We have guided you through the process of writing a complex JavaScript-callback. However, most users might be slightly frustrated by the lack of arguments to customize a standard rBokeh plot fully. Actually, rBokeh is a little bit outdated (structure() warnings all the way!) and lacks some functionalities that are available in its Python equivalent. But don’t toss in the towel right away! I created some workarounds for these, which you hopefully find helpful.

**In general**

This approach is my go-to solution to change a rBokeh plot for which there is an argument missing in rBokeh that is available in python.

* Create the plot.
* Inspect the structure (str(plot)) of the rBokeh object.
* Search for the python’s argument name.
* Overwrite the value with the desired option as derived from python’s bokeh.

So, first of all, I set up an initial rBokeh figure that we manipulate later on.

plot <- figure(data = iris) %>%

ly\_bar(x = Species,

y = Sepal.Length,

hover = TRUE)

**Manipulate the hover functionality**

The first set of tricks deals with the customization of hover effects. Hover effects are essentials of interactive plots, so it makes a lot of sense to invest some time in optimizing them.

**Anchor**

Unlike in python’s bokeh, there is no anchor argument to change the position of a hover tooltip. By default, it appears in the center of the hovered element. To change it, we need to deep dive into the rBokeh object. The object is a deep and complex nested list in which all the information about the plot is stored. While some elements are always structured in the same way, different layers are named by a seemingly arbitrary string (e.g., 51dab389c6209bbf084a86b368f68724). I wrote the following code snippet to change the hover position from center to top\_center.

# Get the position of the anchor argument within the object-list

xyz <- logical()

for (i in seq\_along(plot$x$spec$model)) {

xyz[i] <- !is.null(plot$x$spec$model[[i]]$attributes$point\_policy)

}

# Solution using for loop

for (i in which(xyz)) {

plot$x$spec$model[[i]]$attributes$anchor <- "top\_center"

}

In case you are not very fond of simple for loops, here are also solutions with [purrr](https://www.statworx.com/de/blog/about-risks-and-side-effects-consult-your-purrr-macist/) or lapply:

# Solution using purrr

xyz <- purrr::map\_lgl(plot$x$spec$model, .f = ~ !is.null(.x$attributes$anchor))

plot$x$spec$model[which(xyz)] <- purrr::map(plot$x$spec$model[which(xyz)],

~{

.$attributes$anchor <- "top\_center"

return(.)

})

# Solution using the apply family

xzy <- sapply(plot$x$spec$model, function(x) !is.null(x$attributes$anchor))

plot$x$spec$model[which(xyz)] <- lapply(plot$x$spec$model[xyz],

function(abc) {

abc$attributes$anchor <- "top\_center"

return(abc)

})

All options of the tooltip position can be found [here](https://bokeh.pydata.org/en/latest/docs/reference/core/enums.html#bokeh.core.enums.LegendLocation).

**Point policy**

Another option that can be specified in the same way is whether the tooltip should appear at a specific place (snap\_to\_data) or should follow the courser (follow\_mouse). This point\_policy option is also missing in rBokeh but can be added by the same logic. Here is a solution for the purrr way but all other descriped options work as well.

# Get the position of the point policy argument within the object-list

xyz <- purrr::map\_lgl(plot$x$spec$model, .f = ~ !is.null(.x$attributes$point\_policy))

plot$x$spec$model[which(xyz)] <- purrr::map(plot$x$spec$model[which(xyz)],

~{

.$attributes$point\_policy <- "follow\_mouse"

return(.)

})

**What you see is what you want**

The last hover-related issue I want to address are the shown values. rBokeh is rather inflexible in this context. Sometimes (e.g., in ly\_points) it is possible to define a specific hover information (either a variable from the data or another data frame/list of the same length as the plot data) but in other cases the hover argument is just logical (TRUE or FALSE, like in ly\_bar). If you want to change its default tooltip you need to do this by hand, again.

# Set up the figure

plot <- figure(data = iris) %>%

ly\_bar(x = Species,

y = Sepal.Length,

hover = T)

# get the list elements where tooltips are defined

hover\_info <- purrr::map\_lgl(plot$x$spec$model, .f = ~ !is.null(.x$attributes$tooltips))

# delete a specific tooltip

plot$x$spec$model[[which(hover\_info)]]$attributes$tooltips[[2]] <- NULL

# add a tooltip

plot$x$spec$model[[which(hover\_info)]]$attributes$tooltips[[2]] <-

# list of printed name (test) and name for internal use (@hover\_col\_3)

list("test","@hover\_col\_3")

hover\_data <- purrr::map\_lgl(plot$x$spec$model, .f = ~ !is.null(.x$attributes$data$hover\_col\_1))

# manipulate a tooltip

plot$x$spec$model[which(hover\_data)] <- purrr::map(plot$x$spec$model[which(hover\_data)],

~{

.$attributes$data$hover\_col\_1 <- 1:3

# must match assigned name above

.$attributes$data$hover\_col\_3 <- letters[1:3]

return(.)

})

plot

**Keep plotting!**

I hope you enjoyed my blog post, and it helps you in solving or avoiding some troubles with rBokeh. And who knows, maybe a more intense use of this package might even motivate the developers to update or further develop this excellent package. So, keep plotting!