Post02: Interactive Data Visualization with ggvis

Stat 133, Fall 2017

Abigail Stiris

Interactive Data Visualization with `ggvis`

Introduction

You've probably (hopefully) visualized data in R with base functions and ggplot2, but have you visualized data so that the user can change up the output with just a few clicks or hovers? If you haven't yet, now you can! The package `ggvis` is an awesome way to add interactivity into your data visualizations. Plus, it combines the grammar of graphics from ggplot2 with the reactive programming principles from `shiny` with the grammar from `dplyr` to provide us data scientists with a really cool and useful tool.

Motivation

Once I saw ggvis in lecture, I was immediately interested in the ways it seemed to combine all the knowledge we've already gained from the course into something even cooler than I expected to learn. I'll admit, interactive features have always been especially awesome to me. I love when the user has freedom to experiment with any graphic visualization, even dragging the plot to become larger and easier to interpret. I thought that using the pipe operator from dplyr and some of the features reminiscent of shiny made ggvis both intuitive and slightly intimidating. Don't worry, though! It's not as complicated as it seems.

Background

The package ggvis was created by Winston Chang and Hadley Wickham. Although the package is currently dormant, you can see the code on github. It isn't as popular as ggplot2 and other packages for data visualization, but it has some features that neither base functions nor ggplot2 provides.

Some of the most interesting components of ggvis are the interactive functions, where you're not confined to one parameter or input. The user viewing your graph can have an effect on the final result, which we've only seen in shiny apps before.

In this post, I'll be focusing on a few very useful and super cool functions having to do with interactivity.

Limitations

Before we move on to the interactive functions, it's important to note that these plots won't display in a static document. You'll get a warning like this:

```
"Warning: Can't output dynamic/interactive ggvis plots
in a knitr document.

Generating a static (non-dynamic, non-interactive)
version of the plot."
```

To try these features out for yourself, you'll just have to copy the code into your active R session so that you can "run" the functions, just like a shiny app. Thankfully, it only takes an extra line or two to add interactivity to your plots.

Also, the interactive plots were not built for `.md` files, so the execution will halt if you try to incorporate them into your knitted document. To work around this issue, you can insert screenshots of the static plots into your `.md` file and set `eval` equal to "FALSE" in your code chunks so that they're not executed.

With these limitations in mind, let's move forward onto new territories. And definitely don't just take my word for it! Try them out for yourself.

Some Features

Let's check out what you can add to your visualizations in ggvis to allow the user to manipulate the way they appear. In this post, I'll be discussing the various "input_" functions. Once implemented, they might appear to be similar to the input options that we've seen in shiny apps, but you don't need to worry about the UI and the server. That's all done for you when you use ggvis, but that's also why interactive graphs can't just be static in an `html` file, just like shiny apps can't be knitted, only run.

Data Preparation

We'll start by loading some data so that we can test out these new functions. I decided to explore the 'camping tents' data from our Stat 133 course repository. It contains information such as the name, brand, price, dimensions, best use, seasonality, and capacity of 90 different tents.

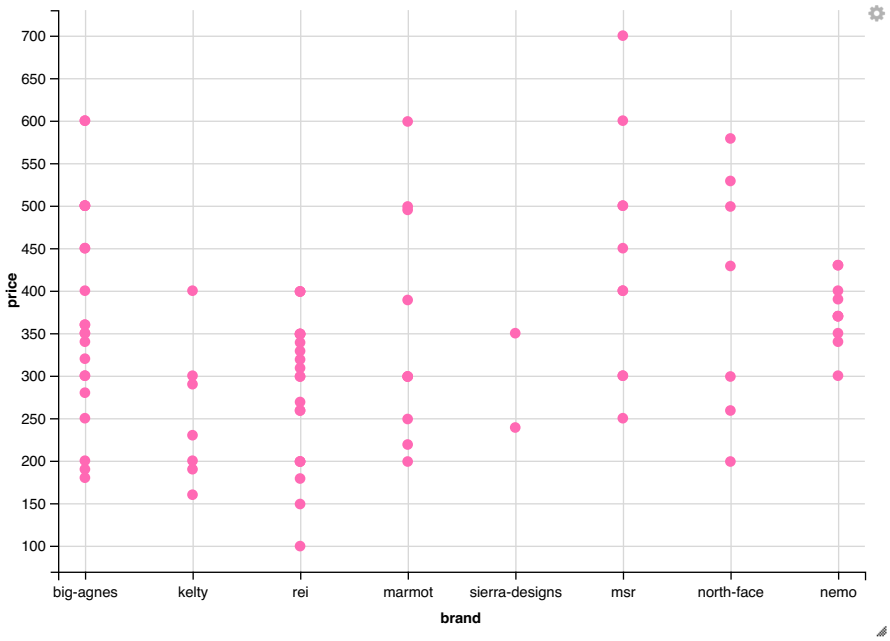
```
# reading the 'tents' data
tents <- read.csv('../data/camping-tents.csv',
                  stringsAsFactors = FALSE)
str(tents)
```

```
## 'data.frame':    90 obs. of  8 variables:
##  $ name    : chr  "fly-creek-ul2" "fly-creek-ul3" "salida-2" "jack-rabbit-sl3" ...
##  $ brand   : chr  "big-agnes" "big-agnes" "kelty" "big-agnes" ...
##  $ price   : num  350 450 160 360 149 ...
##  $ weight  : num  960 1450 1700 2160 2210 1530 2690 1840 3200 850 ...
##  $ height  : int  96 107 102 107 107 107 117 114 122 97 ...
##  $ bestuse : chr  "Backpacking" "Backpacking" "Backpacking" "Backpacking" ...
##  $ seasons : chr  "3-season" "3-season" "3-season" "3-season" ...
##  $ capacity: chr  "2-person" "3-person" "2-person" "3-person" ...
```

