Post02 - Interactivity with ggvis

Jonathan Stuart

12/2/2017

Introduction

Admittedly, the topic I chose for my first post was pretty broad, and for that I was penalized. I selected Principal Component Analysis, and took it on, full-force. Unfortunately, I was unable to go into a sufficient amount of detail to have my post deemed original, so for this post, I would like to focus in on something a little simpler. By focusing on something smaller in scope, I will be able to both fulfill the rubric of the assignment, and maintain the same 'mastery' orientation I attempted to apply during the creation of Post 1.

Now I don't know about you, but when Prof. Sanchez first introduced us to Shiny Apps, I was excited. I was drawn to the idea of a gallery of apps to train my imagination on. I am very interested in data visualization, as most of us are, and the prospect of interactivity combined with data visualization was quite enticing. For that reason, here in my second post, I would like to explore ways to increase the interactivity of plots created with ggvis, through ggvis' built-in hovering functions.

Motivation

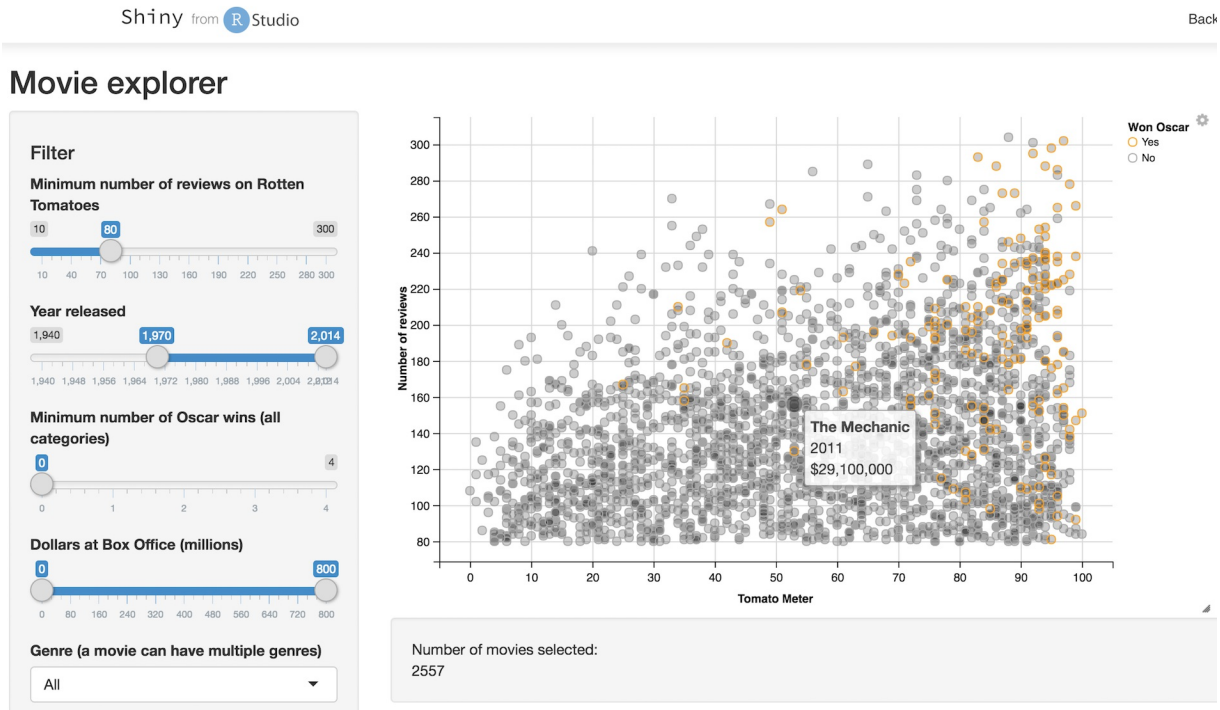
At this point, it may be helpful to state that what follows is, in fact, original with respect to what has been covered in lecture and lab. Think back to the creation of the Shiny App for HW04. A lot of people I worked with ended up wanting to use ggplot for their images, mainly because they felt comfortable with it. Prof. Sanchez, however, wanted us to take a crack at ggvis. Having gone through a simple demonstration of ggivs in lecture and not having dedicated a lab assignment to it, ggvis was pretty new to me. I found that to be perhaps the most frustrating part of HW04: getting ggvis to work.

