autoplotly - Automatic Generation of Interactive Visualizations for Popular Statistical Results

by Yuan Tang

Abstract The **autoplotly** package provides functionalities to automatically generate interactive visualizations for many popular statistical results supported by ggfortify package with **plotly** and **ggplot2** style. The generated visualizations can also be easily extended using **ggplot2** and **plotly** syntax while staying interactive.

Background

With the help of base graphics, grid graphics, and **lattice** graphics (Sarkar, 2008), R users already have many plotting options to choose from. Each has their own unique customization and extensibility options. Nowadays, **ggplot2** has emerged as a popular choice for creating visualizations (Wickham, 2009) and provides a strong programming model based on a "grammar of graphics" which enables methodical production of virtually any kind of statistical chart. The **ggplot2** package provides a suit of succinct syntax and independent components and makes it possible to describe a wide range of graphics. It's based on an object-oriented model that is modular and extensible, which becomes a widely used framework for producing statistical graphics in R.

The distinct syntax of **ggplot2** makes it a definite paradigm shift from base and **lattice** graphics and presents a somewhat steep learning curve for those used to existing R charting idioms. Many industry R users, especially the users that build web applications in R by leveraging **shiny** (Chang et al., 2017) package, may not be satisfied with static plots. Those web applications often involve user interactions so that users can dive into the plots, explore areas of interest, and select relevant data points for more details. **ggiraph** (Gohel, 2017) is an extention of **ggplot2** that provides building blocks for users to build interactive plots and when used within a shiny application, elements associated with an id can be selected and manipulated on client and server sides. There are also other packages such as **d3r** (Bostock et al., 2017) and **plotly** (Sievert et al.) built on top of Javascript visualization frameworks that are totally isolated from **ggplot2** but become popular building blocks for creating interactive visualizations in R.

Often times users only want to quickly iterate the process of exploring data, building statistical models, and visualizing the model results, especially the models that focus on common tasks such as clustering and time series analysis. Some of these packages provide default base `plot` visualizations for the data and models they generate. However, they look out-of-fashion and these components require additional transformation and clean-up before using them in **ggplot2** and each of those transformation steps must be replicated by others when they wish to produce similar charts in their analyses. Creating a central repository for common/popular transformations and default plotting idioms would reduce the amount of effort needed by all to create compelling, consistent and informative charts. The **ggfortify** (Tang et al., 2016) package provides a unified **ggplot2** plotting interface to many statistics and machine-learning packages and functions in order to help these users achieve reproducibility goals with minimal effort. **ggfortify** package has a very easy-to-use and uniform programming interface that enables users to use one line of code to visualize statistical results of many popular R packages using **ggplot2** as building blocks. This helps statisticians, data scientists, and researchers avoid both repetitive work and the need to identify the correct **ggplot2** syntax to achieve what they need. Users are able to generate beautiful visualizations of their statistical results produced by popular packages with minimal effort.

The **autoplotly** (Tang) package is an extension built on top of **ggplot2**, **plotly**, and **ggfortify** to provide functionalities to automatically generate interactive visualizations for many popular statistical results supported by **ggfortify** package with **plotly** and **ggplot2** style. The generated visualizations can also be easily extended using **ggplot2** and **plotly** syntax while staying interactive.

Software Architecture

The **autoplotly** package calls **ggfortify**'s `autoplot()` method that invokes an registered S3 generic functions [1] for the applied object to create the visualizations with **pplot2** style. Next, the generated `ggplot` object is translated to `plotly` object with interactive graphical components leveraging

[1] `http://adv-r.had.co.nz/S3.html`

`plotly::ggplotly`. Additional clean-up and correction are then performed due to the feature parity between **plotly** and **ggplot2** that results in redundant and corrupted components . For example, if we want to generate interactive visualization for principal components analysis results produced from `prcomp(...)`, the following will be executed in order:

- `autoplotly(prcomp(...))` - calls **autoplotly**'s main function
- `autoplot.prcomp(prcomp(...))` - invokes the registered S3 generic function
- `ggplotly(autoplot.prcomp(prcomp(...)))` - translates `ggplot` object to `plotly` object

The final object is of class `plotly` with the corresponding `ggplot` object as one of its attributes. It can be easily extended using either **plotly** or **ggplot2** style. When additional **ggplot2** elements or components are applied, for example:

```
p <- autoplotly(prcomp(iris[c(1, 2, 3, 4)]), data = iris,
  colour = 'Species', label = TRUE, label.size = 3, frame = TRUE)

p <- p +
  ggplot2::ggtitle("Principal Components Analysis") +
  ggplot2::labs(y = "Second Principal Components", x = "First Principal Components")
p
```

The above example adds title and axis labels to the originally generated plot. When `` `+` <-function(e1,e2) `` operator is applied to a `ggplot` element as the second argument `e2`, e.g. `ggplot2::ggtitle(...)`, the `ggplot` object that we attached to the output of `autoplotly()` earlier will be used as the first argument `e1`, borrowing **ggplot2**'s extensibility.

