A Comparison of Plotting with ggvis and ggplot2

Ruijie Zhou

12/2/2017

Introduction

The purpose of this post is to discuss certain features associated with creating graphs and plots with 2 specific packages widely used in R, the `ggvis` package and the `ggplot2` package. The post would mainly consist of plotting and graphing examples from both the `ggvis` and the `ggplot2` packages, and some comparisons between the 2 ways of creating graphs. This post serves as an extension to what I have learned about data visulization using `ggvis` in the past few weeks. The first half of the post will be basic plotting using both packages, and the second half of the post will be on some new features introduced by the `ggvis` package.

Motivation and Background

In my previous post, I focuses on and summarized some of the key features in plotting with the R built-in functions and `ggplot2`, and compared the 2 ways of creating simple graphs. Since we took some time to discuss the data visulization features in the `ggvis` package, I think it would be good idea to see and compare these 2 common packages that we have been using a lot. Essentially `ggplot2` and `ggvis` has a similar underlying graphing logics, with some differences in the syntax. `ggvis` does add the interativity feature into the graphing, which I'll be expanding on later in this post. At the end of the post, I hope I can demonstrate the graphing capabilities of both packages, and how they can convert to each other.

Data Preparation

In this section I will be performing setup for further use in the post. I'll be working with the `landdata-state.csv` data file, which is available and referenced from the Introduction to R website. I'll also work with some of the built-in data frames such as `mtcars` and `diamonds` for convenience.

```
# Importing the libraries
library(ggplot2)
library(ggvis)
```

```
##
## Attaching package: 'ggvis'
```

```
## The following object is masked from 'package:ggplot2':
##
##     resolution
```

```
# Reading the data file
housing <- read.csv(file = "../data/landdata-states.csv", stringsAsFactors = FALSE)
```

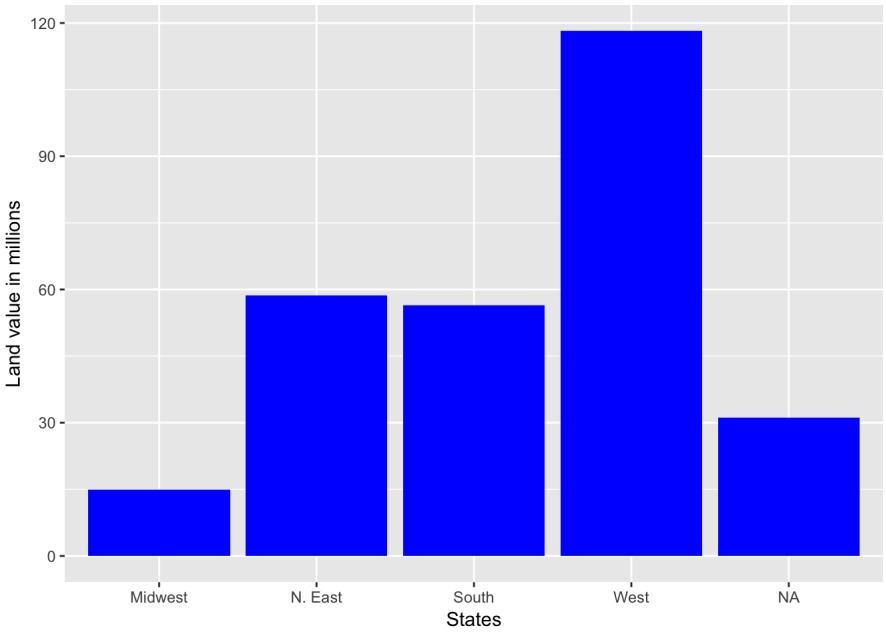
Basic Plotting with `ggvis` and `ggplot2`

In this section, we will be looking at some common statistical plots using functions from both the `ggvis` and the `ggplot2` packages. We will then be examining some of the differences in creating these graphs.