In poring over the sample apps illustrating conditional panels that Prof. Sanchez posted for us, after the assignment was complete, funny enough, I noticed the subtlety of his hovering characteristics on his bar plot. To see what I mean, check out the app here:

https://jstuart3.shinyapps.io/grade_visualizer/

Go ahead and mouse over one or any of the bars in the first tab's bar plot…you'll see a slight change in opacity.

That reminded me of the initial lecture Prof. Sanchez gave on shiny apps, and the first app he opened. The Movie Explorer app. Here's an image of it to jog your memory.



Screen capture from Shiny App Gallery - The Movie Explorer App

Go ahead and direct your attention to the little box with the movie title, release year, and box office earnings. I didn't know it at the time, but that little text box is created by ggvis' hovering functions. That, along with with a few other characteristics, we will explore in the remainder of this post.

If you'd like to play with the Movie Explorer app some, it can be found here:

https://shiny.rstudio.com/gallery/movie-explorer.html

Building a Basic ggvis Plot

Before we start anything, make sure that you've got the required libraries loaded. We're going to need `ggvis` for our plots, `dplyr` for the piping operator, and `datasets` for the R data sets. You load them like this:

```
library(ggvis)
library(dplyr)
library(datasets)
```

Alright. Now that we've got that out of the way, let's now build a basic ggvis plot using some built-in R data so that the data set will be publicly available and easily accessible. This will go a long way toward respecting computational reproducibility. How about that good old `mtcars` data set?

Usually, the `datasets` package loads on start-up of RStudio. If not, go ahead and load it manually, as described above. To access it, you can open an instance of RStudio, and type the following into the console:

```
mtcars
```

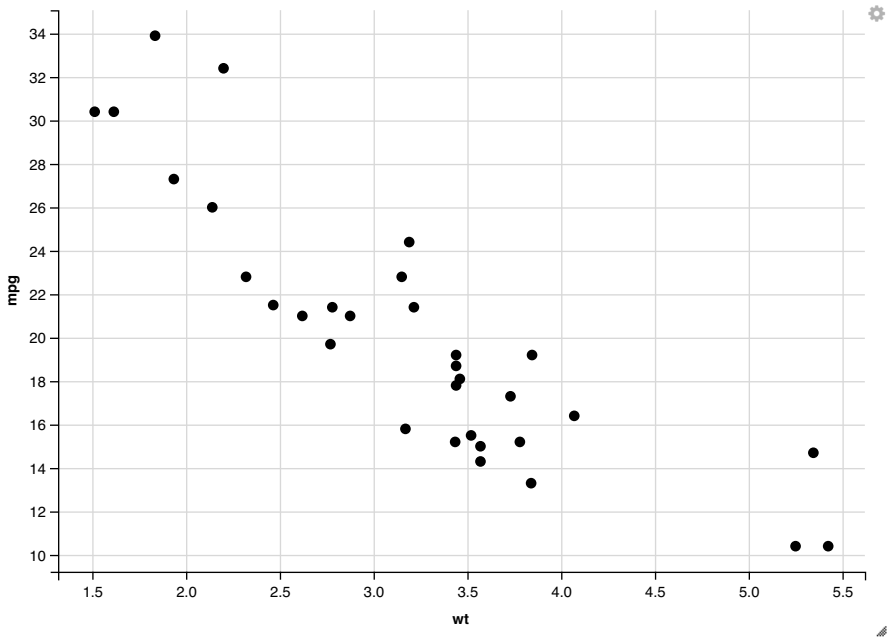
You'll then see something like the following:

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Great. Now let's go ahead and build a simple ggvis plot with the data set we've selected.

