Post 02: Data Visualization Using ggvis

Lisa Zhou

12/2/2017

Introduction

R has many different ways to create visualizations with different data sets, such as ggplot2. Another visualization tool to help you create interactive graphics is `ggvis`. The purpose of this post is to explore the different aspects and features of `ggvis` in creating interactive graphics. The post will go through basics of `ggvis` and interesting applications of how it can be used to create visual graphics.

Motivation

After seeing `ggvis` briefly utilized in lecture and homework, I wanted to explore ggvis more. I was interested in the different features of the package and it's ability to create interactive graphics compatible with Shiny apps and web browsers.

Background

The package ggvis was created by Hadley Wickham in order create an easy way to describe interactive web graphics in R. `ggvis` combines the grammar of graphics from `ggplot2`, reactive programming from shiny, and the grammar of data transformation from dplyr in order to build interactive graphics for exploratory data analysis.

Loading Packages and Data Sets

To start implementing data visualization using `ggvis`, the `ggvis` package has to be installed and loaded into the current R session. Since `ggvis` is available in the CRAN library, we can simply install it using `install.packages()`.

```
# installing the package ggvis
install.packages("ggvis")
```

```
#load packages to use
library(ggvis)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## 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
```

```
library(shiny)
```

Throughout this post, we'll be utilizing data from the built-in datasets in R (mtcars, iris).

Basics

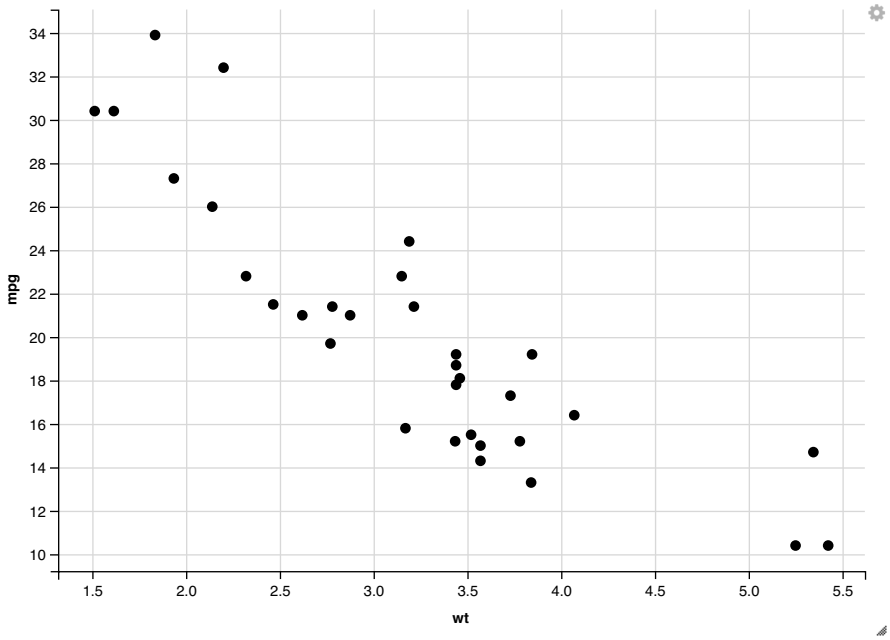
Every `ggvis` graphic starts with a call to `ggvis()`. The first argument is the data set you'll be using and the other arguments describe how to map variables to visual properties. In order to plot something, we have to tell `ggvis` how to display data by layering visual elements, for example with layer_points().

Three basic elemnts to a ggvis plot: - Dataset: we need to pass in the name of dataset in which we want to build plots for - Variables: features we want to use - Plotting Tools: options to plot/ format/add layers

ggvis vs ggplot2 Previously in this course we have learned how to create plots and graphics using `ggplot2`. It turns out that `ggvis` and `ggplot2` share many familiar concepts. Knowing the syntax and ideas behind `ggplot2` make learning `ggvis` much more feasible. Let's talk about some of the main differences between the two packages. Main Differences - ggplot2: layer, geom elements -> ggvis: `layer()` function - ggplot2: `aes()` -> ggvis: `props()` - use pipe operator `%>%` to combine components in `ggvis` instead of `+` in `ggplot2` - Facetting is not supported in `ggvis` like it is in `ggplot2`

