Post02: ggvis

Yuechan Huang

December 1, 2017

Note: If you're reproducing this post in R Markdown, you need to add *runtime: shiny* after *output: html_document* in the header, otherwise the examples will have their interactive features disabled. You can also run the code in R script to see and use the interactive controls.

Introduction

According to https://blog.rstudio.com/2014/06/23/introducing-ggvis/, **ggvis**, a package for data visualization, is built on concepts from grammar of graphics like ggplot2, but it also combines reactive programming from shiny and data transformation pipelines from dplyr. If you want to know more about the differences between **ggvis** and ggplot2, you can check them out: https://ggvis.rstudio.com/ggplot2.html and http://jimhester.github.io/ggplot2ToGgvis/. The goal of **ggvis** is to make it easy to build interactive graphics for exploratory data anyalysis.

Data Preparation

```
# Loading Packages
library(dplyr)
```

```
##
## 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(ggvis)
```

In this post, I choose to use the *airquality* dataset which R already provided, and I would like to replace all NA's in this dataset with 0.

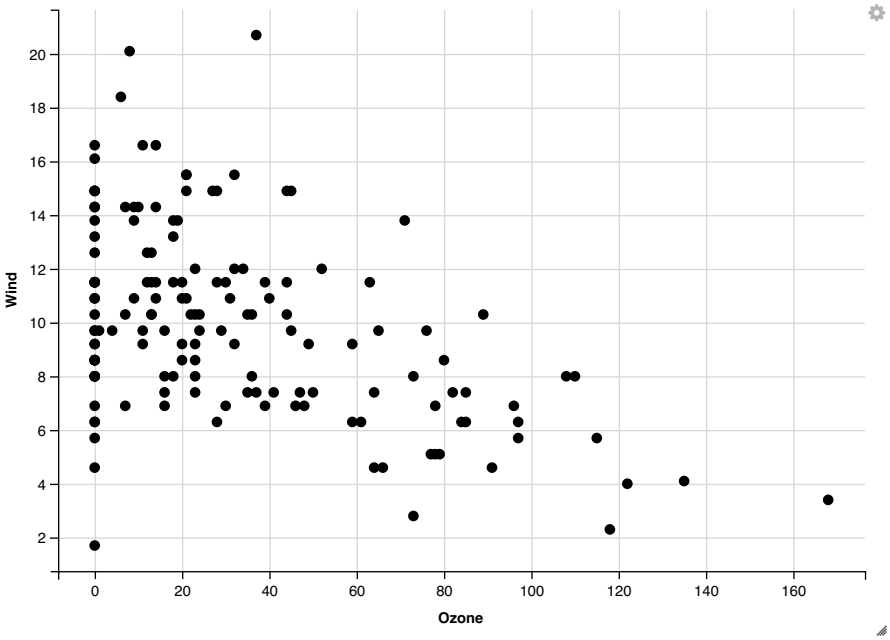
```
# replace NA's with 0
airquality[is.na(airquality)] <- 0
```

ggvis

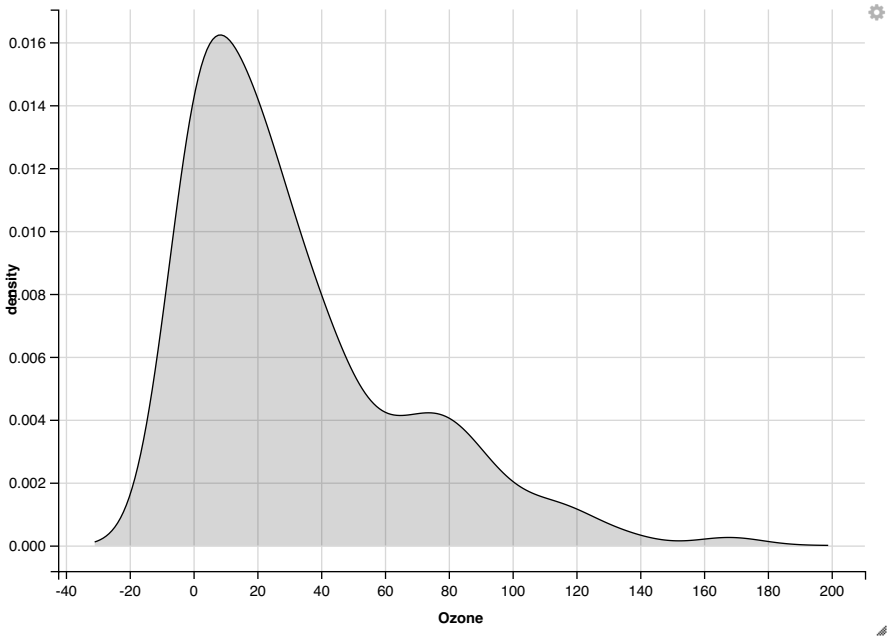
Basic Graphics

Similar to ggplot2, every ggvis graphic starts with a call to **ggvis()**. The first argument is the dataset that you want to plot, and the other arguments describe how to map vairables to visual properties. However, unlike ggplot2 which uses **+** to combine components, ggvis has a function interface that allows you to combine components using **%>%**. For example:

```
airquality %>%
  ggvis(x = ~Ozone, y = ~Wind) %>% # in ggplot2, it will be + instead of %>% here
  layer_points()
```



