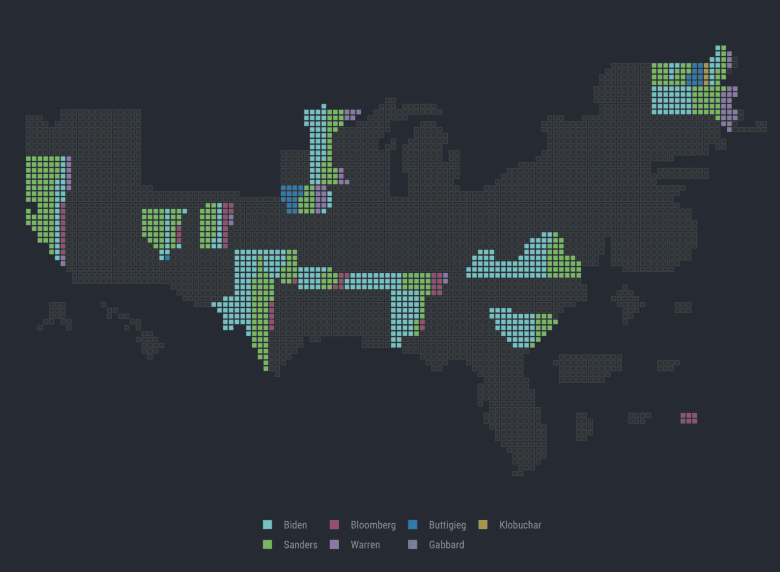
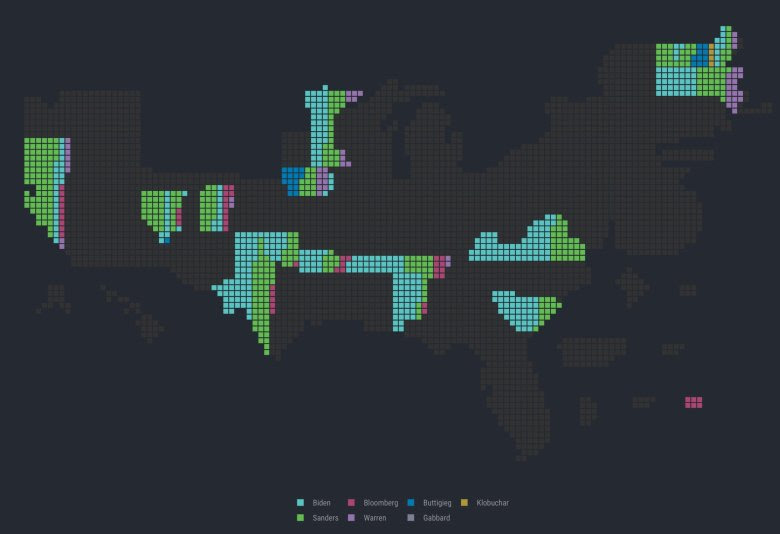
Über Tuesday has come and *almost* gone (some state results will take a while to coalesce) and I’m relieved to say that {catchpole} did indeed work,:



If we tweak the buffer space around the squares, I think the cartogram looks better:



but, you should likely use a different palette

I noted in the previous post that borders might be possible. While I haven’t solved that use-case for individual states, I did manage to come up with a method for making a light version of the cartogram usable:

library(sf)

library(hrbrthemes)

library(catchpole)

library(tidyverse)

delegates <- read\_delegates()

candidates\_expanded <- expand\_candidates()

gsf <- left\_join(delegates\_map(), candidates\_expanded, by = c("state", "idx"))

m <- delegates\_map()

# split off each "area" on the map so we can make a border+background

list(

setdiff(state.abb, c("HI", "AK")),

"AK", "HI", "DC", "VI", "PR", "MP", "GU", "DA", "AS"

) %>%

map(~{

suppressWarnings(suppressMessages(st\_buffer(

x = st\_union(m[m$state %in% .x, ]),

dist = 0.0001,

endCapStyle = "SQUARE"

)))

}) -> m\_borders

gg <- ggplot()

for (mb in m\_borders) {

gg <- gg + geom\_sf(data = mb, col = "#2b2b2b", size = 0.125)

}

gg +

geom\_sf(

data = gsf,

aes(fill = candidate),

col = "white", shape = 22, size = 3, stroke = 0.125

) +

scale\_fill\_manual(

name = NULL,

na.value = "#f0f0f0",

values = c(

"Biden" = '#f0027f',

"Sanders" = '#7fc97f',

"Warren" = '#beaed4',

"Buttigieg" = '#fdc086',

"Klobuchar" = '#ffff99',

"Gabbard" = '#386cb0',

"Bloomberg" = '#bf5b17'

),

limits = intersect(unique(delegates$candidate), names(delegates\_pal))

) +

guides(

fill = guide\_legend(

override.aes = list(size = 4)