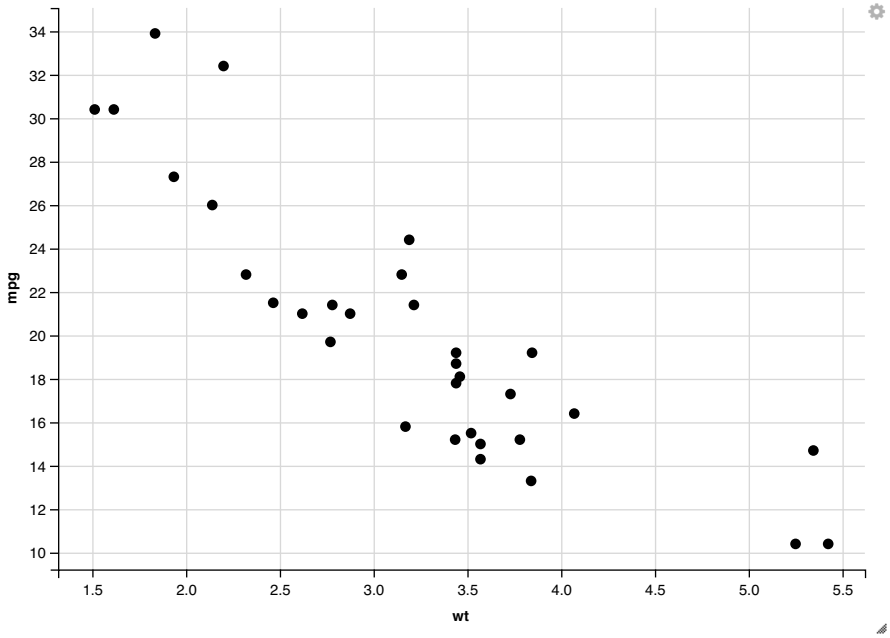
Now that we know some of the fundamentals of `ggvis` let's get started with some examples! First, let's see how to use `ggvis` to visualize the relationship between weight of cars and mileage from the `mtcars` data set already included in R.

```
# creating a  simple scatterplot using ggvis
layer_points(ggvis(mtcars, x = ~wt, y = ~mpg))
```



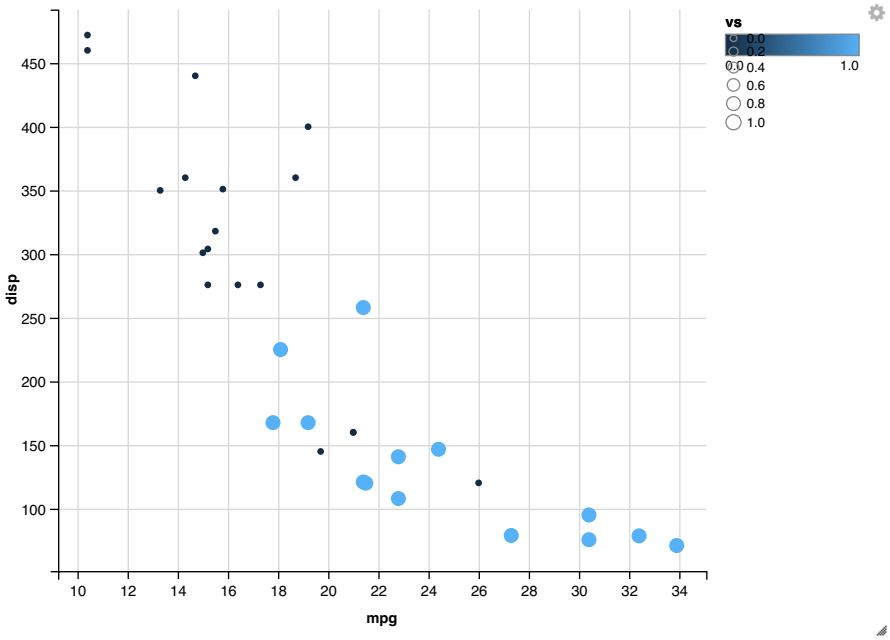
The ggvis package uses the pipe operator %>% to combine multiple calls and add multiple layers to plots.The code above works to create scatterplots, but often times it easier and more convenient to use `%>%` especially if we wanted to add multiple elements to the same plot.

```
# using the pipe operator to create the same plot
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points()
```



Additionally, it is possible to add more variables to the plot by mapping other visual properties like fill, stroke, size, and shape.

