ticks") +
  coord_sf(crs = 3857,
           datum = NA) +
  theme_minimal() +
  theme(panel.border = element_rect(linewidth = .2, fill = NA))
```



Marginalia, cont'd

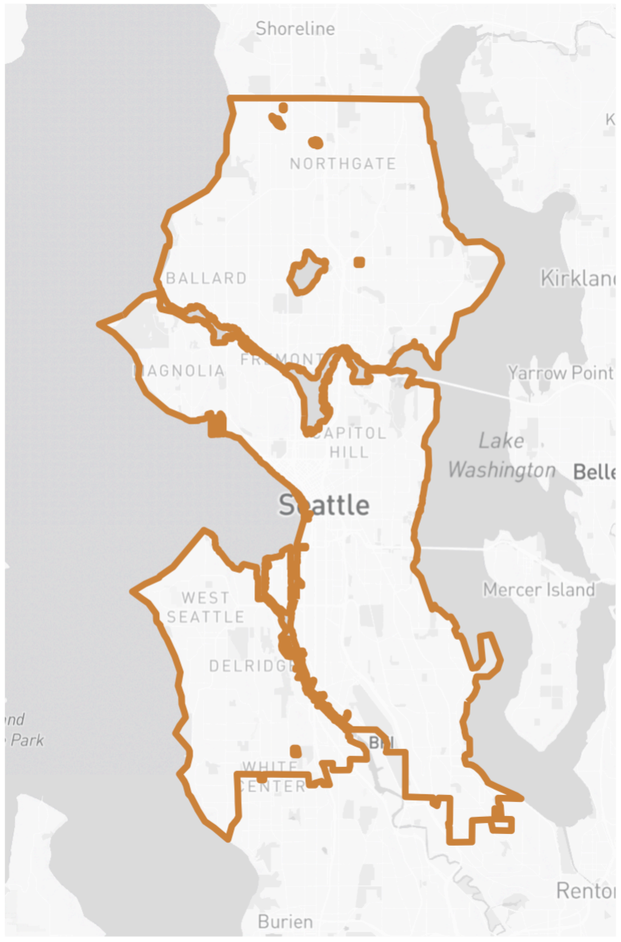
📎 Giving your marginalia a little breathing room can be a good move. So expand your limits within `coord_sf()`!

```
sea <- places("wa",
              cb = TRUE) %>%
  filter(NAME == "Seattle") %>%
  st_transform(3857) %>%
  erase_water(area_threshold = .25)

xlims <- st_buffer(sea, 5000) %>%
  st_bbox() %>%
  .[c(1, 3)]

ylims <- st_buffer(sea, 5000) %>%
  st_bbox() %>%
  .[c(2, 4)]

ggplot() +
  geom_sf(data = sea) +
  ggspatial::annotation_north_arrow(location = "br",
                          which_north = "true",
                          style = north_arrow_minimal) +
  annotation_scale(location = "bl",
                   style = "ticks") +
  coord_sf(crs = 3857,
           datum = NA,
           xlim = xlims,
           ylim = ylims) +
  theme_minimal() +
```

Basemaps

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

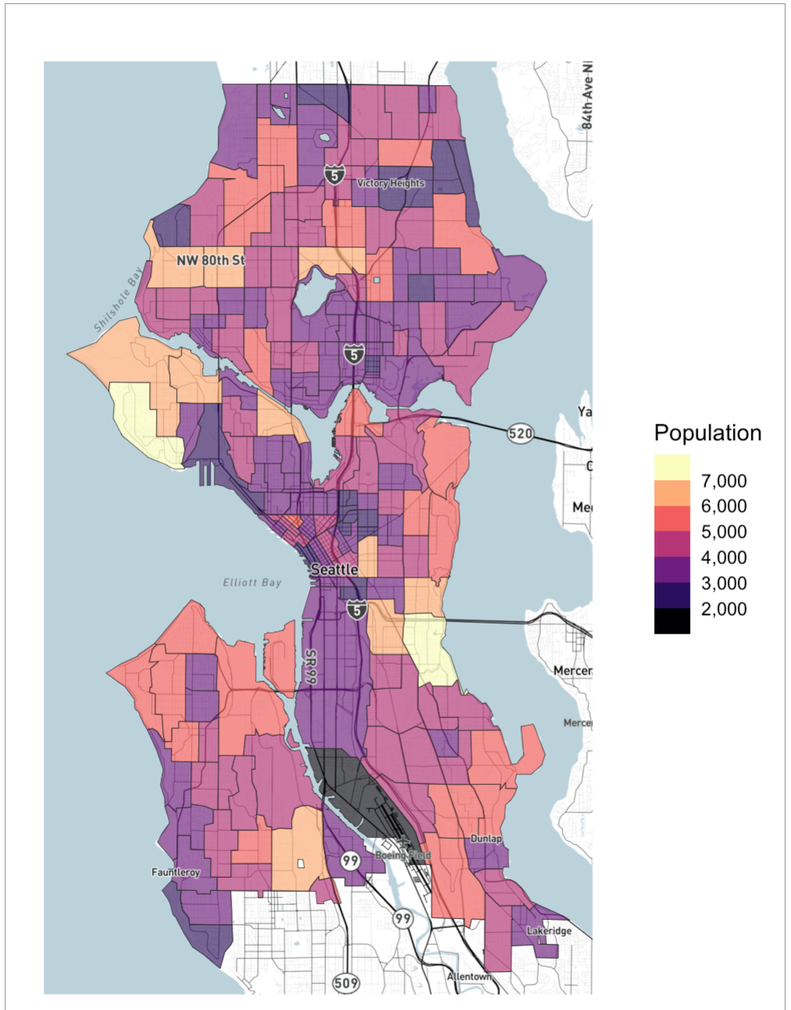
```r
basemap <- get_static_tiles(
  location = st_buffer(sea, 5000),
  zoom = 10,
  buffer_dist = 0,
  scaling_factor = "2x",
  crop = TRUE,
  style_id = basic_id,
  username = me
)

ggplot() +
  layer_spatial(data = basemap) +
  geom_sf(data = sea,
          fill = NA,
          color = "peru",
          linewidth = 1.5) +
  coord_sf(crs = 3857,
           datum = NA,
           xlim = xlims,
           ylim = ylims) +
  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()`

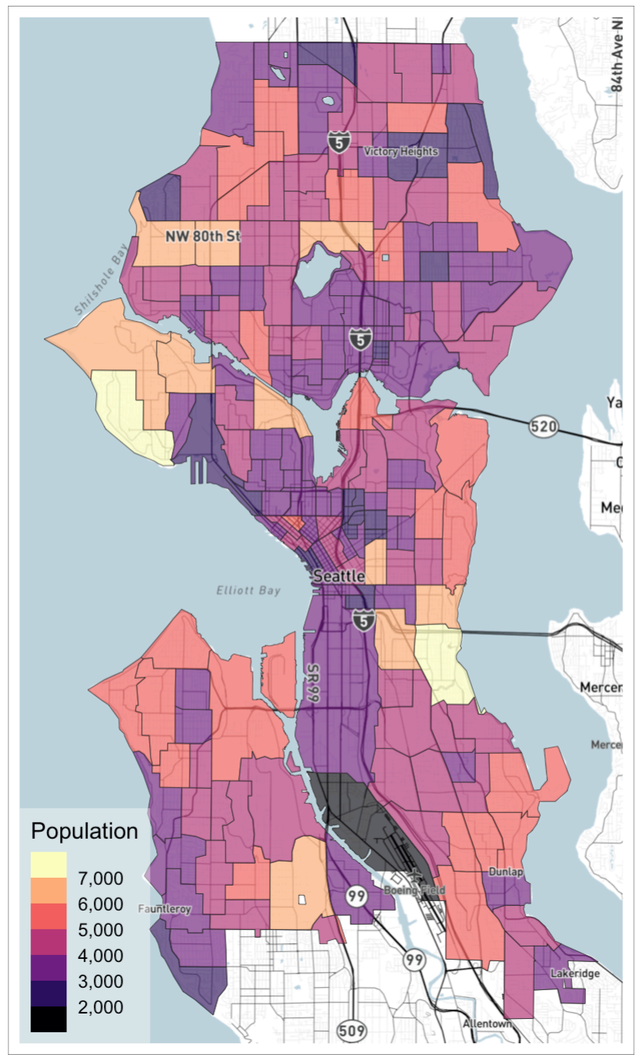
```
tract_pop <- get_acs("tract",
                     state = "WA",
                     county = "King",
                     variables = "B01001_001",
                     geometry = TRUE) %>%
  st_transform(st_crs(sea)) %>%
  st_intersection(sea)

basemap <- get_static_tiles(
  location = sea,
  zoom = 11,
  buffer_dist = 1000,
  scaling_factor = "2x",
  crop = TRUE,
  style_id = basemap_id,
  username = me
)

labels <- get_static_tiles(
  location = sea,
  zoom = 11,
  buffer_dist = 1000,
  scaling_factor = "2x",
  crop = TRUE,
  style_id = labels_id,
  username = me
```



