R Visualizations - plotlyStatistics 506

No plot version

Due to **plotly** plots being interactive, they have a tendency to slow down certain web browsers. This version of the notes does *not* execute any R code and thus does not display any plots.

Introduction

Plotly is a commerical company that makes available open source graphing libraries for R, Python, Julia, and others. In particular, the plots generated are interactive so that users can modify their view of the plot to hopefully make things more clear.

There are two primary ways to interface with **plotly**. The first is to pass any object created via `ggplot()` into the `ggplotly()` function. The second is to use `plot_ly` to generate a plot from scratch.

```
library(plotly)
library(ggplot2)
```

ggplotly

Any `gg` object created by `ggplot` can be passed into `ggplotly`.

```
# This code taken from the previous set of notes
nnmaps <- read.csv("data/chicago-nmmaps.csv")
nnmaps$date <- as.Date(nnmaps$date)
suppressWarnings(nnmaps_month <-
                   aggregate(nnmaps, by = list(nnmaps$month_numeric,
                                               nnmaps$year),
                             FUN = mean, na.rm = TRUE))
nnmaps_month <- nnmaps_month[order(nnmaps_month$year,
```

```
                                        nnmaps_month$month_numeric), ]
nnmaps_month$time <- seq_len(nrow(nnmaps_month))


g <- ggplot(nnmaps, aes(x = date, y = o3, color = season)) +
  geom_point() +
  geom_line(data = nnmaps_month, color = "red", linewidth = 3)
```

Play around with the above plot. You can …

- mouse over individual points
- zoom in and out or move the plot around (changing the tooltip type)
- hide or isolate the subgroups by clicking/double clicking on the legend

However, with more advanced plots, the conversion from a ggplot into a plotly plot can cause issues. Recall this plot the previous set of notes:

```
data(mpg)
g <- ggplot(mpg) +
  geom_jitter(aes(x = hwy, y = cty, color = displ, shape = drv, size = cyl)) +
  scale_radius()
g
```

Note the three separate scales in the legends. Observe what happens when we convert to a plotly object.

```
ggplotly(g)
```

plot_ly

For more advanced plots, we're better off creating plots directly in plotly with the `plot_ly` function. To be honest, plotly is not the easiest package to use, but with a little work (and a lot of online resources) you can figure out how to generate plots.

Let's start by replicating the NNMAPS seasonal data.

```
p <- plot_ly(nnmaps, x = ~ date, y = ~ o3, color = ~ season,
             type = "scatter", mode = "markers")
p
```

This shows the basics of a plot: the `x`, `y` and `color` arguments being passed into `...`, the `type` argument defining what kind of plot, and the `mode` being an argument that scatterplots supports.

A plotly objects consists of some number of "trace" and a "layout". Each trace represents one plotted object, and more specifically the transformation that connects the data to the visual plot.

Behind the scenes, R `plotly` objects are converted into JSON lists before being passed to the browser for drawing. We can examine these objects to get information about the traces.

```
pb <- plotly_build(p) # Convert the plot into the JSON list
sapply(pb$x$data, "[[", "name")
```

So we see that each of the four seasons generated it's own trace. There is obviously a massive amount of information inside the `pb` object. If you are really adventurous, you can manipulate it directly to modify the plotted object.

```
pb$x$data[[1]]$mode <- "markers+lines"
pb
```

We'll add the averaged line by adding another trace to the plot.

```
p2 <- add_trace(p, x = ~ date, y = ~ o3,
          data = nnmaps_month,
          type = "scatter",
          mode = "lines",
          color = "red",
          name = "Average",
          line = list(width = 10))
sapply(plotly_build(p2)$x$data, "[[", "name")
p2
```

`add_trace` is a generic version that requires specifying the `type` (or letting **plotly** guess); there exist specific `add_*`s as well.

```
# Not run
add_lines(p, x = ~ date, y = ~ o3,
          data = nnmaps_month,
          color = "red",
          name = "Average",
          line = list(width = 10))
```

Often it is best to "manually" add each trace to an empty plot.

```
plot_ly() |>
  add_markers(x = ~ date, y = ~ o3, color = ~ season, data = nnmaps) |>
  add_lines(x = ~ date, y = ~ o3,
            data = nnmaps_month,
            color = "red",
            name = "Average",
            line = list(width = 10))
```

Plotly has a version of inheritance similar to **ggplot2**. I do find it more fickle and liable to error.

```
plot_ly(x = ~ date, y = ~ o3, color = ~ season, data = nnmaps,
        type = "scatter", mode = "markers") |>
  add_lines(data = nnmaps_month, # no `x=` or `y=` argument
            color = "red",
            name = "Average",
            line = list(width = 10))
```

3d-plots can be easily created by adding a third dimension and switching to the 3d scatterplot type.

```
plot_ly(z = ~ date, x = ~ o3, y = ~ temp, color = ~ season, data = nnmaps,
        type = "scatter3d", mode = "markers")
```

