# Interactive Visualization

## Interactivity without Shiny 2

Dave Armstrong

May 27, 2020

# linking plots

One of the coolest things about these interactive plots is that they allow linking across plots.

- plotly has a built-in feature that does this.
  - The 'highlight' argument allows linking the plots.
- the R package `{crosstalk}` does this for several compatible plots in the htmlwidgets universe.

# within plotly

```
p1 <- ohio %>%
      highlight_key(~county) %>%
      plot_ly(x = ~lt25k, y=~repvote, text=~county,
              textposition="top", hoverinfo="x+y") %>%
      add_markers()

p2 <- ohio %>%
      highlight_key(~county) %>%
      plot_ly(x = ~over60_pop, y=~repvote, text=~county,
              textposition="top", hoverinfo="x+y") %>%
      add_markers()
```

```
subplot(p1, p2) %>%
  highlight(on = "plotly_selected", off="plotly_doubleclick", persistent=FALSE) %>%
  layout(height=400, width=800, shareY=TRUE, titleX=TRUE,
         xaxis=list(title="White Proportion"),
         xaxis2 = list(title="Over 60 Proportion"),
         yaxis = list(title="Republican Vote Share"))
```

See the plot (https://davidaarmstrong.github.io/deck2/sharing2.html#plotly-with-subplot)

# plotly highlight redux

- Easy to implement
- First plot talks to second, but not other way around.

# Crosstalk - scatterplots

Crosstalk (https://rstudio.github.io/crosstalk/) is a package that provides linkages across plots. Here's an example:

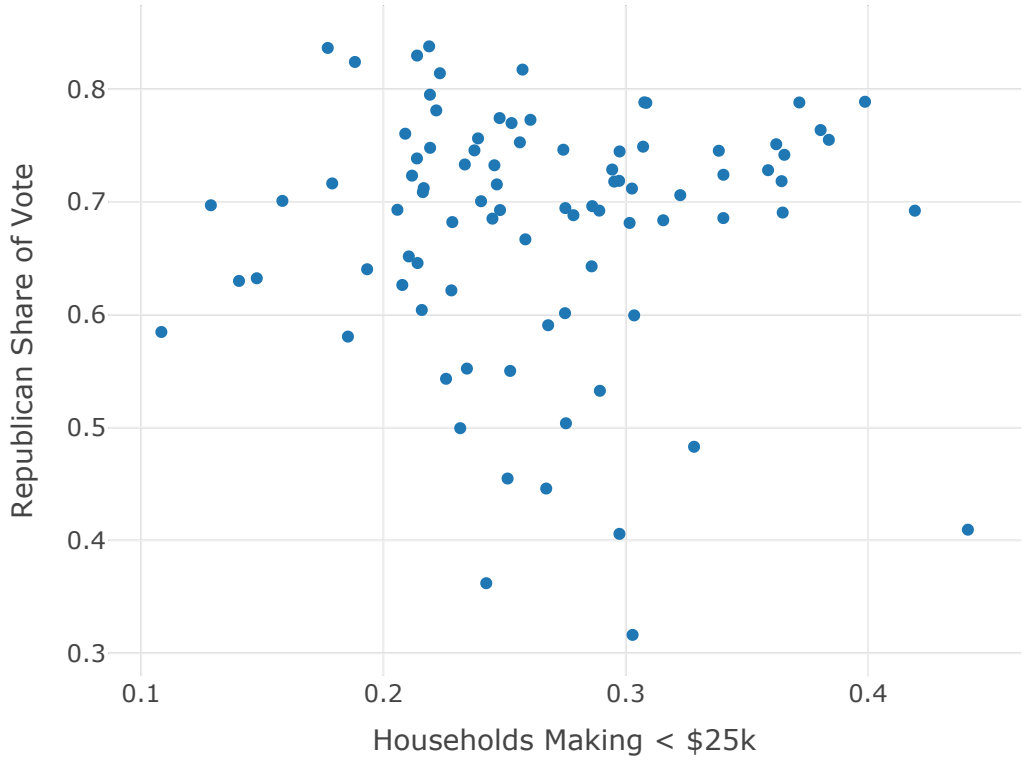
```
remotes::install_github("rstudio/crosstalk")
remotes::install_github("jcheng5/d3scatter")
remotes::install_github("rstudio/leaflet")
```