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

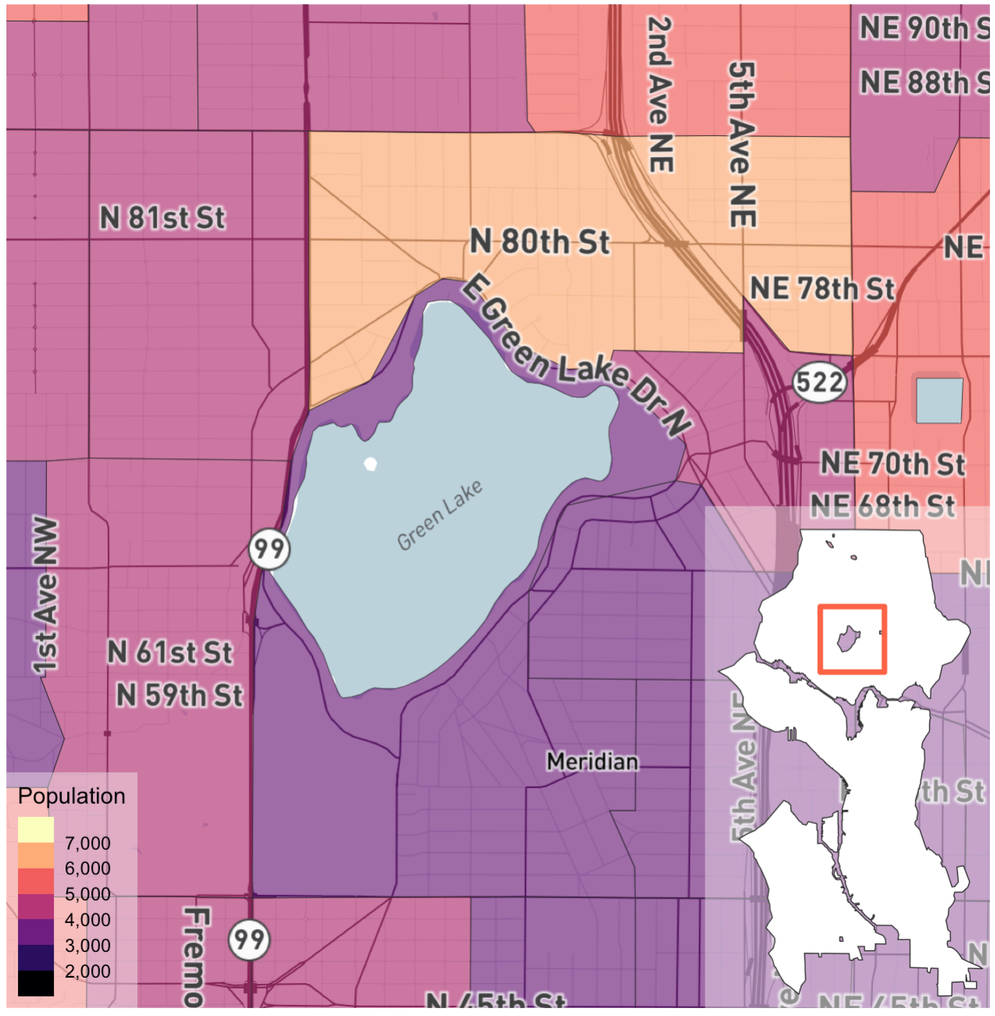
```
ggplot() +
  layer_spatial(data = basemap) +
  geom_sf(data = tract_pop,
          aes(fill = estimate),
          color = "gray20",
          linewidth = .15,
          alpha = .7) +
  layer_spatial(data = labels) +
  scale_fill_viridis_b(option = "A",
                       direction = 1,
                       name = "Population",
                       n.breaks = 7,
                       labels = comma) +
  coord_sf(crs = 3857,
           datum = NA,
           # This one is necessary!
           expand = 0) +
  theme_minimal() +
  theme(legend.position = "inside",
        legend.location = "plot",
        legend.position.inside = c(0, 0),
        legend.justification.inside = c(0, 0),
        # element_rect() doesn't have an alpha argument, but you c
        legend.box.background = element_rect(fill = "#ffffff60", c
  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.

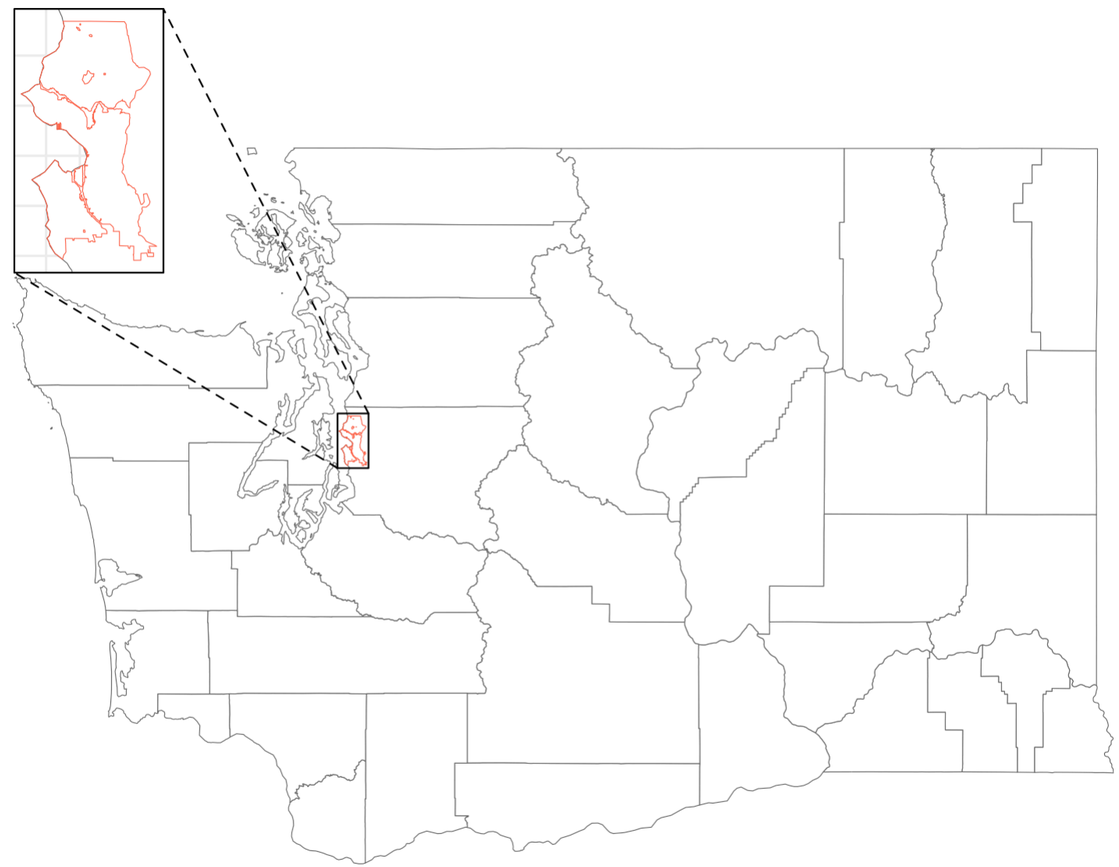
```
greenlake <- st_as_sfc("POLYGON ((-13620987 6050695, -13620987 605
  st_sf()

xlims <- st_bbox(greenlake)[c(1, 3)]

ylims <- st_bbox(greenlake)[c(2, 4)]

basemap <- get_static_tiles(
  location = greenlake,
  zoom = 13,
  buffer_dist = 1000,
  scaling_factor = "2x",
  crop = TRUE,
  style_id = basemap_id,
  username = me
)

labels <- get_static_tiles(
  location = greenlake,
  zoom = 13,
  buffer_dist = 1000,
  scaling_factor = "2x",
  crop = TRUE,
  style_id = labels_id,
  username = me
)
```



Insets

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