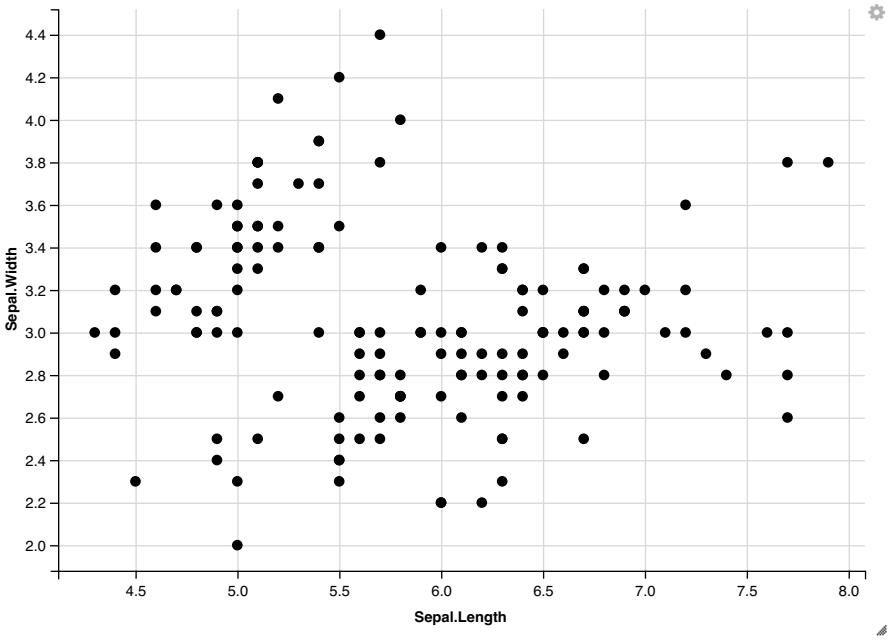
```
# adding fill and size arguments to customize the plot
mtcars %>%
  ggvis(~mpg, ~disp,fill = ~vs, size = ~vs) %>%
  layer_points()
```



The use of the pipe operator gives you a lot of power to also modify the plots using dplyr data visualization.

For this next example, we're going to use dplyr in conjunction with `ggvis` in order to change the scale of the x-axis and and also the axis titles. Let's start off by seeing what happens when we create a scatterplot just like before for another built-on data set `iris`.

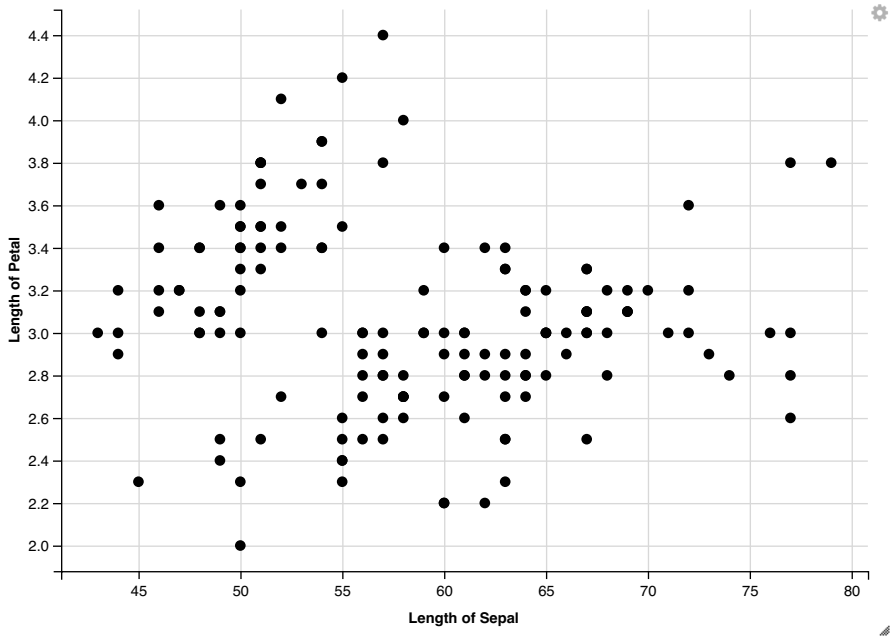
```
# create basic scatterplot in ggvis
iris %>%
ggvis( x = ~Sepal.Length, y = ~Sepal.Width) %>%
layer_points()
```