Now that we have our data and are a bit more familiar with it, we can move on to building a basic `ggvis` plot. You've probably done this step before.

```
# creating a basic ggvis plot
library(ggvis) # loading ggvis

tents %>% # data
  ggvis(~brand, ~price) %>% # visualization
  layer_points(fill := "hotpink") # scatterplot
```



Let's move on to the interactivity that we've all been waiting for.

Disclaimer: this package has not been fully "thought through" yet, as Hadley Wickham himelf admitted. That's why these functions work best with scatterplots specifically.

`input_select`

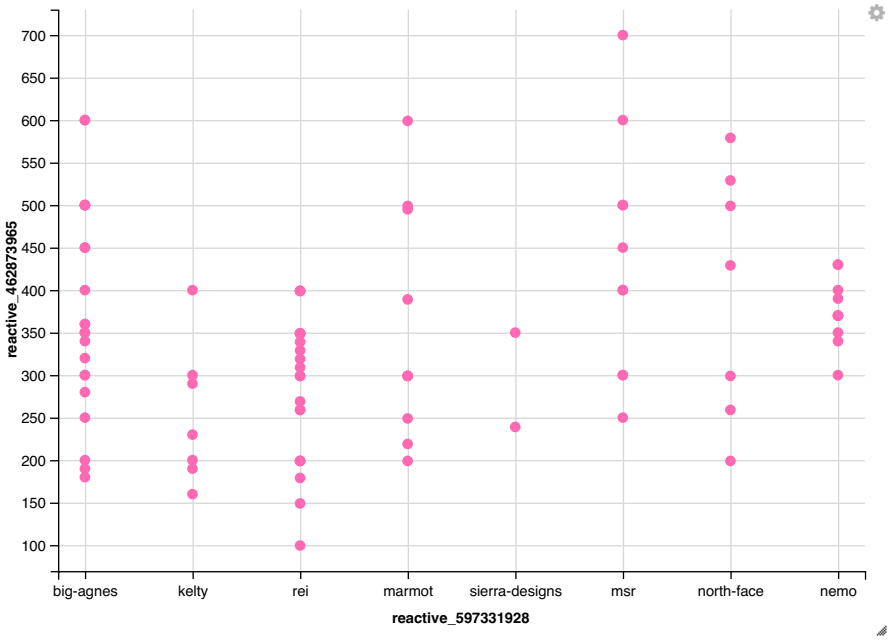
The first function is called `input_select`. It's used for creating a drop-down list from which you can choose a value. In this case, I've added it to the parameters of the `ggvis` function so that the user can decide which two values to choose when creating the visualization.

Passing `as.name` into the `map` paramater ensures that the output of the selected input is converted to a name in the data frame so that it's interpreted correctly by ggvis.