Barplots

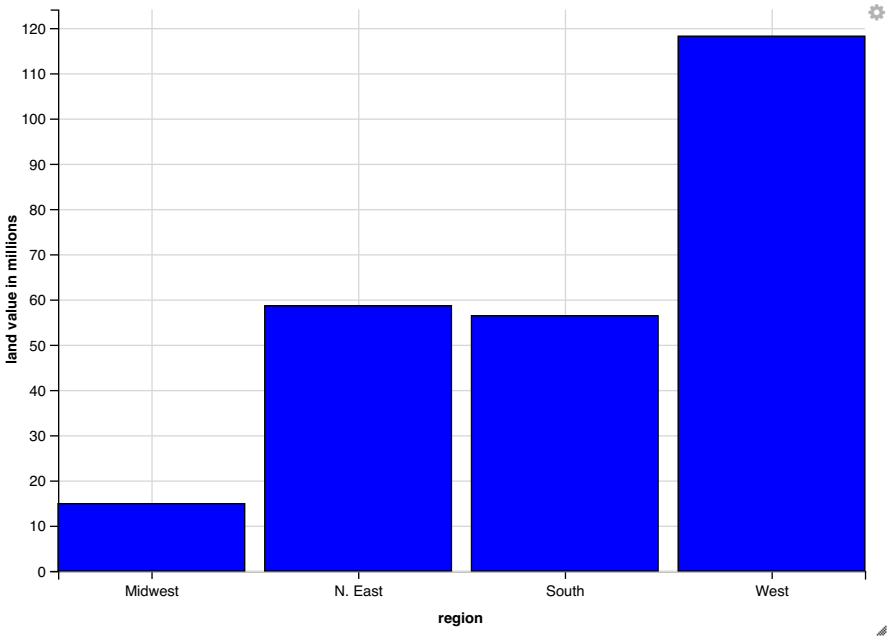
First of all, one of the most commonly used graphs in data visualization are the bar plots. Let us start with some simple examples. For instance, if we want to look at the land values by region, creating a bar plots by each region is a good way to do this. In `ggplot2`, we could do the following to create such a bar plot. The `stat = 'identity'` is used to specify that the heights of the bars are used to represent the values in the data:

```
# Simple barplot in ggplot2
ggplot(data = housing, aes(x = region,
                           y = Land.Value / 1000000)) +
  geom_bar(stat = 'identity', fill = "blue") +
  labs(x = "States", y = "Land value in millions")
```



Here we have the barplots we want above. Now we can start to create a similar graph using the functions in the `ggvis` package. As I have stated above, the general logic of `ggvis` and `ggplot2` is similar, but there are still some key differences between the 2. The most notable change in `ggvis` is that the common way to combine components in `ggvis` is the pronounced pipe `%>%` instead of `+`, even though the `+` is supported in `ggvis` as indicated in the Quick ggvis examples reference. `ggvis` utilizes the `add_axis` function to add additional features to the axes. For more information on axes and legends, see the reference on `ggvis` legends and axes:

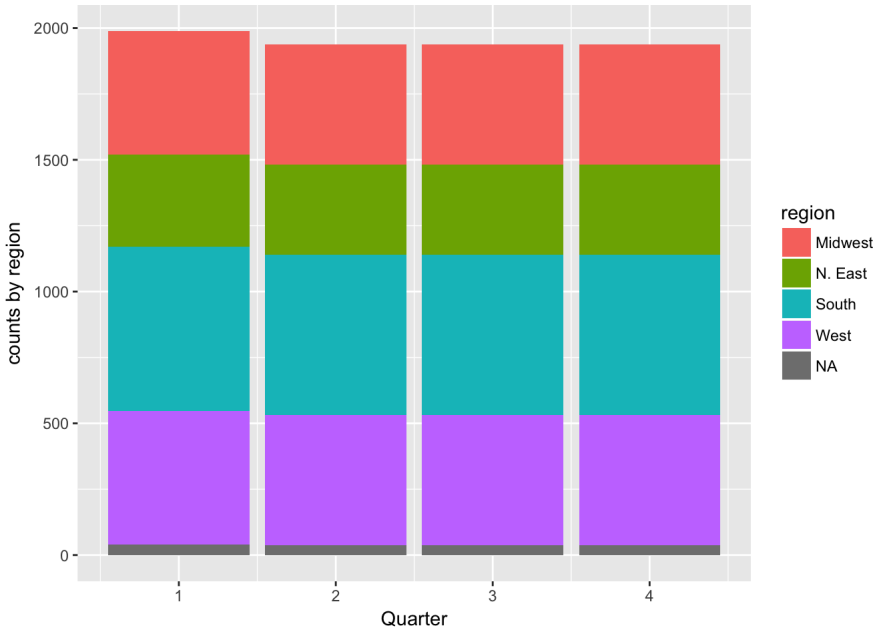
```
# Using add_axis to create labels
housing %>%
  ggvis (x = ~region, y = ~Land.Value / 1000000, fill:= "blue") %>%
  layer_bars() %>%
  add_axis("x", title = "region") %>%
  add_axis("y", title = "land value in millions")
```



Here we have the almost identical graph as the one created using `ggplot2`. Note that we are missing the column of `NA` values. According to the github thread provided in the reference, `ggvis` does not have consistent `NA` handling, which is probably the cause of this issue.