Now let's incorporate the use of dplyr in order to change the scale and axis titles.

```
# using dplyr to change the scale of x-axis
require(dplyr)
iris %>%
ggvis( x = ~Sepal.Length, y = ~Sepal.Width) %>%
mutate(Sepal.Length = Sepal.Length*10) %>%
layer_points() %>%

#change axis titles using add_axis()
add_axis("x",title = "Length of Sepal") %>%
add_axis("y",title = "Length of Petal")
```



Interaction

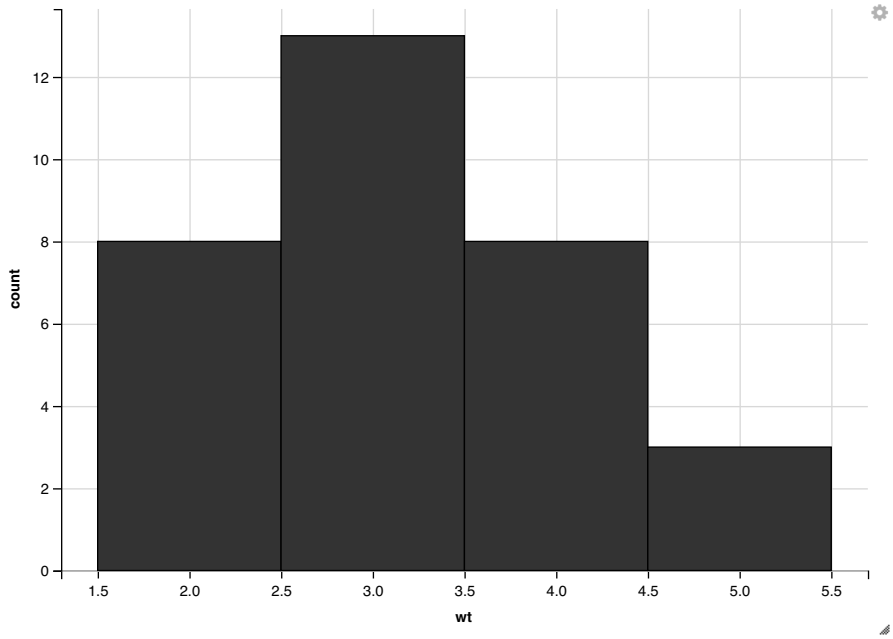
After covering the fundamentals of creating graphics using `ggvis`, let's make more sophisticated graphs by adding an element of interacton! We can use interactive controls to map visual properties to variable or set them to specific values. `ggvis` interactivity is built to complement the reactive programming in Shiny. In order to create something interactive, we can replace constant values with functions(usually starting with `input_`) that produce interactive controls.

Note: The interactive plots won't show up correctly in the knitted file, but will be reproducible in R Studio. The knitted file shows the plot for a set slider input. The knitted file will generate a static (non-interactive) version of the plot.

Let's go back to the `mtcars` data set and make some interactive plots! Let's start off by creating an interactive histogram.

```
# adding interactive slider to change the width of the bins in a histogram
mtcars %>%
  ggvis(~wt) %>%
  layer_histograms(width =  input_slider(0, 2, step = 0.10, label = "width"))
```

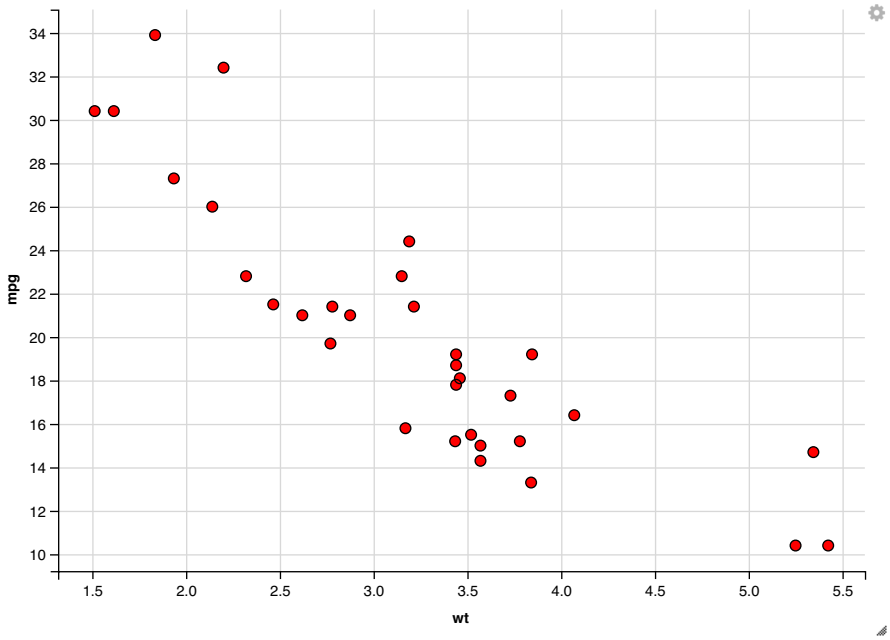
```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```



Next, let's make a slider to change the size of the points on the plot.

