Create interactive time series charts with plotly in R

19. April 2020

When analysing time series data I often draw on a small set of R helper functions to quickly visualise information using the excellent plotly package. The code creates interactive charts that allow the user to selectively display series and switch between transformations (level, quarterly and annual growth rates). I recently decided to put the code up online to make these functions available to others. This post provides a quick run-down of how these functions work.

First, grab the file `plotly_chart_helpers.R` from this GitHub repository, then load it alongside the required packages:

```r
library("xts")
library("plotly")
library("lubridate")

download.file(url = "https://raw.githubusercontent.com/stefanangrick/r-helpers/master/plotly_chart_helpers.R", destfile = "plotly_chart_helpers.R")
source("plotly_chart_helpers.R")
```

Interactive line plot

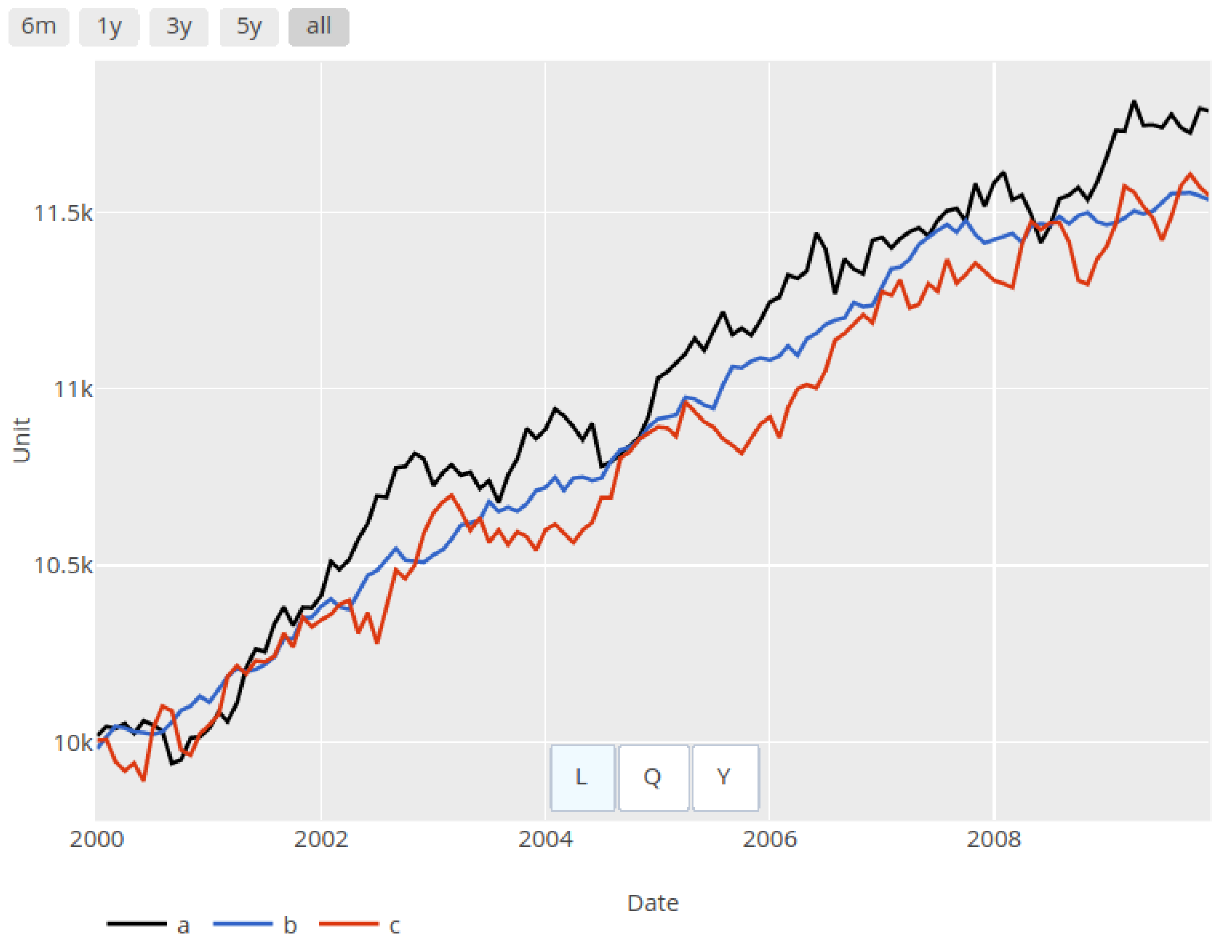
To create an interactive line plot, let's first generate some sample data. Note that we need to turn our data into an xts object for it to work with our helper functions.

```r
dat <- data.frame("a" = sin(seq(1, 9*pi, length.out = 120)) +
                    rnorm(120, 0, 50),
                  "b" = sin(seq(1, 8*pi, length.out = 120)) +
                    rnorm(120, 0, 20),
                  "c" = sin(seq(1, 7*pi, length.out = 120)) +
                    rnorm(120, 0, 50))
```

```
dat <- cumsum(dat) + seq(10000, by = 10, length.out = 120)
dat <- xts::xts(x = dat,
                order.by = seq(as.Date("2000-01-01"),
                               by = "month",
                               length.out = 120))
```

Let's now call the `plotly_line_growth()` function with our new data set:

```
plotly_line_growth(dat, x_lab = "Date", y_lab = "Unit")
```



[Interactive chart](#)

Let's briefly look at what happens in the background. The function takes our `xts` object (and, optionally, a second `xts` object of different frequency) and calculates quarterly and annual growth rates. This can be disabled by setting the parameters `add_q` and `add_y` to `FALSE`. By default, percentage growth rates are calculated, but if you prefer simple differences set the parameter `rate` to `FALSE`. The function then merges these new data series with the growth rates into the main set, appending `_q` and `_y` suffixes to the respective column names. While doing this, it keeps track of the colour palette we supplied (the included default palette can be overridden using the `colpal` parameter) to make sure the colours for growth rates match those for level series.