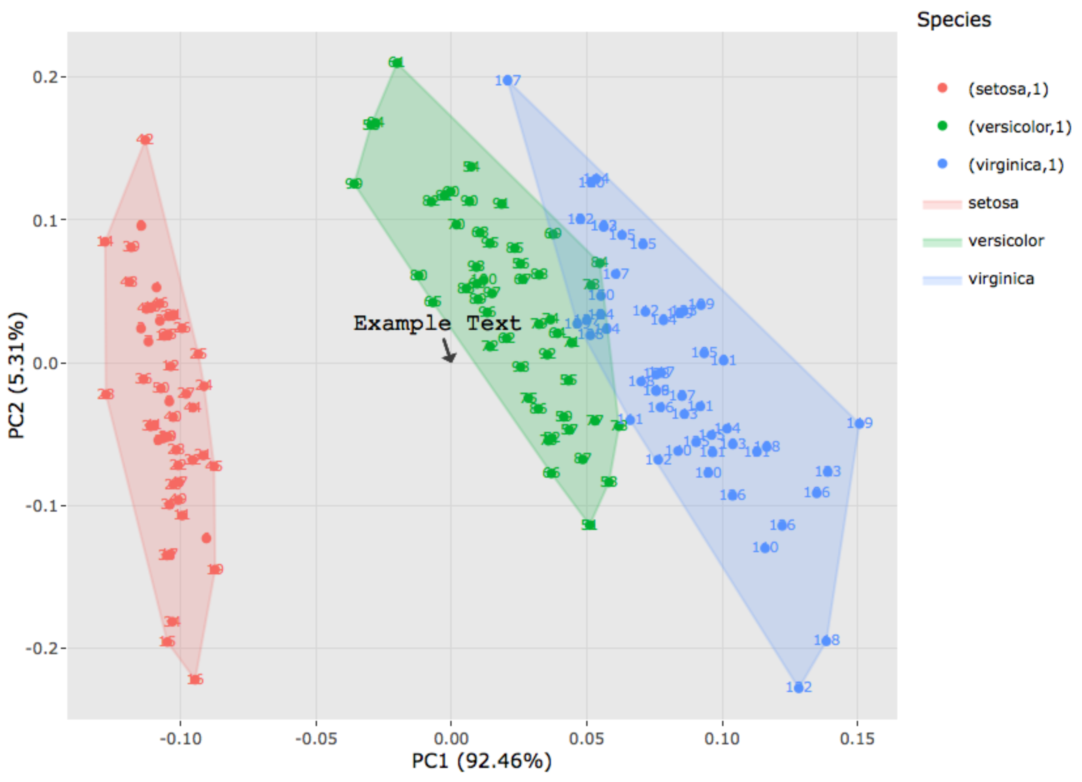

Figure 1: PCA with custom **plotly** style annotation element.

Similarly, if we are adding **plotly** interactive components, the `plotly` object from the output of `autoplotly()` `p` will be used instead. The following code adds a custom **plotly** annotation element placed to the center of the plot with an arrow, as shown in Figure 1:

```
p <- autoplotly(prcomp(iris[c(1, 2, 3, 4)]), data = iris,
  colour = 'Species', label = TRUE, label.size = 3, frame = TRUE)

p %>% layout(annotations = list(
  text = "Example Text",
  font = list(
    family = "Courier New, monospace",
    size = 18,
    color = "black"),
  x = 0,
  y = 0,
  showarrow = TRUE))
```

autoplotly re-exports `plotly::subplot()` to enable users to stack multiple interactive plots generated via `autoplotly()` together. Some statistical results produce multiple plots, e.g. `lm()` fitted model objects, which are given extra attention and additional manipulations are performed in order to make sure users can choose whether to share axis labels, change margins, and change layout strategy while keeping the interactive control of multiple sub-plots independently.

TODO: Individual-level zooming TODO: Hoverover metadata TODO: Zooming in details TODO: Exportability with export(p, "inst/images/iris_pca_full.png")

Illustrations

As demonstrated earlier, **autoplotly** package provides a `autoplotly()` function to work with objects of different classes produced from various popular statistical packages. This section highlights some of the example of automatic interactive visualizations from different types of statistical results.

