The Wonders of ggvis

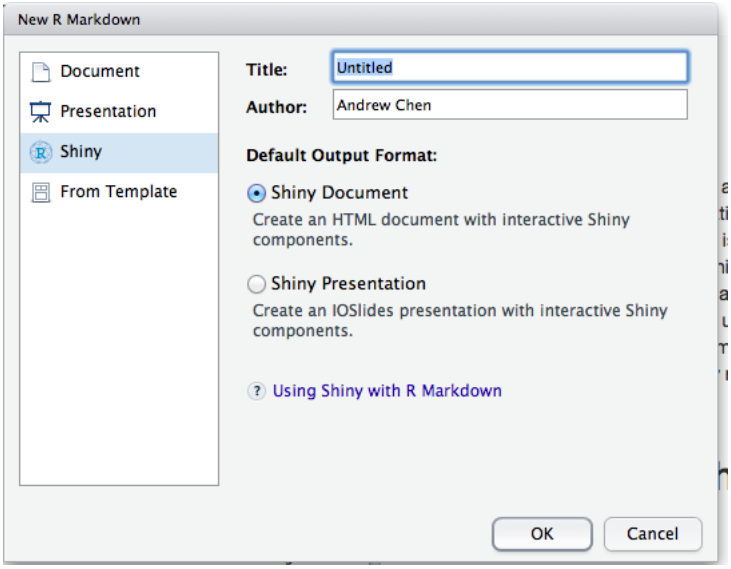
Andrew Chen

December 3, 2017

Introduction

We have played with ggplot2 for quite a while throughout this class, but this time I would like to explore another slightly different data visualization package called ggvis. The ggvis package, created in 2016 by Winston Chang and Hadley Wickham, is claimed to have taken "the best parts of ggplot2" and "combined them with the reactive framework of shiny." In other words, it can be described as an enhanced form of ggplot2 that allows analysts to render an interactive version of their data that users can play around with.

Due to its interactivity, ggvis is usually used alongside the implementation of interactive Shiny apps, but they can also be implemented in an Rmd file! Simply create an Rmd file and select the Shiny option, to create a Rmd file that can interactively render shiny components (including ggvis graphics)!

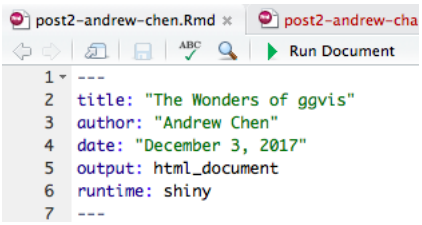


Creating an Rmd file with Shiny components

The heading of the Rmd file should then look like this:

```
---
title: "The Wonders of ggvis"
author: "Andrew Chen"
date: "December 3, 2017"
output: html_document
runtime: shiny
---
```

Normally, we can knit Rmd files to html, but unfortunately Rmd files with interactive Shiny components cannot be knitted into a static HTML webpage. Therefore, the code using ggvis that I provide here will be followed by screenshots of what the resulting plots should look like; you can copy all the code, paste it into the "Shiny Document" Rmd, and press "Run Document" to see and play around with the plots.



The "Run Document" button appears on the top right of the image.

Comparisons with ggplot2

Similarities

Below is a simple table displaying various basic ggplot functions to their corresponding ggvis functions. As you can see, there isn't too much of difference between the two; there's only the fact that "geom" is replaced with "layer" (and that the ggvis functions are plural). So once you've learned ggplot, ggvis isn't too hard to learn.

ggplot	ggvis	function
geom_point	layer_points	scatterplot
geom_histogram	layer_histograms	histogram
geom_boxplot	layer_boxplots	boxplot
geom_line	layer_lines	line plot
geom_smooth	layer_smooths, layer_model_predictions	regression lines
geom_text	layer_text	text as data points

ggplot	ggvis	function
geom_bar	layer_bars	barplot

Differences

I would then like to visualize certain plots using both ggplot2 and ggvis, to get a sense of the slight difference between the functionalities of both packages. The data I will be using is from the built-in 'iris' dataset, which lists the measurements (in centimeters) of the sepal length and width and petal length and width of 50 flowers from each of 3 species of iris: *Iris setosa*, *versicolor*, and *virginica*.