```
counties <- counties("wa",
                     cb = TRUE) %>%
  st_transform(3857)

to <- st_as_sfc("POLYGON ((-13881692 6185920, -13881692 6376113, -
                crs = 3857) %>%
  st_sf()

xlims <- st_bbox(st_union(counties, to))[c(1, 3)]

ylims <- st_bbox(st_union(counties, to))[c(2, 4)]

county_map <- ggplot() +
  geom_sf(data = counties,
          fill = "white",
          color = "gray40",
          linewidth = .2) +
  geom_sf(data = sea,
          fill = NA,
          color = "tomato",
          linewidth = .2) +
  coord_sf(crs = 3857,
           datum = NA,
           # This one is necessary!
           expand = 0,
           xlim = xlims,
```



Labels

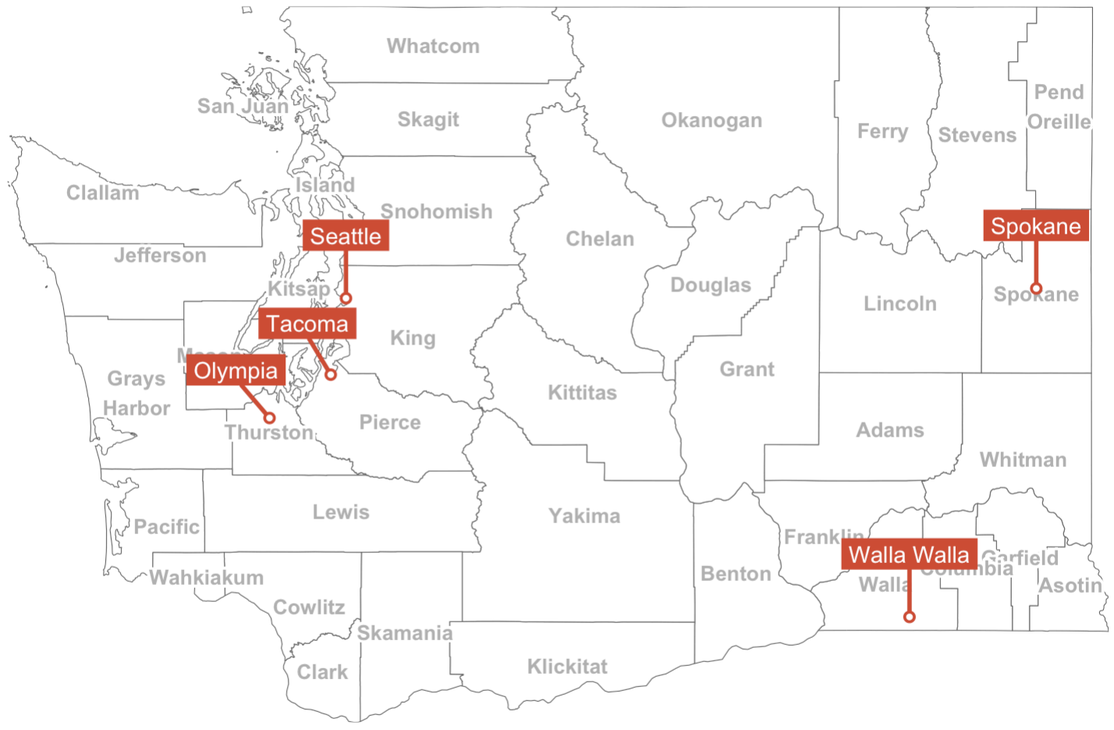
📎{ggrepel} is a very powerful tool for labeling your map, especially when you have a lot of labels to place. It can be a bit fussy though, so it will likely take lots of trial and error. And don't sleep on the `bg.colour` argument for `geom_text_repel()`!

```
cities <- get_acs("place",
                                    state = "wa",
                                    variables = "B01001_001",
                                    year = 2022,
                                    geometry = TRUE) %>%
    filter(estimate > 7.5e4) %>%
    mutate(NAME = str_remove_all(NAME, paste0(c(" city, Washington
    mutate(class = santoku::chop(estimate, breaks = c(1e5, 1.5e5,
  # filter(NAME %in% c("Seattle", "Tacoma", "Olympia", "Spokane",
  st_transform(3857)

ggplot() +
  geom_sf(data = counties,
          fill = "white",
          color = "gray40",
          linewidth = .2) +
  geom_label_repel(data = st_centroid(cities),
                   aes(x = st_coordinates(geometry)[, 1],
                       y = st_coordinates(geometry)[, 2],
                       label = NAME,
                        size = class),
                   min.segment.length = 0,
                   color = "white",
                   fill = "tomato3",
                   label.r = unit(0, "lines"),
                   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

