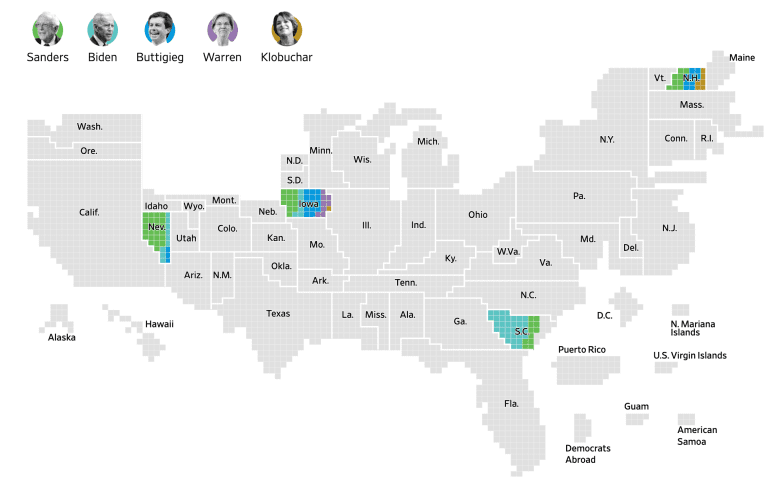
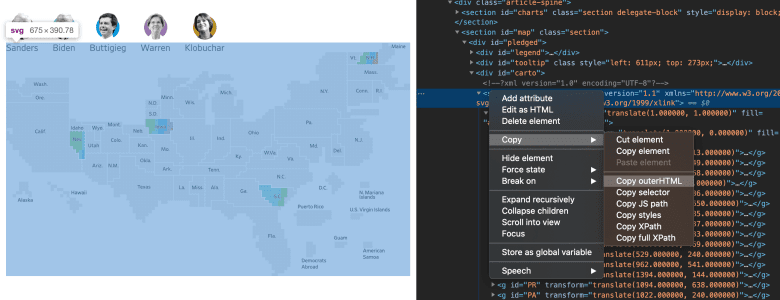
For folks who are smart enough not to go near Twitter, I’ve been on a hiatus from the platform insofar as reading the Twitter feed goes. “Why” isn’t the subject of this post so I won’t go into it, but I’ve broken this half-NYE resolution on more than one occasion and am very glad I did so late January when I caught a RT of this tweet by WSJ’s Brian McGill:

You guys! I spent 2 days making a cartogram by hand with 3,979 freaking squares on it. One for each delegate to the Democratic National Convention. Never doing it again.\* Apologies to Georgia, you got the short end of the stick shape-wise.  
.  
.  
.  
.  


I kinda wanted to try to make a woefully imperfect static version of it in R with {ggplot2} so poked around at that URL’s XHR objects and javascript to see if I could find the cartogram and the data source.

The data source was easy as it’s an XHR loaded JSON file: <https://asset.wsj.net/wsjnewsgraphics/election/2020/delegates.json>.

The cartogram bits… were *not*. Brian’s two-days of manual effort still needed to be put into something that goes onto a web page and news outlets are super-talented at making compact, fast-loading interactive visualizations, which means one tool they use is “webpack”-esque tools to combine many small javascript files into one. I did traipse through it seeing if there as a back-end JSON or CSV somewhere but could not locate it. However, their cartogram library builds the SVG you see on the page. If you use Developer Tools to inspect any element of the SVG then copy the whole SVG “outer HTML” and save it to a local file:



That SVG has three top layer groups and has some wicked transforms in it. There was no way I was going to attempt a {statebins}-esque approach to this copycat project (i.e. convert the squares to a grid and map things manually like Brian did) but I had an idea and used Adobe Illustrator to remove the state names layer and the background polygon layer, then “flatten” the image (which — to over-simplify the explanation — flattens all the transforms), and save it back out.

Then, I added a some magic metadata prescribed by svg2geojson package to turn the SVG into a GeoJSON file (which {sf} can read!). (That sentence just made real cartographers & geocomp’ers weep, btw).

