Interactive Plotting

By Zhen Hui Trinh, Ben Johnson and Eileen Burns

A picture is worth a thousand words. Does an idiom exist to better describe what advanced data visualization methods aim to accomplish? As data scientists, our purpose for using graphics is to reduce massive amounts of information into bite-sized deliverables to be easily consumed by our intended audience. Depending on the audience and the type of deliverable—be it a lengthy report, a quick-and-dirty email or perhaps an elaborate slide deck—the sort of visuals we use will inevitably vary. Invariable is the need to convey as complete an analysis as possible without overcomplicating the visuals. Producing a too-busy plot is about as helpful to your reader as donning a track jacket when confronted with Snowmageddon.

As mediums for publishing the results of our analysis move online, we gain a number of advantages that paper reports cannot provide. The ability to incorporate interactive plots within our output is a great convenience, encoding additional information into graphics by utilizing mouse clicks, hovers, highlights and more. That's why we call them "interactive plots," but the interactivity does not stop there. Not only can a reader interact with a plot, but that plot may interact with other visuals. You can design a report so that clicking on one plot could trigger a chain of events that alters the entire display of the document. While paper formats may force us to split plots into several facets to avoid overcrowding, interactive plots enable us to communicate an equivalent amount of information in a much more compact space. Acknowledging the irony of touting the advantages of interactive plots within a static publication, we provide a link in the next section to a GitHub repository that includes code for producing said interactive plots.

Several interactive plotting packages have been developed far past infancy and into what we feel comfortable calling adolescence. Others also feel that there remains functionality to be desired in the available packages and have decided to delay adoption of interactive plots until packages have further matured. This is exactly how our team first felt when we considered migrating from the standard, static visuals. As a precursor to implementing interactive plots in our online web applications, we've compared the capabilities of some of the more popular interactive plotting packages. We build our web applications in Shiny, so we've focused on graphics packages in R, though we note there is similar functionality available in Python. Although the existing packages may not yet have everything you want, they still bring a lot to the table.

During the search, our team identified seven packages for consideration, building off the work of a former teammate, Mandy Zhuang, in 2017. (Thanks Mandy!) Among these, plotly seemed the most popular option within the R and Python communities. However, RStudio's ggvis package seemed a strong contender to our team, due to its similarity to ggplot. And, of course, we were compelled to investigate googleVis with its association to the tech giant. The list also included rCharts, rbokeh, highcharter and billboarder, with friends and colleagues in similar fields highly recommending the latter two. There are several packages not included in our investigation that provide other types of useful interactive graphics, such as maps, time series and tables.

Rather than provide a comparison of all of the packages across each relevant dimension (which would take more than the space available for an article), we're sharing a detailed comparison of two packages: plotly and billboarder. We built these packages into a Shiny app that we used to illustrate the differences below. We've also shared the Shiny app via GitHub and will continue to develop it as we investigate further: *https://github.com/milliman/SOA_PAF_Section_Newsletter_Plotly_vs_Billboarder.*

PLOTLY VERSUS BILLBOARDER

Zoom

A zoom-by-dragging property is enabled for both plotly and billboarder. Plotly allows for point selection, while billboarder allows for range selection. In other words, with plotly, the range of the y-axis can be changed after zooming, but with billboarder, the range of y-axis is fixed. As shown in Figures 1 to 4, we zoomed into the dots representing setosa in the Iris data set with both plotly and billboarder.