First we will load the necessary packages. Note: Piping is already built into the ggvis package, so there is no need to load the "magrittr" package when using ggvis; I'm just loading it to use for ggplot2 implementations.

```
knitr::opts_chunk$set(echo = TRUE)
library(ggvis)
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

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

```
library(magrittr)
library(dplyr) #we will see the use of this soon!
```

```
##
## Attaching package: 'dplyr'
```

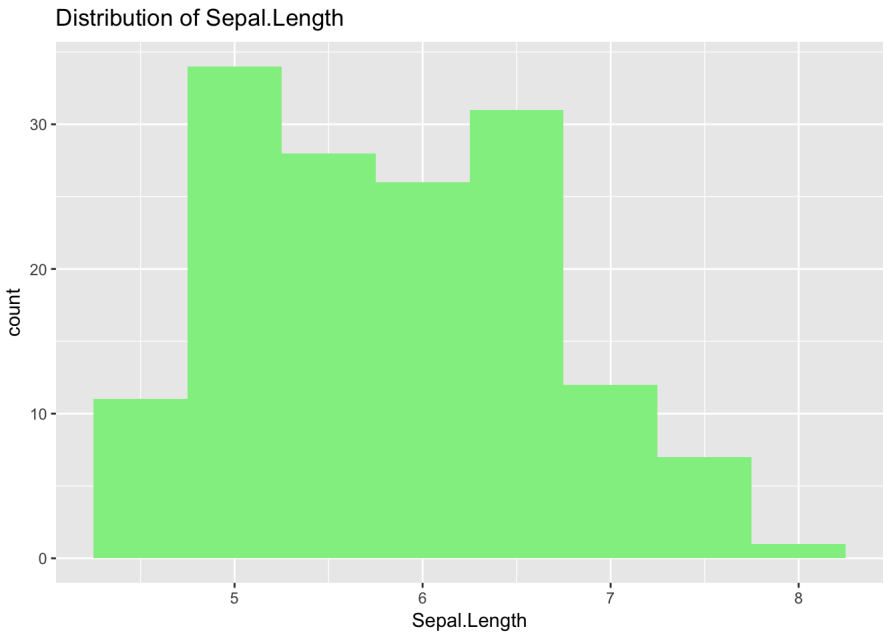
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Histogram

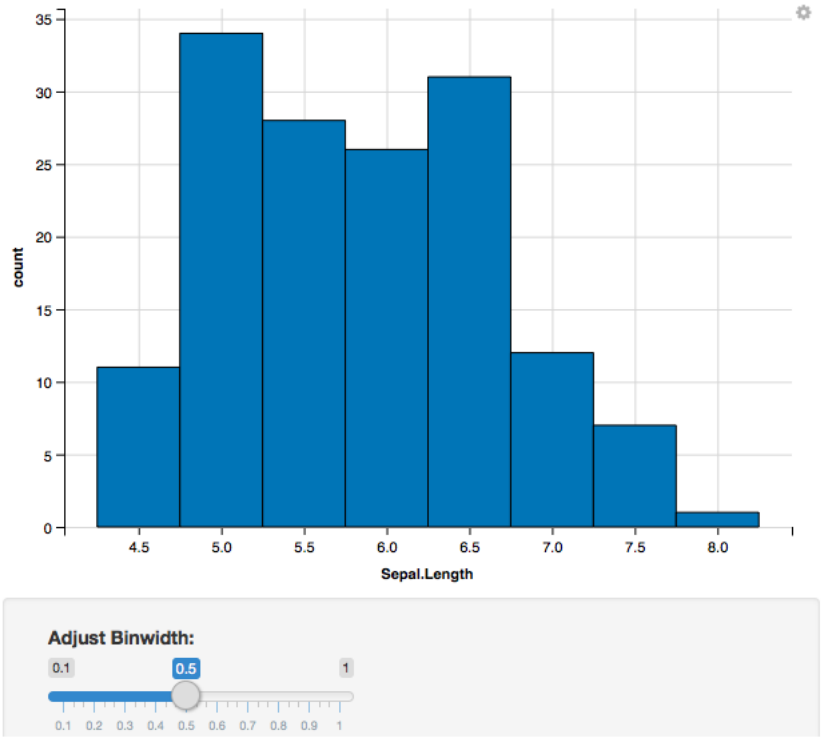
We will start first with the histogram of Sepal Length. For ggplot this is pretty straightforward; every variable that is assigned a value is set in stone.

