Post 02 - Using ggvis to visualize data

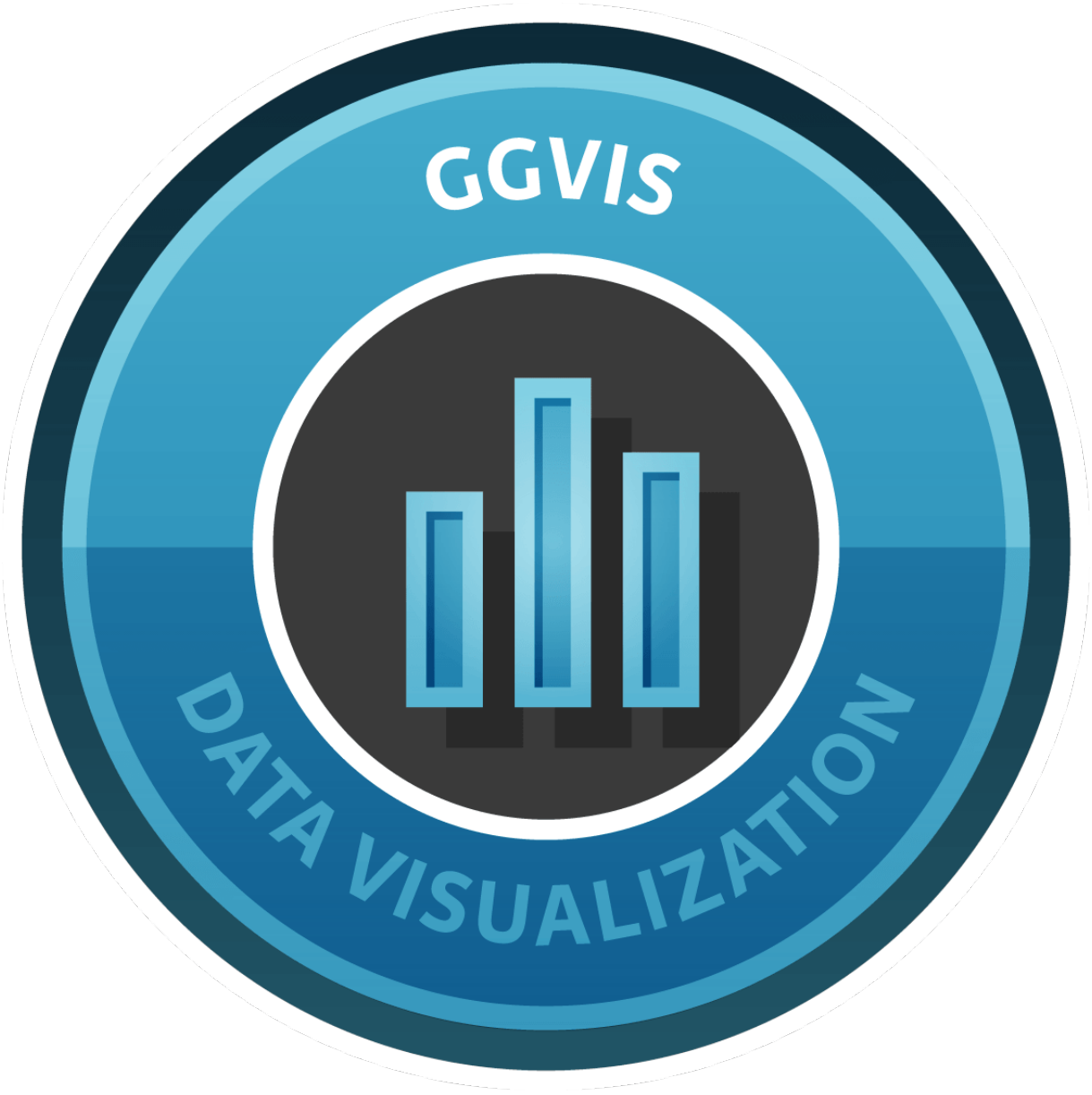
Sandeep Tiwari

12/3/2017

Using ggvis to visualize data

Introduction

If one is familiar with ggplot, then getting familiarized with ggvis shouldn't bee very difficult, as it borrows many concepts. It is a data visualization package for R which lets you declaratively describe data graphics with a syntax similar to ggplot, and create rich interactive graphics that you can play with locally. There are many special applications that ggvis allows you to run, but we will be looking at how we can create interactive tools using ggvis and shiny. We will also be embedding a ggvis in a Shiny app, giving us complete freedom to make any component of the plot interactive, to display multiple plots on one page, and to freely arrange controls. The downside is the we'll need more code, and will need at least basic understanding of Shiny.