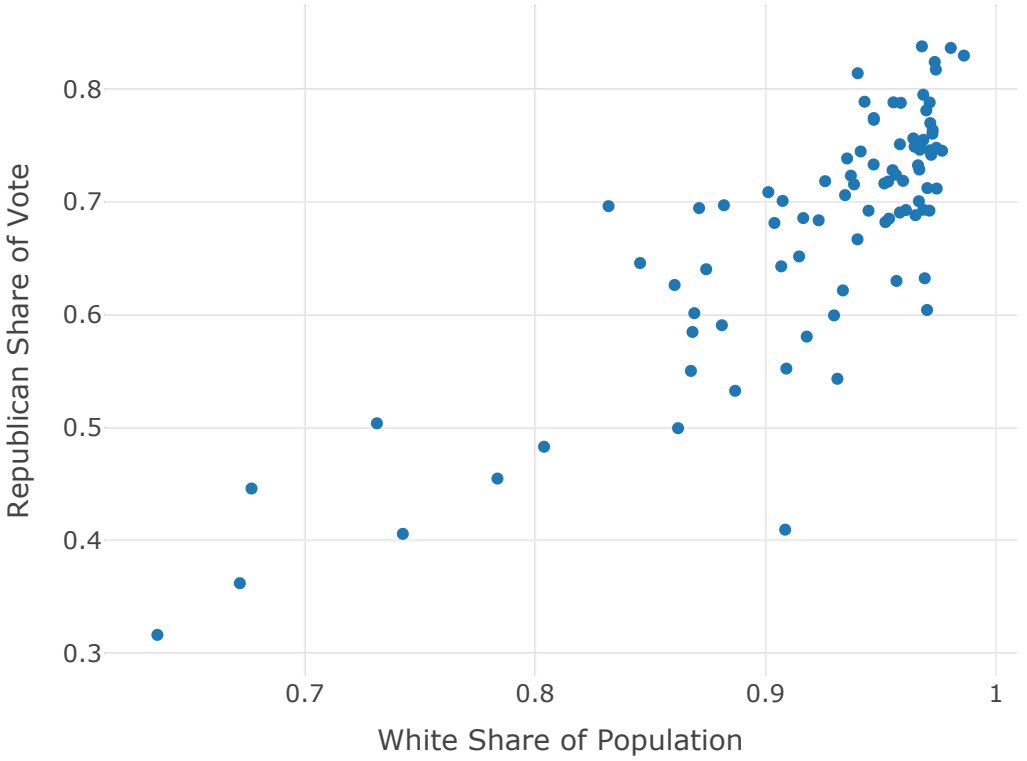
```
library(crosstalk)
s <- SharedData$new(ohio)

p1 <- s %>%
  plot_ly(x=~lt25k, y=~repvote, text=~county) %>%
  add_markers() %>%
  layout(xaxis=list(title="Households Making < $25k"),
         yaxis = list(title="Republican Share of Vote"))
p2 <- s %>%
  plot_ly(x=~white_pop, y=~repvote, text=~county) %>%
  add_markers() %>%
  layout(xaxis=list(title="White Share of Population"),
         yaxis = list(title="Republican Share of Vote"))
```