Performance of `ggplotly` vs `plot_ly`

```
library(microbenchmark)
mb <- microbenchmark(
  ggplot = ggplotly(ggplot(nnmaps, aes(x = date, y = o3, color = season)) +
                      geom_point() +
                      geom_line(data = nnmaps_month,
                                color = "red", linewidth = 3)),
  plot_ly =
    plot_ly() |>
    add_markers(x = ~ date, y = ~ o3, color = ~ season, data = nnmaps) |>
    add_lines(x = ~ date, y = ~ o3,
              data = nnmaps_month,
              color = "red",
              name = "Average",
              line = list(width = 10))
```

```
)
print(mb, unit = "s", signif = 4)
```

This implies that `ggplotly` is over many times slower! This could have a big impact on larger documents with many embedded interactive plots.

Layout

The `layout` function allows manipulation of the non-trace attributes of the plot. It takes in a `plotly` object and returns one as well so its appropriate in a pipe chain.

```
p <- plot_ly(x = ~ date, y = ~ o3, color = ~ season, data = nnmaps,
             type = "scatter", mode = "markers")
p2 <- p |> layout(title = "O3 over time",
                  yaxis = list(type = "log",
                               title = "log(o3)"))
p2
```

Similar to how we extracted or could modify aspects of the JSON to affect the trace, we likewise can do so to affect the layout.

```
plotly_build(p)$x$layout$title
plotly_build(p2)$x$layout$title
p3 <- plotly_build(p2)
p3$x$layout$title <- "My new title!"
p3
```

Subplots

The `subplot` function can easily combine multiple plots into a single output.

```
p1 <- plot_ly() |> add_markers(x = ~ date, y = ~ o3,
                               data = nnmaps, name = "o3")
p2 <- plot_ly() |> add_markers(x = ~ date, y = ~ temp,
                               data = nnmaps, name = "type")
subplot(p1, p2)
subplot(p1, p2, nrows = 2)
```

The `share_` arguments control whether axes are linked.

```
subplot(p1, p2, shareY = TRUE)
subplot(p1, p2, nrows = 2, shareX = TRUE)
```

While in this example we're placing both scattergraphs, they can be any types of traces.

Adding additional interactivity

So far all the interactivity we've seen has been zooming or panning or hovering. However, we can also fairly easily add functionality to let users modify parts of the plot. For example, let's allow users to choose what predictor goes on the Y axis. This takes place in the `layout` as well.

```
p <- plot_ly(data = nnmaps) |>
  add_markers(x = ~ date, y = ~ o3) |>
  add_markers(x = ~ date, y = ~ temp, visible = FALSE)
p
```

We've added a second trace, but we've made it not visible. Now we can add a menu where a user can toggle which trace is visible.

```
p |> layout(updatemenus = list(
  list(
    y = 0.5,
    buttons = list(
      list(method = "update",
           args = list(list(visible =  list(TRUE, FALSE)),
                       list(yaxis = list(title = "o3"))),
           label = "o3"),

      list(method = "update",
           args = list(list(visible =  list(FALSE, TRUE)),
                       list(yaxis = list(title = "temp"))),
           label = "temp"))
  )
))
```

We can also let users control whether to plot lines or scatterplot, but this time let's make them buttons rather than a menu.

```
p |> layout(updatemenus = list(
  list(
```

```
    y = .8,
    buttons = list(
      list(method = "update",
           args = list(list(visible =  list(TRUE, FALSE)),
                       list(yaxis = list(title = "o3"))),
           label = "o3"),

      list(method = "update",
           args = list(list(visible =  list(FALSE, TRUE)),
                       list(yaxis = list(title = "temp"))),
           label = "temp"))),
  list(
    type = "buttons",
    y = .7,
    buttons = list(
      list(method = "update",
           args = list(list(mode = "markers")),
           label = "markers"),
      list(method = "update",
           args = list(list(mode = "lines")),
           label = "lines"))
  )
))
```

Finally, let's let users zoom in and out of the range in the x-axis more naturally.

```
p |> layout(
  xaxis = list(
    rangeslider = list(type = "date")
  ),
  updatemenus = list(
  list(
    y = .8,
    buttons = list(
      list(method = "update",
           args = list(list(visible =  list(TRUE, FALSE)),
                       list(yaxis = list(title = "o3"))),
           label = "o3"),

      list(method = "update",
           args = list(list(visible =  list(FALSE, TRUE)),
```

```
                      list(yaxis = list(title = "temp"))),
           label = "temp"))),
  list(
    type = "buttons",
    y = .7,
    buttons = list(
      list(method = "update",
           args = list(list(mode = "markers")),
           label = "markers"),
      list(method = "update",
           args = list(list(mode = "lines")),
           label = "lines"))
  )
))
```