Motivation

After the struggle of completing the shiny app for the most recent homework, I thought it would be interesting to explore the different interactive tools provided by 'ggvis.'

Background

Ggvis interactivity is built on top of Shiny's reactive programming model. It's possible to use ggvis without understanding how Shiny works, but you'll be limited to relatively simple interactions. The first part of this vignette describes basic interactive controls, which provide a quick and easy way to add basic interactivity to a plot. They are not very flexible, but they cover the most common interactive needs and you don't need to know anything about Shiny to use them.

Examples

Interactive graphics are very similar to statics plots. A plot with basic interactive controls looks the same, but you just replace constant values with functions that produce interactive controls, such as input_slider().

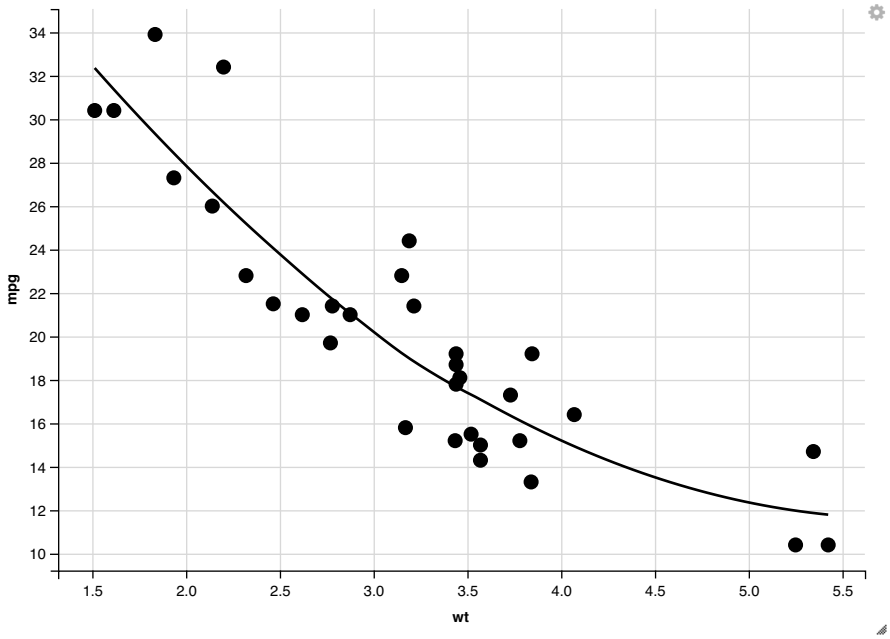
Installing Packages

```
#install.packages('ggvis')
library(ggvis)
library(shiny)
```

A plot with basic interactive controls looks very similar to a static plot. You just replace constant values with functions that produce interactive controls like this one:

```
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_smooths(span = input_slider(0.5, 1, value = 1)) %>%
  layer_points(size := input_slider(100, 1000, value = 100))
```

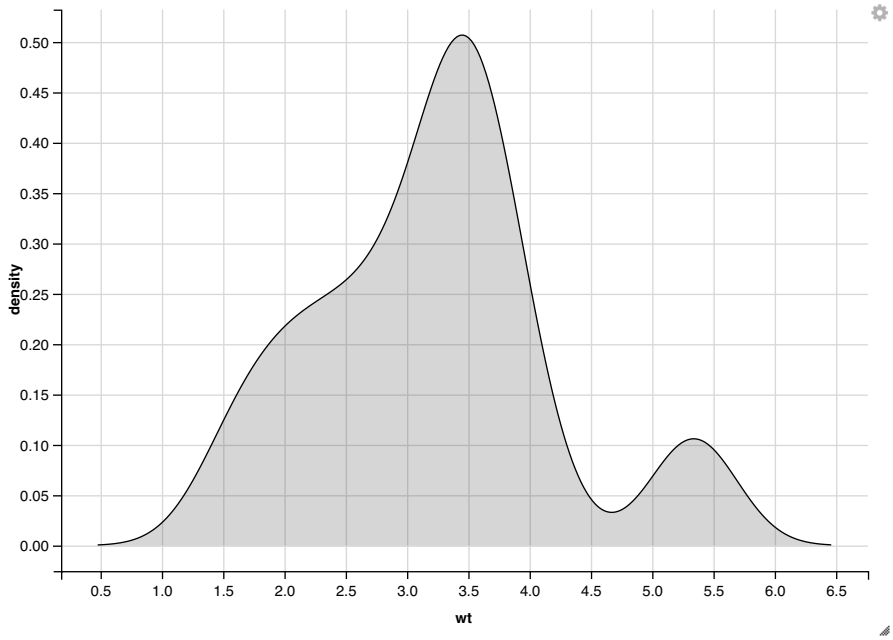
```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```



As well as input_slider(), which displays an interactive slider at the end, there are a number of other interactive controls such as input_checkbox, input_numeric, input_radiobuttons, input_select, and input_text. Here we have another example, which contains both a slider and a select box.