# plot

```
bscols(p1, p2)
```

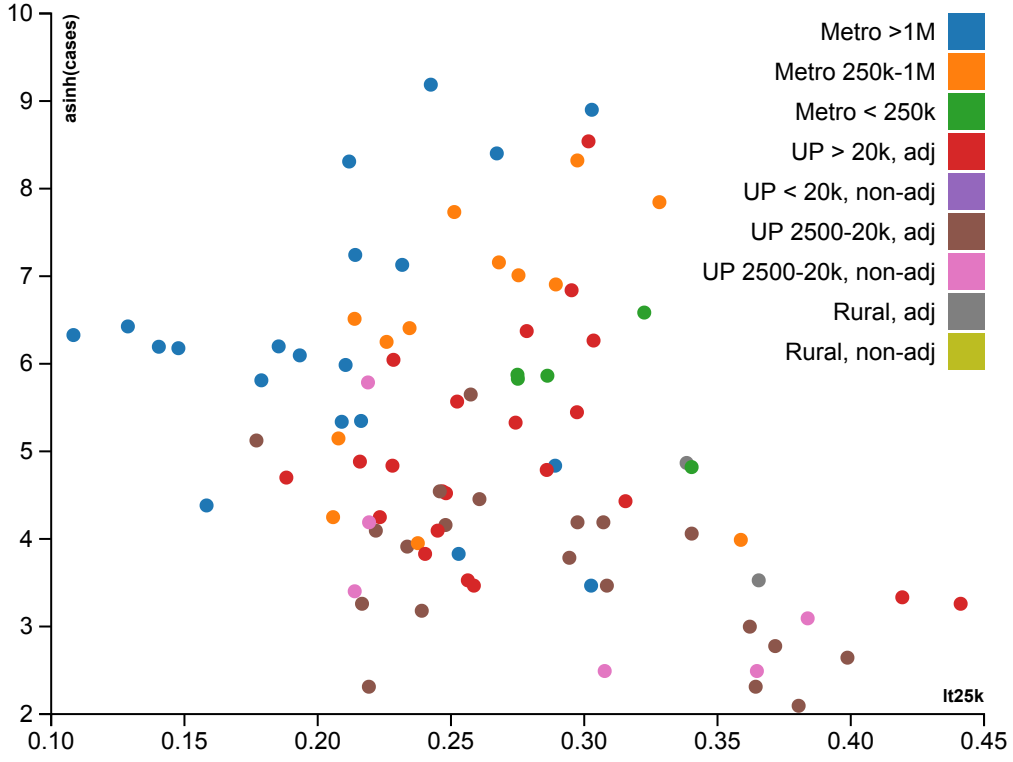


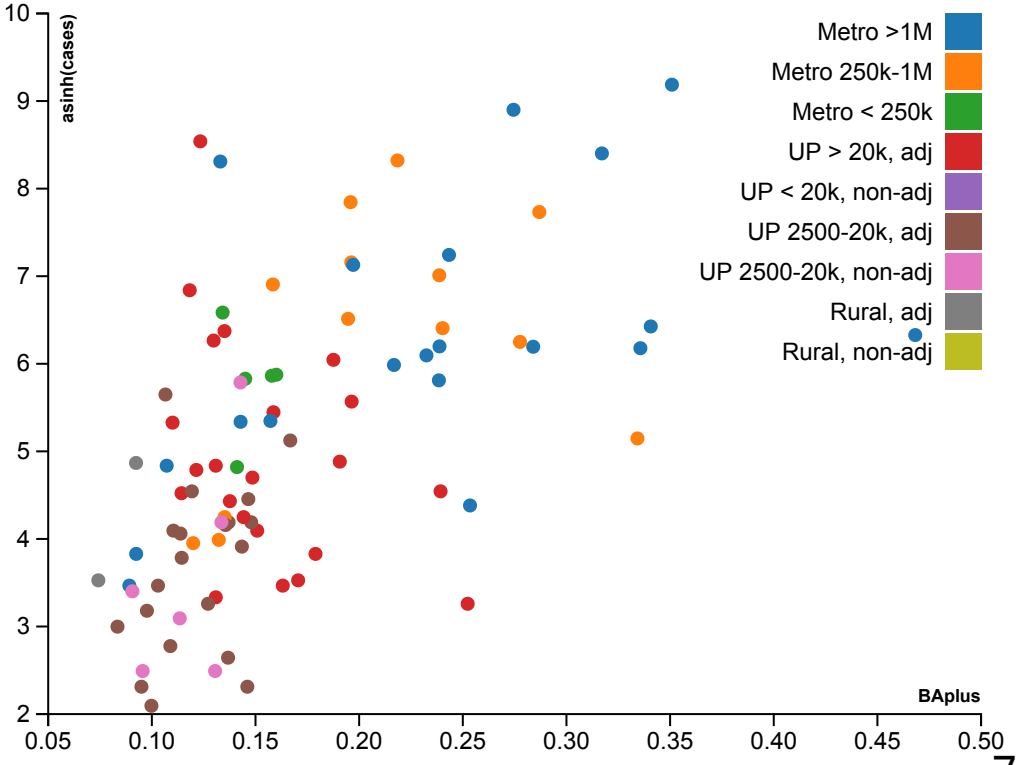



# crosstalk - d3scatter

```r
library(d3scatter)

bscols(
  d3scatter(s, ~lt25k, ~asinh(cases), ~urban_rural, width="100%", height=400),
  d3scatter(s, ~BAplus, ~asinh(cases), ~urban_rural, width="100%", height=400)
)
```



# crosstalk - map scatter data

```
library(rio)
library(DT)
library(mapview)
shoot <- import("stanford_mass_shooting.dta")

shoot <- shoot %>% select( latitude, longitude, date, city, numberofcivilianfatalities,
                           numberofcivilianinjured, numberofenforcementfatalities, numberofenforcementinjured)
shoot$date <- as.Date(shoot$date, format="%m/%d/%Y")
names(shoot) <- c("lat", "long", "date", "city", "num_civ_fatal", "num_civ_injur", "num_pol_fatal", "num_pol_injur")
shoot <- shoot %>% mutate(year = lubridate::year(date))


shoot2 <- SharedData$new(shoot)
library(leaflet)
bscols(widths = c(6,6),
  leaflet(shoot2, width="100%", height=400) %>% addTiles() %>% addMarkers(),
  datatable(shoot2, extensions="Scroller", style="bootstrap", class="compact", width="100%",
            colnames=c("Longitude", "Latitude", "Date", "City", "# Civilians Killed",
                       "# Civilians Injured", "# Police Killed", "# Police Injured"),
            options = list(paging=FALSE, pageLength = 20, scrollY = "200px"),
            fillContainer = TRUE) %>%
    formatRound(columns=c("lat", "long"), digits=1)
)
```

See the plot (https://davidaarmstrong.github.io/deck2/sharing2.html#crosstalk-with-leaflet-and-dt)

# selection and filtering

```
bscols(widths = c(6,6),
  list(
    leaflet(shoot2, width="100%", height=400) %>% addTiles() %>% addMarkers(),
    filter_slider("yr", "Year", shoot2, ~year, sep="")
  ),