1. Use the piping operator `%>%` on the data set `mtcars` as shown below.
2. Create a ggvis object using `ggvis(x = ~variable, y = ~variable)` as shown below.
3. Again use the piping operator `%>%` to add the `layer_points()` function that will give us our data points, as should below.

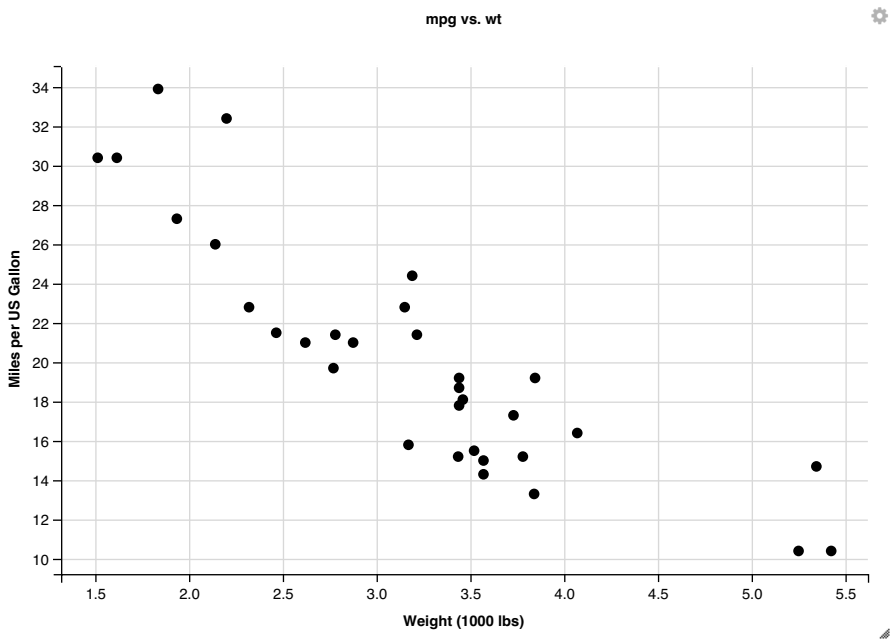
```
# building simple ggivs plot
mtcars %>% ggvis(x = ~wt, y = ~mpg) %>%
  layer_points()
```



Ok, great. Now we've got a simple ggvis plot using the built-in data set, `mtcars`. Nothing new here. Let's go ahead and fix it up some.

1. Create the same base plot as described above.
2. Add axis titles using the `add_axis()` function as shown below.
3. Add a title using the 'dirty hack' of adding an oriented axis as shown and described below.

```
# adding title and axis labels to simple ggvis plot
mtcars %>% ggvis(x = ~wt, y = ~mpg) %>%
  layer_points() %>%
  add_axis("x", title = "Weight (1000 lbs)") %>%
  add_axis("y", title = "Miles per US Gallon") %>%
  add_axis("x", orient = "top", ticks = 0, title = "mpg vs. wt",
           properties = axis_props(
             axis = list(stroke = "white"),
             labels = list(fontSize = 0))
           )
```