Now, that I had something R could use in a bit of an easier fashion there was still work to be done. The SVG 1-px elements ended up coming across as POLYGONs and many, many more point-squares came along for the ride (in retrospect, I think they may have been the borders around the states, more on that in a bit).

I used {purrr} and {st\_coordinates} to figure out where all the 1-px “polygons” were in the {sf} object and isolated them, then added an index field (1:n, n being the number of delegate squares for a given state).

I read in the original SVG with {xml2} and extracted the named state groups. Thankfully the order and number of “blocks” matched the filtered {sf} object. I merged them together, turned the 1-px POLYGONs into POINTs, and made the final {sf} object which I put in the nascent {catchpole} package (location below). Here’s a quick view of it using plot():

library(catchpole) # hrbrmstr/catchpole

plot(delegates\_map()[1])

delegates\_map()

## Simple feature collection with 3979 features and 2 fields

## geometry type: POINT

## dimension: XY

## bbox: xmin: -121.9723 ymin: 37.36802 xmax: -121.9581 ymax: 37.37453

## epsg (SRID): 4326

## proj4string: +proj=longlat +datum=WGS84 +no\_defs

## First 10 features:

## state idx geometry

## 1 WY 1 POINT (-121.9693 37.37221)

## 2 WY 2 POINT (-121.9693 37.37212)

## 3 WY 3 POINT (-121.9691 37.37221)

## 4 WY 4 POINT (-121.9691 37.37212)

## 5 WY 5 POINT (-121.9691 37.37203)

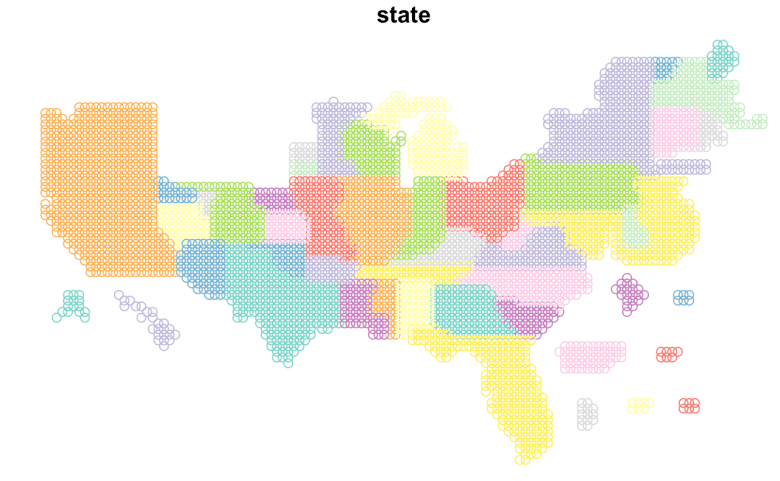
## 6 WY 6 POINT (-121.9691 37.37194)

## 7 WY 7 POINT (-121.9691 37.37185)

## 8 WY 8 POINT (-121.969 37.37221)

## 9 WY 9 POINT (-121.969 37.37212)

## 10 WY 10 POINT (-121.969 37.37203)



All that was needed was to try it out with the real data.

I simplified that process quite a bit in {catchpole} but also made it possible to work with the individual bits on your own. {gg\_catchpole()} will fetch the WSJ delegate JSON and build the basic map for you using my dark “ipsum” theme:

library(sf)