```
iris %>% ggplot(aes(x = Sepal.Length)) + geom_histogram(binwidth = .5, fill = "light green") + ggtitle("Distributi
on of Sepal.Length")
```



However, this is not necessarily the case for ggvis. The cool thing about this package is that you can freely change the features of a plot by assigning a certain feature to a function that produces interactive controls, such as "input_slider" or "input_select." In this case I used an input_slider to allow users to adjust the binwidth of the histogram.

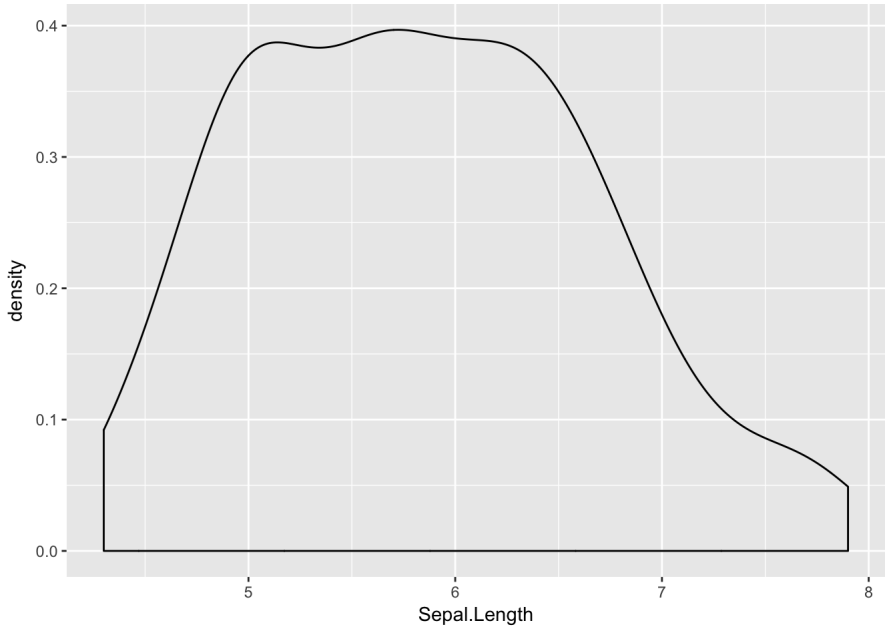
```
iris %>% ggvis(x = ~Sepal.Length) %>%
  layer_histograms(
    fill = "light blue",
    width = input_slider(.1, 1, value = .5, step = .1, label = "Adjust Binwidth:")
    #creating slider with min = .1, max = .1, selected value as 0.5, and intervals of .1
    %>% hide_legend("fill")
  )
```



Density Plot

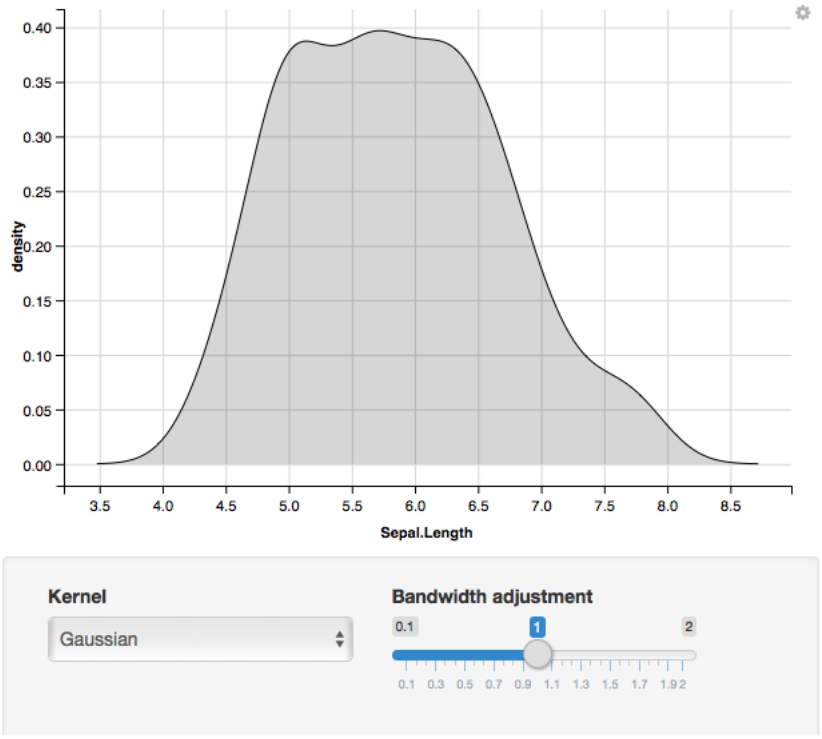
We will continue on to a continuous version of the histogram, the density plot. The following compares the density plots from ggplot and ggvis.