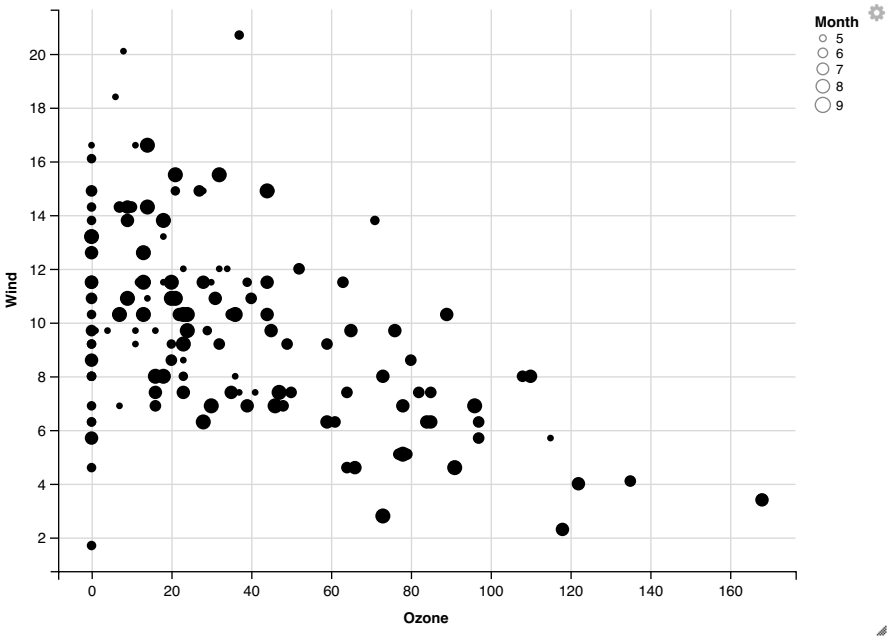

```
airquality %>%
  ggvis(x = ~Ozone) %>%
  layer_densities()
```



In fact, the sign **~** before the variable names is very important. We use **~** to indicate that we want to use the **Ozone** varaible in the dataset but not the value of **Ozone** variable (which doesn't exist).

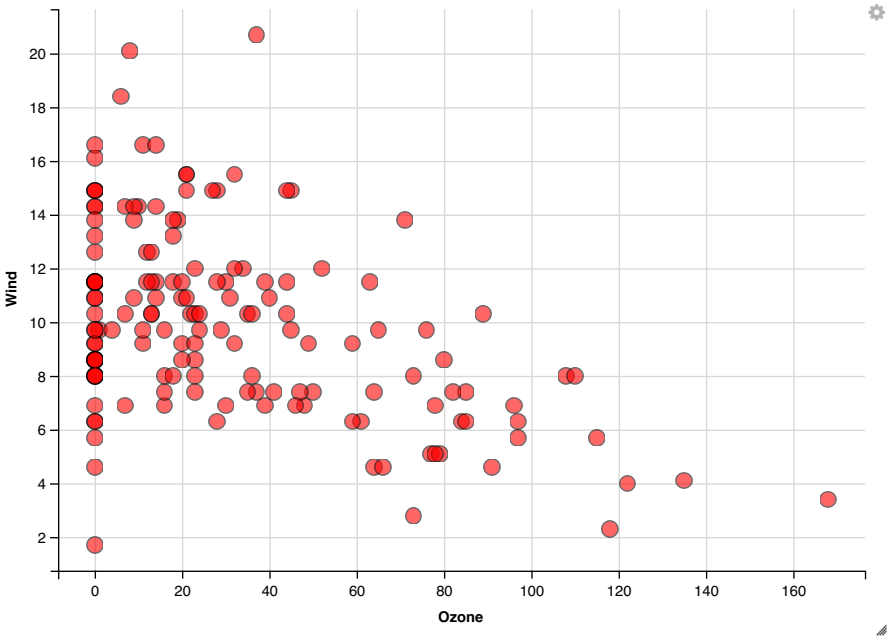
There are many visual properties in ggvis like *fill*, *stroke*, *size* and *shape*, and we can add more variables to the plot by mapping them to these properties. For example,

```
airquality %>%
  ggvis(x = ~Ozone, y = ~Wind, size = ~Month) %>%
  layer_points()
```



If you want to make the points to a fixed color, you need to use **:=** instead of **=**. The **:=** operator means to use a raw, unscaled value.

```
airquality %>%
  ggvis(x = ~Ozone, y = ~Wind, fill := "red",
        stroke := "black", opacity := .6, size := 100) %>%
  layer_points()
```

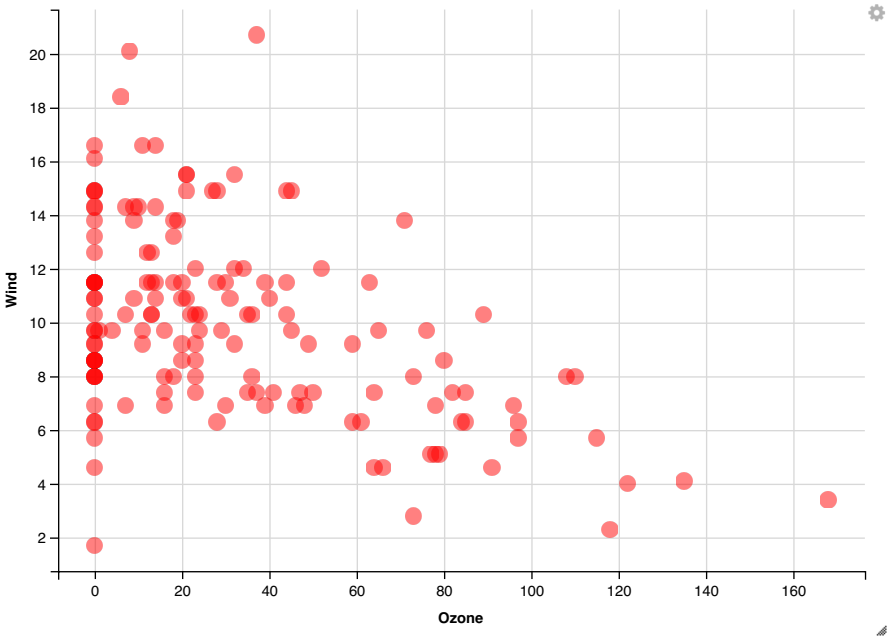


Interaction

In addition to mapping visual properties to variables or to specific values, we can also connect them to interactive tools. For example,