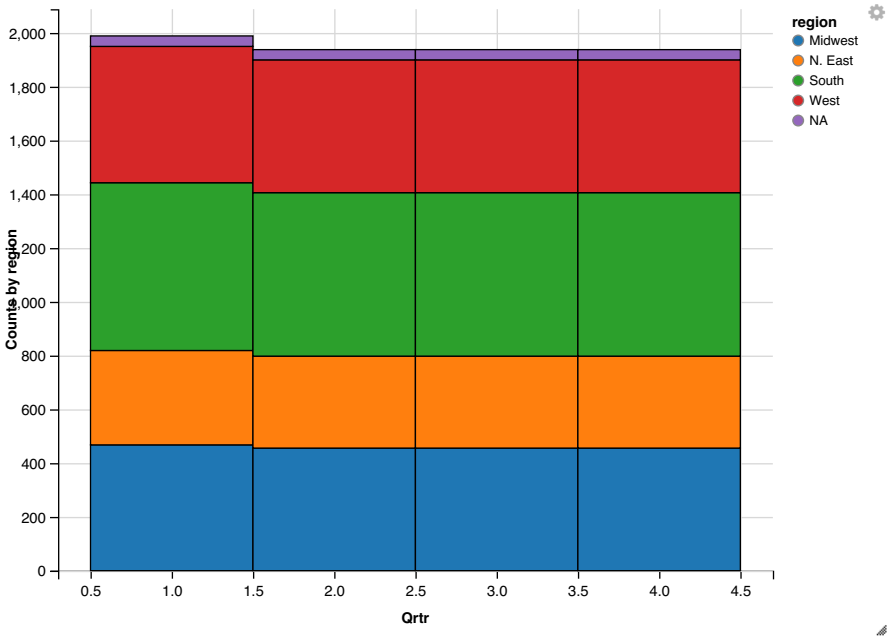
Having seen how the basic syntax works in `ggvis`, we can now look at a more complex example in creating bar plots. Here we want to examine the house distributions by its region and the quarter in which the house is sold. A good way to see this distribution is to use a stacked barplot. Here is generally how we do this in `ggplot2`:

```
# Generating a stacked barplot
ggplot(housing, aes(Qrtr, fill = region) ) +
  geom_bar() +
  labs(x = "Quarter", y = "counts by region")
```



The way we generate this graph with `ggvis` should be quite similar to the way in `ggplo2` . There is also a `fill` argument for the `ggvis` function. Note that in order for this to work, we need a `group_by` function for the colors to be grouped by the regions. With the pronounced pipe, the syntax is much like what we have in the `dplyr` package:

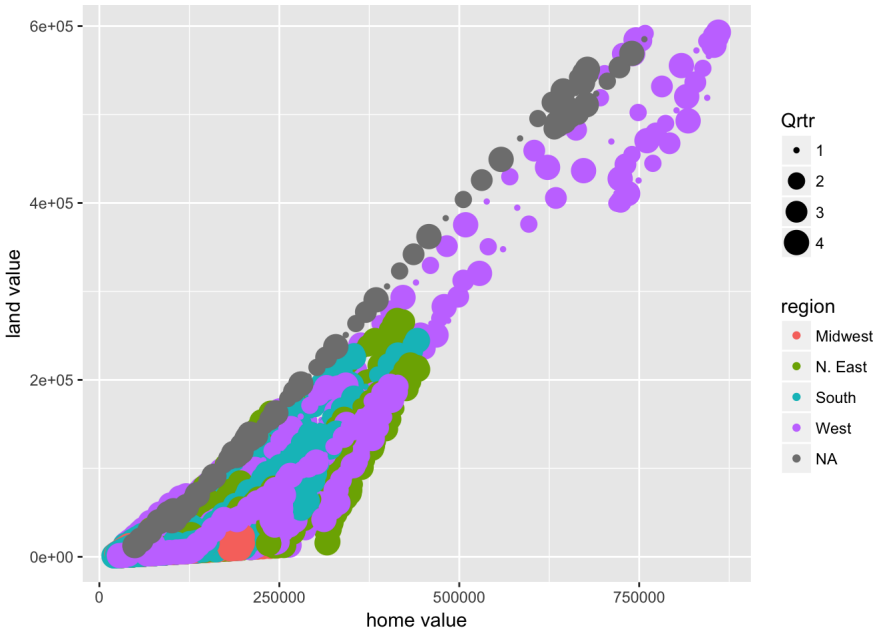
```
# Creating stacked barplot in ggvis
housing %>%
  ggvis (x = ~Qrtr, fill = ~region) %>%
  group_by(region) %>%
  layer_bars() %>%
  add_axis("x", title = "Qrtr") %>%
  add_axis("y", title = "Counts by region")
```