```
mtcars %>%
  ggvis(~wt, ~mpg, size := input_slider(1, 100)) %>%
  layer_points(fill := "red") %>%
  layer_points(stroke := "black", fill := NA)
```

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

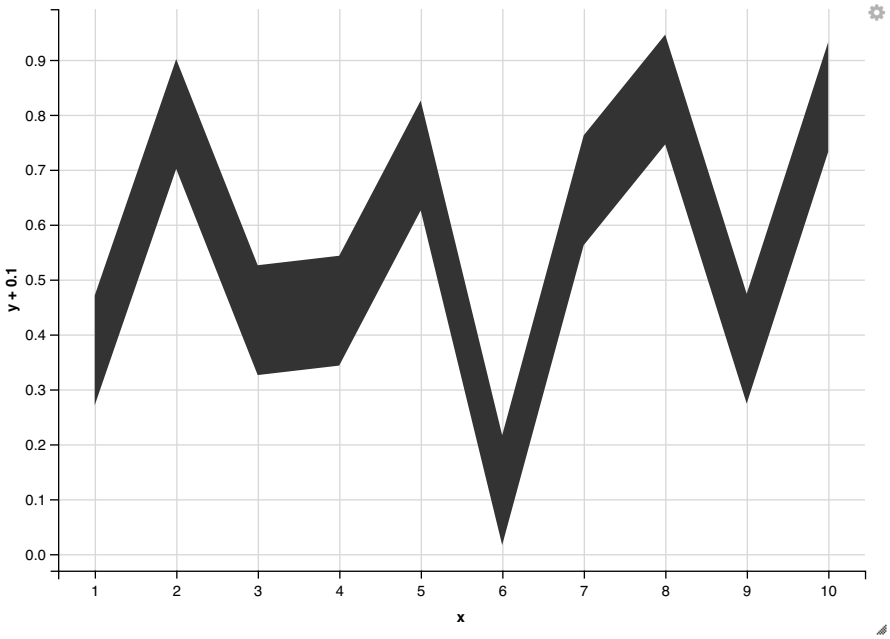


Layers

In ggvis there are two types of layers: simple and compound. Simple layers represent primitives (points, lines, rectangles), while compound layer combine data transformations with one or more simple layers.

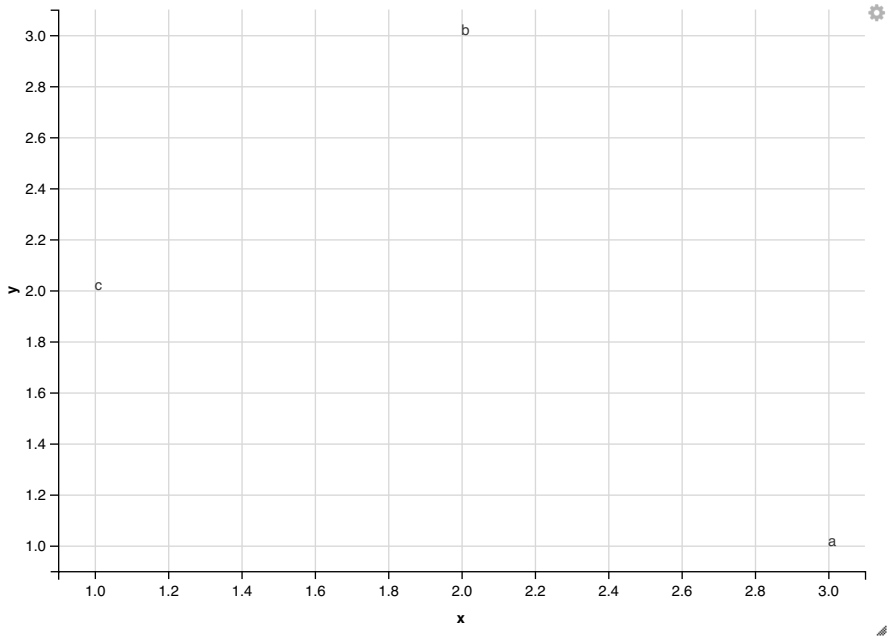
Simple Layers: Let's explore some examples of simple layers. We can create a ribbon in out plot to represent our data with the `layer_ribbon()` function.