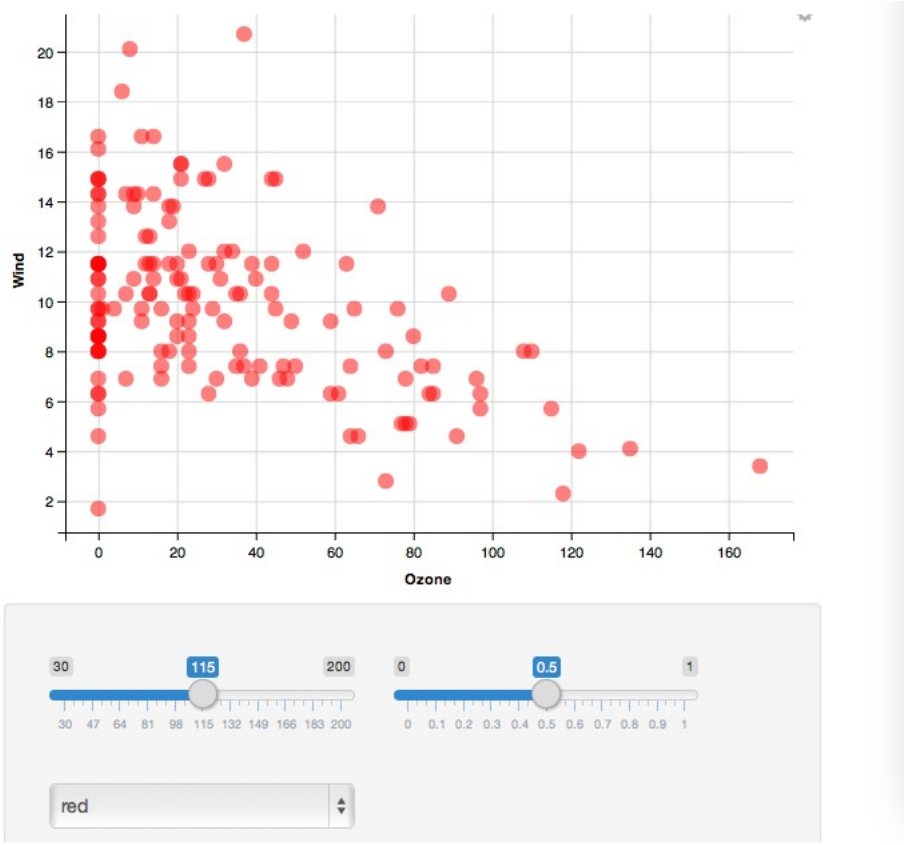
```
airquality %>%
  ggvis(x = ~Ozone, y = ~Wind,
        size := input_slider(30, 200),
        opacity := input_slider(0, 1),
        fill := input_select(c("red", "blue", "grey", "black"))) %>%
  layer_points()
```

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

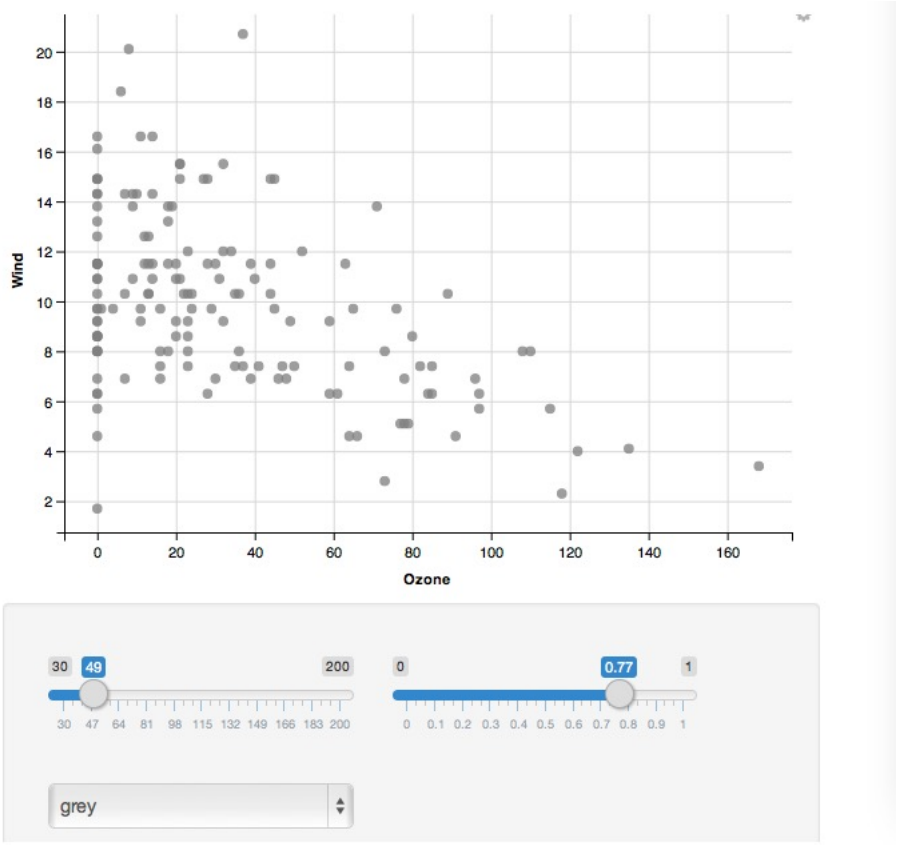


(Because this is the html version, the interactive features are disabled. The two pictures below are the outputs with interactive features.)