Scatterplots

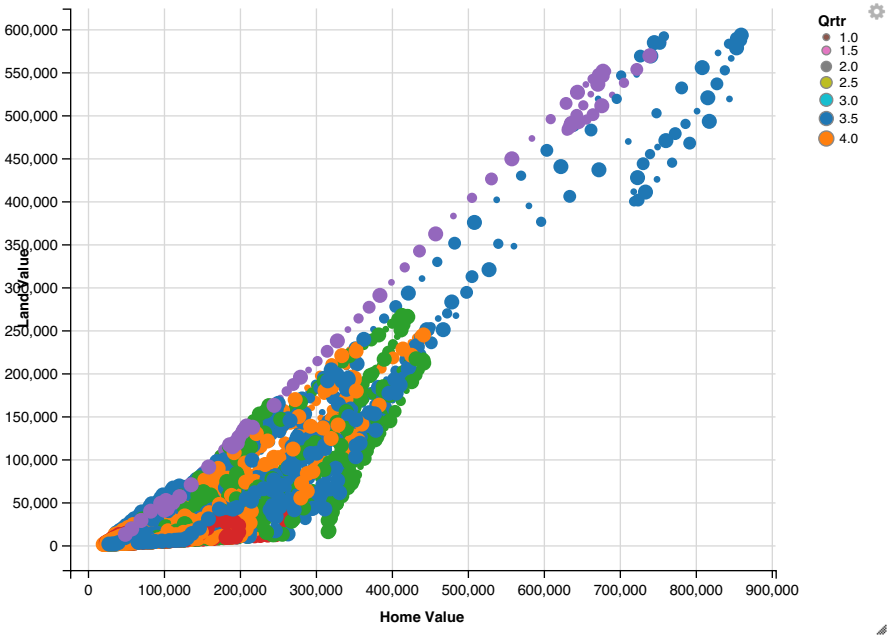
Having seen some examples of barplots, we can now examine some examples with scatterplots. Suppose we want to see the distribution between home value and land value. The best way to do this is clearly creating a scatterplot between the 2 variables. To make things more interesting, we make the size correspond to the quarter of the year, and the color correspond to the region of the house. Note that the legends generated in `ggplot2` on the right side. The legend is automatically generated and placed properl:

```
# A scatterplot on home.vaue vs land.value
# size = quarter
# color = region
ggplot(data = housing, aes(x = Home.Value, y = Land.Value)) +
  geom_point(aes(color = region, size = Qrtr)) +
  labs(x = "home value", y = "land value")
```



Now we want to do the same with `ggvis`. The general structure is similar to what we have created in barplots, except that we are using `layer_points` instead of `layer_bars`. We still have to group the data by `region` and `Qrtr` to correspond to the size and color. Pay special attention to the legend on the right side. `ggvis` does not really offer automatic adjusting the positions of multiple legends, so we have to manually add the `add_lenged` to generate the legend according to our graph. (size and color in this case). Without this line, the legends will overlap together and seem really messy; however, once properly generated, it will combine multiple scales into one and it is really nice to look at:

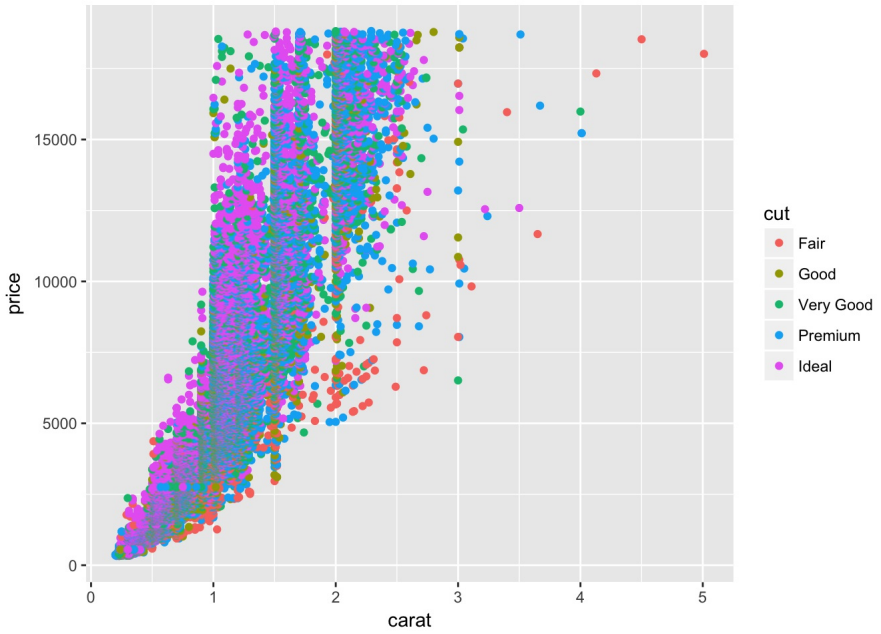
```
# A scatterplot between home value and land value
# Generate single legend using add_legend
housing %>%
  ggvis (x = ~Home.Value, y = ~Land.Value, fill = ~region, size = ~Qrtr) %>%
  group_by(region) %>%
  group_by(Qrtr) %>%
  layer_points() %>%
  add_axis("x", title = "Home Value") %>%
  add_axis("y", title = "Land Value") %>%
  add_legend(c("size", "fill"))
```