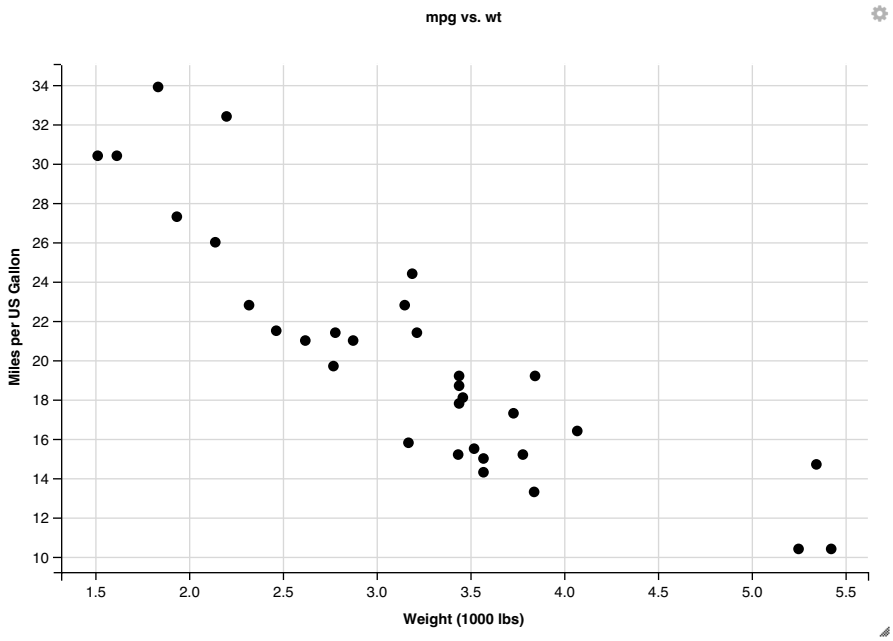
Ok great. Now here, we do actually have something new. Apparently, ggvis doesn't yet support adding a title to your plots! Prof. Sanchez wasn't kidding when he said it was new, and that we could expect it to develop over the next few years! What we've done here is use the 'dirty hack' of adding another axis, orienting it to the top of the plot, and making the axis white (so that it isn't visible) while keeping the title text black.

The Main Event - Hovering

Ok, great. Now that we've got a simple plot, let's go ahead and figure out how to make it a bit more interactive. Go ahead and mouse over one of the data points. As with the rest of the plots and the shiny app, you can recreate the output by following the accompanying code.

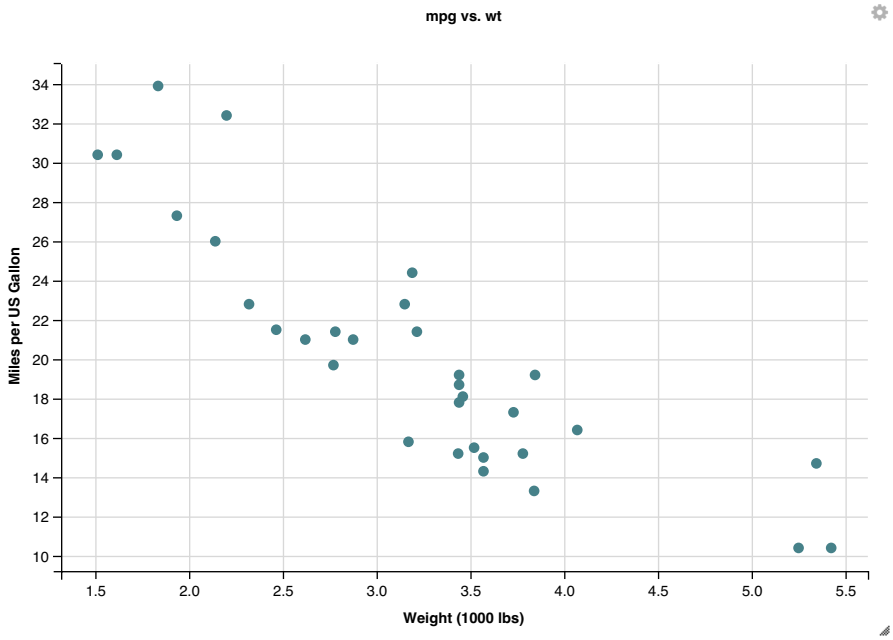
1. Make sure to add `size.hover := 80` when you create your ggvis object…this is how we get the hover functionality.

```
# adding hover functionality for changing size
mtcars %>% ggvis(x = ~wt, y = ~mpg, size.hover := 80) %>%
  layer_points() %>%
  add_axis("x", title = "Weight (1000 lbs)") %>%
  add_axis("y", title = "Miles per US Gallon") %>%
  add_axis("x", orient = "top", ticks = 0, title = "mpg vs. wt",
           properties = axis_props(
             axis = list(stroke = "white"),
             labels = list(fontSize = 0))
           )
```