```
iris %>% ggplot(aes(x = Sepal.Length)) + geom_density(kernel = "gaussian")
```



Here, along with the binwidth adjustment, we can also use input_select to allow users to select the method of kernel density estimation (KDE) to estimate the probability density function.

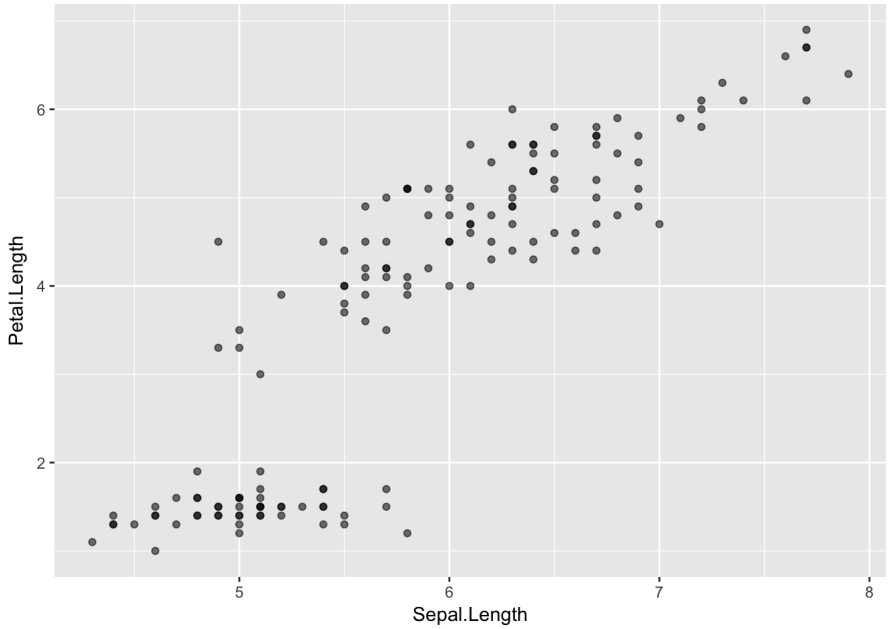
```
iris %>% ggvis(x = ~Sepal.Length) %>%
    layer_densities(
      adjust = input_slider(.1, 2, value = 1, step = .1, label = "Bandwidth adjustment"),
      kernel = input_select(
        c("Gaussian" = "gaussian",
          "Epanechnikov" = "epanechnikov",
          "Rectangular" = "rectangular",
          "Triangular" = "triangular",
          "Biweight" = "biweight",
          "Cosine" = "cosine",
          "Optcosine" = "optcosine"),
        #matching input_select choices with corresponding KDE methods
        label = "Kernel")
    )
```



Scatterplot

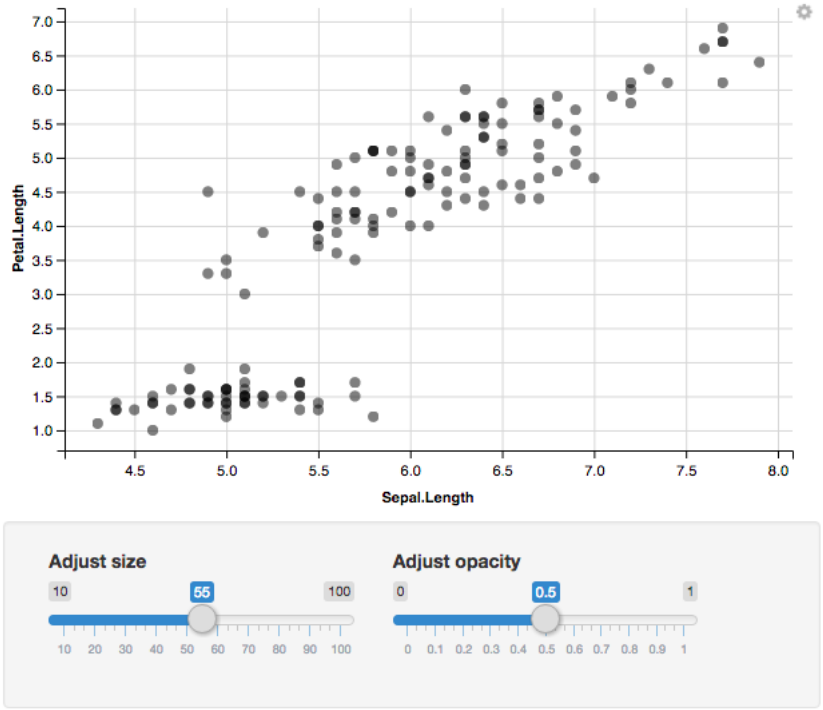
We will then explore the scatterplots of the dataset for both packages. Here is the ggplot scatterplot of Sepal Length vs. Petal Length:

```
iris %>%
  ggplot(aes(x = Sepal.Length, y = Petal.Length)) + geom_point(aes(alpha = .5)) + theme(legend.position = "none")
```



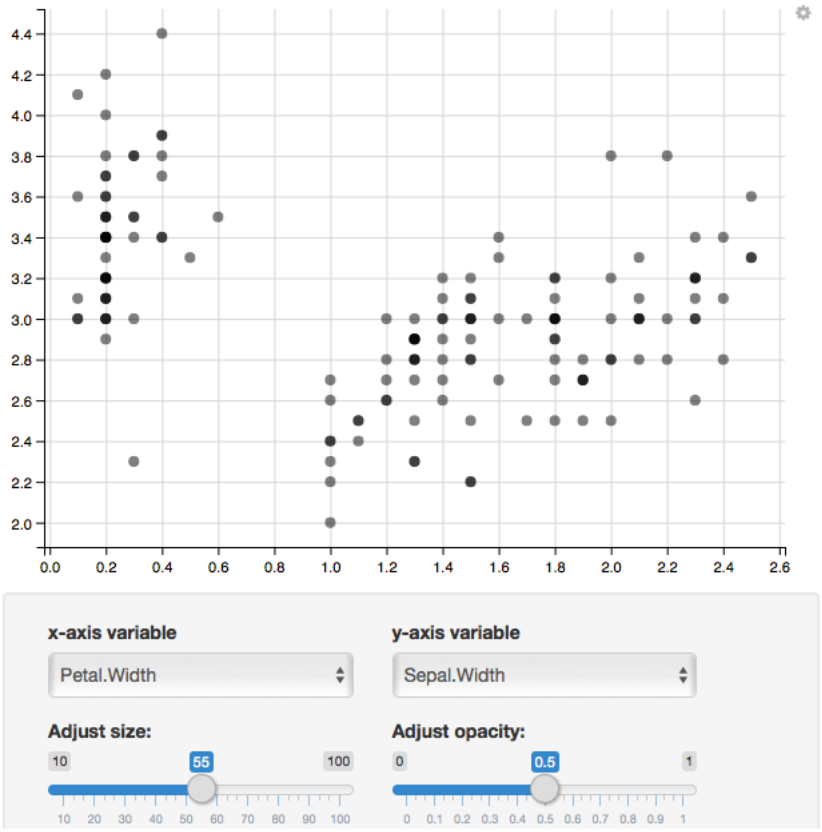
Here's a simple variation of the ggplot scatterplot, just with the options to change the size and opacity of data points.