The `autoplotly()` function works for the two essential classes of objects obtained from **stats** package: `stats::prcomp` and `stats::princomp`, for example:

```
autoplotly(prcomp(iris[c(1, 2, 3, 4)]), data = iris, frame = TRUE, colour = 'Species')
```

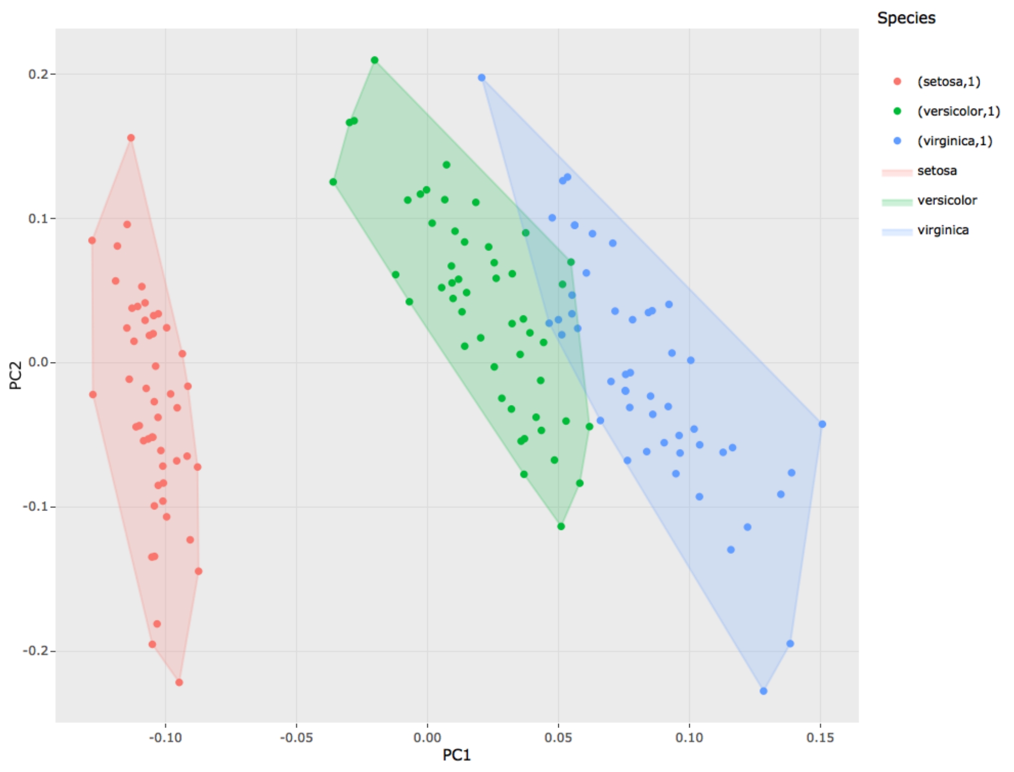

Figure 2: PCA with clolors and boundary for each flower species.

The above example automatically plots the PCA results from **stats** package. `autoplotly()` accepts parameters such as `frame` to draw the boundaries for each flower species and `colour` to indicate the column name to use to color each data points, as shown in Figure 2.

Users can also hover the mouse over to each point in the plot to see more details, such as principal components information and the species this particular data point belongs to, as shown in Figure 3.