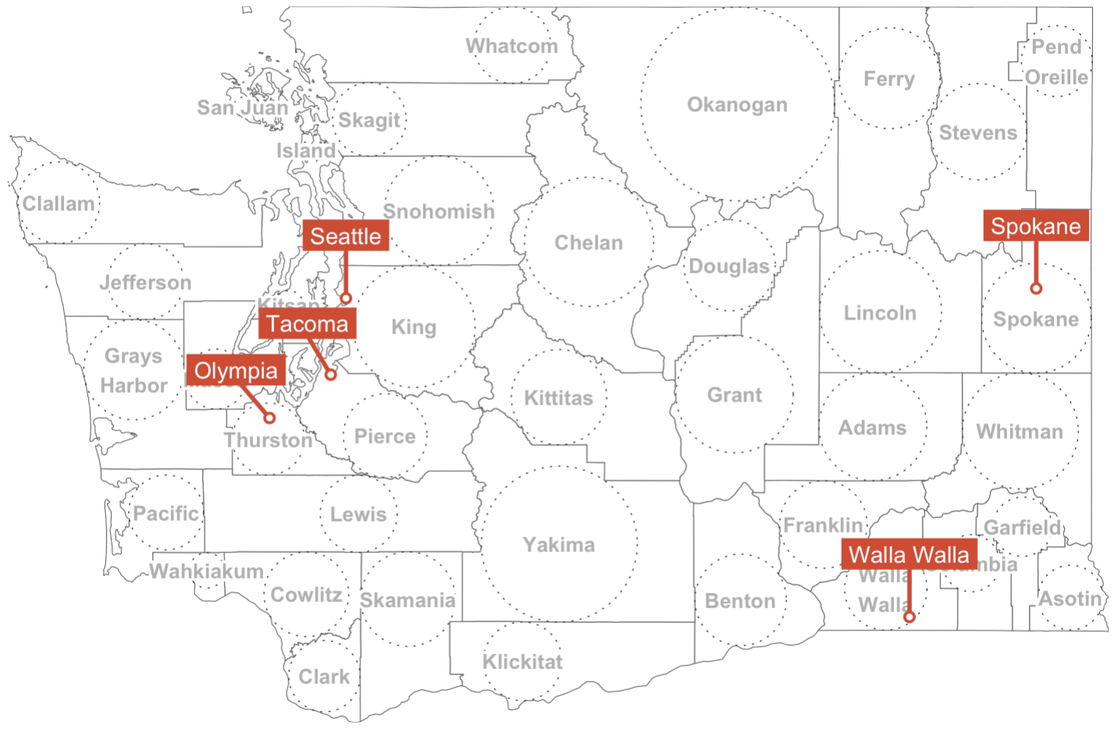
```
cities <- places("wa") %>%
  filter(NAME %in% c("Seattle", "Tacoma", "Olympia", "Spokane", "W
  st_transform(3857)

county_lbls <- st_point_on_surface(counties) %>%
  transmute(name = NAME,
            x = st_coordinates(.)[, 1],
            y = st_coordinates(.)[, 2])

ggplot() +
  geom_sf(data = counties,
          fill = "white",
          color = "gray40",
          linewidth = .2) +
    geom_text_repel(data = county_lbls,
                    aes(x = x,
                        y = y,
                        label = str_wrap(name, 10)),
                    # ggrepel functions have this bg.colour argume
                    bg.colour = "white",
                    bg.r = .2,
                    force = 0,
                    color = "gray70",
                    fontface = "bold",
                    size = 3.5) +
  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

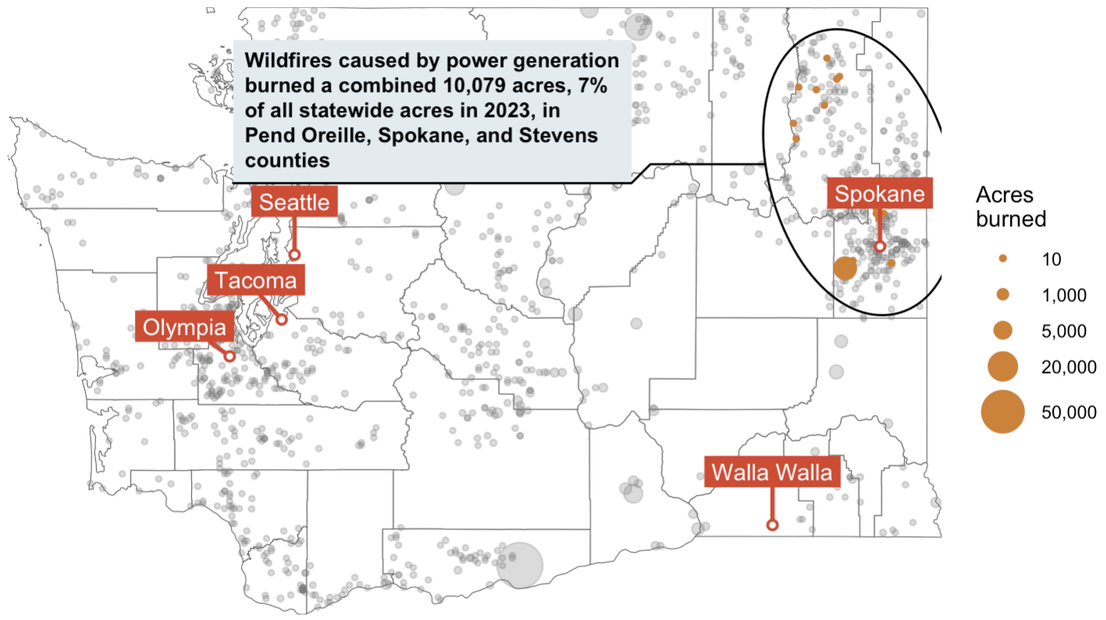
```
circles <- counties %>%
    pull(geometry) %>%
  st_inscribed_circle() %>%
  st_sf() %>%
  filter(!st_is_empty(geometry))

county_lbls <- circles %>%
  st_centroid() %>%
  transmute(name = counties$NAME,
                   x = st_coordinates(.)[, 1],
                   y = st_coordinates(.)[, 2])

ggplot() +
  geom_sf(data = counties,
          fill = "white",
          color = "gray40",
          linewidth = .2) +
    geom_sf(data = circles,
                   fill = NA,
                   color = "gray20",
                   linewidth = .25,
                   linetype = "dotted") +
    geom_text_repel(data = county_lbls,
                   aes(x = x,
                           y = y,
                           label = str_wrap(name, 10)),
```



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

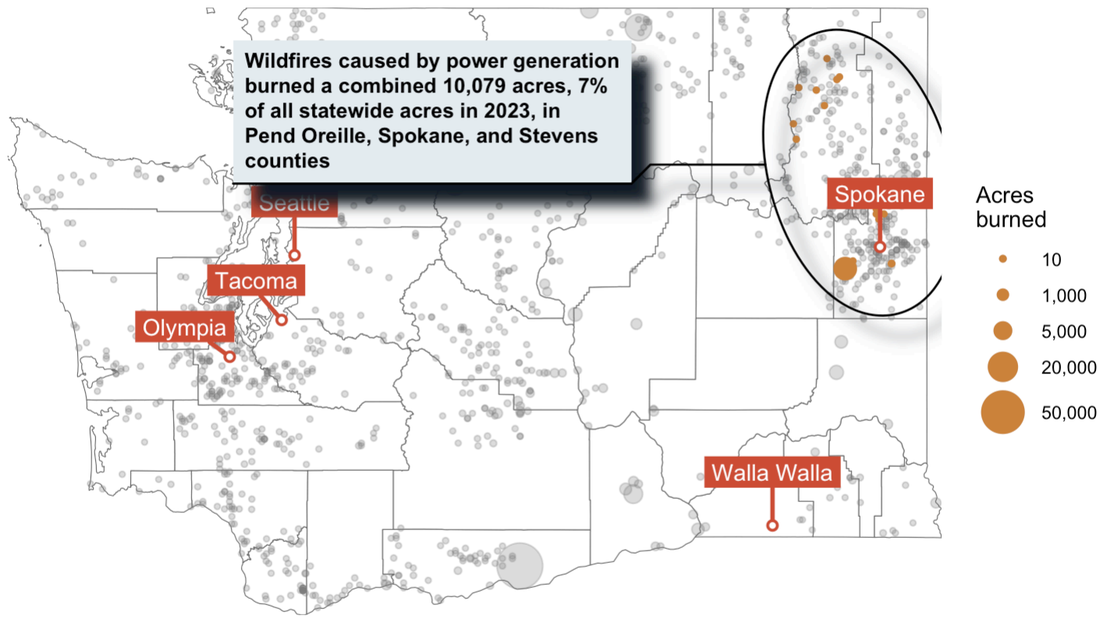
```
# Read in the data, in this case this CSV from WA DNR https://data
fires <- read_csv("/Users/sherrill/Projects/CascadiaR_Cartographic
  st_as_sf(coords = c("LON_COORD", "LAT_COORD"), crs = 4326) %>%
    st_transform(3857) %>%
    transmute(acres_burned = ACRES_BURNED,
                        cause = FIREGCAUSE_LABEL_NM,
                        year = str_sub(DSCVR_DT, 1, 4)) %>%
    st_as_sf() %>%
    filter(year == 2023,
                  acres_burned > 0)

fires_ne <- fires %>%
    filter(year == 2023,
                  cause == "Power Gen") %>%
    st_filter(counties %>%
                            filter(NAME %in% c("Pend Oreille", "Sp
    mutate(x = st_coordinates(geometry)[,1],
                  y = st_coordinates(geometry)[,2])

tot <- sum(fires_ne$acres_burned)

tot_2023 <- sum(fires$acres_burned)

