

## Lecture 18: Interactive plots with ggvis

### STAT598z: Intro. to computing for statistics

---

#### Vinayak Rao

#### Department of Statistics, Purdue University

```
In [ ]: options(repr.plot.width=5, repr.plot.height=3)
```

ggvis is a simple way to get interactive plots

- provides a simpler interface to shiny
- is still experimental

Like ggplot this expects a dataframe/tibble as an input

Some differences:

- add layers using %>% instead of +
- instead of aes(color=group), write color = ~group
- we still write color:=clr\_val
- aesthetics have different names:
  - color becomes stroke
  - alpha becomes opacity

```
In [ ]: library('tidyverse')
library('shiny')
library('ggvis')
load('Data/HomeValues.RData')
HomeValues$qtr <- as.double(HomeValues$qtr)
print(HomeValues$qtr)
```

```
In [ ]: plt<- ggvis(HomeValues,x=~qtr,y=~Home.Value,stroke=~State) %>%
  layer_lines(); plt
```

```
In [ ]: plt <- plt %>% hide_legend('stroke'); plt
```

```
In [ ]: plt %>% layer_points()
```

```
In [ ]: plt %>% layer_points(size=1, fillOpacity=.1) # Bug!
```

ggvis uses both = and := for assignments

Use = to map a variable to a property

- Then use ~ to refer to a column of a dataframe

Use := when we set a property based on a *value*

```
In [ ]: plt %>% layer_points(size:=1, fillOpacity:=.1)
```

In the end, set properties using = ~column or := value

So why use ggvis instead of ggplot?

- Interactive plots!

```
In [ ]: plt %>% layer_points(size:=input_slider(.1,50,1),
                             fill =~ State, fillOpacity:=.5) %>%
  hide_legend('fill')
```

```
In [ ]: plt %>% add_tooltip(function(x)
                             {paste(x$State, ":", x$Home.Value)}, 'hover')
```

add\_tooltip needs a function to read value and return a string

- we used an *anonymous function* to print State, Value

For lines, add\_tooltip only prints first value (<http://stackoverflow.com/questions/28540504/mouse-hover-in-layer-lines-ggvis-r>)

- add layer\_points() for all values

```
In [ ]: plt <- plt %>% layer_points(size:=input_slider(0,5))
```

```
In [ ]: plt %>% add_tooltip(function(x)
                             {paste(x$State, ":", x$Home.Value)}, 'hover')
```

```
In [ ]: mtcars %>% ggvis(~wt, ~mpg) %>% layer_points() %>%
  layer_smooths(span = input_slider(0.2, 1))
```

```
In [ ]: plt <-ggvis(HomeValues, x=~qtr, y=~Home.Value, stroke=~State) %>%
  layer_smooths(span:=input_slider(0,5)) %>%
  hide_legend('stroke')
```

Error because ggvis doesn't do grouping for you (unlike ggplot)

```
In [ ]: plt <- HomeValues %>% group_by(State) %>%
  ggvis(x=~qtr,y=~Home.Value, stroke=~State) %>%
  layer_smooths(span=input_slider(0,2,step=.1)) %>%
  hide_legend('stroke')
```

```
In [ ]: ggvis(HomeValues) %>% layer_histograms(x=~Home.Value)
```

```
In [ ]: ggvis(HomeValues) %>%
  layer_histograms(x=~Home.Value,
    width=input_slider(min=1000,max=100000))
```

```
In [ ]: ggvis(HomeValues) %>%
  layer_histograms(x=~Home.Value, fill.hover:='red',
    width=input_slider(min=10^3,max=10^5))
```

```
In [ ]: plt<- ggvis(HomeValues,x=~qtr,y=~Home.Value,stroke=~State) %>%
  layer_lines(stroke.hover:='black', size.hover:=3) %>% # watch out for s
  trokewidth.hover:=3
  hide_legend('stroke')
plt %>% add_tooltip(function(x)
  {paste(x$State,":",x$Home.Value)},'hover')
```

tidyverse commands can be overloaded for use with ggvis:

<https://rdr.io/cran/ggvis/man/dplyr-ggvis.html> (<https://rdr.io/cran/ggvis/man/dplyr-ggvis.html>)

```
In [ ]: plt <- HomeValues %>% group_by(State) %>%
  ggvis(x=~qtr,y=~Home.Value, stroke=~State) %>%
  filter(State %in% eval(input_select(choices =
    unique(as.character(HomeValues$State)),
    multiple=TRUE, label='States list')))) %>%
  layer_lines(strokeWidth:=2)
```