With size = 115, opacity = 0.5, and fill = red



With size = 49, opacity = 0.77, fill = grey



According to https://ggvis.rstudio.com/interactivity.html and https://cran.r-project.org/web/packages/ggvis/ggvis.pdf, there are a number of interactive controls in ggvis:

- *input_checkbox()*: a check-box
- *input_checkboxgroup()*: a group of check boxes
- *input_numeric()*: a spin box
- *input_radiobuttons()*: pick one from a set options
- *input_select()*: create a drop-down text box
- *input_text()*: arbitrary text input
- *input_slider()*: a double ended range slider

Layers

I have shown you two layer functions in the beginning of the post: layer_points() and layer_densities(). According to https://ggvis.rstudio.com/layers.html, there are many layers functions in ggvis, and they can be categorized into two types:

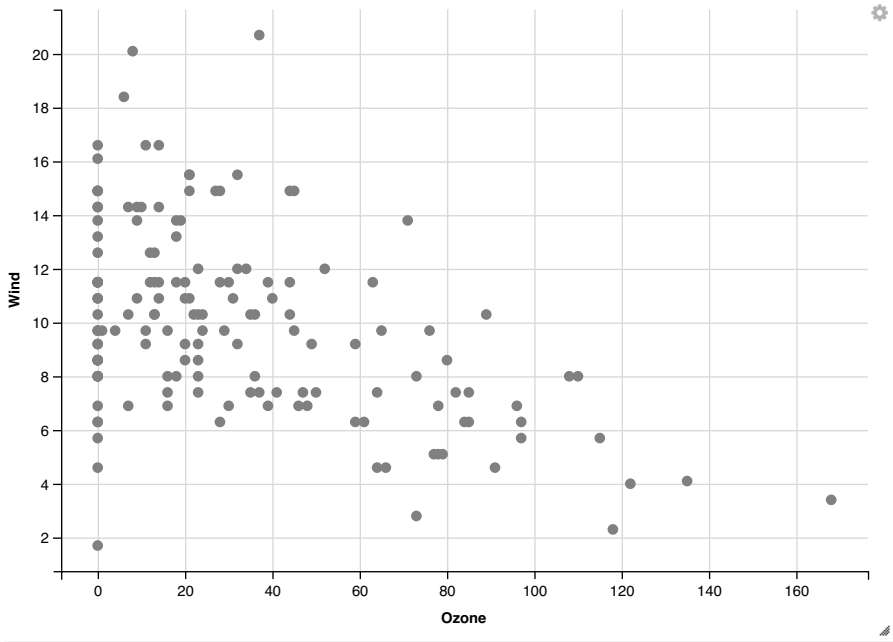
- simple, includes geometric primitives like points, lines, and rectangles.
- compound, combines data transformations with one or more simple layer functions.

Simple Layers

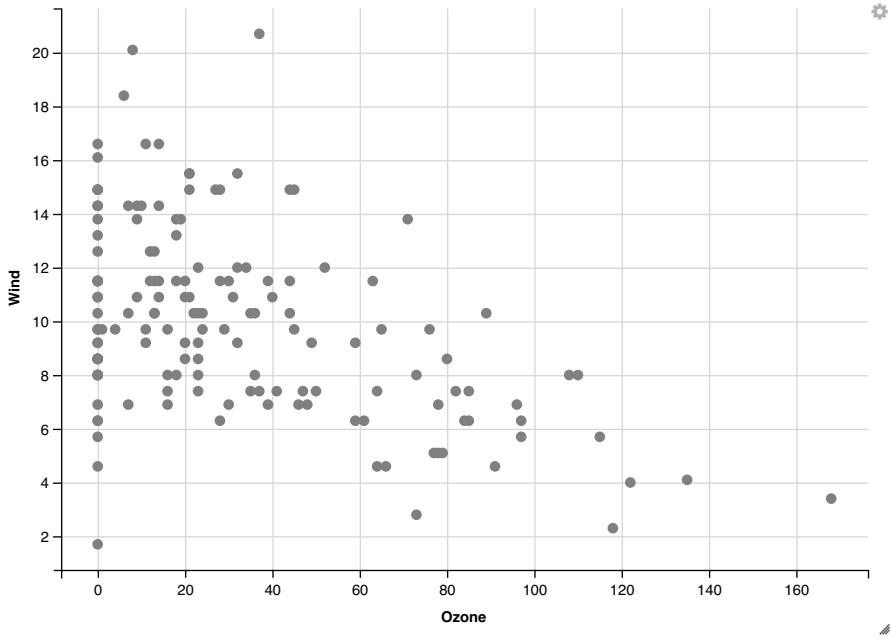
There are five simple layers:

1. layer_points(): points, with properties *x*, *y*, *shape*, *fill*, *stroke*, *strokeOpacity*, *fillOpacity*, and *opacity*.

```
airquality %>% ggvis(~Ozone, ~Wind, fill := "grey") %>% layer_points()
```

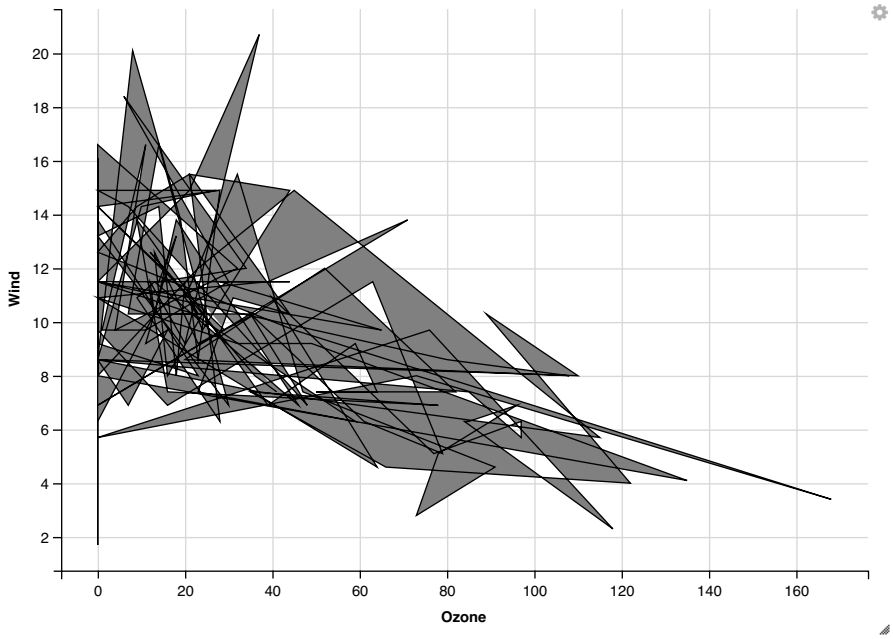



```
airquality %>% ggvis() %>% layer_points(~Ozone, ~Wind, fill := "grey")
```