  datatable(shoot2, extensions="Scroller", style="bootstrap", class="compact", width="100%",
            colnames=c("Longitude", "Latitude", "Date", "City", "# Civilians Killed",
                       "# Civilians Injured", "# Police Killed", "# Police Injured"),
            options = list(paging=FALSE, pageLength = 20, scrollY = "200px"),
            fillContainer = TRUE) %>%
    formatRound(columns=c("lat", "long"), digits=1)
)
```

See the plot (https://davidaarmstrong.github.io/deck2/sharing2.html#selection-and-filtering)

# You try it

Using the `scad_africa.csv` file, investigate some of the methods we've discussed so far. You can find the codebook here (https://www.strausscenter.org/wp-content/uploads/SCAD_33_Codebook.pdf). There are two other filters,

- `filter_checkbox()`
- `filter_select()`

Give those a try - using them to filter countries or certain types of events.

20:00

# Dahsboards

Dashboards have one key difference from full-blown Shiny apps.

- They don't do analytics on the server-side. All of the action takes place on the client side.
- You can build shiny dashboards, but they must be hosted on a shiny server.
- We're going to use the `flexdashboard` template for building a our dashboard.

```
install.packages("flexdashboard")
```

Go to File $\rightarrow$ New $\rightarrow$ RMarkdown and choose "From Template".

- When the dialog box comes up, choose "Flex Dashboard"

# flexdashboard

There is a ton of great advice about using flexdashboards here (https://rmarkdown.rstudio.com/flexdashboard/using.html).

- You can set the layout as either row or column oriented and can then identify where you want the columns or rows and how you want them distributed.
- You can build multi-page dashboards with storyboards.
- You can include any element and text you want in the dashboard.

Let's look at the template we get when we start.

