The article breaks out eight tech stock with attractive dividends: IBM, HPQ, TXN, CSCO, INTC, ORCL, AAPL and MSFT. It also mentions QCOM as an interesting stock to watch. We’ll piggyback on the substance of the article and visualize the dividend history of those nine tickers.

First, let’s load up our packages and create a vector of tickers called barrons\_tickers. We will pass that vector to tq\_get(get = "dividends") just as we did last time. Indeed, we’re not going to do much differently today, but hopefully it’s a nice way to see how previous work can be applied to other situations. Ah, the joys of code that can be reused!

library(tidyverse)

library(tidyquant)

library(janitor)

library(plotly)

barrons\_tickers <-

c("IBM", "HPQ", "TXN", "CSCO", "INTC", "ORCL", "AAPL", "MSFT", "QCOM")

barrons\_dividends <-

barrons\_tickers %>%

tq\_get(get = "dividends")

We can reuse our code from the previous post to quickly visualize these tickers’ dividend histories, along with a detailed tooltip setting in plotly.

ggplotly(

barrons\_dividends %>%

group\_by(symbol) %>%

mutate(info = paste(date,

'  
symbol:', symbol,

'  
div: $', dividends)) %>%

ggplot(aes(x = date,

y = dividends,

color = symbol,

label\_tooltip = info)) +

geom\_point() +

scale\_y\_continuous(labels = scales::dollar) +

scale\_x\_date(breaks = scales::pretty\_breaks(n = 10)) +

labs(x = "", y = "div/share", title = "Nasdaq dividends") +

theme(plot.title = element\_text(hjust = 0.5)),

tooltip = "label\_tooltip"

)

With a handful of stocks, our visualization really tells a nice story. We can more clearly see the four annual payments by each company, and it pops off the chart that IBM has been raising it’s dividend consistently. Not bad for a company that also owns Red Hat.

Let’s move beyond the dividend history and compare the dividend yields for each of these tickers. We’ll grab yesterday’s closing price by calling tq\_get(get = "stock.prices", from = "2019-08-05").

barrons\_price <-

barrons\_tickers %>%

tq\_get(get = "stock.prices", from = "2019-08-05")

Now, we estimate the annual dividend payment by taking the most recent quarterly dividend via slice(n()) and multiplying by four.

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4)

# A tibble: 9 x 4

# Groups: symbol [9]

symbol date dividends total\_div

1 AAPL 2019-08-09 0.77 3.08

2 CSCO 2019-07-03 0.35 1.4

3 HPQ 2019-06-11 0.16 0.64

4 IBM 2019-08-08 1.62 6.48

5 INTC 2019-08-06 0.315 1.26

6 MSFT 2019-08-14 0.46 1.84

7 ORCL 2019-07-16 0.24 0.96

8 QCOM 2019-06-05 0.62 2.48

9 TXN 2019-07-30 0.77 3.08

Next, we use left\_join(barrons\_price, by = "symbol") to add the most recent closing price.

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(barrons\_price, by = "symbol") %>%

head()

# A tibble: 6 x 11

# Groups: symbol [1]

symbol date.x dividends total\_div date.y open high low close

1 AAPL 2019-08-09 0.77 3.08 2019-08-05 198. 199. 193. 193.

2 AAPL 2019-08-09 0.77 3.08 2019-08-06 196. 198. 194. 197

3 AAPL 2019-08-09 0.77 3.08 2019-08-07 195. 200. 194. 199.

4 AAPL 2019-08-09 0.77 3.08 2019-08-08 200. 204. 199. 203.

5 AAPL 2019-08-09 0.77 3.08 2019-08-09 201. 203. 199. 201.

6 AAPL 2019-08-09 0.77 3.08 2019-08-12 200. 202. 199. 200.

# … with 2 more variables: volume , adjusted

That worked, but note how we now have two date columns, called date.x and date.y, since both of our tibbles had a date column before we joined them. In the past we have dealt with that by deleting the duplicate but this time let’s use a select() inside left\_join() to remove the duplicate before joining. The full call is left\_join(select(barrons\_price, -date), by = "symbol").

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(select(barrons\_price, -date), by = "symbol") %>%

head()

# A tibble: 6 x 10

# Groups: symbol [1]

symbol date dividends total\_div open high low close volume

1 AAPL 2019-08-09 0.77 3.08 198. 199. 193. 193. 5.24e7

2 AAPL 2019-08-09 0.77 3.08 196. 198. 194. 197 3.58e7

3 AAPL 2019-08-09 0.77 3.08 195. 200. 194. 199. 3.34e7

4 AAPL 2019-08-09 0.77 3.08 200. 204. 199. 203. 2.70e7

5 AAPL 2019-08-09 0.77 3.08 201. 203. 199. 201. 2.46e7

6 AAPL 2019-08-09 0.77 3.08 200. 202. 199. 200. 2.25e7

# … with 1 more variable: adjusted

Now, we calculate the yield with mutate(yield = total\_div/close).