Layers

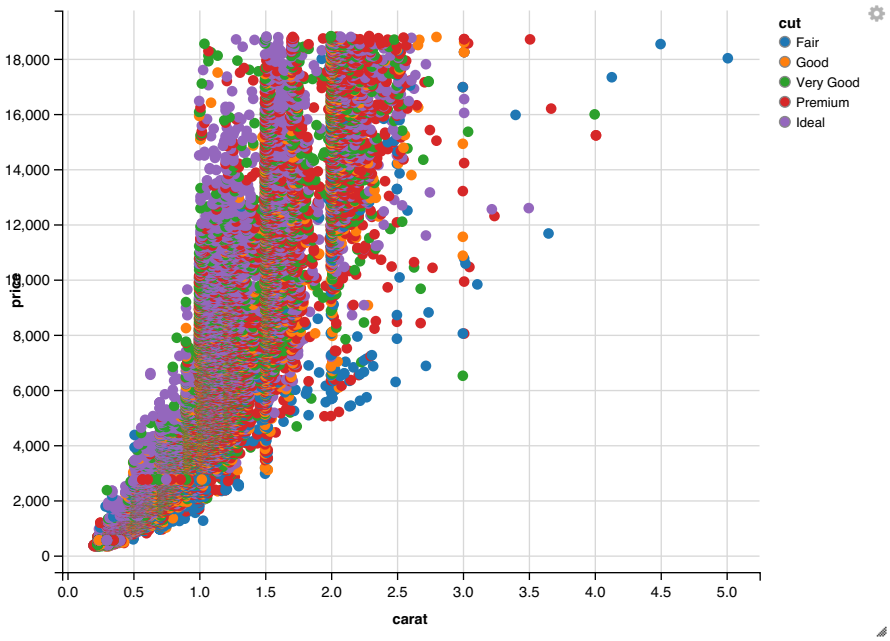
By now we should know that unlike `ggplot2` that utilizes functions `geom_point` or `geom_bar`, `ggvis` has a special set of commands that are called `layers` such as `layer_points` or `layer_bars`. In truth, the `geom` functions in `ggplot2` are just as the same as the `layers` in `ggvis`. There is in fact a `layer` function in the `ggplot2` package that makes the two package syntax seems even similar. We can look at the following example referenced from ggplot2 to ggvis, which utilizes the built-in diamond data frame. Note that the use of `layer` in the blog post is actually incorrect because one has to specify the `stat` and `position` argument for the `layer` function, otherwise it will error out:

```
# init. plot, specifying data and aes
# Adding layer using the layer function
p <- ggplot(data=diamonds, aes(x=carat,y=price,colour=cut)) +
  layer(geom="point", stat = "identity", position = "identity")
p
```



Similarly, we could create the same graph using `ggvis` and the `layer_points`, which is the common way we do this in `ggvis`:

```
# Create the graph using ggvis
p2 <- ggvis(diamonds, x=~carat, y=~price, fill=~cut) %>%
  layer_points()
p2
```



Interactive Graphing with ggvis

Now that we have gone through basic plottings with both packages, it seems that the functions that the two packages provide are quite similar; however, `ggvis` introduces the concept of interactivity, which is not included in the `ggplot2` package. The interactive document is somewhat similar to the shiny app, with widgets, sliders and other functions. However, it is called an interactive document because the interactive components could be nicely include into a markdown file and generate a document with such components. Here, we will discuss the two most common widgets, the input slider and the select box, in the `ggvis` interactive documents.