pct <- tot/tot_2023

anchor <- st_centroid(st_union(fires_ne)) %>%
```



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

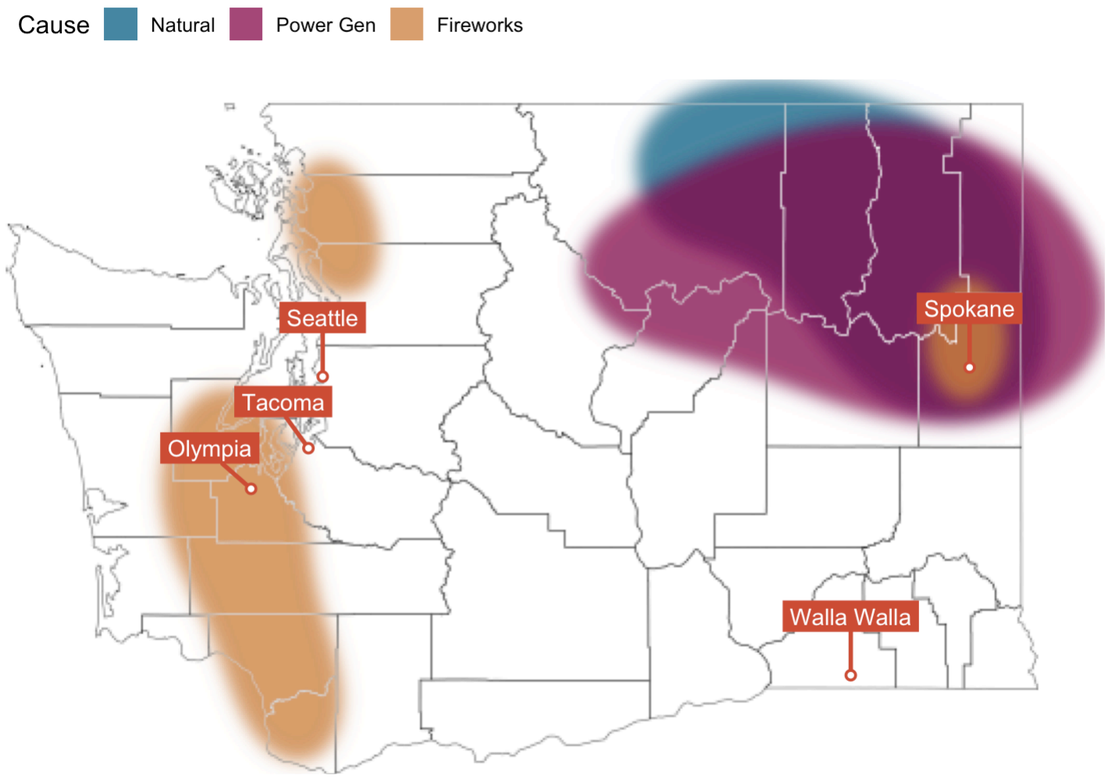
```
all_fires <- read_csv("/Users/sherrill/Projects/CascadiaR_Cartogra
  st_as_sf(coords = c("LON_COORD", "LAT_COORD"), crs = 4326) %>%
    st_transform(3857) %>%
    transmute(acres_burned = ACRES_BURNED,
                         cause = FIREGCAUSE_LABEL_NM,
                         year = str_sub(DSCVR_DT, 1, 4)) %>%
    st_as_sf()

fires <- all_fires %>%
    filter(year == 2023,
                  acres_burned > 0)

fires_ne <- fires %>%
    filter(year == 2023,
                  cause == "Power Gen") %>%
    st_filter(counties %>%
                           filter(NAME %in% c("Pend Oreille", "Sp
    mutate(x = st_coordinates(geometry)[,1],
                  y = st_coordinates(geometry)[,2])

tot <- sum(fires_ne$acres_burned)

tot_2023 <- sum(fires$acres_burned)

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

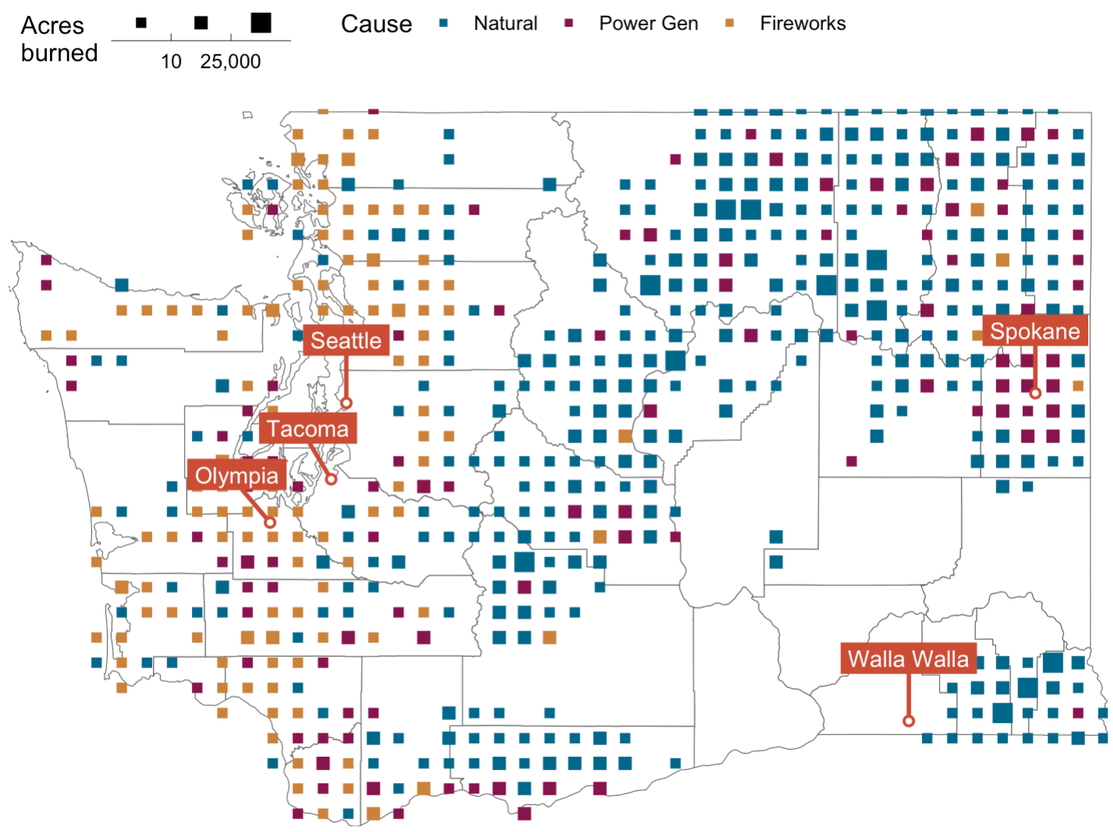
```
fires_hdr <- fires %>%
    mutate(x = st_coordinates(geometry)[,1],
                y = st_coordinates(geometry)[,2]) %>%
    st_drop_geometry() %>%
    filter(cause %in% c("Natural", "Power Gen", "Fireworks")) %>%
    mutate(cause = ordered(cause,
                                                levels = c("Natur


ggplot() +
    as_reference(with_blur(
        geom_hdr(
            data = fires_hdr,
            aes(
                x = x,
                y = y,
                group = cause,
                fill = cause
            ),
            probs = c(.33),
            n = 300,
            # Expand the limits to give the geometry some breathin
            xlim = st_bbox(counties)[c(1, 3)] * c(.9, 1.1),
            ylim = st_bbox(counties)[c(2, 4)] * c(.9, 1.1),
            alpha = .8
        ),
```