```
iris %>%
  ggvis(x = ~Sepal.Length,
        y = ~Petal.Length,
        size := input_slider(10, 100, step = 5, label = "Adjust size"),
        opacity := input_slider(0, 1, step = .05, label = "Adjust opacity")
        ) %>%
  layer_points()
```



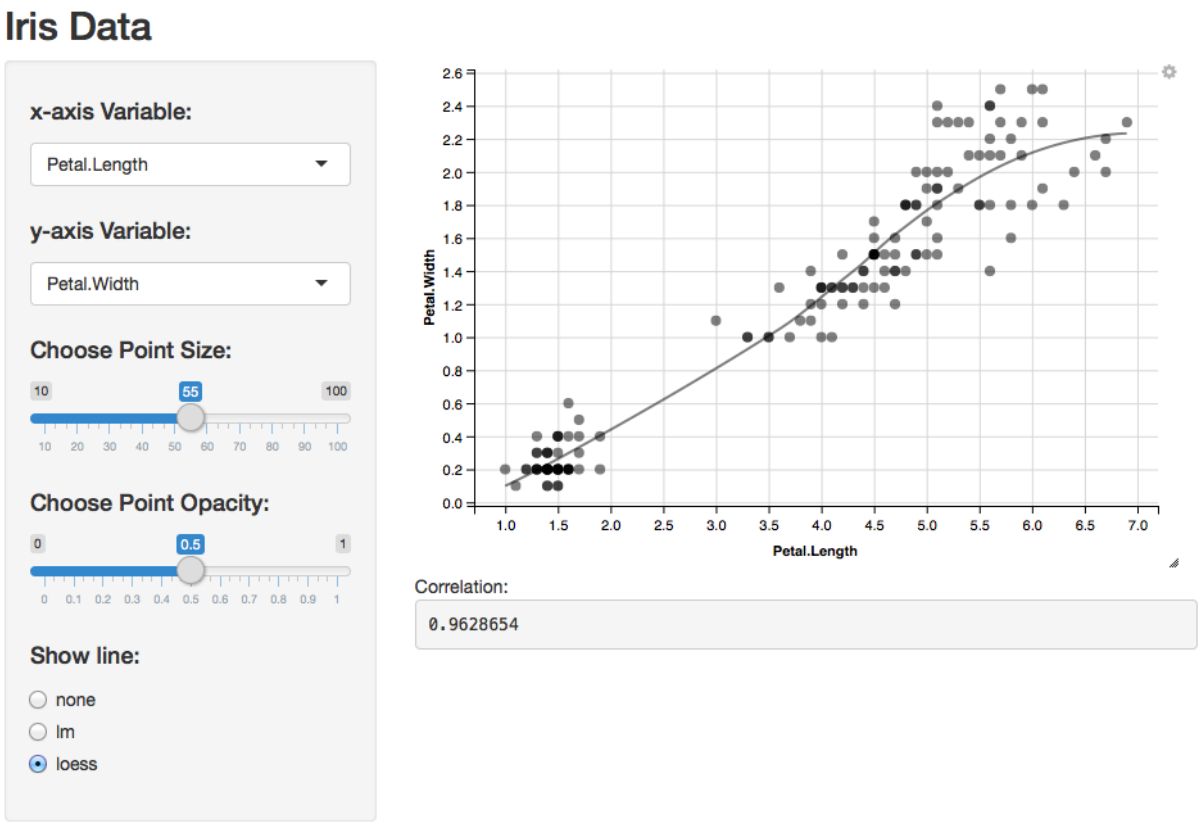
It might be fun to see the dots change according to your free will, but is there anything else we can alter in the scatterplot to produce more versatile results? The answer is yes; we can even assign the x and y variables that are plotted to input_select functions, so that you can select which two variables you want to compare. If there are numerous columns to the data, you can simply use names(insert name of data) instead of a c() vector to include all the columns as selectable options. And if only a number of those columns are numeric, you can use the bracket symbol [] to subset specific column names. I just enumerated the column names in the input_select function to give you a better sense of what's going on.

```
iris %>%
  ggvis(x = input_select(c('Petal.Width', 'Sepal.Length',
                           'Petal.Length', 'Sepal.Width'), map = as.name,
                         label = "x-axis variable"),
        y = input_select(c('Petal.Width', 'Sepal.Length',
                           'Petal.Length', 'Sepal.Width'), map = as.name,
                         label = "y-axis variable",
                         selected = 'Sepal.Width'),
        #creating lists of options for numeric columns; this can also be achieved through names(iris)[1:4], omitti
ng the "Species" column
        size := input_slider(10, 100, step = 5, label = "Adjust size:"),
        opacity := input_slider(0, 1, step = .05, label = "Adjust opacity:")) %>%
  layer_points() %>%
  add_axis("x", title = "") %>%
  add_axis("y", title = "")
```



We can go even further to introduce regression lines based off the data, but at that point it's best to implement ggvis within a Shiny app to produce such interactive results. The following code can be copied into a Shiny app and ran to visualize such lines. This is largely similar to what we did in HW04, including the display of the correlation between the two variables. Note: The regression lines do not work when a variable is plotted against itself.