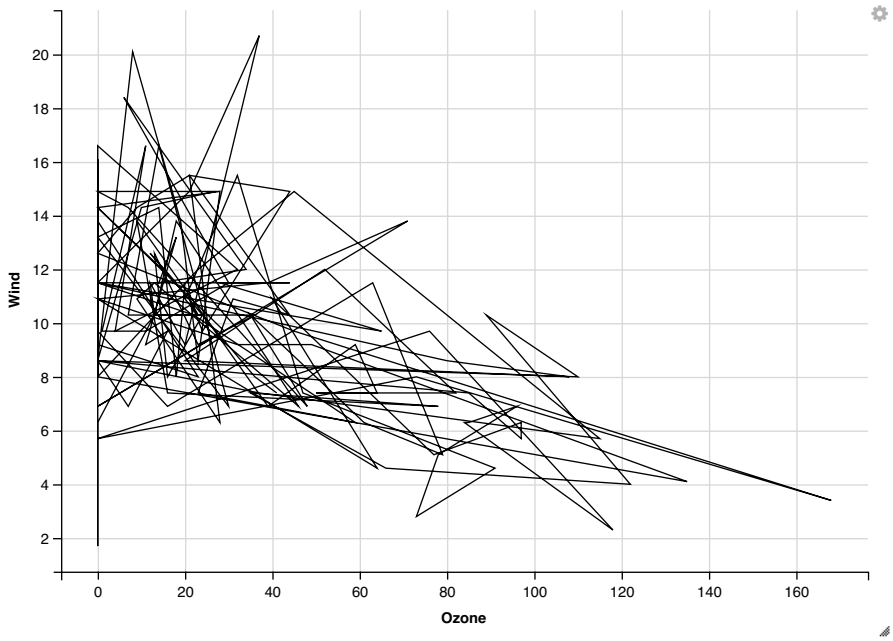


2. layer_paths(): paths and polygons

```
airquality %>% ggvis(~Ozone, ~Wind, fill := "grey") %>% layer_paths()
```

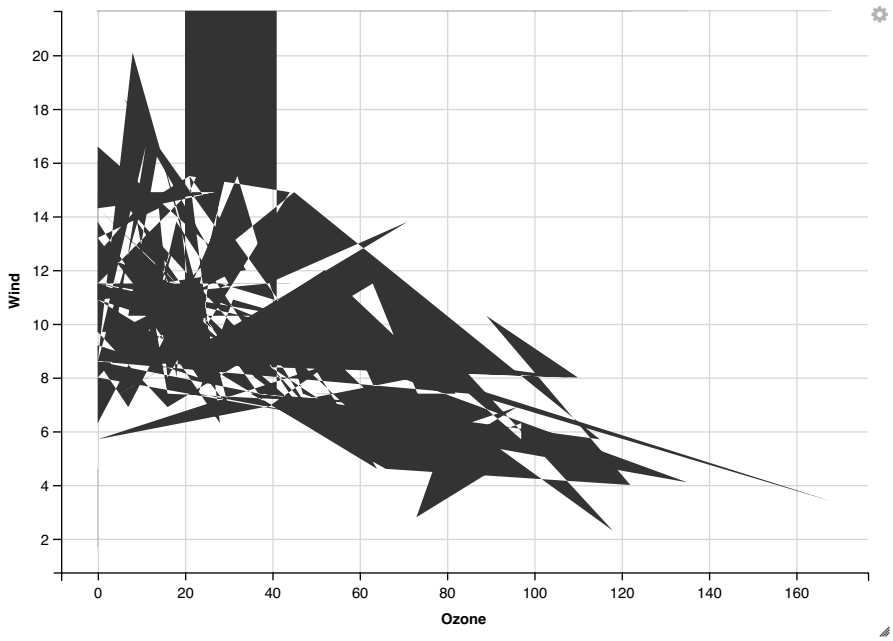


```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_paths()
```

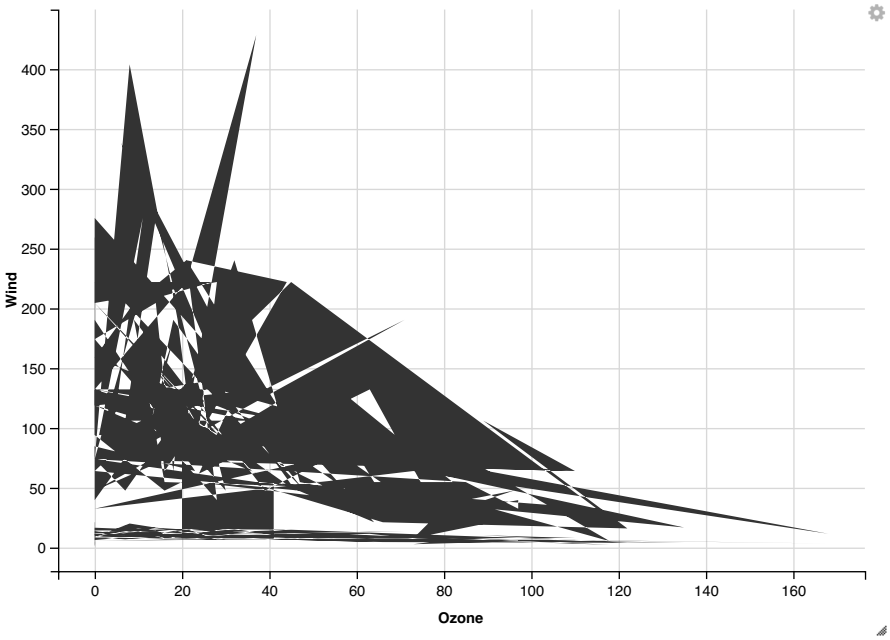


3. layer_ribbons(): filled areas, use properties *y* and *y2* to control the extent of area

```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_ribbons()
```