```
# layering ribbons
df <- data.frame(x = 1:10, y = runif(10))
df %>% ggvis(~x, ~y + 0.1, y2 = ~y - 0.1) %>% layer_ribbons()
```



We can also add text labels to plots with the `layer_text()` function.

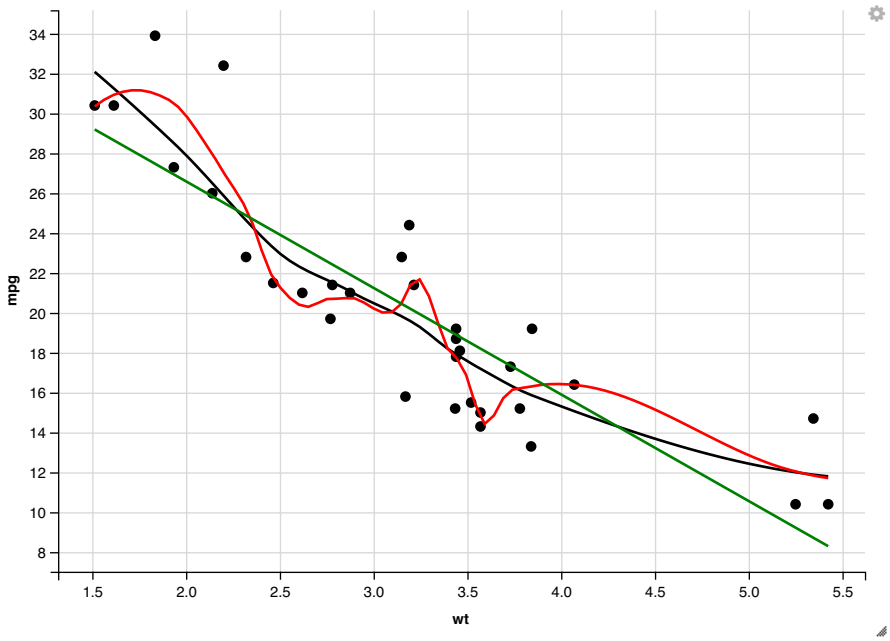
```
# layering text
df <- data.frame(x = 3:1, y = c(1, 3, 2), label = c("a", "b", "c"))
df %>% ggvis(~x, ~y, text := ~label) %>% layer_text()
```



Compound Layers: Next, let's explore some compound layers! To create more complex graphics, it is possible to add many different elements and layers to one plot. Going back to the `mtcars` data set, we create an even better plot by adding some smoothing lines to get a better idea of the relationship between weight and mileage of cars.

```
# adding multiple layers to plots
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  # plot the points
  layer_points() %>%
  # adding first smoothing line
  layer_smooths() %>%
  # second smoothing line
  layer_smooths(span = 0.3, stroke := "red") %>%
  # adding third prediction/regression line
  layer_model_predictions(model = "lm", stroke := "green")
```