Important Notice

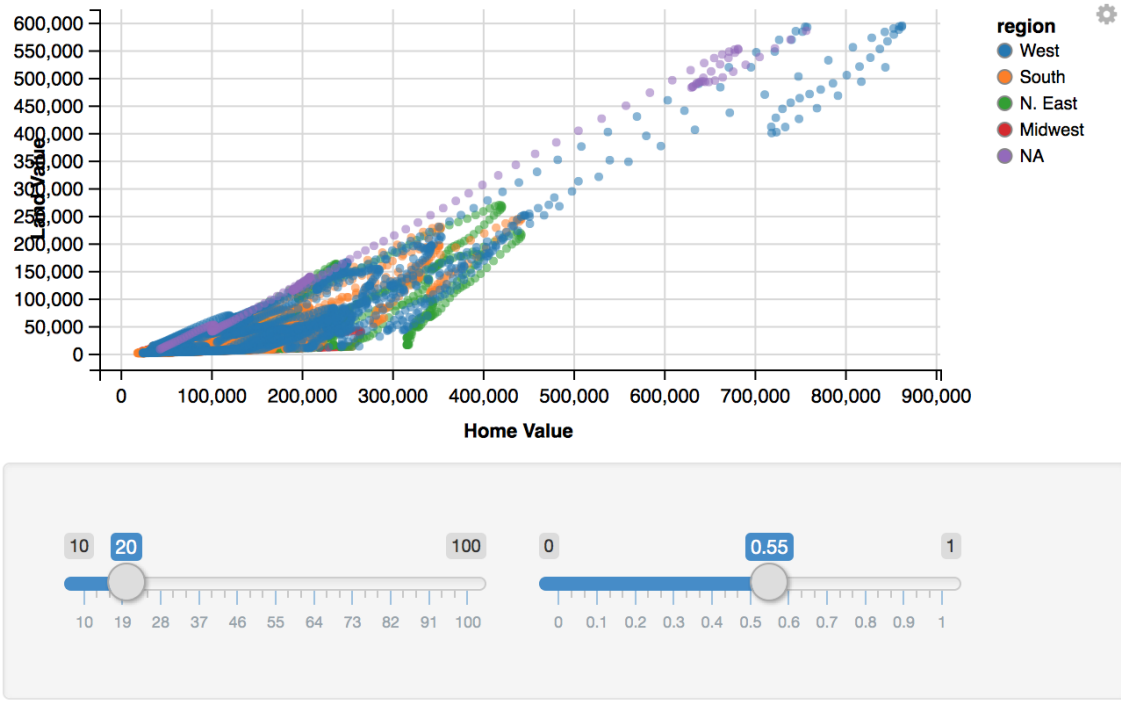
The interacivity comes at a cost. While it is true that the interactive component can be included in the document, it cannot be knitted into a html file since an html file can only show static components. So the only way to reproduce the following content is to copy the code into RStudio, add `runtime: shiny` to the header of the file and click on `run document` to see the interactive component. In a knitted html file, these components will be static with the warning: **Can't output dynamic/interactive ggvis plots in a knitr document**. I have set `eval` to be false for interactive components to avoid the constant warnings.

Input Slider

Here we still use the scatterplot from above. We want to be able to control the size of the points, so we assign a `input_slider` to size. We also want to be able to control the opacity of the points, so we assign a `input_slider` to opacity:

```
# With input slider for size and opacity
housing %>%
  ggvis (x = ~Home.Value, y = ~Land.Value, fill = ~region,
         size:= input_slider(10, 100),
         opacity:= input_slider(0,1)) %>%
  group_by(region) %>%
  layer_points() %>%
  add_axis("x", title = "Home Value") %>%
  add_axis("y", title = "Land Value") %>%
  add_legend(c("size", "fill"))
```