Note the eval, this is because of we are calling input\_select inside filter

<http://stackoverflow.com/questions/25891020/dynamic-filtering-with-input-select-using-ggvis-in-r>  
(<http://stackoverflow.com/questions/25891020/dynamic-filtering-with-input-select-using-ggvis-in-r>)

```
In [ ]: library('ggplot2');library('maps')
my_state_map <- map_data('state');
my_state_map$region <- tolower(my_state_map$region)
# Add DC
state.name[51]<-"district of columbia"; state.abb[51]<-"DC"

# Get abbreviations for state names
get_ab <- function(x) state.abb[x == tolower(state.name)]

# Get home prices in state st in year yr
get_house_pr <- function(st,yr) {
  HomeValues[HomeValues$State==st & HomeValues$qtr==yr,2] }

# apply get_ab to each row of my_state_map
my_state_map$region <- purrr::map_chr(my_state_map$region, get_ab)
# Function to get vector of prices of yr
get_yr_pr <- function(yr) {
  pr <- my_state_map$pr
  for(st in state.abb)
    pr[my_state_map$region == st] <- get_house_pr(st,floor(yr))
  return(pr)
}
```

```
In [ ]: yr <- 1976
stamp <- reactive({invalidateLater(2000,NULL) # note reactive
  my_state_map$pr <- get_yr_pr(yr)
  yr <<- yr + 4; if(yr>=2013) yr <<- 1976
  print(yr)
  my_state_map })

stamp %>% ggvis(~long, ~lat,fill=~pr) %>%
  group_by(region) %>%
  layer_paths(strokeOpacity := 0.5,
              strokeWidth := 0.5) %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(width=960, height=600, keep_aspect=TRUE) %>%
  hide_legend('fill') %>%
  add_tooltip(function(x) {
    paste(x$region)}, 'hover')
```

```
In [ ]: my_state_map %>% ggvis(~long, ~lat) %>%
  mutate(pr = eval(input_slider(1976,2013,
    map= function(x) get_yr_pr(x)))) %>%
  group_by(region) %>% layer_paths(fill=~pr) %>%
  hide_axis("x") %>% hide_axis("y") %>%
  set_options(duration=0,width=960,height=600,keep_aspect=TRUE) %>%
  hide_legend('fill') %>% hide_legend('stroke') %>%
  add_tooltip(function(x) {
    paste(isolate(x$curr),":",x$region,":",x$pr)}, 'hover') %>%
  scale_numeric("fill", range = c("yellow","red"))
```

```
In [ ]: yr <- reactiveValues(curr=1976)
stmp <- reactive({my_state_map$pr <- get_yr_pr(yr$curr);
my_state_map})

stmp %>% ggvis(~long, ~lat, fill=~pr,
stroke=input_slider(1976,2010,
map= function(x) yr$curr <<-x)) %>%
group_by(region) %>%
layer_paths(strokeOpacity := 0.5,
strokeWidth := 0.5) %>%
hide_axis("x") %>% hide_axis("y") %>%
set_options(duration=0,width=960, height=600, keep_aspect=TRUE) %>%
hide_legend('fill') %>%
hide_legend('stroke') %>%
add_tooltip(function(x) {paste(isolate(yr$curr),":",
x$region,":",x$pr)},'hover') %>%
scale_numeric("fill", range = c("yellow","red"))
```

ggvis is also compatible with *reactive programming*

This is a programming paradigm imported from shiny (<https://shiny.rstudio.com/articles/reactivity-overview.html>) (<https://shiny.rstudio.com/articles/reactivity-overview.html>)

At a high level a reactive source feeds inputs to reactive end-points

- whenever the source changes, the end point is automatically updated

ggvis automatically updates when a reactive input changes

```
In [ ]: #https://r2014-mtp.sciencesconf.org/file/92631
#library(shiny)
dat <- data.frame(time=1:10, value=runif(10))

# Create a reactive that returns a data frame, adding a new
# row every 2 seconds
ddat <- reactive({
  invalidateLater(500, NULL) # wait of 2 seconds
  dat$time <<- c(dat$time[-1], dat$time[length(dat$time)] + 1)
  dat$value <<- c(dat$value[-1], runif(1))
  dat
})

ddat %>% ggvis(x = ~time, y = ~value, key := ~time) %>%
  layer_points() %>% layer_paths()
```

```
In [ ]: dat <- data.frame(time = 1, value = c(0), mn = c(0))

ddat <- reactive({
  invalidateLater(200, NULL);
  len <- length(dat$time) + 1;
  dat[len,] <<- c(len, rnorm(1),0)
  dat$mn[len] <<- mean(dat$value)
  dat
})

ddat %>% ggvis(x = ~time, y = ~mn, key := ~time) %>%
  layer_paths()

ddat %>% ggvis(x = ~value) %>%
  layer_histograms()
```