Figure 1
Before Zooming in Plotly

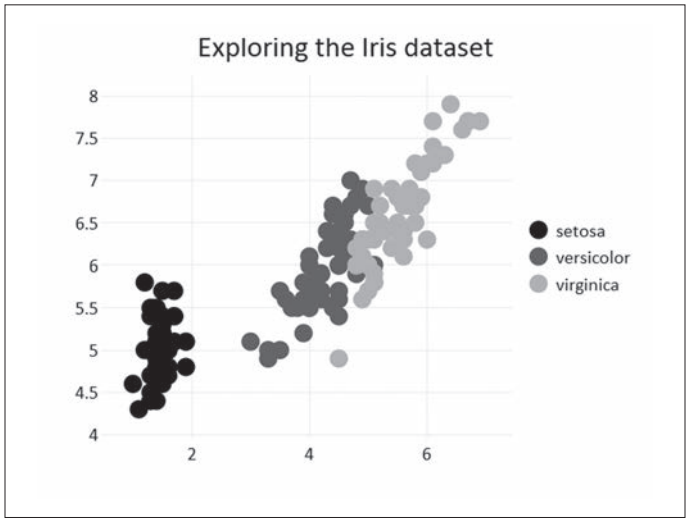


Figure 2
After Zooming in Plotly

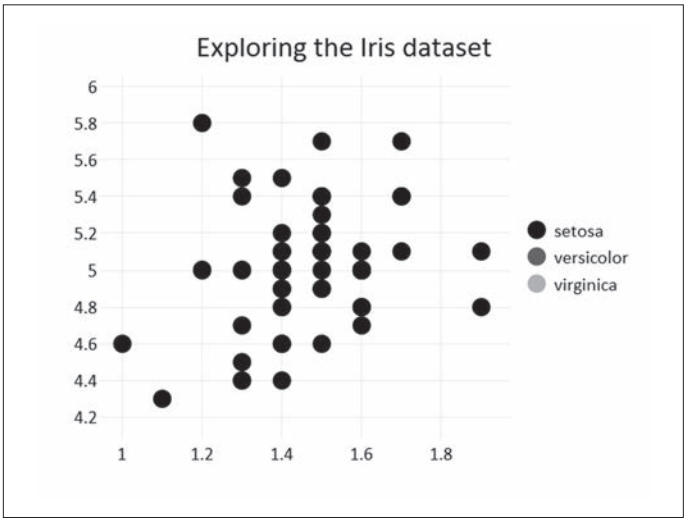


Figure 3
Before Zooming in Billboarder

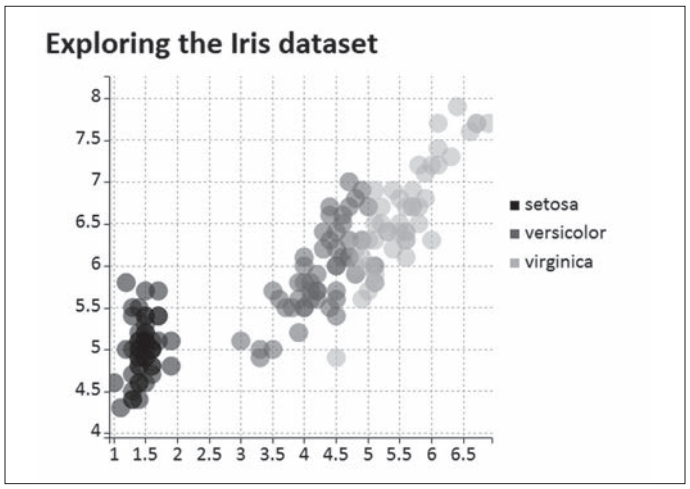


Figure 4
After Zooming in Billboarder

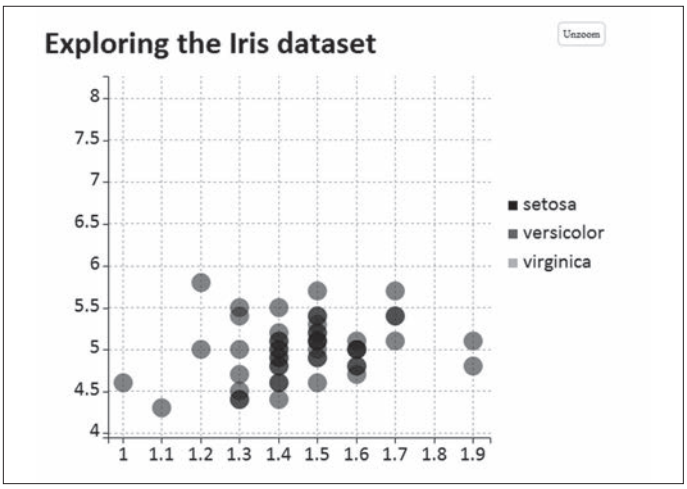


Figure 5
Default Hover Information in Plotly

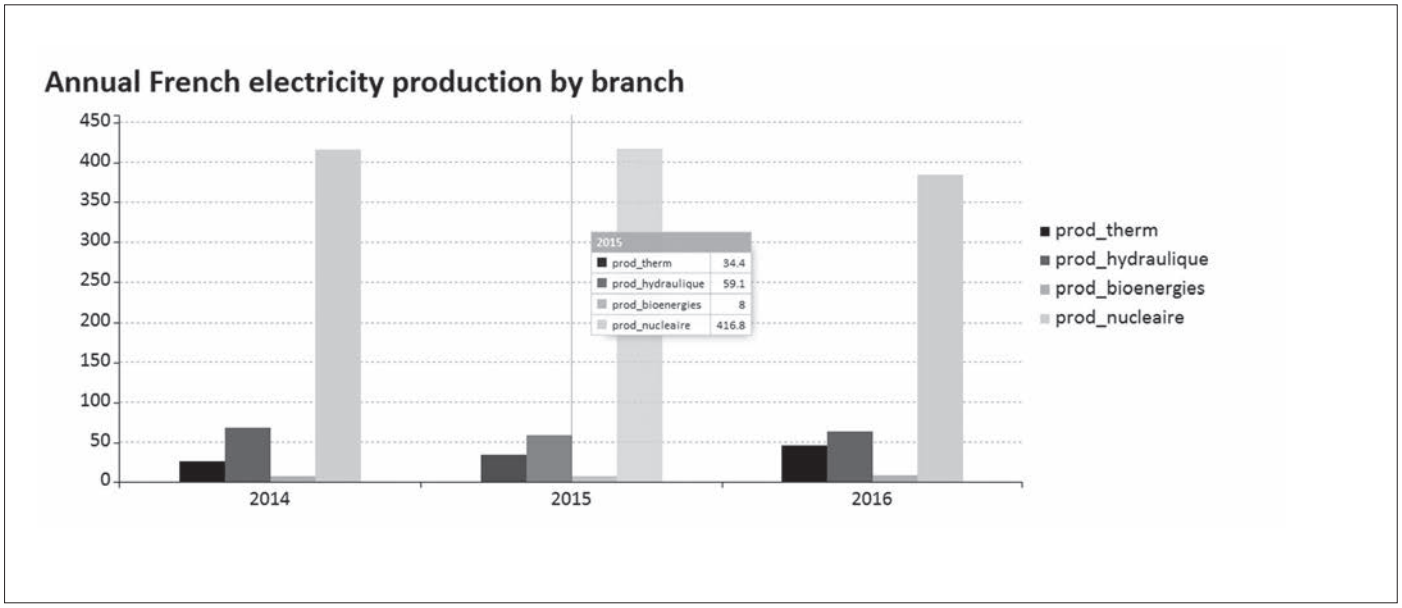