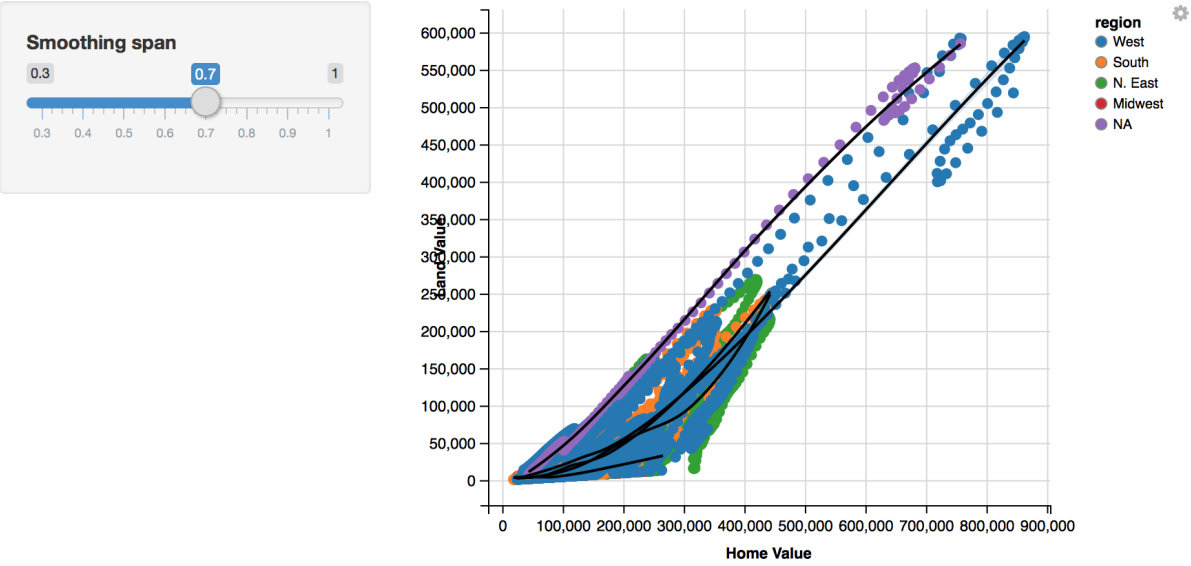
After proper execution, the resulting interactive component should look like the following image:



Not only can we control the points using input slider, we can pretty much assign anything to an input slider. For instance, if we want to add loess smoother to the above graph, and control the span with a slider, we could do the following. This example is inspired by examples from Quick ggvis examples in the reference:

```
# Housing Scatterplot with smoother input slider
housing %>%
  ggvis (x = ~Home.Value, y = ~Land.Value, fill = ~region) %>%
  group_by(region) %>%
  layer_points() %>%
  layer_smooths(se = TRUE, span = input_slider(min = 0.3, max = 1,
                                               value = 0.8, step = 0.1,
                                               label = "Smoothing span")) %>%
  add_axis("x", title = "Home Value") %>%
  add_axis("y", title = "Land Value") %>%
  add_legend(c("size", "fill"))
```

If it is executed properly, we could have the interactive component in the following image:

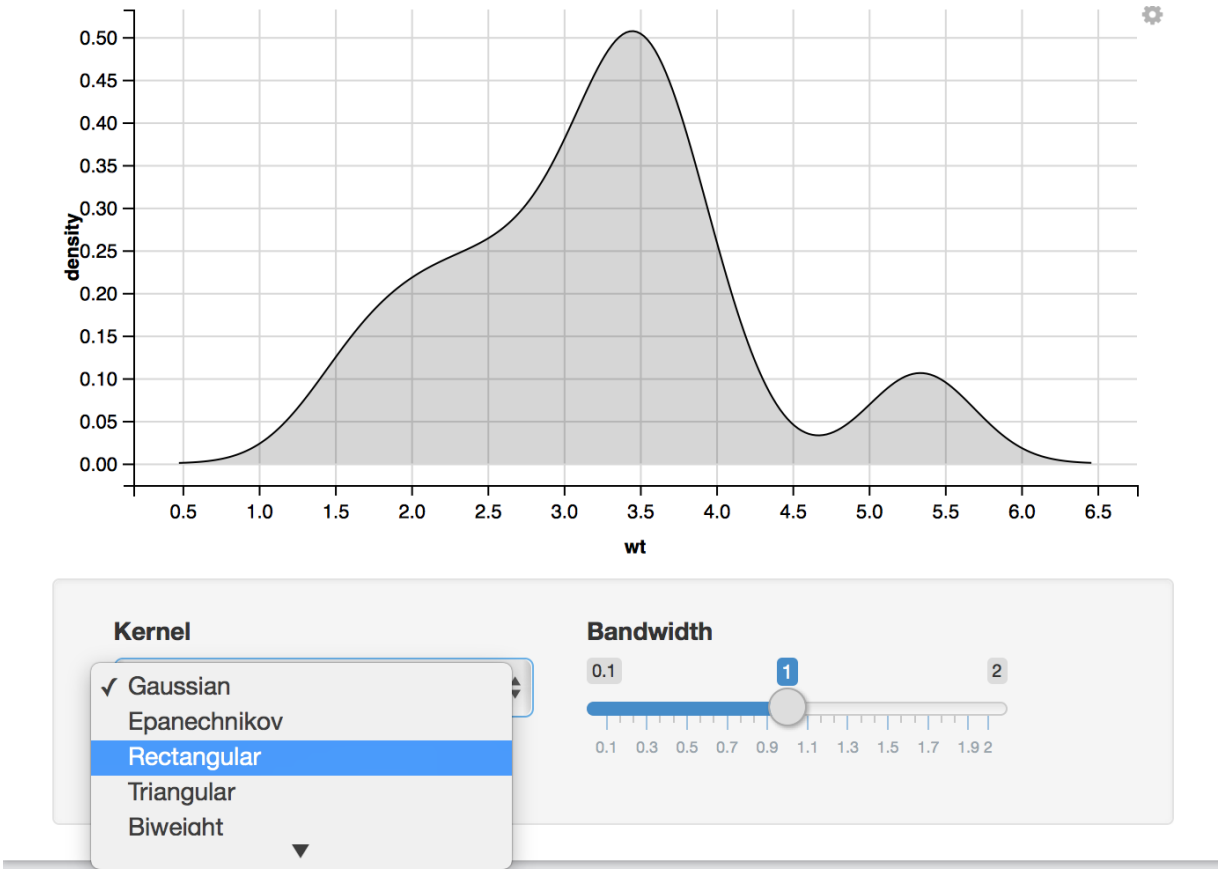


Select Box

Apart from the slider, we also have the select box option. Here I use an example from the quick ggvis examples in the reference, but change it to follow the `%>%` pipeline syntax. In this example, we are using `layer_densities` to plot the density using different density functions provided in the `kernel`. The reference site might be a little bit outdated since `layer_density` does not exist anymore. We also have an input slider here for the bandwidth:

```
# Mtcars density plot of wt
# with slider input for bandwidth
# select input for kernel
mtcars %>%
  ggvis(x = ~wt) %>%
  layer_densities(
    adjust = input_slider(.1, 2, value = 1, step = .1, label = "Bandwidth"),
    kernel = input_select(
      c("Gaussian" = "gaussian",
        "Epanechnikov" = "epanechnikov",
        "Rectangular" = "rectangular",
        "Triangular" = "triangular",
        "Biweight" = "biweight",
        "Cosine" = "cosine",
        "Optcosine" = "optcosine"),
      label = "Kernel")
  )
```

If it is executed properly, we should have the interactive component that looks like the following:



For more examples of `ggvis` interactive components, please see my first three references.

Comparisons Between `ggvis` and `ggplot2`

So far we have seen several comparions between the two packages, and also some new features introduced by `ggvis`. We can conclude some basic differences to help with the conversions between the 2 packages:

- For the layer function, we have `layer` -> `geom`
- For constructing mappings, we have `prop()` -> `aes()`
- For combining components, we have `%>%` -> `+`
- For creating graphs, we have `ggvis()` -> `ggplot()`

There are also some differences between the 2 packages, which I have not really covered in the post, but would like to point out here:

- Unlike `ggplot2`, there is no faceting in `ggvis`.
- Unlike `ggplot2`, which has a two-level hierachy (data and aes specifications in the plot in each layer), `ggvis` has essentially an unlimited hierachy.
- Unlike `ggplot2` which has a `qplot()` function for quick plotting, `ggvis` does not have this function. But it can be achieved by using `ggvis()` with no layers.

For more information on the similaries and differences between the 2 packages, please refer to the ggvis vs. ggplot2 article in the reference section.

Conclusion

In conclusion, this post mainly focuses on the graphing features of both `ggplot2` and `ggvis`, with some shifts to the interactivite components introduced by `ggvis`. As we can see from above, both packages provide a high level of flexibility in creating graphs. There is no way to tell which package is better for static plotting. However, `ggvis` has a significant advantage for creating dynamic document with interactive components, which are perfect for demonstration whether it is in a dynamic document or a website. I hope the reader of this post will find some of the examples to be interesting and helpful.

References

- ggvis basics: https://ggvis.rstudio.com/ggvis-basics.html#simple-layers
- Quick ggvis examples: http://ggvis.rstudio.com/0.1/quick-examples.html
- ggvis Interativity: http://ggvis.rstudio.com/0.1/interactivity.html
- ggvis vs. ggplot2: http://ggvis.rstudio.com/ggplot2.html
- ggplot2 to ggvis: http://jimhester.github.io/ggplot2ToGgvis/#conclusion
- Introduction to R graphics dataset: http://tutorials.iq.harvard.edu/R/Rgraphics.zip
- Guides: Axes and Legends: https://ggvis.rstudio.com/axes-legends.html
- Github Thread: https://github.com/rstudio/ggvis/issues/148