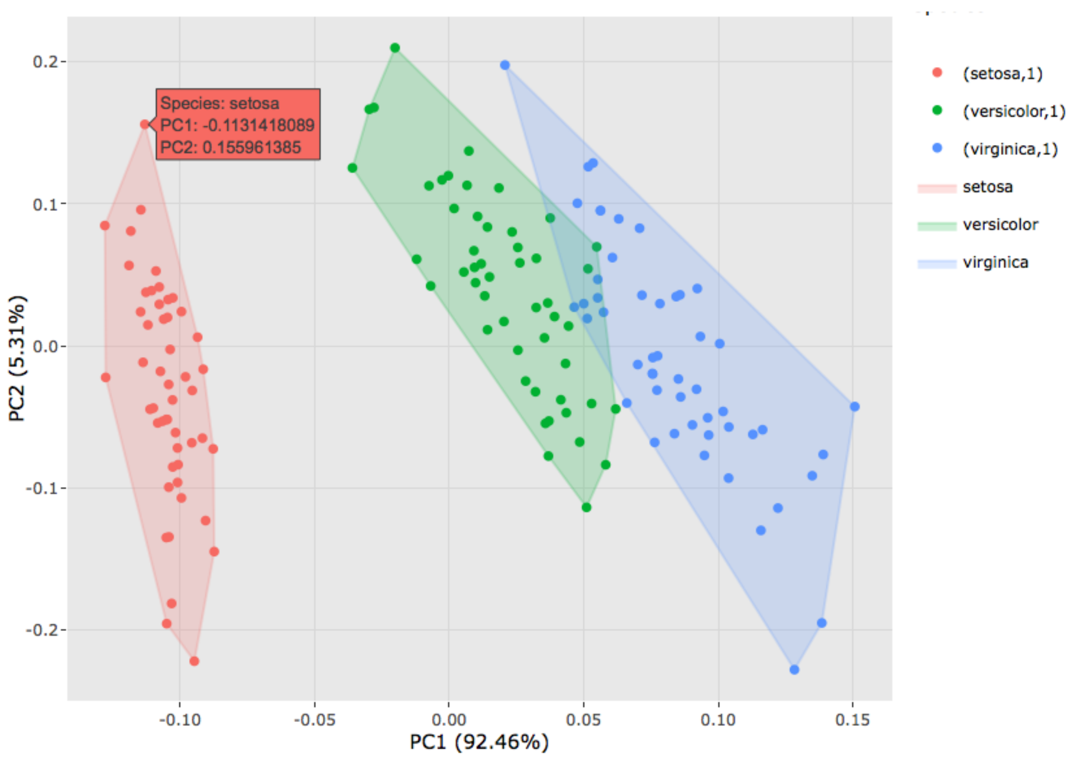

Figure 3: PCA with clolors and boundary for each principal component.

Forecasting packages such as **forecast** (Hyndman, 2015), **changepoint** (Killick et al., 2016), **strucchange** (Zeileis et al., 2002), and **dlm** (Petris, 2010), are popular choices for statisticians and researchers. Interactive visualizations of predictions and statistical results from those packages can be generated automatically using the functions provided by **autoplotly** with the help of **ggfortify**.

The **autoplotly** function automatically plots the change points with optimal positioning for the `AirPassengers` data set found in the **changepoint** package using the `cpt.meanvar` function, shown in Figure 4.

```
library(changepoint)
autoplotly(cpt.meanvar(AirPassengers))
```

The **autoplotly** function automatically plots the original and smoothed line from Kalman filter function in **dlm** package as shown in Figure 5.

```
library(dlm)
form <- function(theta){
  dlmModPoly(order = 1, dV = exp(theta[1]), dW = exp(theta[2]))
}
model <- form(dlmMLE(Nile, parm = c(1, 1), form)$par)
filtered <- dlmFilter(Nile, model)
autoplotly(filtered)
```

Additionally, **autoplotly** plots the optimal break points where possible structural changes happen in the regression models built by the `strucchange::breakpoints`, shown in Figure 6.

```
library(strucchange)
autoplotly(breakpoints(Nile ~ 1), ts.colour = "blue", ts.linetype = "dashed",
           cpt.colour = "dodgerblue3", cpt.linetype = "solid")
```

The `autoplotly` can also automatically generate interactive plots for results producuced by **splines**, such as B-spline basis points visualization for natural cubic spline with boundary knots shown in Figure 7.

```
library(splines)
autoplotly(ns(diamonds$price, df = 6))
```