Can't see it? How about now?

```
# increasing impact of hover functionality; size and stroke
mtcars %>% ggvis(x = ~wt, y = ~mpg, fill:= "#468189", size.hover := 300, stroke := NA, stroke.hover := "black") %>
%
  layer_points() %>%
  hide_legend('fill') %>%
  add_axis("x", title = "Weight (1000 lbs)") %>%
  add_axis("y", title = "Miles per US Gallon") %>%
  add_axis("x", orient = "top", ticks = 0, title = "mpg vs. wt",
           properties = axis_props(
             axis = list(stroke = "white"),
             labels = list(fontSize = 0))
           )
```

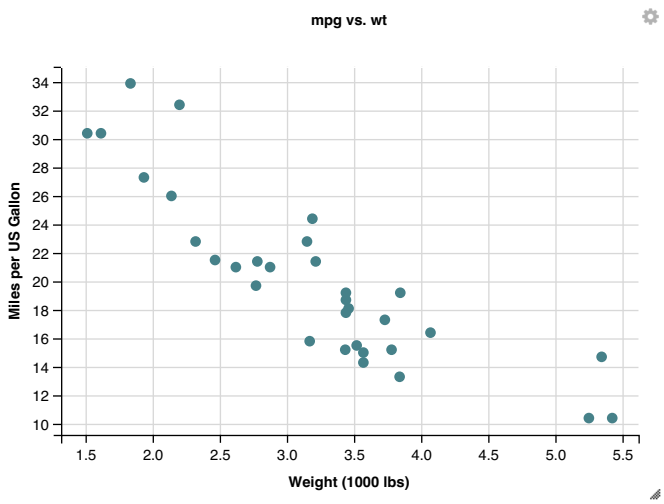


If you are reproducing this plot, don't forget to:

1. Hide the automatic legend associated with changing the fill color that pops up by piping in `hide_legend('fill')` as shown above.
2. For this one, also be sure to include `stroke := NA` and `stroke.hover := "black"` parameters when you create the ggvis object. This

Great. Study the code a little bit. What we've done so far is change the color of the data points with the `fill` option, and used the `size.hover` and `stroke.hover` parameters to increase the size of a given data point when the mouse hovers over it. The `size.hover` parameter increases the size of a data point when it is hovered over by the mouse, while the `stroke.hover` adds a border to the data point when it is hovered over. Pay special attention to the `stroke := NA` parameter. What do you think happens if we don't include it? Let's see.

```
# demonstration of what happens without `stroke := NA` parameter
mtcars %>% ggvis(x = ~wt, y = ~mpg, fill:= "#468189", size.hover := 300,
                 stroke.hover := "black", strokeWidth.hover := 2) %>%
  layer_points() %>%
  hide_legend('fill') %>%
  add_axis("x", title = "Weight (1000 lbs)") %>%
  add_axis("y", title = "Miles per US Gallon") %>%
  add_axis("x", orient = "top", ticks = 0, title = "mpg vs. wt",
           properties = axis_props(
             axis = list(stroke = "white"),
             labels = list(fontSize = 0))
           ) %>%
  set_options(height = 375, width = 500)
```



As you can see, without the `stroke := NA` parameter acting as a default, the border around a data point won't leave when you're finished hovering! In this plot, we've also started changing the default size of the displayed plot with the `set_options()` function.

In our last plot, did you also notice that we changed the stroke size using the `strokeWidth.hover` parameter? ggvis has got other parameters that will allow you to do things like change the opacity of a data point or object when you hover over it, change its fill color, etc. If you've graphed lines using, say, a density plot or a line plot, you can also code a plot to alter the thickness, stroke, or fill of that line.

The Crown Jewel

Ok, so the best effect I've come across related to interactivity is, by far, the ability to display a text box by hovering. This is also the aspect of the Movie Explorer app that excited me most. Let's create that interactivity such that when we hover over an element, we're shown some information.

Unfortunately, interactive plots cannot be rendered by the `knittr` package. So, for computational reproducibility, we'll have to host these interactive plots online in a shiny app, while including the code for it here.