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

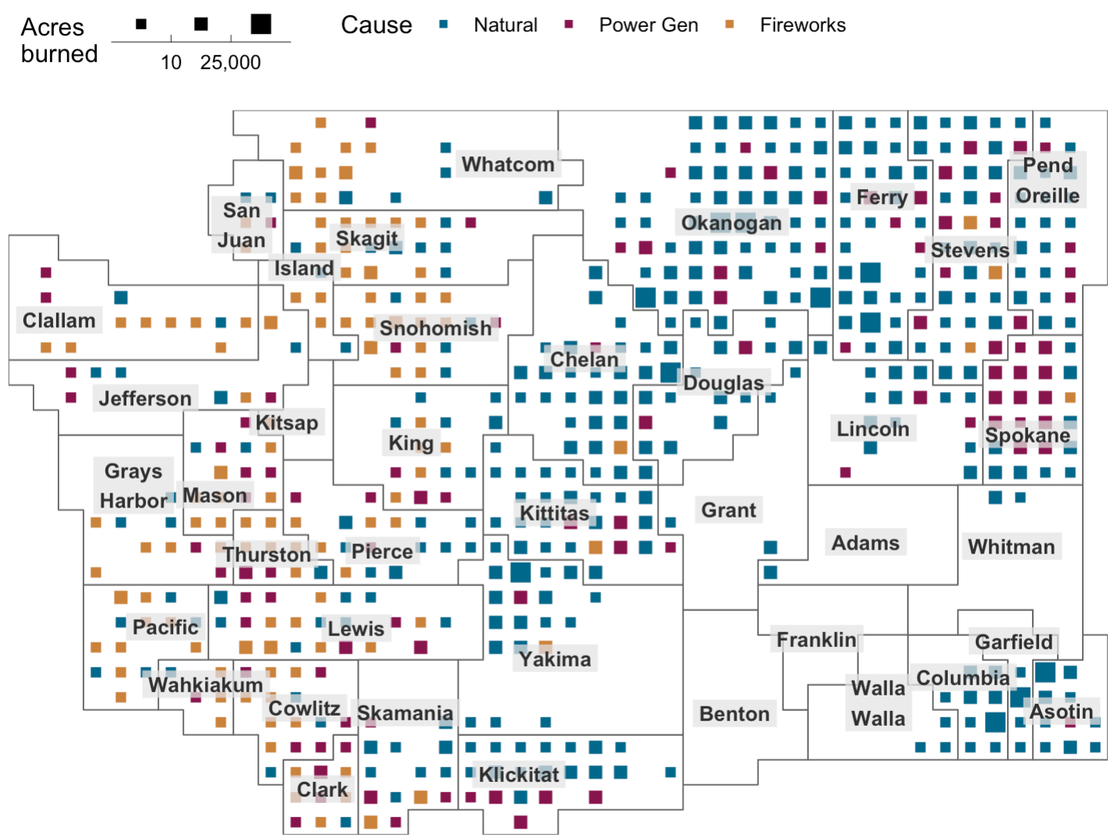
```
wa_grid <- st_make_grid(counties,
                                                cellsize = 20000)
    st_sf() %>%
    rowid_to_column("gridid")

fire_grid <- st_join(wa_grid, all_fires %>%
                                            filter(acres_burned >
                                                     cause %in
    group_by(gridid, cause) %>%
    summarise(acres_burned = sum(acres_burned)) %>%
    group_by(gridid) %>%
    filter(acres_burned == max(acres_burned, na.rm = TRUE)) %>%
    ungroup() %>%
    mutate(acres_burned = ifelse(is.na(acres_burned), 0, acres_bur
    st_centroid() %>%
    mutate(cause = ordered(cause,
                                              levels = c("Natur

ggplot() +
    geom_sf(
        data = counties,
        fill = "white",
        color = "gray40",
        linewidth = .2
    ) +
    geom_sf(data = fire_grid,
```



Patterns

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

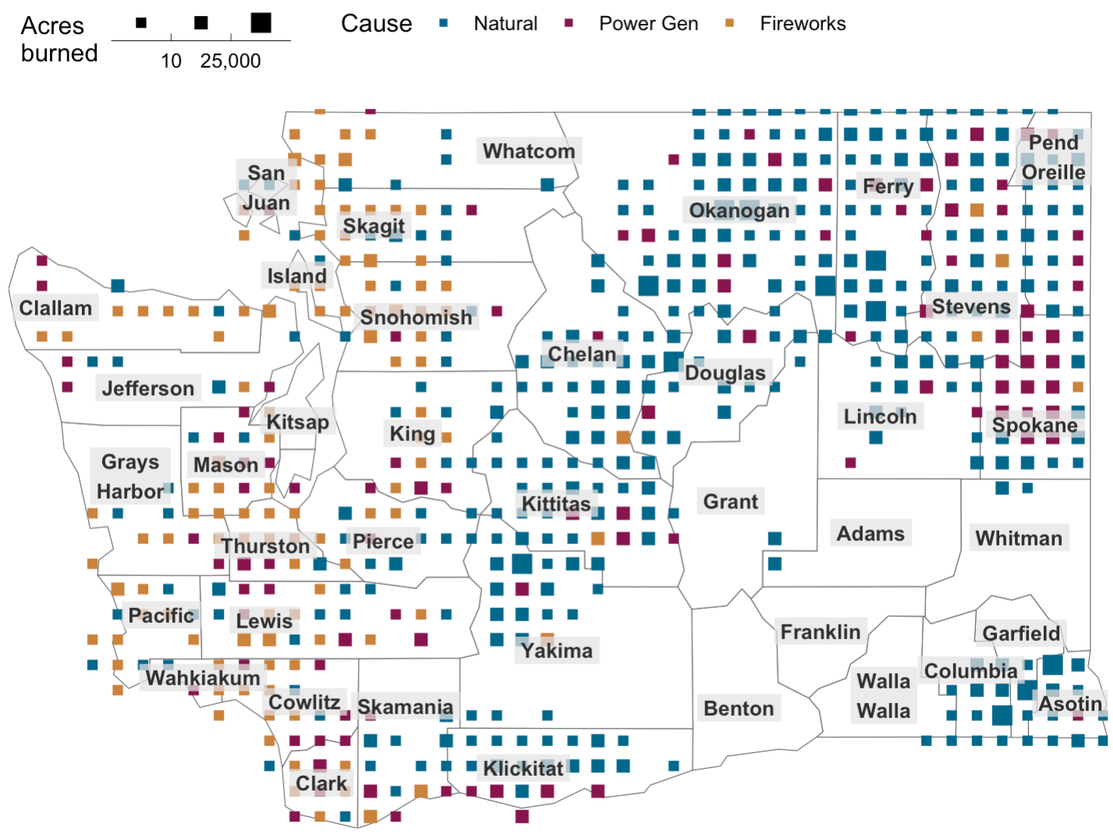
```
wa_grid <- st_make_grid(counties,
                                                cellsize = 20000)
    st_sf() %>%
    rowid_to_column("gridid")

county_grid <- wa_grid %>%
    st_join(counties, largest = TRUE, left = FALSE) %>%
    group_by(county = NAME) %>%
    summarise()

county_grid_lbls <- county_grid %>%
    pull(geometry) %>%
  st_inscribed_circle() %>%
  st_sf() %>%
  filter(!st_is_empty(geometry)) %>%
  st_centroid() %>%
  transmute(name = county_grid$county,
                    x = st_coordinates(.)[, 1],
                    y = st_coordinates(.)[, 2])

fire_grid <- st_join(wa_grid, all_fires %>%
                                          filter(acres_burned >
                                                   cause %in
    group_by(gridid, cause) %>%
    summarise(acres_burned = sum(acres_burned)) %>%
    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

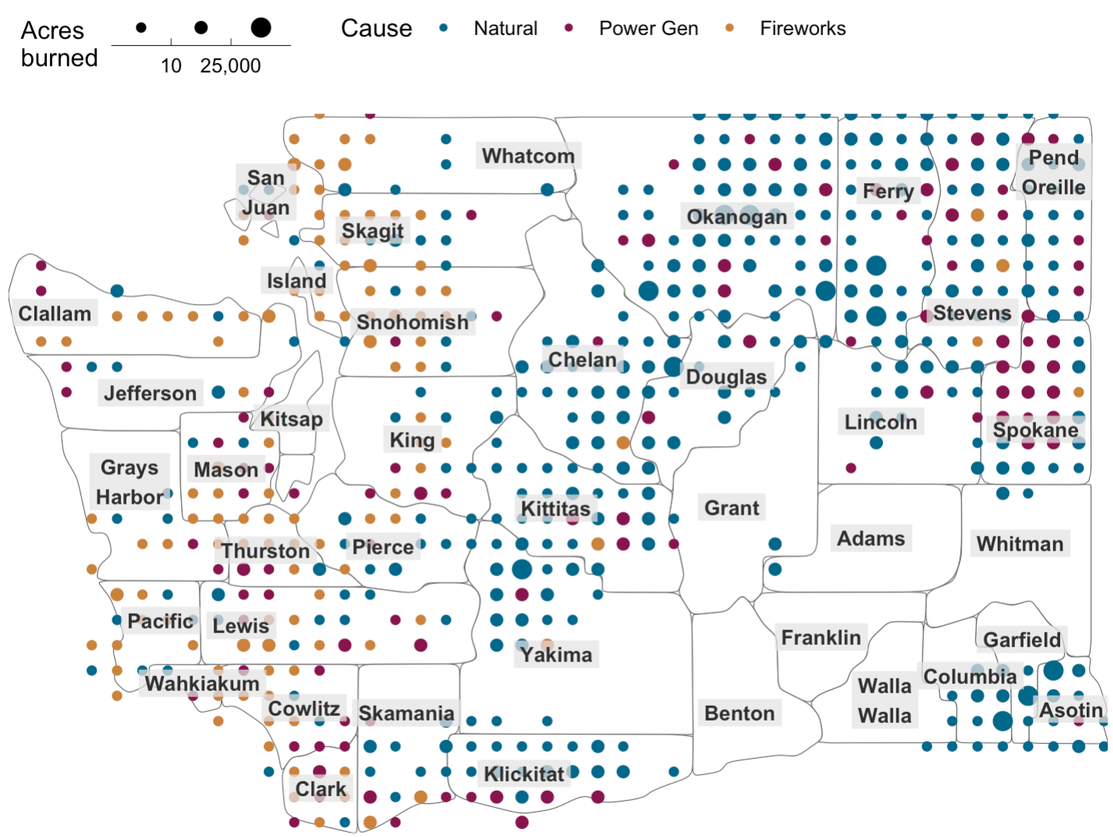
```r
counties_simp <- ms_simplify(counties, keep = .02, keep_shapes = T