```
library(shiny)
library(ggvis)

ui <- fluidPage(

   titlePanel("Iris Data"),

   sidebarLayout(
      sidebarPanel(
        #interactive select list inputs for choosing variables and slider inputs for adjusting plot features, alon
g with radio buttons to select regression line methods
         selectInput("x",
                     label = h4("x-axis Variable:"),
                     names(iris)[1:4],
                     selected = "Sepal.Length"),
         selectInput("y",
                     label = h4("y-axis Variable:"),
                     names(iris)[1:4],
                     selected = "Sepal.Width"),
         sliderInput("size",
                     label = h4("Choose Point Size:"),
                     min = 10, max = 100, value = 55, step = 5),
         sliderInput("opacity",
                     label = h4("Choose Point Opacity:"),
                     min = 0, max = 1, value = .5),
         radioButtons("line",
                      label = h4("Show line:"),
                      c("none", "lm", "loess"))
      ),

      mainPanel(
        #outputting the scatterplot along with the correlation of the plotted variables
         ggvisOutput("scatterplot"),
         textOutput("correlation"),
         verbatimTextOutput("corr")
      )
   )
)

server <- function(input, output) {
   output$scatterplot <- reactive({
     #We can use the prop function to assign the inputs that are selected from the selectInputs into variables tha
t can be used in ggvis. In this case, x is "assigned" to whatever gets selected in the first list, while y is "ass
igned" to whatever gets selected in the second list. R can then recognize var1 and var2 when they are actually ass
igned to x and y when calling ggvis.
     var1 <- prop("x", as.symbol(input$x))
     var2 <- prop("y", as.symbol(input$y))
     method <- input$line

     #assigning x and y to var1 and var2
     plot <- iris %>%
       ggvis(x = var1, y = var2, opacity:= input$opacity) %>%
       layer_points(size := input$size)

     #using if statements for the situations in which the "loess" or "lm" radio buttons are selected
     if (method == "loess") {
       plot <- plot %>% layer_smooths()
     } else if (method == "lm") {
       plot <- plot %>% layer_model_predictions(model = "lm")
     }

     #renders the finalized plot onto the app
     plot %>% bind_shiny("scatterplot")

   })

   output$correlation <- renderText({
     "Correlation:"
   })

   output$corr <- renderPrint({
     c1 <- which(names(iris) == input$x)
     c2 <- which(names(iris) == input$y)
     cat(cor(iris[, c1], iris[, c2]))
   })
}

#running the app
shinyApp(ui = ui, server = server)
```

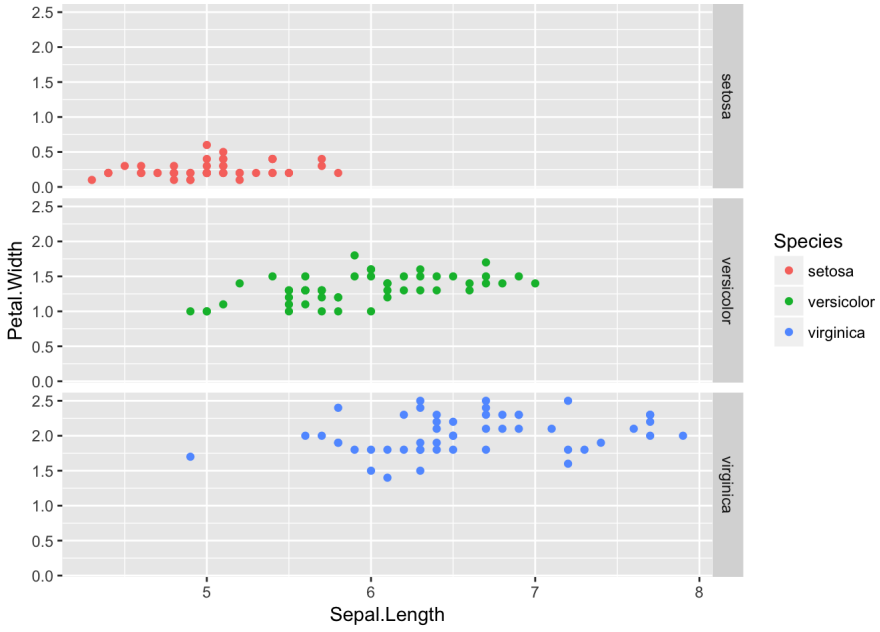


An example from running the app (using the most correlated data)

Disadvantages

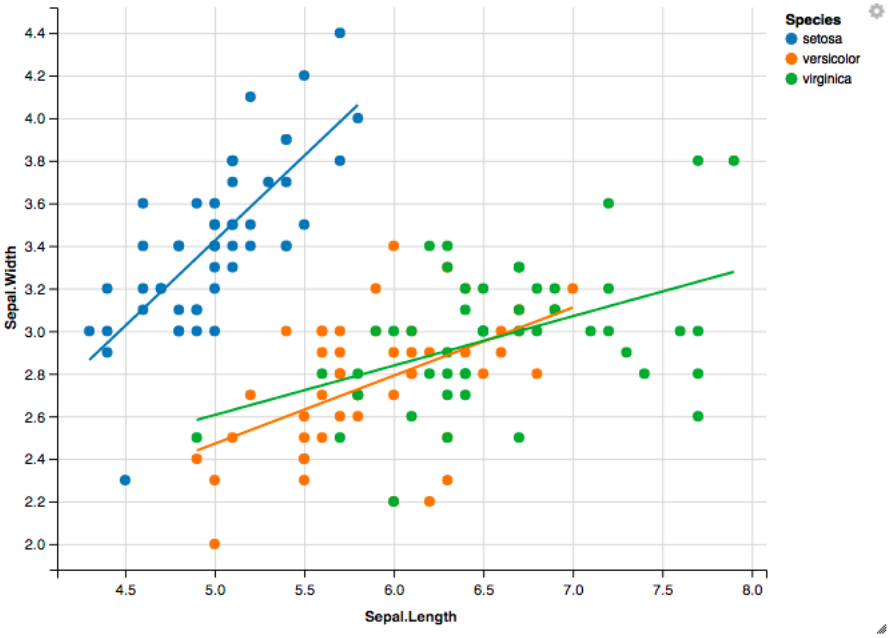
While it seems that ggvis surpasses ggplot2 in terms of interactivity, there are still some features that ggvis lacks compared to ggplot. First of all, ggvis does not have the facetting feature that ggplot has.