By default, I've set the `selected` parameter to "brand" and "price" so that they display first. Just some simple additions like these make the graph more exciting and interesting.

```
# input_select function
tents %>%
  ggvis(input_select(names(tents), map = as.name,
                     selected = 'brand',
                     label = "Variable 1"), # first drop-down
        input_select(names(tents), map = as.name,
                     selected = 'price',
                     label = "Variable 2")) %>% # second drop-down
  layer_points(fill := "hotpink") # scatterplot
```

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

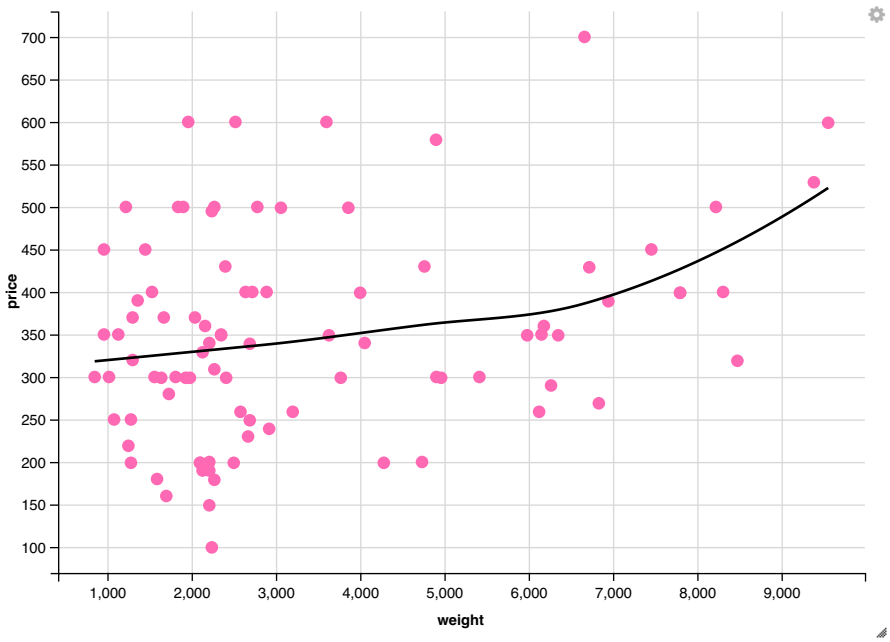


`input_slider`

Next, we'll look at `input_slider`. Again, you can see the inspiration from shiny here. Here, we can use it to add some more interactivity into the sizing of the points, and the span of the loess smoother.

```
# input_slider function
tents %>%
  ggvis(~weight, ~price) %>%
  layer_points(fill := "hotpink",
               # slider for size
               size := input_slider(10, 100, value = 70,
                                    label = "Size")) %>%
  layer_smooths(# slider for span
    span = input_slider(0.5, 1, value = 1,
                                    label = "Span"))
```

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



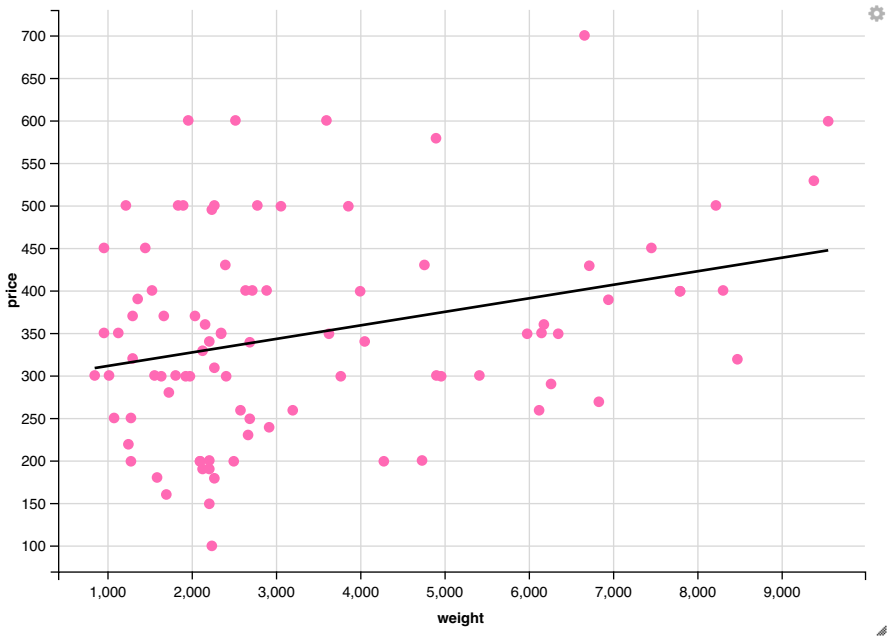
`input_radiobuttons`

This function is also reminiscent of a shiny widget. It's used to create different buttons that the user can select. In this case, I'll use it to allow for different options when it comes to adding a regression or lowess line.

```
# input_radiobuttons function
tents %>% ggvis(~weight, ~price) %>%
  layer_points(fill := "hotpink") %>%
  layer_model_predictions(# radio buttons for model type
    model = input_radiobuttons(
    choices = c("Regression" = "lm", "Loess" = "loess"),
    selected = "lm",
    label = "Model type"))
```

```
## Guessing formula = price ~ weight
```