Figure 6
Default Hover Information in Billboarder

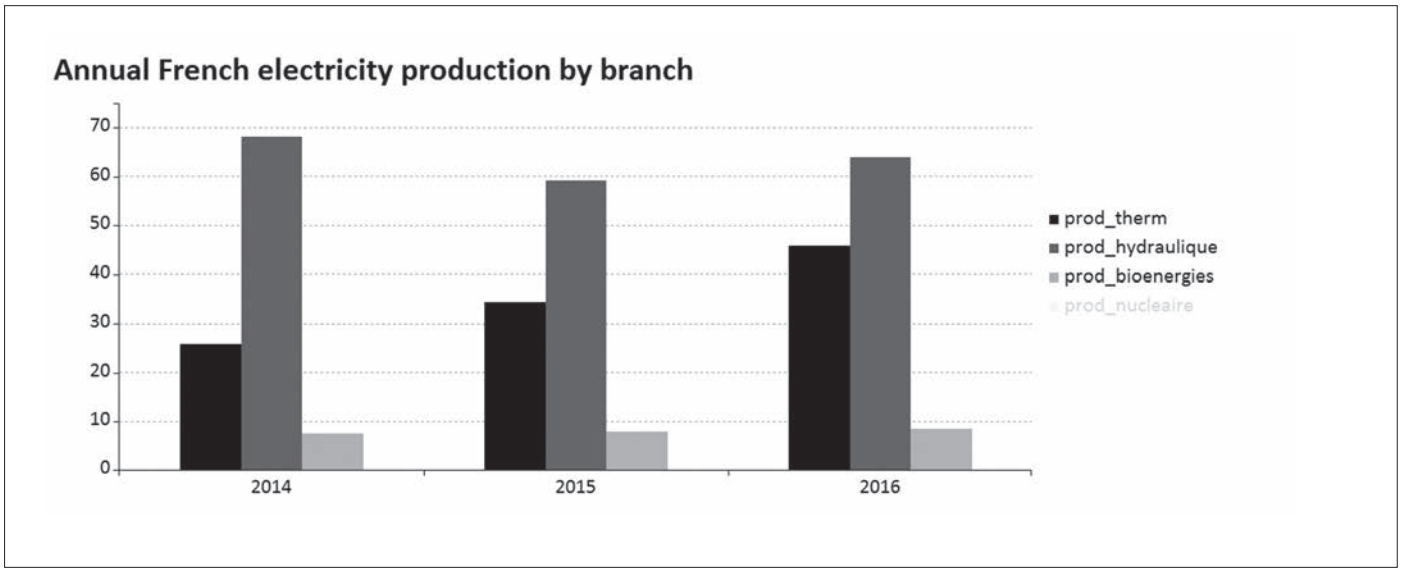
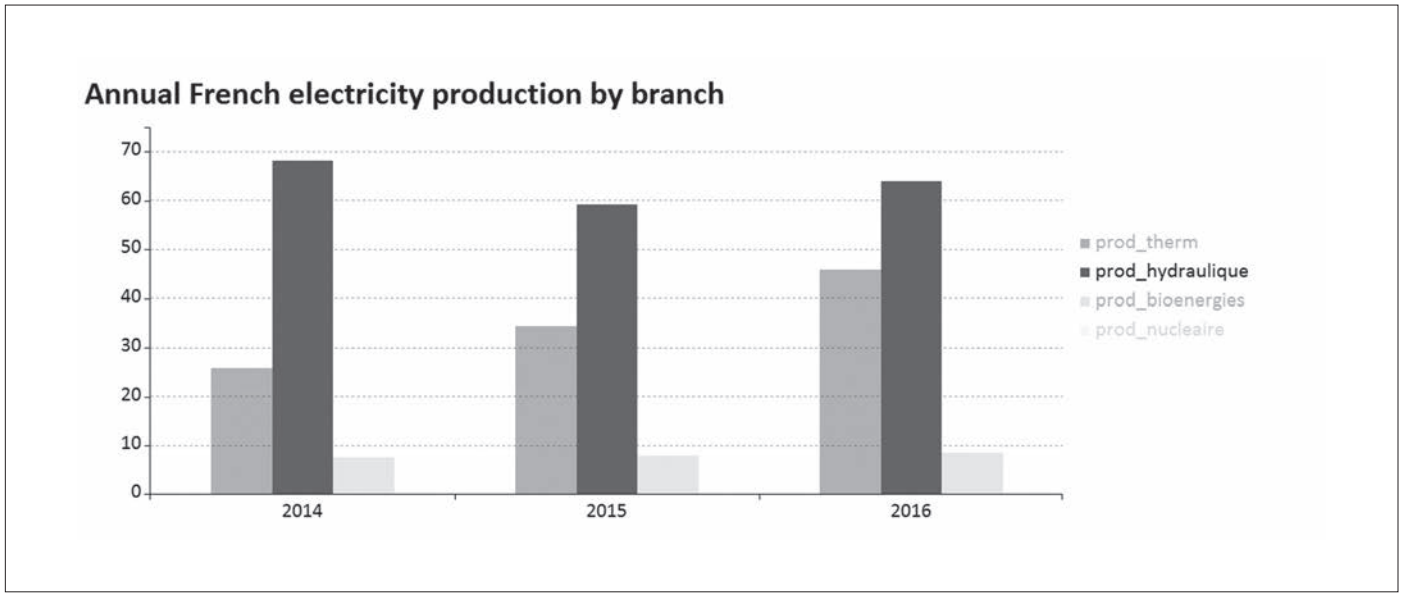


Figure 7
Before Hovering Over the Legend in Billboarder



Hover Information

For plotly, the default display format is: (x value, y value) series label, but there is a lot of freedom to design how you want to display information for a point on hover. Billboarder has a similar display for scatter plots but has an enhanced default option for bar plots and line charts. The x-axis value of the selected data point is displayed on the first line of the box, and the series label and y-value are displayed on the second line of the box with a vertical line in between. This feature can add value because all the y-values for a specific x-value are displayed at the same time.