It then creates a plotly object, adding each series with its appropriate colour while making sure series showing growth rates are set to invisible. This is because we then add buttons to the chart that selectively show and hide certain data series: A button “Q” that when toggled will show quarterly growth rates while hiding annual growth rates and level data. A button “Y” that does the same for annual growth rates and a button “L” that does the same for level data.

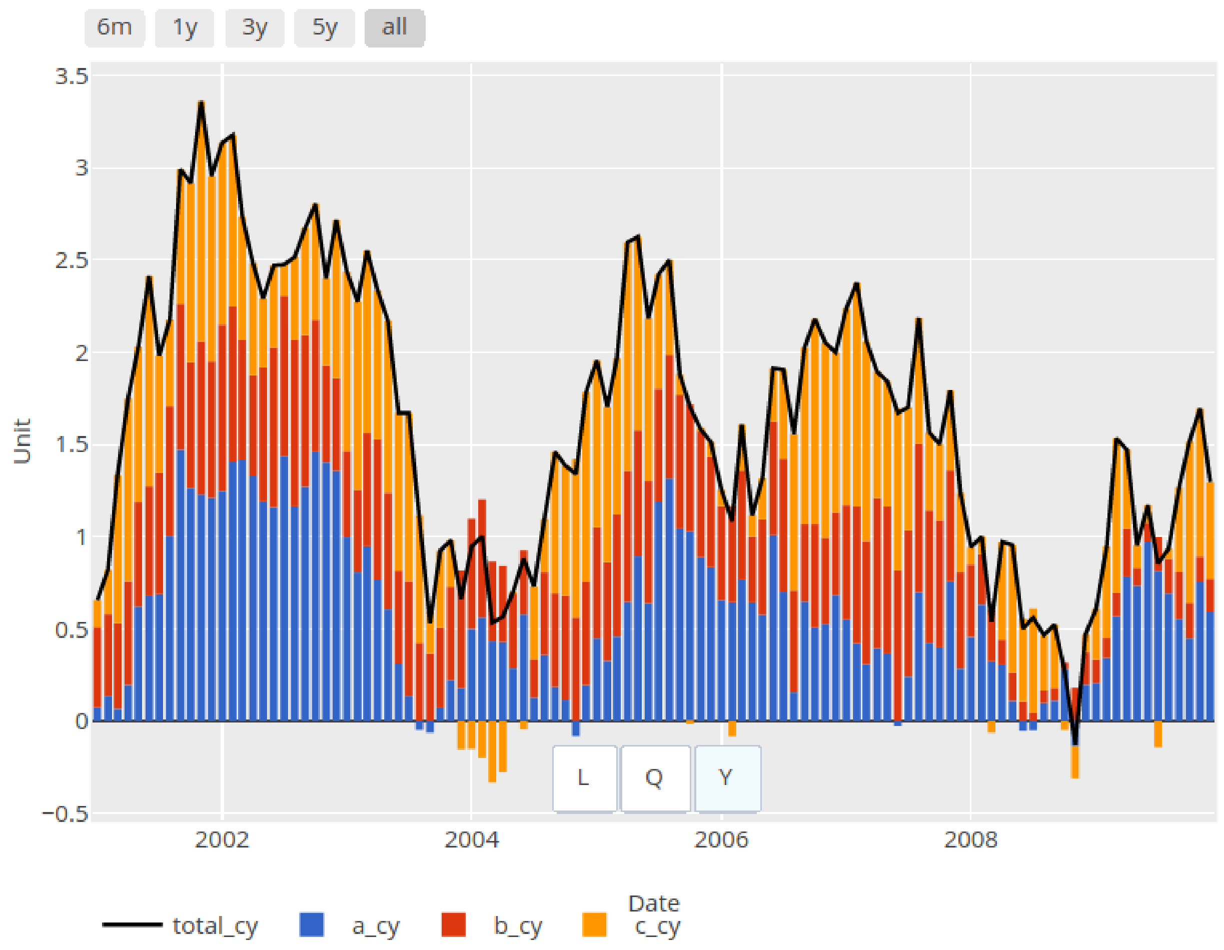
Interactive bar plot

Let's now look at the equivalent function for bar plots. This function is supposed to streamline plotting of data where several series add up to a total. Think GDP and expenditure components, or total exports split up by country of destination. To simulate this, let's add a “total” column to our sample data set. Note that the total must be the first column of the data set.

```
dat$total <- rowSums(dat[, c("a", "b", "c")])
dat <- dat[, c("total", "a", "b", "c")]
```

Let's now call the `plotly_line_bar_growth()` function with this data set:

```
plotly_line_bar_growth(dat, x_lab = "Date", y_lab = "Unit", add_others =
FALSE)
```



Interactive chart

Internally, this function is very similar to the previous one, with one key difference: Instead of calculating *growth rates*, it calculates the *contribution to growth* for each component. For the contributions to add to the change in the total, we need to ensure that our series add up. With real world data this is often not the case, usually for methodological reasons or due to rounding, in which case the function will automatically add a category "others" unless told not to (by setting `add_others` to `FALSE`). Another difference is that this function adds some custom hover text which shows the percentage share of each component on mouseover.

I mostly use this code to construct dashboards that show charts with daily, weekly, monthly, or quarterly frequency data, but it can be used with lower frequencies (annual or semi-annual) by setting `add_q` to `FALSE`.