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



input_checkbox

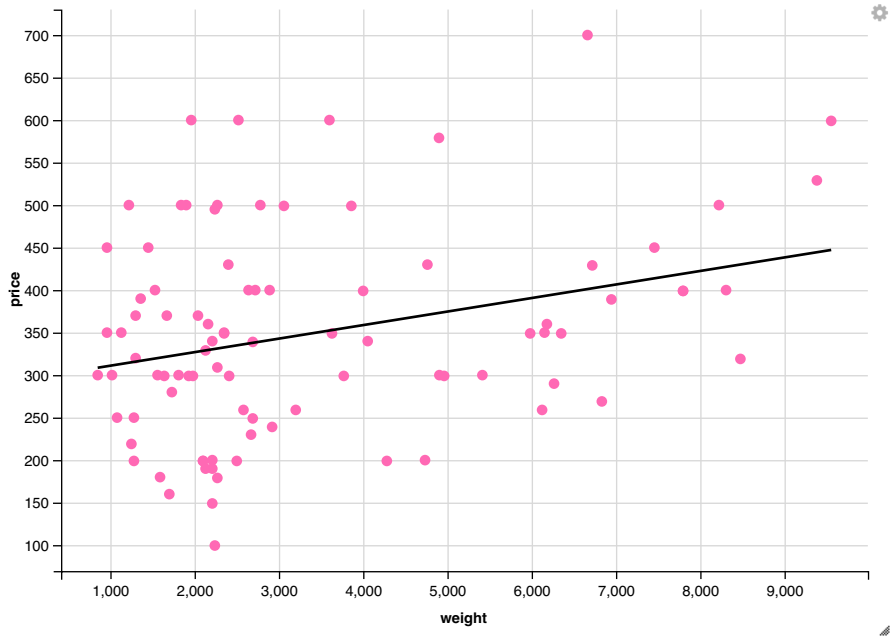
The `input_checkbox` function gives the user the choice of turning on a certain feature. For this plot, I'll incorporate it so that it can be used to add a loess line, while setting the default to a regression line.

The "map" parameter is very useful for this: you can create a function within it. Here, if the box is checked and "loess" is selected, the model will switch to "loess" rather than "lm".

```
# input_checkbox function
tents %>% ggvis(~weight, ~price) %>%
  layer_points(fill := "hotpink") %>%
  layer_model_predictions(# checkbox for model type
    model = input_checkbox(label = "loess",
    map = function(x) if(x) "loess" else "lm"))
```

```
## Guessing formula = price ~ weight
```

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



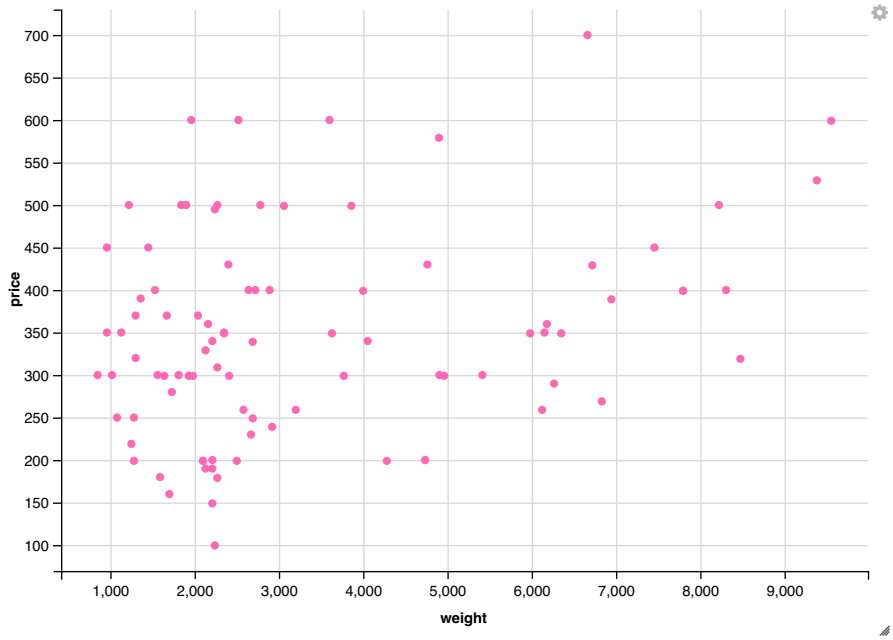
The next two functions are probably my favorite. The user can type their own input and immediately change the output! There are two varieties: numeric and text. Check them out for yourself.