```
#facetting scatterplots of Sepal Length vs. Petal Width according to Species
iris %>%
  ggplot(aes(x = Sepal.Length, y = Petal.Width)) + geom_point(aes(color = Species)) + facet_grid(Species ~ .)
```



While we are unable to break up ggvis plots into groups, we can still create a plot that is somewhat similar to ggplot facetting, using the group_by function in the dplyr package. The following code generates a plot that is grouped and color-coded based on iris species and draws regression lines for each group.

```
iris %>%
  ggvis(x = ~Sepal.Length, y = ~Sepal.Width) %>%
  layer_points(fill = ~Species) %>%
  group_by(Species) %>%
  layer_model_predictions(stroke = ~Species, model = "lm") #Drawing regression lines for each Species
```



ggvis also lacks some of the plots that ggplot2 supports, such as contour plots, dotplots, violin plots, and correlation charts. Furthermore, since it is still a relatively new data visualization package, there are still a few bugs and abnormalities that have been discovered or have yet to be discovered, as its usage increases.

So what should I use?

If you just need to plot a set of data with fixed variables and features, with no need for interactive inputs and effects, ggplot2 works fine. In other words, for *static* plots, use ggplot2.

But if you want to allow users to adjust certain features of the plot, ggvis (along with Shiny app) is a useful tool. In other words, for *interactive* plots, use ggvis.

Conclusion

Last time I talked about "the wonders of ggplot2," in which I introduced various extensive features of ggplot2 to reinforce the idea that even just one subset of data can yield so many visual representations. I would like to further extend such a message in this post, adding that among such data visualizations, there are even options to make your plots interactive using ggvis, allowing users to freely adjust features of the visualization as they wish. Also, although ggvis does have some slight differences with ggplot2 in terms of function names and additional interactive functions, it's really not a hard package to use, since its essence is largely similar to that of ggplot2. Hopefully this post was helpful in furthering your understanding of the versatility of data visualizations!

Sources

https://stackoverflow.com/questions/40322713/how-to-embed-ggvis-interactive-charts-in-rmarkdown

https://stackoverflow.com/questions/24840804/is-it-possible-with-ggvis-to-interactively-change-the-variables-for-the-x-and-y

https://stats.stackexchange.com/questions/117078/for-plotting-with-r-should-i-learn-ggplot2-or-ggvis

https://ggvis.rstudio.com/cookbook.html

http://www.londonr.org/presentations/2014/11/LondonR_-_Introduction_to_ggvis_-_Aimee_Gott_-_20141125.pdf

http://ggvis.rstudio.com/0.1/quick-examples.html#regression-lines

https://shiny.rstudio.com/articles/interactive-docs.html

https://cran.r-project.org/web/packages/ggvis/ggvis.pdf

https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/iris.html