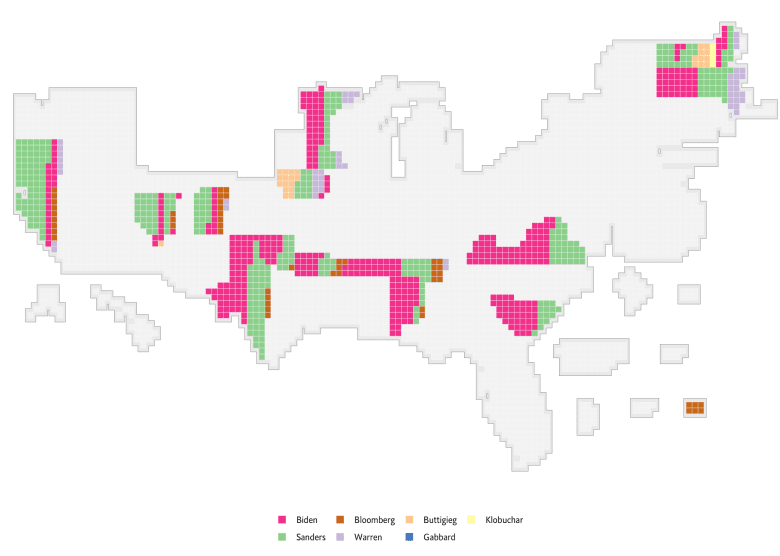
)

) +

coord\_sf(datum = NA) +

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

theme(legend.position = "bottom")



**{ssdeepr}**

Researcher pals over at Binary Edge [added web page hashing](https://blog.binaryedge.io/2020/03/02/webv2-the-future-for-web-scanning-at-binaryedge/) (pre- and post-javascript scraping) to their platform using [ssdeep](https://ssdeep-project.github.io/). This approach is in the category of context triggered piecewise hashes (CTPH) (or local sensitivity hashing) .

Since I’ll be working with BE’s data off-and-on and the ssdeep project has a well-crafted library (plus we might add ssdeep support at $DAYJOB), I went ahead and [packaged that up as well](https://cinc.rud.is/web/packages/ssdeepr/).

I recommend using the hash\_con() function if you need to read large blobs since it doesn’t require you to read everything into memory first (though hash\_file() doesn’t either, but that’s a direct low-level call to the underlying ssdeep library file reader and not as flexible as R connections are).

These types of hashes are great at seeing if something has changed on a website (or see how similar two things are to each other). For instance, how closely do CRAN mirror match the mothership?

library(ssdeepr) # see the links above for installation

cran1 <- hash\_con(url("<https://cran.r-project.org/web/packages/available_packages_by_date.html>"))

cran2 <- hash\_con(url("<https://cran.biotools.fr/web/packages/available_packages_by_date.html>"))

cran3 <- hash\_con(url("<https://cran.rstudio.org/web/packages/available_packages_by_date.html>"))

hash\_compare(cran1, cran2)

## [1] 0

hash\_compare(cran1, cran3)

## [1] 94

I picked on [cran.biotools.fr](http://cran.biotools.fr) as I saw they were well-behind CRAN-proper on the monitoring page.

I noted that BE was doing pre- and post-javascript hashing as well. Why, you may ask? Well, websites behave differently with javascript running, plus they can behave differently when different user-agents are set. Let’s grab a page from Wikipedia a few different ways to show how they are not alike at all, depending on the retrieval context. First, let’s grab some web content!

library(httr)

library(ssdeepr)

library(splashr)

# regular grab

h1 <- hash\_con(url("<https://en.wikipedia.org/wiki/Donald_Knuth>"))

# you need Splash running for javascript-enabled scraping this way

sp <- splash(host = "mysplashhost", user = "splashuser", pass = "splashpass")

# js-enabled with one ua

sp %>%

splash\_user\_agent(ua\_macos\_chrome) %>%

splash\_go("<https://en.wikipedia.org/wiki/Donald_Knuth>") %>%

splash\_wait(2) %>%

splash\_html(raw\_html = TRUE) -> js1

# js-enabled with another ua

sp %>%

splash\_user\_agent(ua\_ios\_safari) %>%

splash\_go("<https://en.wikipedia.org/wiki/Donald_Knuth>") %>%

splash\_wait(2) %>%

splash\_html(raw\_html = TRUE) -> js2

h2 <- hash\_raw(js1)

h3 <- hash\_raw(js2)

# same way {rvest} does it

res <- httr::GET("<https://en.wikipedia.org/wiki/Donald_Knuth>")

h4 <- hash\_raw(content(res, as = "raw"))

Now, let’s compare them:

hash\_compare(h1, h4) # {ssdeepr} built-in vs httr::GET() => not surprising that they're equal

## [1] 100

# things look way different with js-enabled

hash\_compare(h1, h2)

## [1] 0

hash\_compare(h1, h3)

## [1] 0

# and with variations between user-agents

hash\_compare(h2, h3)

## [1] 0

hash\_compare(h2, h4)

## [1] 0

# only doing this for completeness

hash\_compare(h3, h4)

## [1] 0

For this example, just content size would have been enough to tell the difference (mostly, note how the hashes are equal despite more characters coming back with the {httr} method):

length(js1)

## [1] 432914

length(js2)

## [1] 270538

nchar(

paste0(

readLines(url("<https://en.wikipedia.org/wiki/Donald_Knuth>")),

collapse = "\n"

)

)

## [1] 373078

length(content(res, as = "raw"))

## [1] 374099

**FIN**

If you were in a U.S. state with a primary yesterday and were eligible to vote (and had something to vote for, either a (D) candidate or a state/local bit of business) I sure hope you did!

The ssdeep library works on Windows, so I’ll be figuring out how to get that going in {ssdeepr} fairly soon (mostly to try out the Rtools 4.0 toolchain vs deliberately wanting to support legacy platforms).

As usual, drop issues/PRs/feature requests where you’re comfortable for any of these or other packages.