```
## Guessing formula = mpg ~ wt
```

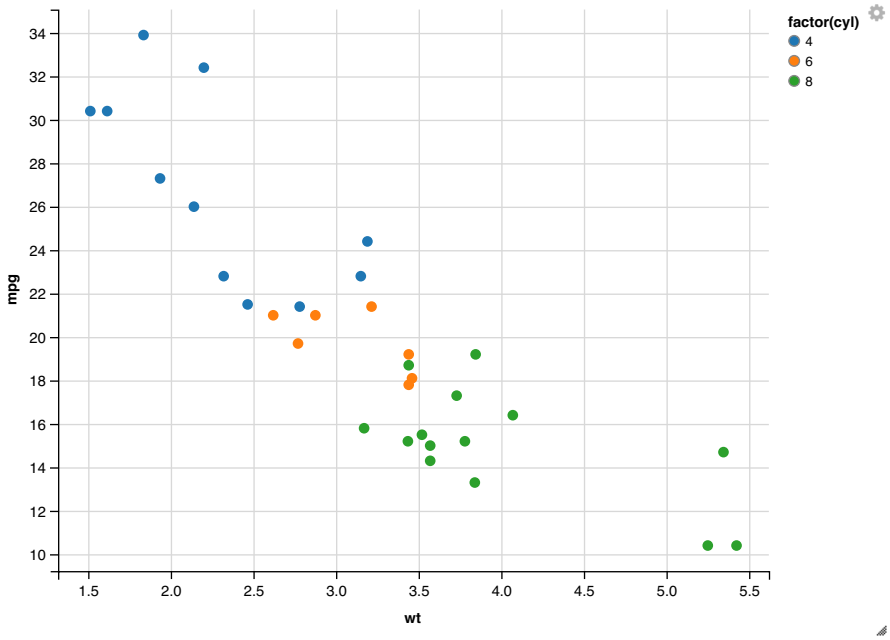


Grouping

In `ggivs`, grouping is another way of transforming data using the `group_by()` function to split up data in pieces given a specific variable. Also, `auto_split()` can be used to split up any categorical variable in the plot.

For example, we can create a scatter plot by coloring points by a variable according.

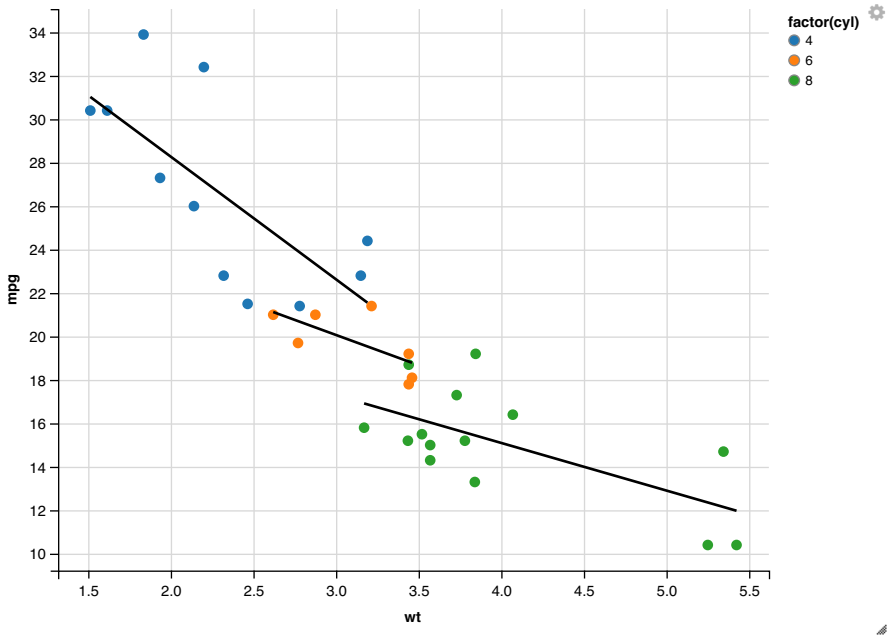
```
# coloring poiints by a variable
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_points(fill = ~factor(cyl))
```



We can also color points, and add another layer by adding a smoother for each group. We specify the grouping varaible with `group_by`.

```
# creating multiple layers by grouping and then adding a smoothing line for each group
mtcars %>%
  ggvis(~wt, ~mpg, fill = ~factor(cyl)) %>%
  layer_points() %>%
  group_by(cyl) %>%
  layer_model_predictions(model = "lm")
```

```
## Guessing formula = mpg ~ wt
```



Conclusion

Overall, `ggvis` is a powerful package that enables us to create beautiful interactive graphics. The package includes many features in order to endlessly customize plots and graphics. `ggvis` is a very powerful data visualization tool in R beacuse of its simplicity and power to create informative visualization. Because of its interactive element, `ggvis` and Shiny are often used together to help data professionals create data visualizations in apps and web content. Thus, `ggvis` is a captivating package with various uses, so once you master `ggvis` you'll be able to easily create powerful graphics and applications.

References

https://ggvis.rstudio.com/ggvis-basics.html http://ggvis.rstudio.com/0.1/quick-examples.html https://ggvis.rstudio.com/interactivity.html https://ggvis.rstudio.com/cookbook.html https://ggvis.rstudio.com/ggplot2.html https://www.dezyre.com/data-science-in-r-programming-tutorial/ggvis https://www.r-bloggers.com/introducing-ggvis/ https://ggvis.rstudio.com/layers.html