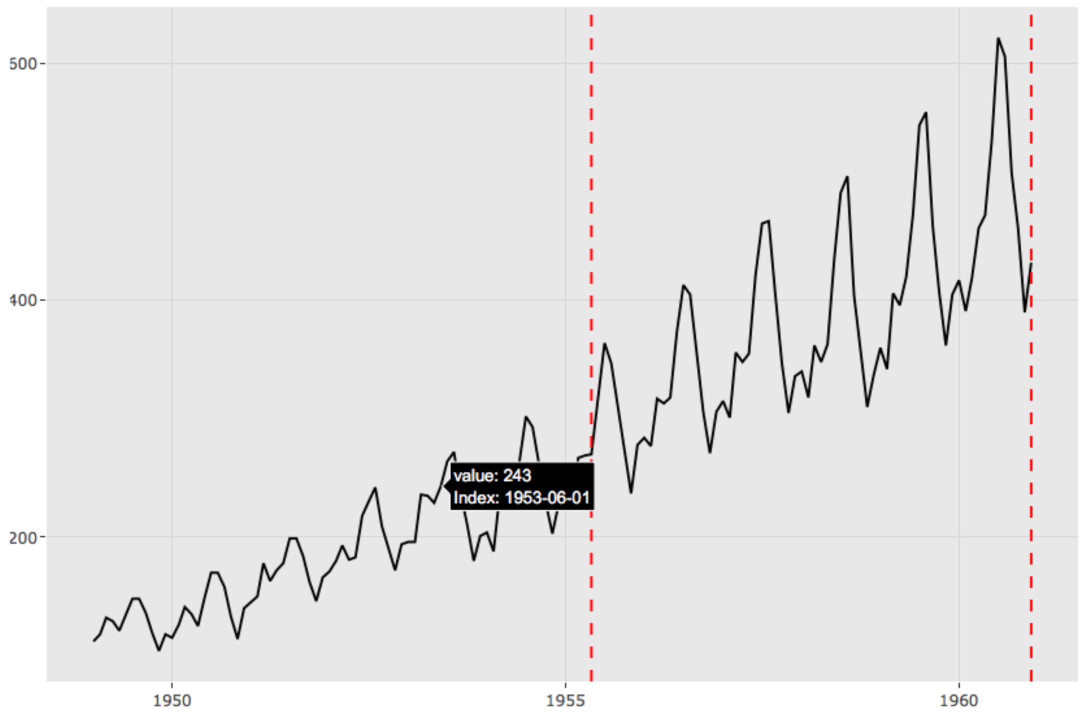

Figure 4: Change points with optimal positioning for AirPassengers.

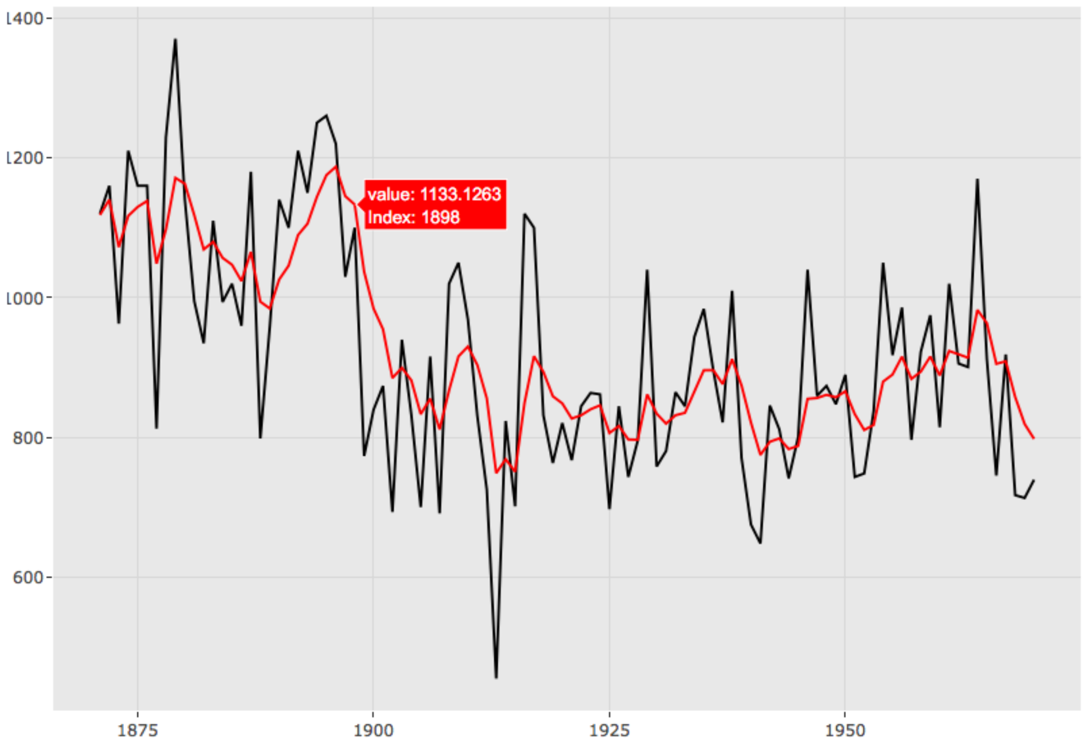

Figure 5: Smoothed time series by Kalman filter.