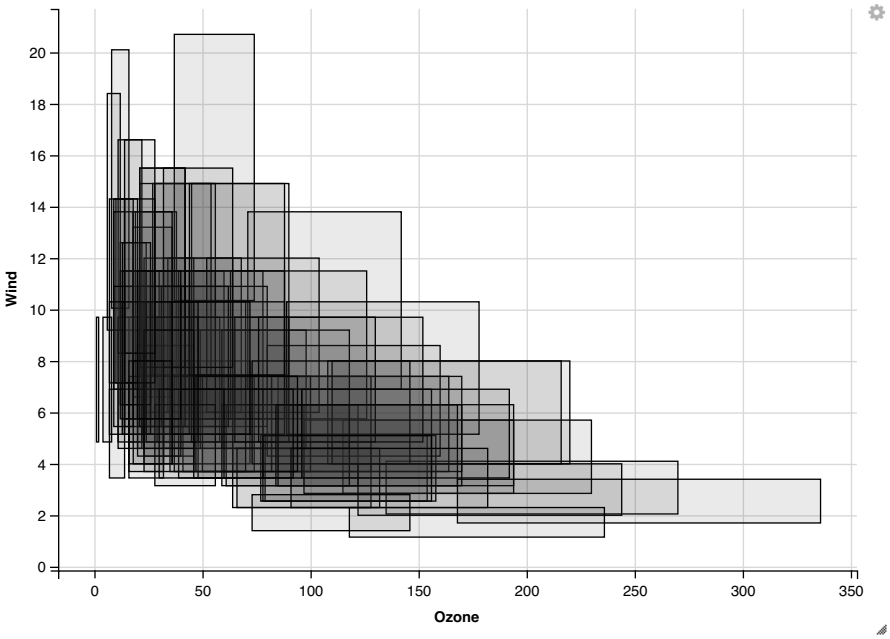



```
airquality %>% ggvis(~Ozone, ~Wind, y2 = ~Wind ^ 2) %>% layer_ribbons()
```



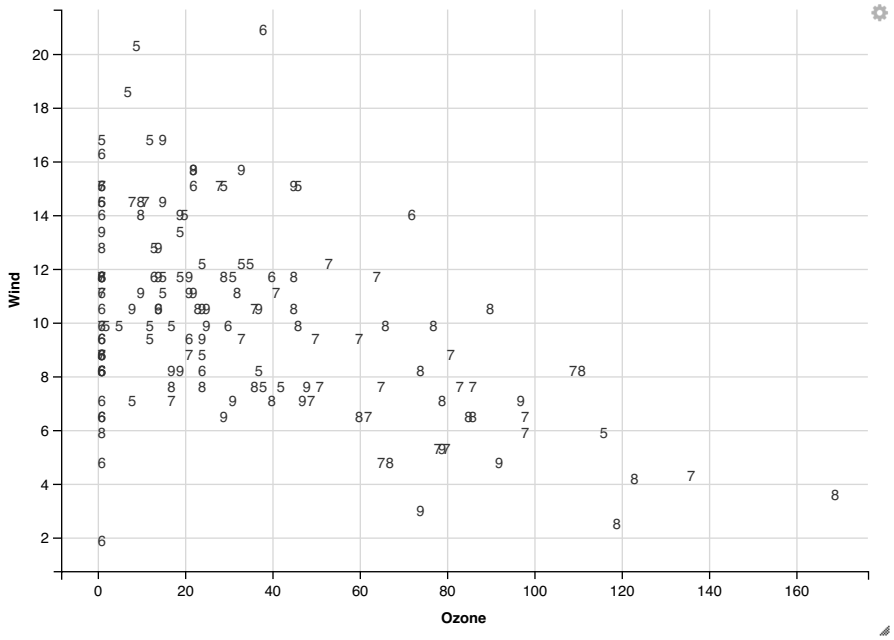
4. layer_rects(): rectangles, use properties *x*, *x2*, *y*, *y2* to control the location and size of the rectangle.

```
airquality %>% ggvis(~Ozone, ~Wind, x2 = ~Ozone * 2, y2 = ~Wind / 2,
                     fillOpacity := 0.1) %>% layer_rects()
```



5. layer_text(): text, with properties *text*, *font*, *angle*, *fontStyle*, *fontSize*, etc.

```
airquality %>% ggvis(~Ozone, ~Wind, text := ~Month) %>% layer_text()
```

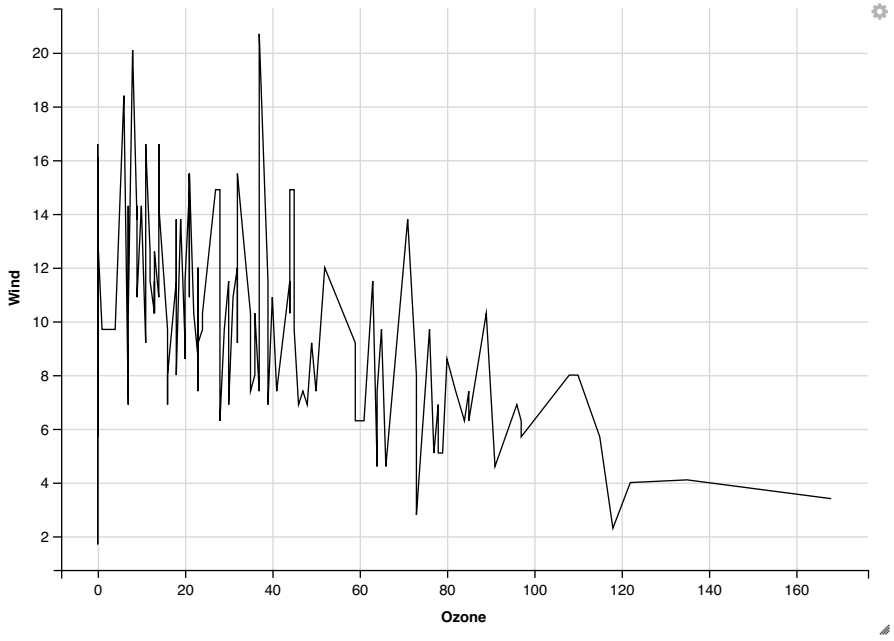


Compound Layers

There are four most common compound layers:

1. layer_lines(), automatically put x variable in order

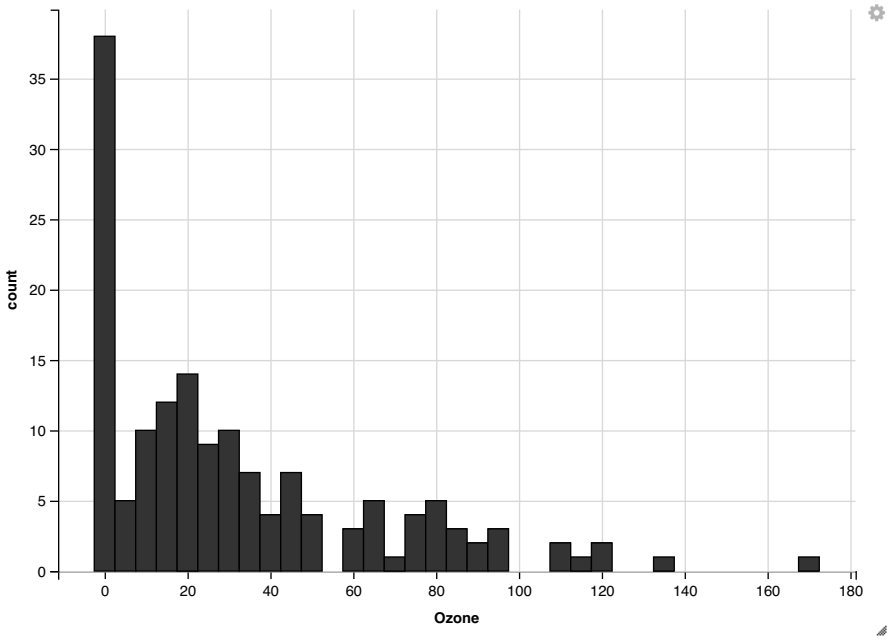
```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_lines()
```



2. layer_histograms()

```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_histograms()
```

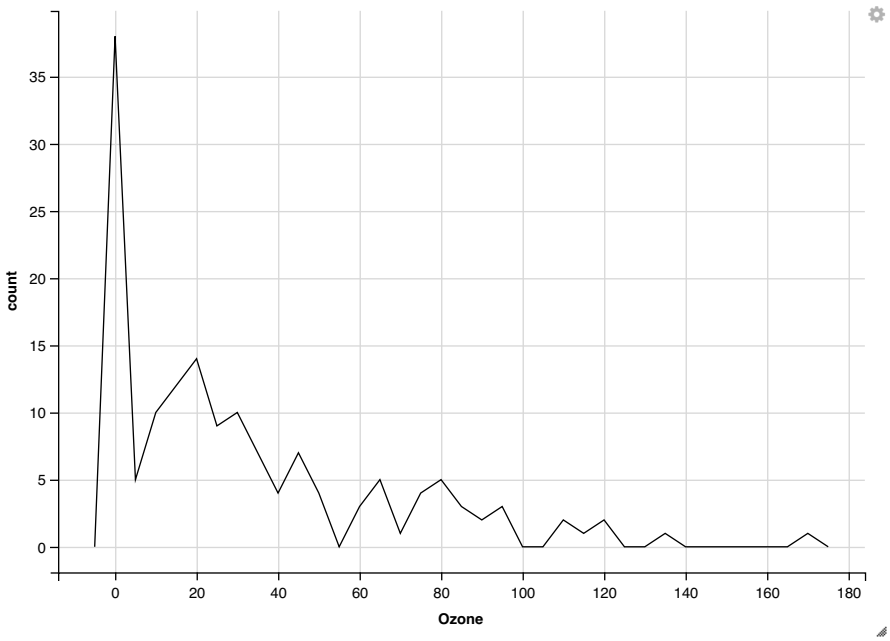
```
## Guessing width = 5 # range / 34
```



3. layer_freqpolys()

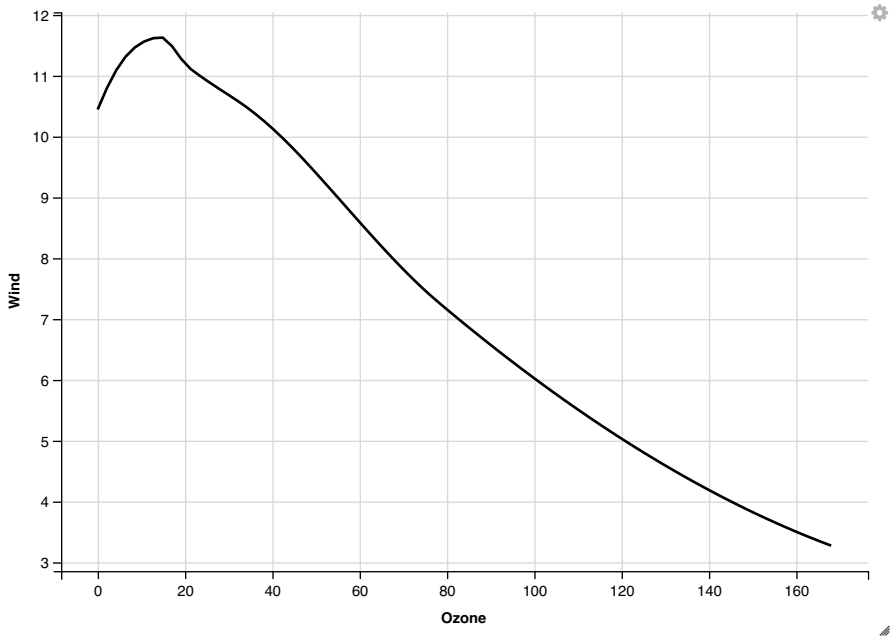
```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_freqpolys()
```

```
## Guessing width = 5 # range / 34
```



4. layer_smooths(), fits a smooth model to the data, and displays predications with a line.