```
mtcars %>% ggvis(x = ~wt) %>%
    layer_densities(
      adjust = input_slider(.1, 2, value = 1, step = .1, label = "Bandwidth adjustment"),
      kernel = input_select(
        c("Gaussian" = "gaussian",
          "Epanechnikov" = "epanechnikov",
          "Rectangular" = "rectangular",
          "Triangular" = "triangular",
          "Biweight" = "biweight",
          "Cosine" = "cosine",
          "Optcosine" = "optcosine"),
        label = "Kernel")
    )
```

```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```

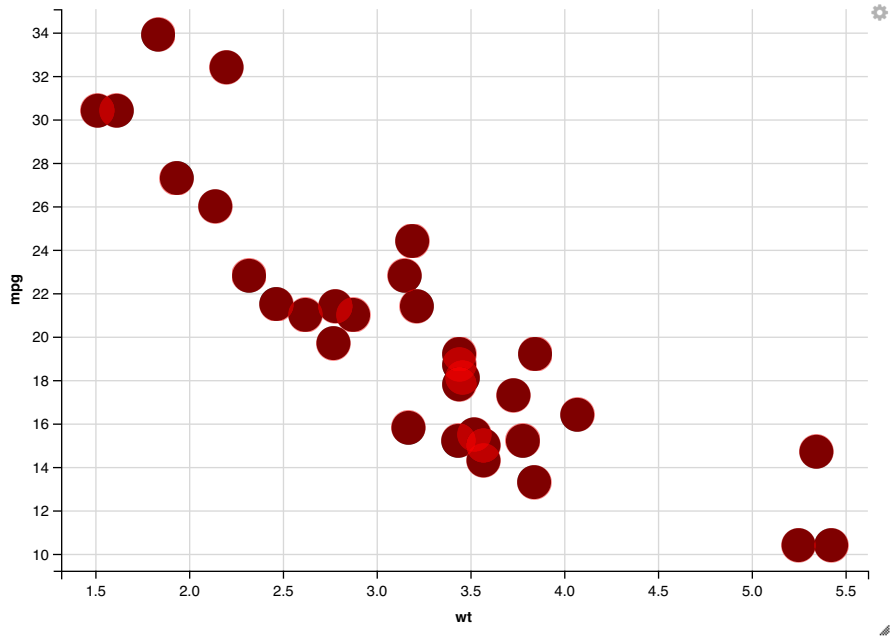


Note that all interactive functions begin with 'input.'

It is also possible to have an interactive input control more than one setting on the plot. For example, we can create an input slider that affects both the size of botht the red and blackpoint:

```
mtcars %>% ggvis(~wt, ~mpg) %>%
  layer_points(size := input_slider(10, 1000)) %>%
  layer_points(fill := "red", opacity := 0.5, size := input_slider(10, 1000))
```

```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```



However, there are some limitations when using interactive inputs. Currently, interactive input can only be used in two places: * as arguments to transforms: layer_smooth(span = input_slider(0, 1)) * as properties: props(size = input_slider(10, 1000))

This means that interactive inputs can only modify the data, not the underlying plot specification. So, with only basic interactivity there's no way to add or remove layers, or switch between different datasets. This is a reasonable limitation because if you're doing exploration you can always create a new ggvis with R code, or if you're polishin a plot for preentation, you can embed it in a Shiny app.

Compared to Shiny

When comparing ggvis to shiny, you'll notice that these functions have very similar equivalents: sliderInput(), selectInput(), etc. There are two main differences: * arguments are tweaked to create basic control with minimal arguments * interactive input are not necessarily created in a reactive context, so they cannot return reactives. Instead, they return delayed reactives, which are activated and connected together when the plot is displayed.

Conclusion

In general, being able to produce valid interactive html charts from R markdown is great! All of the packages great sensible outputs, but there are also a lot of differences. While ggplot is great, so is ggvis, as it pays attention to graphical details. It's ability to scale and layer multiple data sets as well as it's functionality with shiny and vega is what makes it unique and powerful.

References

1. http://ouzor.github.io/blog/2014/11/21/interactive-visualizations.html
2. https://ggvis.rstudio.com/interactivity.html
3. https://ggvis.rstudio.com/axes-legends.html
4. https://ggvis.rstudio.com/ggvis-basics.html
5. https://ggvis.rstudio.com/data-hierarchy.html
6. https://ggvis.rstudio.com/layers.html
7. https://ggvis.rstudio.com/ggplot2.html