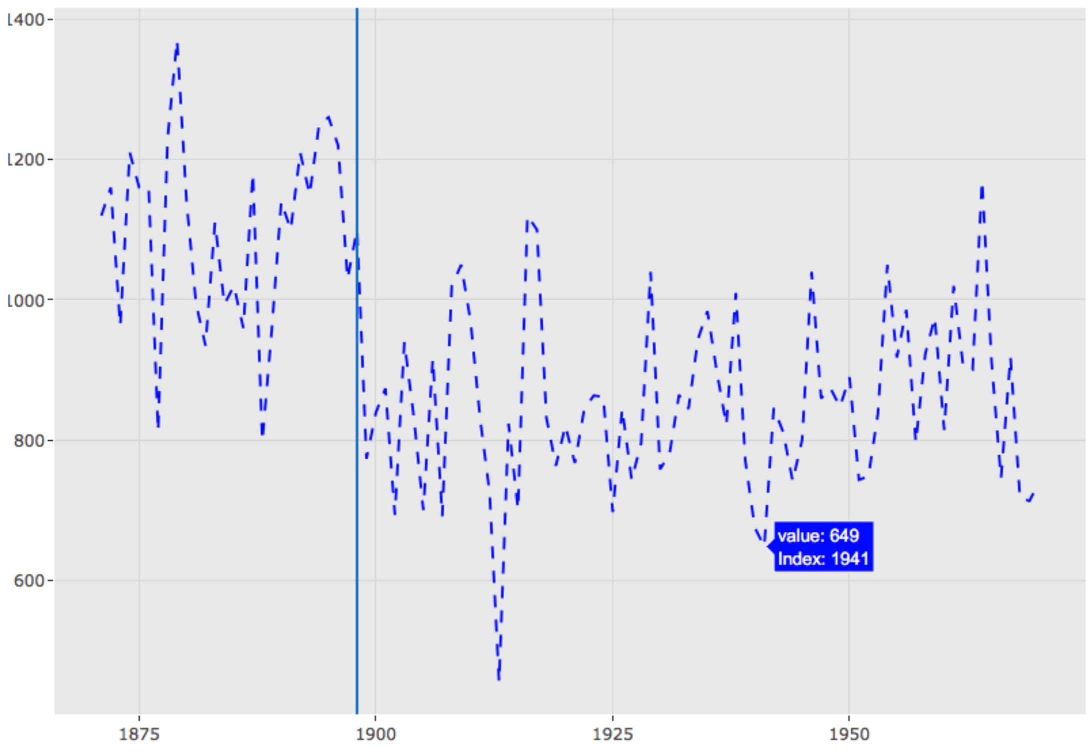

Figure 6: Optimal break points with possible structural changes.

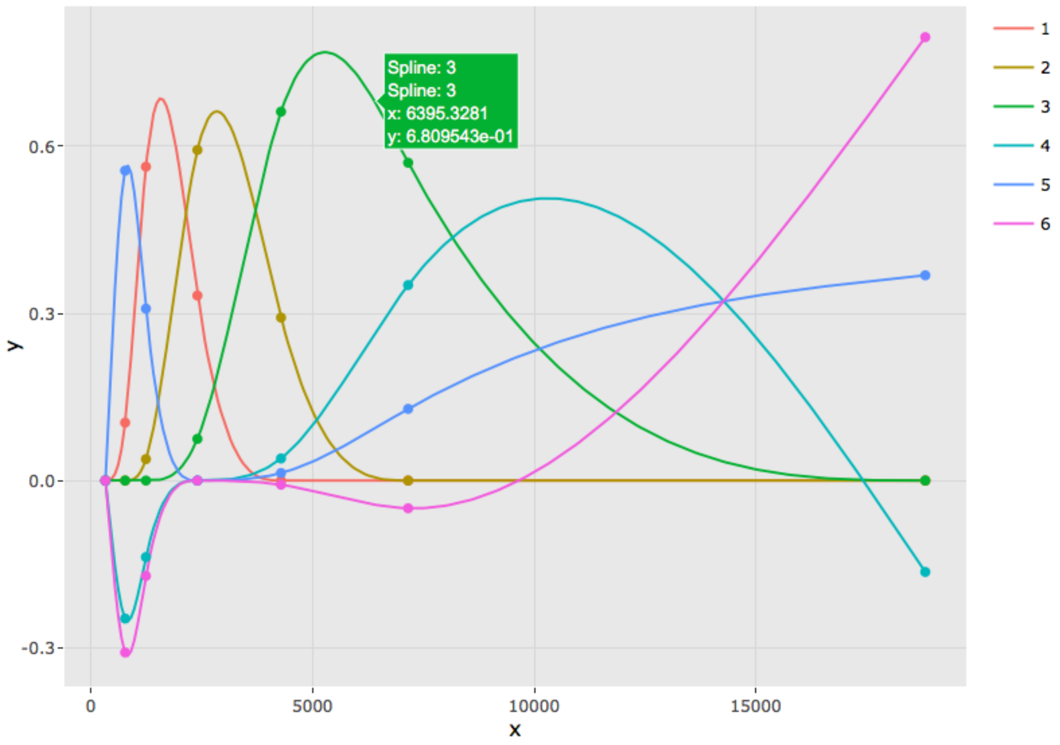

Figure 7: B-spline basis points for natural cubic spline with boundary knots.