counties_simp_lbls <- counties_simp %>%
    pull(geometry) %>%
  st_inscribed_circle() %>%
  st_sf() %>%
  filter(!st_is_empty(geometry)) %>%
  st_centroid() %>%
  transmute(name = counties_simp$NAME,
                    x = st_coordinates(.)[, 1],
                    y = st_coordinates(.)[, 2])

ggplot() +
    geom_sf(
        data = counties_simp,
        fill = "white",
        color = "gray40",
        linewidth = .2
    ) +
    geom_sf(data = fire_grid,
                    aes(color = cause, size = acres_burned),
                    shape = 15) +
    geom_label(
        data = counties_simp_lbls,
        aes(
            x = x,
```



Simplification

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

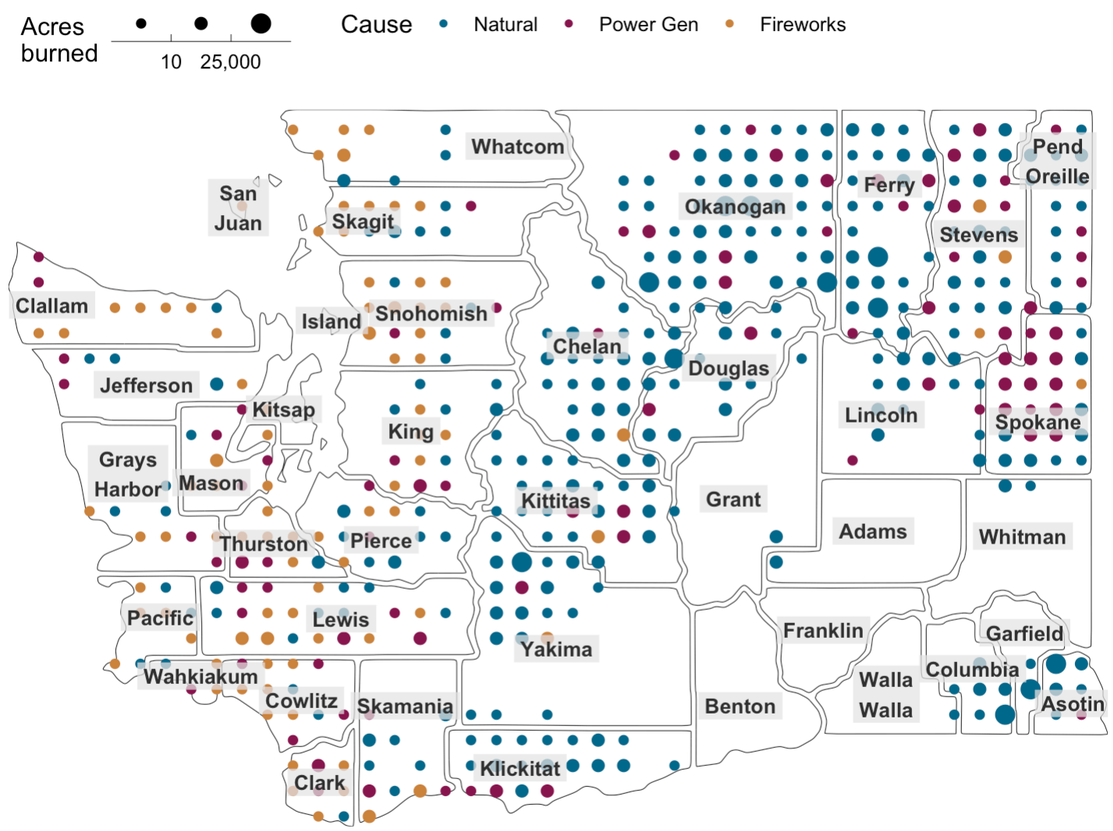
```r
counties_simp <- counties %>%
    ms_simplify(keep = .02, keep_shapes = TRUE) %>%
    smooth(method = "ksmooth",
                  smoothness = .35) %>%
    # Sometimes you can create geometry errors when smoothing, so
    st_make_valid()

counties_simp_lbls <- counties_simp %>%
    pull(geometry) %>%
  st_inscribed_circle() %>%
  st_sf() %>%
  filter(!st_is_empty(geometry)) %>%
  st_centroid() %>%
  transmute(name = counties_simp$NAME,
                    x = st_coordinates(.)[, 1],
                    y = st_coordinates(.)[, 2])