library(catchpole) # hrbrmstr/catchpole

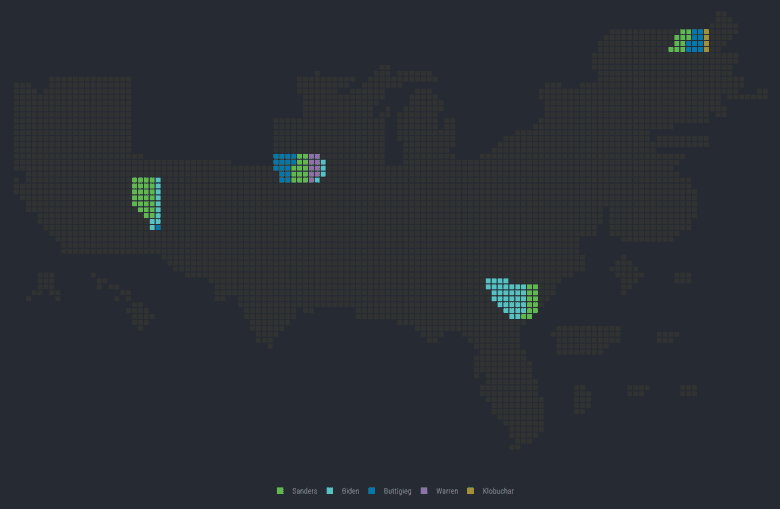
library(hrbrthemes)

library(tidyverse)

gg\_catchpole() +

theme\_ft\_rc(grid="") +

theme(legend.position = "bottom")



**BONUS!**

Now that you have the WSJ JSON file, you can do other, basic visualizations with it:

library(hrbrthemes)

library(waffle)

library(geofacet)

library(tidyverse)

jsonlite::fromJSON(

url("<https://asset.wsj.net/wsjnewsgraphics/election/2020/delegates.json>"),

simplifyDataFrame = FALSE

) -> del

c(

"Biden" = "#5ac4c2",

"Sanders" = "#63bc51",

"Warren" = "#9574ae",

"Buttigieg" = "#007bb1",

"Klobuchar" = "#af973a",

"Bloomberg" = "#AA4671",

"Steyer" = "#4E4EAA",

"Yang" = "#C76C48",

"Gabbard" = "#7B8097"

) -> dcols

bind\_cols(del$data$US$delCount) %>%

gather(candidate, delegates) %>%

filter(delegates > 0) %>%

arrange(desc(delegates)) %>%

mutate(candidate = fct\_inorder(candidate)) %>%

ggplot(aes(candidate, delegates)) +

geom\_col(fill = ggthemes::tableau\_color\_pal()(1), width = 0.55) +

labs(

x = NULL, y = "# Delegates",

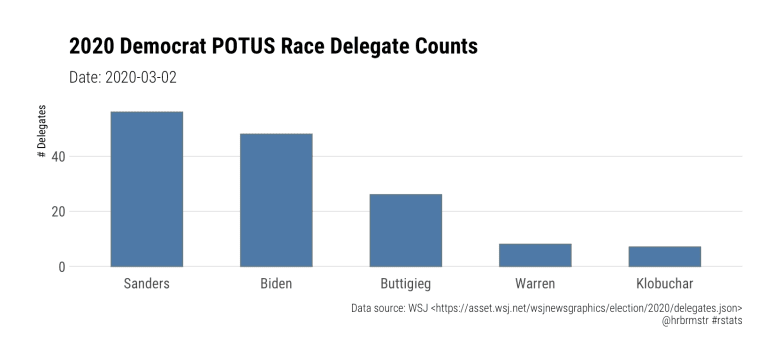
title = "2020 Democrat POTUS Race Delegate Counts",

subtitle = sprintf("Date: %s", Sys.Date()),

caption = "Data source: WSJ \n@hrbrmstr #rstats"

) +

theme\_ipsum\_rc(grid="Y")