Users can also stack multiple plots generated from `autoplotly()` together in a single view using `subplot()`, two interactive splines visualizations with different degree of freedom are stacked into one single view in the following example, as shown in Figure 8:

```
library(splines)
subplot(
  autoplotly(ns(diamonds$price, df = 6)),
  autoplotly(ns(diamonds$price, df = 3)), nrows = 2, margin = 0.01)
```

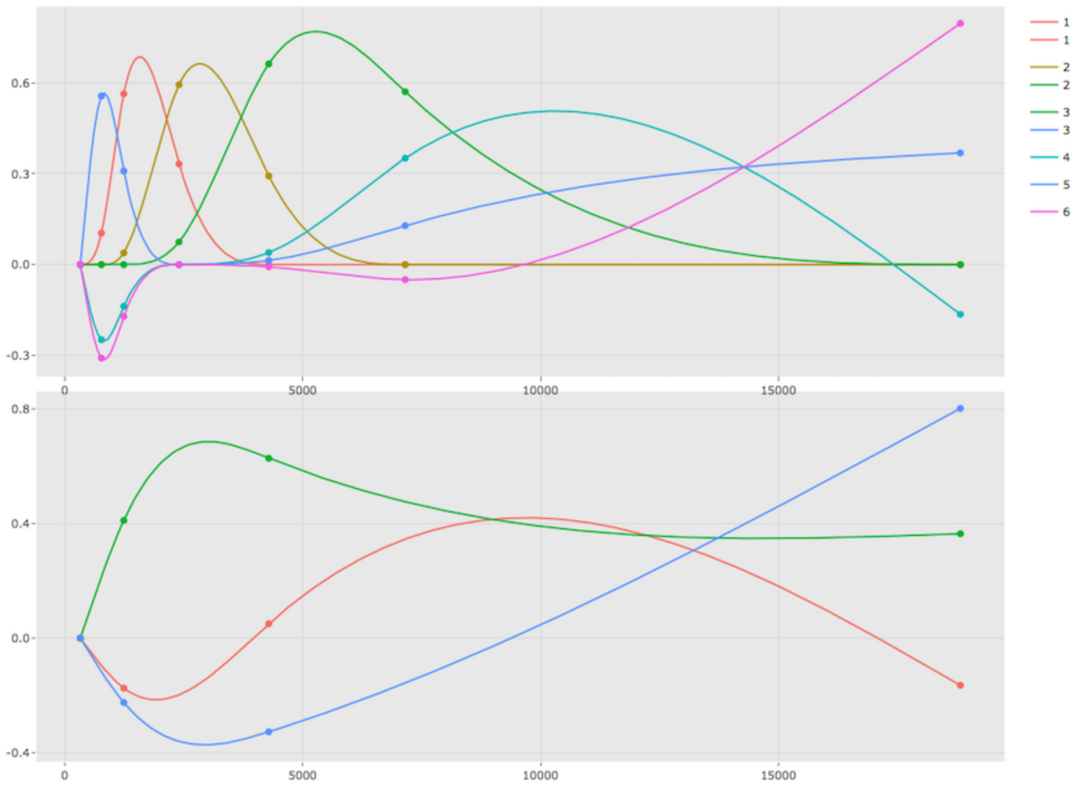

Figure 8: Multiple splines visualizations with different degree of freedom.

Note that the interactive control of those two plots are independent. In other words, you can control two sub-plots separately without affecting each other. Additional options for `subplot()` are available to select whether to share axises, titles, and which layout of the plots to adopt, etc.

The `autoplotly()` function is able able to interpret `lm` fitted model objects and allows the user to select the subset of desired plots through the `which` parameter (just like the `plot.lm` function). The `which` parameter allows users to specify which of the subplots to display. Many plot aesthetics can be changed by using the appropriate named parameters. For example, the `colour` parameter is for coloring data points, the `smooth.colour` parameters is for coloring smoothing lines and the `ad.colour` parameters is for coloring the auxiliary lines, as demonstrated in Figure 9 and the following code:

```
autoplotly(
  lm(Petal.Width ~ Petal.Length, data = iris),
  which = c(4, 6), colour = "dodgerblue3",
  smooth.colour = "black", smooth.linetype = "dashed",
  ad.colour = "blue", label.size = 3, label.n = 5,
  label.colour = "blue")
```

Summary

This file is only a basic article template. For full details of *The R Journal* style and information on how to prepare your article for submission, see the Instructions for Authors.