ggplot() +
    geom_sf(
        data = counties_simp,
        fill = "white",
        color = "gray40",
        linewidth = .2
    ) +
    geom_sf(data = fire_grid,
                    aes(color = cause, size = acres_burned),
```



Simplification

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

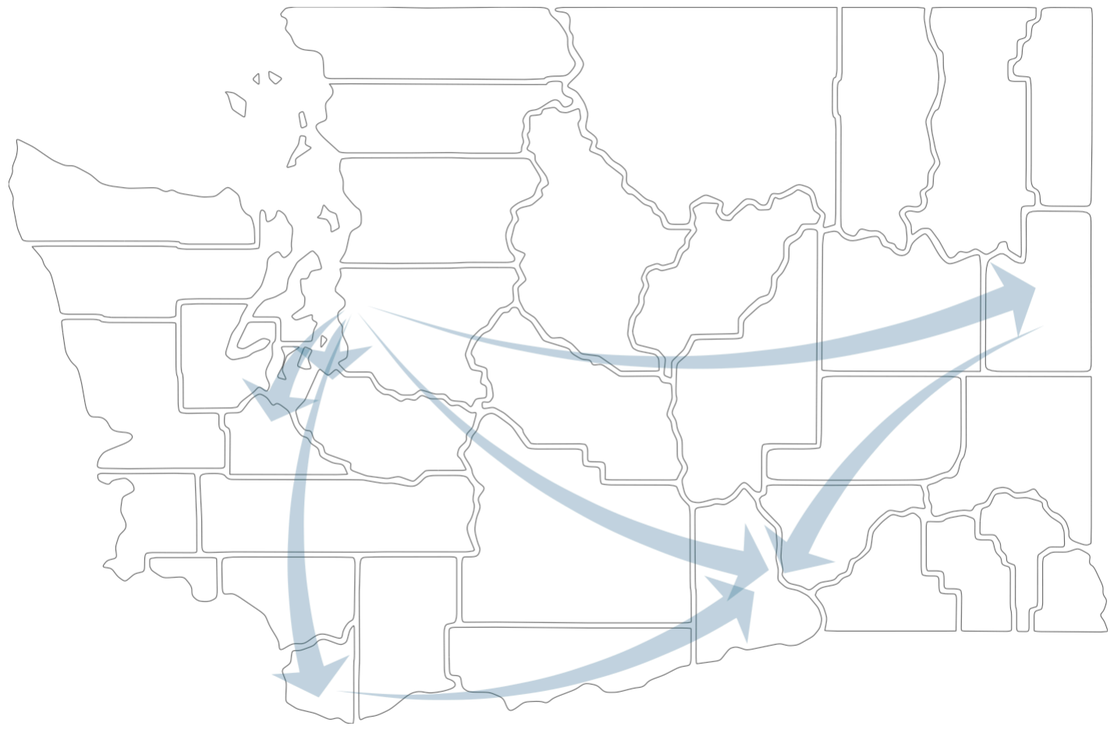
```r
counties_simp <- counties %>%
    st_buffer(-2000) %>%
    ms_simplify(keep = .02, keep_shapes = TRUE) %>%
    smooth(method = "ksmooth", smoothness = .35) %>%
    # Sometimes you can create geometry errors when smoothing, so
    st_make_valid()

counties_simp_lbls <- counties_simp %>%
    pull(geometry) %>%
  st_inscribed_circle() %>%
  st_sf() %>%
  filter(!st_is_empty(geometry)) %>%
  st_centroid() %>%
  transmute(name = counties_simp$NAME,
                    x = st_coordinates(.)[, 1],
                    y = st_coordinates(.)[, 2])

ggplot() +
    geom_sf(
        data = counties_simp,
        fill = "white",
        color = "gray20",
        linewidth = .2
    ) +
    geom_sf(
        data = fire_grid %>%
```



Symbols - Lines

📎 Brand new package {ggarrow} 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!

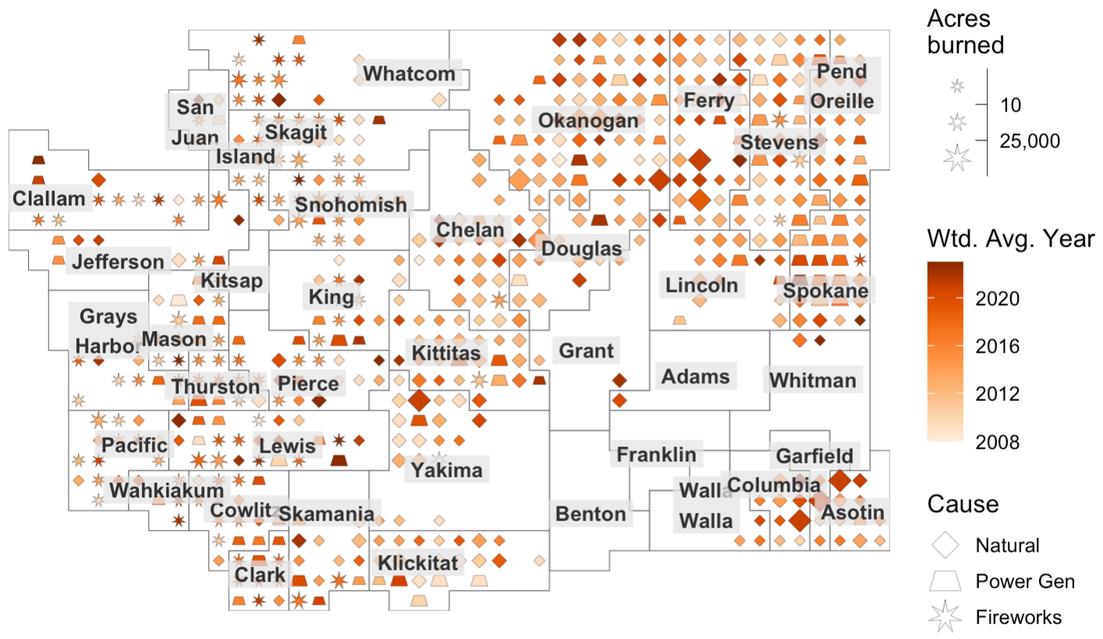
```
lines <- st_as_sfc(c("LINESTRING (-122.3207 47.52453, -122.418 47.
"LINESTRING (-122.2829 47.54782, -122.5151 45.6868)", "LINESTRING
"LINESTRING (-122.2388 47.55275, -119.3356 46.312)", "LINESTRING (
"LINESTRING (-117.3921 47.49202, -119.2271 46.29595)"),
crs = 4326) %>%
    st_transform(3857) %>%
    st_sf() %>%
    rowid_to_column("lineid") %>%
    st_cast("POINT") %>%
    mutate(x = st_coordinates(.)[,1],
                y = st_coordinates(.)[,2]) %>%
    st_drop_geometry() %>%
    group_by(lineid) %>%
    mutate(pt = 1:2) %>%
    pivot_wider(names_from = pt,
                           values_from = c(x, y))

ggplot() +
    geom_sf(
        data = counties_simp,
        fill = "white",
        color = "gray40",
        linewidth = .2
    ) +
    geom_arrow_curve(
        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

```
star_fire_grid <- st_join(wa_grid, all_fires %>%
                                       filter(acres_burned >
                                                year >= 2
                                                cause %in
                                                !is.na(ca
                                       left = FALSE) %>%
    st_drop_geometry() %>%
    group_by(gridid, year, cause) %>%
    summarise(acres_burned = sum(acres_burned)) %>%
    ungroup() %>%
    complete(year, cause, nesting(gridid), fill = list(acres_burne
    filter(!is.na(cause)) %>%
    group_by(gridid, cause) %>%
    mutate(cause_acres = sum(acres_burned)) %>%
    group_by(gridid) %>%
    mutate(main_cause = first(cause[which(cause_acres == max(cause
    group_by(gridid, main_cause, year) %>%
    summarise(acres_burned = ifelse(is.na(acres_burned), 0, acres_
    group_by(gridid, main_cause) %>%
    summarise(year = weighted.mean(as.numeric(year), w = acres_bur
                      acres_burned = sum(acres_burned)) %>%
    ungroup() %>%
    inner_join(wa_grid) %>%
    st_as_sf() %>%
    st_centroid() %>%
    mutate(main_cause = ordered(main_cause,
```



Symbols - Points

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

```
# Read in a downloaded static SVG
svg_text <- paste(readLines("./01_data/01_raw/clippy.svg"), collap

ggplot() +
    geom_sf(
        data = counties,
        fill = "white",
        color = "gray40",
        linewidth = .2
    ) +
    geom_point_svg(
        data = clippy[1,],
        aes(x = x, y = y),
        size = 5,
        svg = svg_text,
        svg_width = 600,
        svg_height = 800,
        defaults = list(fill_inner = 'white', fill_outer = 'red')
    ) +
    geom_label_repel(
        data = clippy[1,],
        aes(
            x = x,
            y = y,
            label = str_wrap(lines, 24)
        ),
```

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!