Legend

For both plotly and billboarder, when a data series in the legend is clicked, the data points of the series will disappear in the graph. This will also lead to changes to the axes—readjusting the range of x- and y-axes to produce a better-scaled plot. For instance, in Figure 5, prod_nucleaire has an exceptionally large value compared to other variables, so the y-axis limit is set to 450 in the plots. After unclicking its legend, as seen in Figures 6 and 7, we see that the y-axis limit is now set to 70.

Of course, they will reappear after you click on their label in the legend again. One extra advantage of using billboarder is that when you hover your mouse over the legend, the data points of that data series will be highlighted in the graph and those of the other groups will become transparent in the background. This feature could be very useful if there are multiple series to compare and you only wish to focus on one. Figures 6 and 7 demonstrate how the bar for prod_hydraulique is highlighted after we hover the mouse over its legend. Plotly does not have this feature.

Display of Large Numbers

In plotly, large numbers are automatically scaled and labeled with k (for values in the thousands), m (millions), b (billions), etc. This feature is very helpful if the values you are displaying are very large. In billboarder, this is not a default

Table 1
Web Resources

Package	Examples	GitHub
plotly	https://plot.ly/r/	https://github.com/ropensci/plotly
billboarder	https://cran.r-project.org/web/packages/billboarder/vignettes/billboarder-intro.html	https://github.com/dreamRs/billboarder
highcharter	http://jkunst.com/highcharter/	https://github.com/jbkunst/highcharter
ggvis	http://ggvis.rstudio.com/ggvis-basics.html	https://github.com/rstudio/ggvis
googleVis	https://cran.r-project.org/web/packages/googleVis/vignettes/googleVis_examples.html	https://github.com/mages/googleVis
rCharts	https://ramnathv.github.io/rCharts	https://github.com/ramnathv/rCharts
rbokeh	http://hafen.github.io/rbokeh	https://github.com/bokeh/rbokeh

setting, but you can make it display the same way as plotly does manually.

ADDITIONAL POINTS OF COMPARISON AND FURTHER RESOURCES

Some packages have commercial or home websites with helpful information and examples, and all have at least a place to see the package in action. Other dimensions you may consider when selecting a package include the following.

1. **Accessibility.** Most are on the Comprehensive R Archive Network (CRAN), enabling easy installation directly from R, though rCharts must be installed from GitHub using the devtools package.
2. **Usability.** All of the packages have a GitHub repository where users can track suggestions and edits to get an idea for how well-used and maintained the package is. Plotly, highcharter, googleVis and rbokeh appear to be the best maintained.
3. **Flexibility.** Packages have varying levels of flexibility for altering basic plot components such as label size.
4. **Cost.** Most packages are free, but some like plotly[2,3] and highcharter have limits. This may be a concern if you wish to enable the users of your Shiny app to download plots after they have adapted them. For example, plotly has a limit to the number of downloads allowed per day, and highcharter charges for commercial use.
5. **Cross-functionality.** All of the packages we looked into work well with Shiny (important for our use case), and some also work with knitr. Depending on your use case, this may qualify or disqualify a given package.

Given our own use case, we're leaning toward using billboarder in our web applications, but we will continue to watch for developments in this space by using the resources we share in Table 1. For any of you who are interested in incorporating interactive plots into your products, we encourage you to consider your own needs and decide which package is best for you. ■

Zhen Hui Trinh is a data scientist for Milliman. She can be contacted at *zhenhui.trinh@milliman.com*

Ben Johnson is an actuarial data scientist for Milliman Financial Risk Management LLC. He can be contacted at *ben.johnson@milliman.com.*

Eileen S. Burns, FSA, MAAA, is a principal and consulting actuary with Milliman. She can be contacted at *eileen.burns@milliman.com.*

ENDNOTES

1 The html widgets for R website shows more about what's available; *http://www.htmlwidgets.org/showcase_plotly.html.*

2 Plotly. Chart Studio Cloud. *https://plot.ly/products/cloud* (Accessed Jan. 29, 2019).

3 Plotly. Chart Studio API Rate Limits. *https://help.plot.ly/api-rate-limits/#plotly-api-rate-limits* (Accessed Jan. 29, 2019).