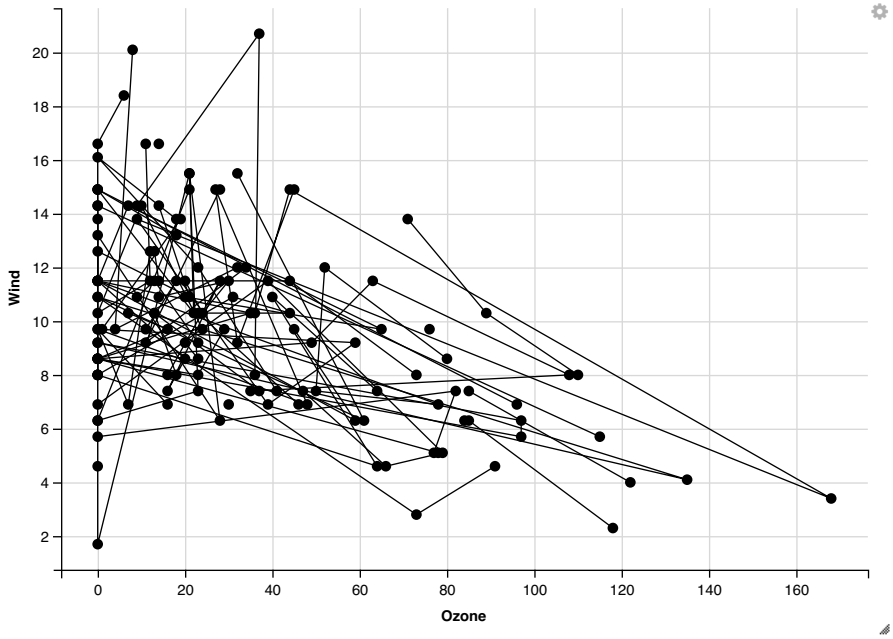
```
airquality %>% ggvis(~Ozone, ~Wind) %>% layer_smooths()
```



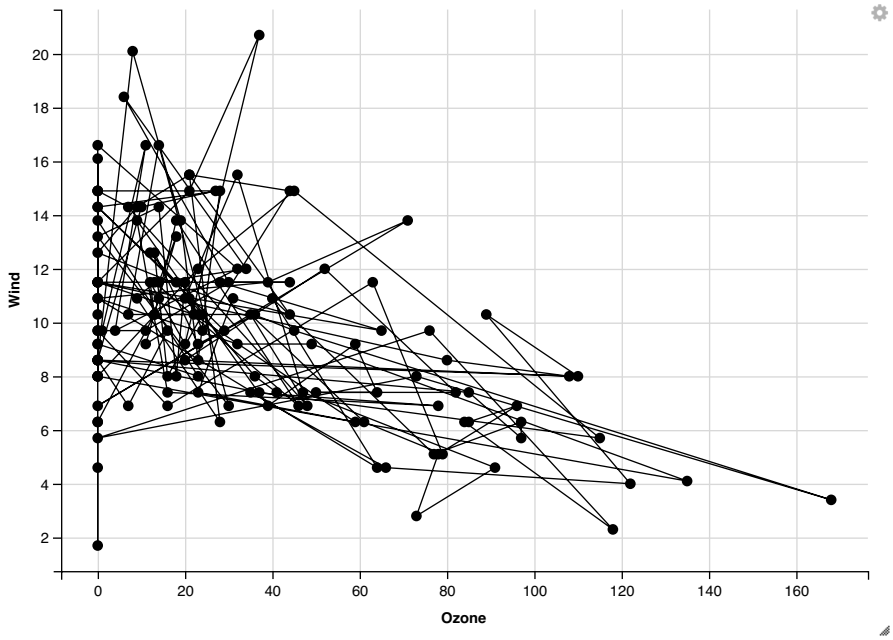
Grouping

Like ggplot2, we can also split data into pieces given a specificed variable using group_by() in ggvis.

```
airquality %>% ggvis(~Ozone, ~Wind) %>%
  layer_points() %>%
  group_by(Temp) %>%
  layer_paths()
```




```
airquality %>% ggvis(~Ozone, ~Wind) %>%
  layer_points() %>%
  auto_group() %>%
  layer_paths()
```



Conclusion

In this post, I perform some common tasks when working with ggvis: basic graphics, working with interactive tools, and different layer functions. In fact, you can also embed a ggvis in a Shiny app as it is in this example, https://github.com/ucb-stat133/stat133-fall-2017/blob/master/apps/conditional-panels/app.R. In addition, if you want to learn more about ggvis, you can also check this out https://www.cheatography.com/shanly3011/cheat-sheets/data-visualization-in-r-ggvis-continued/pdf_bw/.

Reference

- https://blog.rstudio.com/2014/06/23/introducing-ggvis/
- https://ggvis.rstudio.com/ggplot2.html
- http://jimhester.github.io/ggplot2ToGgvis/
- https://cran.r-project.org/web/packages/ggvis/ggvis.pdf
- https://ggvis.rstudio.com/interactivity.html
- https://ggvis.rstudio.com/layers.html
- https://www.cheatography.com/shanly3011/cheat-sheets/data-visualization-in-r-ggvis-continued/pdf_bw/
- https://github.com/ucb-stat133/stat133-fall-2017/blob/master/apps/conditional-panels/app.R