`input_numeric`

Instead of manually inputting a certain number, you can use this function to let the user do the work for you! In my plot, I'm manipulating the size parameter with `input_numeric`, but it would definitely work for other variables that take numeric inputs as well.

```
# input_numeric function
tents %>%
  ggvis(~weight, ~price) %>%
  layer_points(fill := "hotpink",
               # numeric text box
               size := input_numeric(value = 30,
                                     label = "Point size"))
```

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



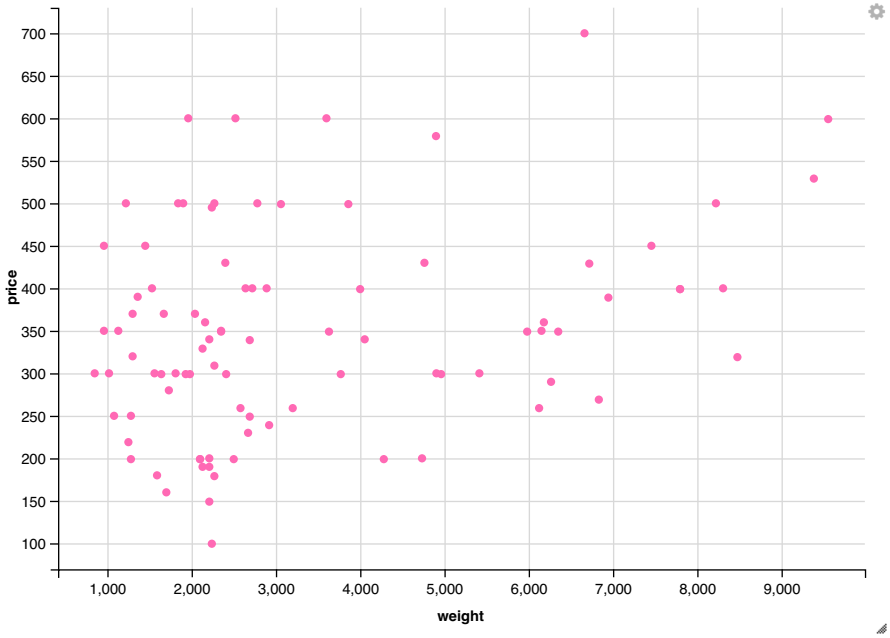
The user is totally free to input *any* number and the points will change to that size!

`input_text`

Just like the `input_numeric` function, `input_text` allows the user to decide what string to pass through your function as an input. For this plot, type any color you want into the text box and the color of the points will change!

```
# input_text function
tents %>%
  ggvis(~weight, ~price) %>%
  layer_points(# text box
    fill := input_text(value = "hotpink",
                                  label = "Point color"),
                size := 30)
```

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



Discussion

Hopefully, this post gave you a better understanding of the world of ggvis. It definitely has a lot to offer beyond the packages we've been using before, and gives you more freedom as a data scientist to create compelling and fun visualizations. This has been only a part of what you can do. Even with these six functions alone, there are endless ways to add exciting dimensions to your graphs and let the user take part in your work. There are definitely some downsides to using it since it's not the most comprehensively developed package and doesn't support as much flexibility as the more popular ones, but I think the code is pretty intuitive and can be very useful.

Conclusions

Even if you have a good grasp on the best way to visualize your data and craft a clear story, your freedom to experiment doesn't stop there. I think that incorporating additional features which make the user experience more exciting will definitely pay off in the end. With ggvis, that doesn't have to be time-consuming at all, and I'm sure your audience will appreciate the added interaction.

References

1. https://www.r-bloggers.com/how-to-create-interactive-data-visualizations-with-ggvis/
2. https://cran.r-project.org/web/packages/ggvis/index.html
3. https://cran.r-project.org/web/packages/ggvis/README.html
4. https://www.rdocumentation.org/packages/ggvis/versions/0.4.3
5. https://www.datacamp.com/community/tutorials/make-histogram-ggvis-r
6. https://github.com/rstudio/ggvis
7. https://www.dezyre.com/data-science-in-r-programming-tutorial/ggvis
8. https://www.youtube.com/watch?v=rf55oB6xX3w
9. https://groups.google.com/forum/#!topic/ggvis/F5C4HQjtgPY