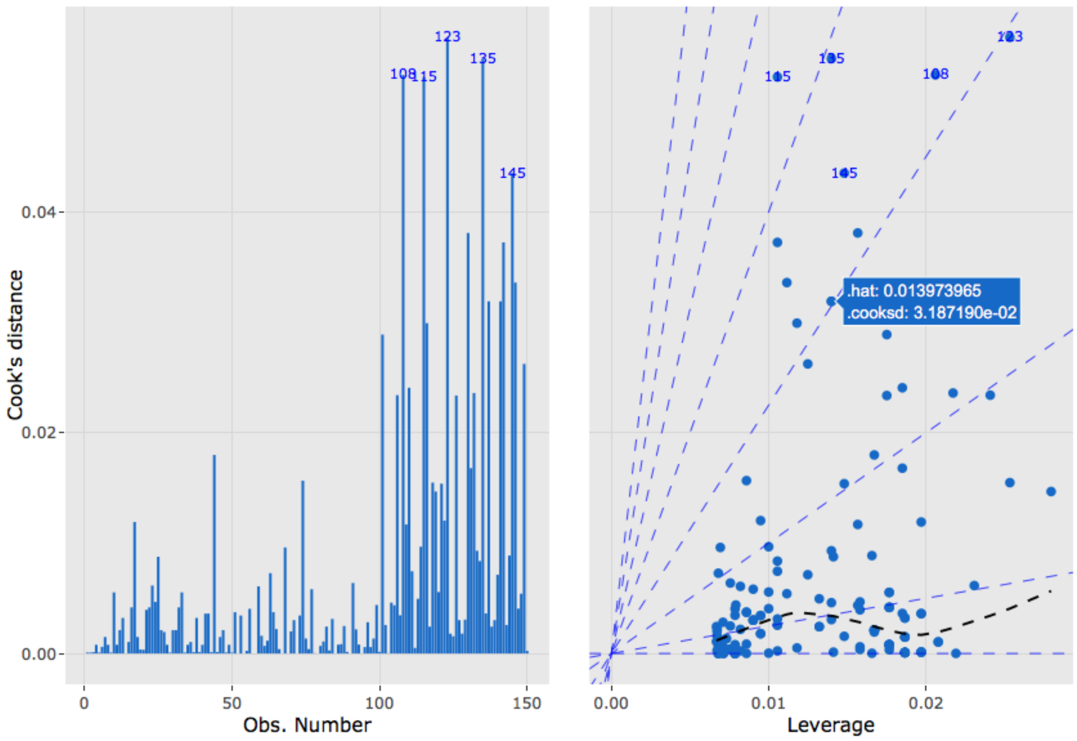

Figure 9: Linear model results.

Bibliography

M. Bostock, K. Russell, G. Aisch, and A. Pearce. *d3r: 'd3.js' Utilities for R*, 2017. URL https://CRAN.R-project.org/package=d3r. R package version 0.7.1. [p1]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.5. [p1]

D. Gohel. *ggiraph: Make 'ggplot2' Graphics Interactive*, 2017. URL https://CRAN.R-project.org/package=ggiraph. R package version 0.4.2. [p1]

R. J. Hyndman. *forecast: Forecasting functions for time series and linear models*, 2015. URL http://github.com/robjhyndman/forecast. R package version 6.2. [p4]

R. Killick, K. Haynes, and I. A. Eckley. *changepoint: An R package for changepoint analysis*, 2016. URL http://CRAN.R-project.org/package=changepoint. R package version 2.2.1. [p4]

G. Petris. An R package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16, 2010. URL http://www.jstatsoft.org/v36/i12/. [p4]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL http://lmdvr.r-forge.r-project.org. ISBN 978-0-387-75968-5. [p1]

C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy. *plotly: Create Interactive Web Graphics via 'plotly.js'*. [p1]

Y. Tang. *autoplotly: Automatic Generation of Interactive Visualizations for Popular Statistical Results*. URL https://github.com/terrytangyuan/autoplotly. R package version 0.1.0. [p1]

Y. Tang, M. Horikoshi, and W. Li. ggfortify: Unified interface to visualize statistical result of popular r packages. *The R Journal*, 8, 2016. URL https://journal.r-project.org/. [p1]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer Science & Business Media, 2009. [p1]

A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. strucchange: An R package for testing for structural change in linear regression models. *Journal of Statistical Software*, 7(2):1–38, 2002. URL http://www.jstatsoft.org/v07/i02/. [p4]

Yuan Tang
H2O.ai
2309 Wake Robin Drive
West Lafayette, IN 47906
terrytangyuan@gmail.com