# first dashboard

Let's make a static dashboard using the `diamonds` data in the `{ggplot2}` package. Let's have the following three elements, we'll work through it in pieces:

- Scatterplot of `price` (y) by `carat` (x), facet on `cut`, color by `color` where "Chart A" is:
- Bar plot of `clarity` where "Chart B" is.
- Heat Map of `color` and `clarity` where the color fill is proportional to average `price` just for diamonds that are between .75 and 1 carat where "Chart C" is.

My dashboard for this problem is here (https://quantoid.net/files/iviz/diamonds-dashboard.html)

05:00

# COVID Dashboard

I want you to make a dashboard that has the following elements:

1. A `plotly` map of the log of cases where "Chart A" is.
2. A `d3scatter` plot of `cases` by `repvote`.
3. A `datatable` using the (`DT`) package

Use the `crosstalk` package to make a `SharedData()` object so that these three elements will be linked.

15:00

# Your turn!

Give this a try with your own data - see what you can come up with.

Feel free to take a break, stretch your legs, etc... We'll reconvene in 15 minutes.

15:00

# parameterized RMarkdown

You can build parameters into your RMarkdown document.

- These aren't parameters that are changeable in the HTML,
- but you can generate different HTML documents by changing the call to `rmarkdown::render()`
- You add parameters as unquoted `key: value` pairs under the `parameters:` heading in the document's YAML header. For example

For example:

```
---
title: "Ohio Covid Map"
output:
  flexdashboard::flex_dashboard:
    theme: yeti
    orientation: columns
    vertical_layout: fill
params:
  state: Ohio
---
---
title: `r paste0(params$state, " COVID-19 Dashboard")`
---
```

# include parameters

You can access the parameter(s) with `params$state`.

- You could use the parameter to filter the data to different States.

For example:

```
rmarkdown::render("covid_dashboard_param.Rmd",
  params = list(state = "Ohio")
)
```

This will typeset a new document called `covid_dashboard_param.html` that has Ohio's data in it. To automate for many plots, you could execute a loop (chaing the `output_file` argument each time:

```
states <- c("Maine", "Michigan", "Wisconsin", "Maryland")
for(i in 1:length(states)){
rmarkdown::render("covid_dashboard_param.Rmd",
  params = list(state = states[i]),
  output_file = paste0(states[i], " COVID-19 Dashboard")
)
}
```

# Your turn!

Try this out either with the dashboard you made with your own data or with the Diamonds dashboard that we made before.

- if you're using the Diamonds dashboard, make `clarity` the parameter variable and replace the barpolot of `clarity` (which will no longer be interesting) with a scatterplot of `table` and `depth`.

15:00

# Animation

One of the last pieces we're going to talk about is animation in `plotly` plots.

- This happens by using the `frame = ~variable` argument, where the `variable`.
- The animation will move across the values of the `variable` specified in the `farme` argument.

We'll use the `gapminder_unfiltered` data

```
library(gapminder)
gapminder %>%
  mutate(text = paste0(
    country, "\n Life Expectancy = ", lifeExp,
    "\n GDP/Capita = ", round(gdpPercap),
    "\n Population = ", round(pop/1000000), "M")) %>%
  plot_ly(x = ~gdpPercap,  y = ~lifeExp, size = ~pop,
          color = ~continent, frame = ~year, text = ~text,
          hoverinfo = "text", type = 'scatter',
          mode = 'markers') %>%
  layout(xaxis = list(type = "log"))
```

See the plot here (https://davidaarmstrong.github.io/deck2/sharing2.html#plotly-with-animation)

# plot

# states

Using the `states_covid.rda` file, make an animated scatterplot of COVID cases versus deaths.

- you can either try to do it by day or aggregate to the week.
  - both `date` and `week` are variables in the dataset.
  - Hint: the `deaths` and `cases` variables are both cumulative, so if you use the week aggregation, you'll want to take the maximum in each week.

10:00

# Your turn!

Build some animation into your dashboard.

- Either use the dashboard that you created above with your own data and add in an animated plot.
- Alternatively, use the `states_covid.rda` data and create a dashboard that has two maps - most recent cases and most recent deaths as well as the animated plot of cases by deaths.

15:00