bind\_cols(del$data$US$delCount) %>%

gather(candidate, delegates) %>%

filter(delegates > 0) %>%

arrange(desc(delegates)) %>%

mutate(candidate = fct\_inorder(candidate)) %>%

ggplot(aes(fill=candidate, values=delegates)) +

geom\_waffle(color = "white", size = 0.5) +

scale\_fill\_manual(name = NULL, values = dcols) +

coord\_fixed() +

labs(

x = NULL, y = "# Delegates",

title = "2020 Democrat POTUS Race Delegate Counts",

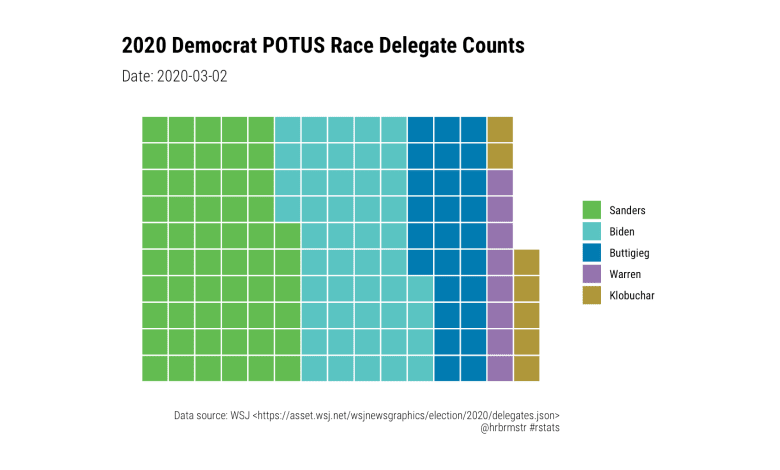
subtitle = sprintf("Date: %s", Sys.Date()),

caption = "Data source: WSJ \n@hrbrmstr #rstats"

) +

theme\_ipsum\_rc(grid="") +

theme\_enhance\_waffle()



state\_del <- del

state\_del$data[["US"]] <- NULL

map\_df(state\_del$data, ~bind\_cols(.x$delCount), .id = "state") %>%

gather(candidate, delegates, -state) %>%

filter(delegates > 0) %>%

ggplot(aes(candidate, delegates)) +

geom\_col(aes(fill = candidate), col = NA, width = 0.55) +

scale\_fill\_manual(name = NULL, values = dcols) +

facet\_geo(~state) +

labs(

x = NULL, y = "# Delegates",

title = "2020 Democrat POTUS Race Delegate Counts by State",

subtitle = sprintf("Date: %s", Sys.Date()),

caption = "Data source: WSJ \n@hrbrmstr #rstats"

) +

theme\_ipsum\_rc(grid="Y") +

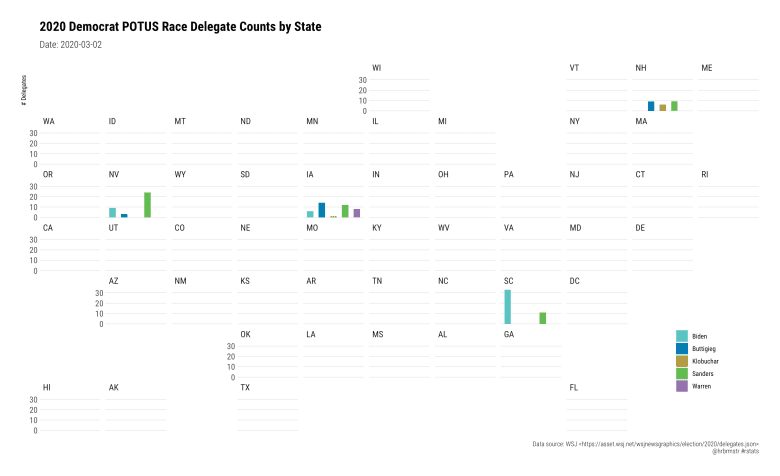
theme(axis.text.x = element\_blank()) +

theme(panel.spacing.x = unit(0.5, "lines")) +

theme(panel.spacing.y = unit(0.1, "lines")) +

theme(legend.position = c(0.95, 0.1)) +

theme(legend.justification = c(1, 0))



**FIN**

More work needs to be done on the map and {catchpole} itself but there’s a sufficient base for others to experiment with (PRs and your own blog posts welcome!).

W/r/t “more on that later” bits: The extra polygons were very likely borders and I think borders would help the cartogram, but we can make them with {sf}, too. We can also add in a layer for state names and/or just figure out the centroid for each point grouping (with {sf}) and get places for labels that way). Not sure I’ll have time for any of that (this whole process went quickly, believe it or not).

Also: ggiraph::geom\_sf\_interactive() can be used as a poor-dude’s popup to turn this (quickly) into an interactive piece.