This is the code:

```
hover-app1.app

# required packages
library(shiny)
library(ggvis)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Hover App"),

    # Placing ggvis output in main panel
    mainPanel(ggvisOutput("scatterplot"))
)

# Define server logic
server <- function(input, output) {

    # Creating a new data frame to work with
    mtc <- mtcars

    # Add an id column to use as the key
    mtc$id <- 1:nrow(mtc)

    # Creating our function for `add_tooltip()` (see `add_tooltip() documentation`)
    display_values <- function(x) {
      if(is.null(x)) return(NULL)
      row <- mtc[mtc$id == x$id, ]
      paste0(rownames(row), " - ", names(row), ": ", format(row), collapse = "<br />")
    }

  # Barchart (for 1st tab)
  vis_barchart <- reactive({

    # creating ggvis object
    mtc %>% ggvis(x = ~wt, y = ~mpg, fill:= "#468189", key := ~id) %>%
      layer_points() %>%
      add_tooltip(display_values, "hover") # this is the part of the code that adds the hover functionality
  })

  # binding ggvis object to shiny display with id "scatterplot"
  vis_barchart %>% bind_shiny("scatterplot")
}

# Run the application
shinyApp(ui = ui, server = server)
```

```
hover-app2.app

# required packages
library(shiny)
library(ggvis)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Hover App"),

  # Placing ggvis output in main panel
  mainPanel(ggvisOutput("scatterplot"))
)

# Define server logic
server <- function(input, output) {

  # Creating a new data frame to work with
  mtc <- mtcars

  # Add an id column to use as the key
  mtc$id <- 1:nrow(mtc)

  # Creating our function for `add_tooltip()` (see `add_tooltip() documentation`)
  display_values <- function(x) {
    if(is.null(x)) return(NULL)
    row <- mtc[mtc$id == x$id, ]
    paste0(rownames(row), " - ", names(row), ": ", format(row), collapse = "<br />")
  }

  # Barchart (for 1st tab)
  vis_barchart <- reactive({

    # creating ggvis object
    mtc %>% ggvis(x = ~wt, y = ~mpg, fill:= "#468189", size.hover := 300,
                  stroke := NA, stroke.hover := "black", key := ~id) %>%
      layer_points() %>%
      add_tooltip(display_values, on = "click")  # this is the portion of the code that adds the click functionali
ty
  })

  # binding ggvis object to shing display with id "scatterplot"
  vis_barchart %>% bind_shiny("scatterplot")
}

# Run the application
shinyApp(ui = ui, server = server)
```

These are some images of the apps:

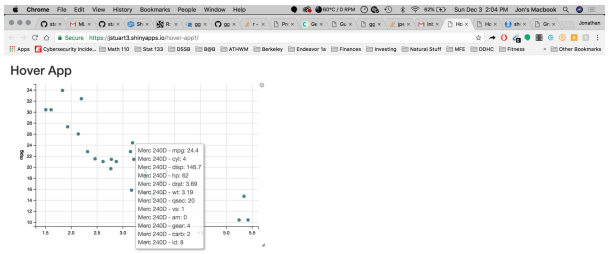


Image of web-hosted hover-app1.app

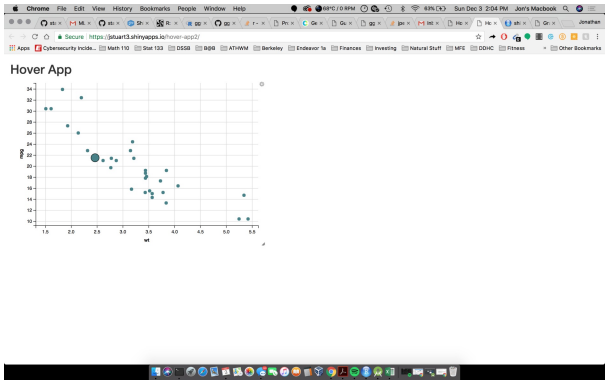


Image of web-hosted hover-app2.app

And here are some links to the apps, hosted online for your access:

- Hover App 1 - https://jstuart3.shinyapps.io/hover-app1/
- Hover App 2 - https://jstuart3.shinyapps.io/hover-app2/

Go ahead and play with the apps a little. To make sure it's clear, I'd like to emphasize that the first hover app displays the text box simply by hovering, while the second only has a size and stroke effect for hovering. The second hover app has the added functionality of displaying the text box when a particular data point is clicked on. I like this because it makes the plot feel truly interactive. Study the code for the ggvis plots a little (in the server sections of the shiny apps) and become familiar with how they're created. They both involved creating functions that can be passed to the `tool_tip()` function to achieve the interactive text box.

Now, the downside. As proof of what Prof. Sanchez mentioned in lecture, the ggvis package really isn't very well documented. See for yourself:

1. Open an instance of RStudio.
2. Locate the pane with the `Help` tab.

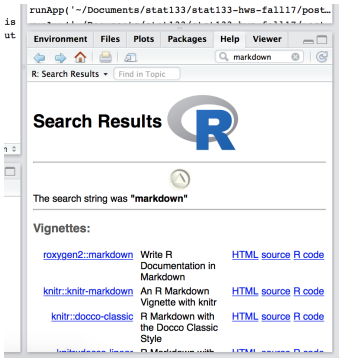


Image of pane contaning Help tab in RStudio

3. Search "hide_tooltip( )"

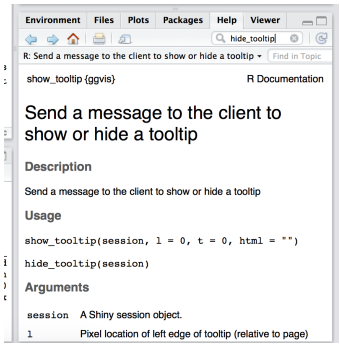


Image of hide_tooltip( ) documentation

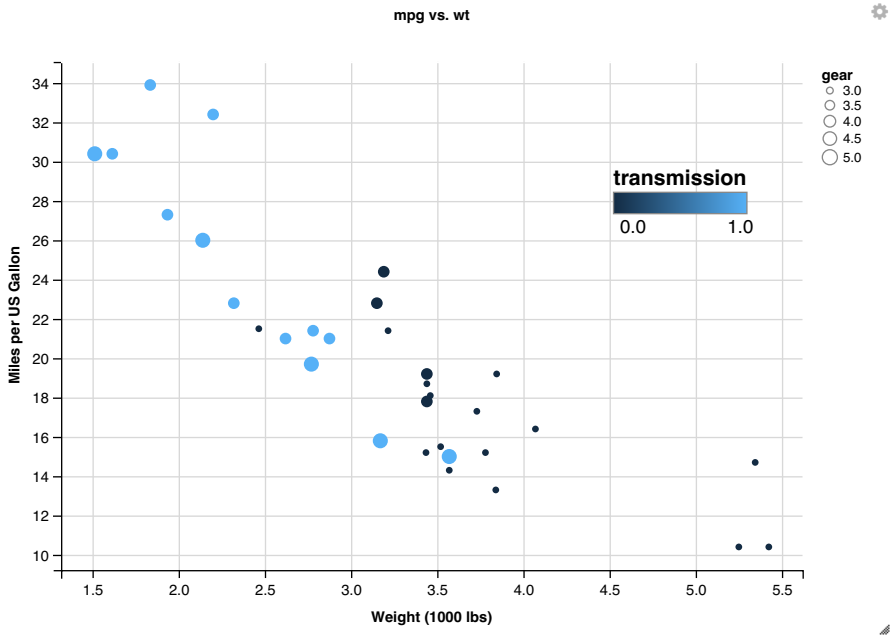
This is the issue I ran into with the second app, hover-app2: I can induce the text box to show when a data point is clicked, but I can't get the text box to disappear. Clearly, `hide_tooltip()` would be the function to use, but its documentation is sparse. Searching the internet for an idea of what the "session" parameter means I came up empty, so for now, we'll just keep the text boxes created by clicking on a particular data point open. For now, we can just refresh the app in-browser to close them.

Some Extras - Multi-dimensionality

Finally, I'd like to discuss something related to data visualization, and not necessarily interactivity. Take a look at the following plot.

1. Create the same base plot as described above.
2. Add the `size = ~gear` and `fill = ~am` parameters when creating the ggvis object as shown below.
3. Don't forget to hide the automatic legend by using the piping operator `%>%` to add `hide_legend('fill')`.
4. Add a legend associated with `"fill"`, with title `"transmission"` and coordinates `x = scaled_value("x", 4.5)` and `y = scaled_value("y", 30)`.

```
# adding two dimensions through size and fill
mtcars %>% ggvis(x = ~wt, y = ~mpg, size = ~gear, fill = ~am, size.hover := 300, stroke := NA, stroke.hover := "bl
ack") %>%
  layer_points() %>%
  hide_legend('fill') %>%
  add_axis("x", title = "Weight (1000 lbs)") %>%
  add_axis("y", title = "Miles per US Gallon") %>%
  add_axis("x", orient = "top", ticks = 0, title = "mpg vs. wt",
           properties = axis_props(
             axis = list(stroke = "white"),
             labels = list(fontSize = 0))
           ) %>%
  add_legend("fill", title = "transmission",
    properties = legend_props(
      title = list(fontSize = 16),
      labels = list(fontSize = 14, dx = 5),
      symbol = list(stroke = "black", strokeWidth = 2,
        shape = "square", size = 200),
      legend = list(
        x = scaled_value("x", 4.5),
        y = scaled_value("y", 30)
      )
    )
  )
```



We've done something interesting here. Though the graph has two axes, it is actually four dimensional. As before, the x and y coordinates pertain to the weight and miles per gallon of each vehicle, respectively. Here, though, we've also added the elements of size and color. The size has been tied to the number of gears of the vehicle, and the fill color has been tied to transmission type (0 for automatic transmission, 1 for manual transmission). What I find interesting about this is that, once we have familiarized ourselves with the scales (the vocabulary of the graph, if you will), we are able to get a very quick, very rich view of the data. Quicker and richer, perhaps, than looking at tabular data.

Take Home Message

So what's the take home message? It's that ggvis is well suited to data visualization through both interactive elements and multi-dimensional plotting. Through allowing us to vary this sizes, shapes and colors of data points through the hovering and clicking functionalities, ggvis nicely draws users into the data through interactivity. Further, through its ability to represent more than 2 dimensions of data in a two-dimensional space, ggvis opens the door to novel and creative ways to quickly and clearly communicate data in an increasingly data-driven world.

References

Here we have a list of the references used while putting together this post.

- Videos
  - YouTube Videos
    - https://www.youtube.com/watch?v=nO-j7Qkwr9w - ggvis Introductory Video
    - https://www.youtube.com/watch?v=pGMHkgK0x3o - Short Video on ggvis Sliders
- Websites
  - Github Repositories
    - https://github.com/rstudio/ggvis/tree/master/demo
  - General Websites
    - https://stackoverflow.com/questions/25018598/add-a-plot-title-to-ggvis - How to Add a Title to a ggvis Plot
    - https://github.com/rstudio/ggvis/issues/229 - Hiding the Legend
    - https://ggvis.rstudio.com/layers.html - Useful Conversion Chart from ggplot2 to ggvis
    - https://ggvis.rstudio.com/axes-legends.html - Manipulating Axes and Legends
    - https://ggvis.rstudio.com/ggvis-basics.html - Multi-dimensional Plotting, Sliders

        - https://www.rdocumentation.org/packages/shinyBS/versions/0.61/topics/addTooltip - Adding a Tooltip for Interactivity
- Help Searches
    - Within RStudio Help Tap
        - `set_options()`
        - `props()`
        - `add_tooltip()`
        - `hide_tooltip()`
        - `add_legend()`
- Publications
    - PDF Documents
    - https://cran.r-project.org/web/packages/ggvis/ggvis.pdf