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(barrons\_price, by = "symbol") %>%

select(symbol, total\_div, close) %>%

mutate(yield = total\_div/close)

# A tibble: 81 x 4

# Groups: symbol [9]

symbol total\_div close yield

1 AAPL 3.08 193. 0.0159

2 AAPL 3.08 197 0.0156

3 AAPL 3.08 199. 0.0155

4 AAPL 3.08 203. 0.0151

5 AAPL 3.08 201. 0.0153

6 AAPL 3.08 200. 0.0154

7 AAPL 3.08 209. 0.0147

8 AAPL 3.08 203. 0.0152

9 AAPL 3.08 202. 0.0153

10 CSCO 1.4 51.4 0.0273

# … with 71 more rows

We can plot the dividend yields as bar heights using geom\_col().

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(barrons\_price, by = "symbol") %>%

select(symbol, total\_div, close) %>%

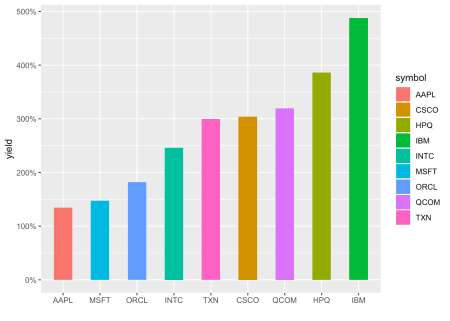
mutate(yield = total\_div/close) %>%

ggplot(aes(x = reorder(symbol, yield), y = yield, fill = symbol)) +

geom\_col(width = .5) +

labs(x = "") +

scale\_y\_continuous(labels = scales::percent)



We could wrap this up with a call to plotly, but let’s totally change directions and add some animation. Animate a chart? That sounds really hard, I guess we’ll need to loop through the dates and add dots as we go. A lot of work and who has the time…wait…boom…gganimate to the rescue!

The gganimate package makes this so painless it’s a shame. We add transition\_reveal(date) to the end of the code flow, and that’s it! Well, not quite; on my machine, I needed to load the gifski and png packages before any of this works, but then we’re good to go.

library(gganimate)

library(gifski)

library(png)

barrons\_dividends %>%

group\_by(symbol) %>%

ggplot(aes(x = date,

y = dividends,

color = symbol)) +

geom\_point() +

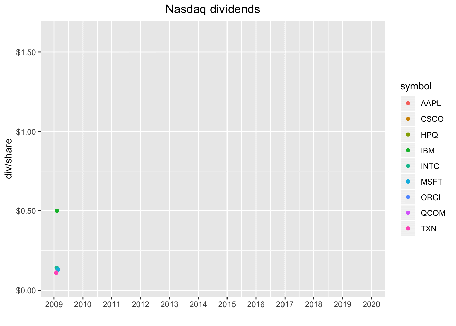
scale\_y\_continuous(labels = scales::dollar) +

scale\_x\_date(breaks = scales::pretty\_breaks(n = 10)) +

labs(x = "", y = "div/share", title = "Nasdaq dividends") +

theme(plot.title = element\_text(hjust = 0.5)) +

transition\_reveal(date)



Nice!

What about animating our chart that shows the dividend yield as bar heights? Well, we can’t reveal by date here, so we use transition\_states(symbol).

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(select(barrons\_price, -date), by = "symbol") %>%

select(symbol, total\_div, close) %>%

mutate(yield = total\_div/close) %>%

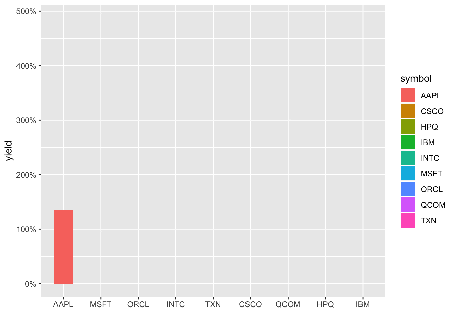
ggplot(aes(x = reorder(symbol, yield), y = yield, fill = symbol)) +

geom\_col(width = .5) +

labs(x = "") +

scale\_y\_continuous(labels = scales::percent) +

transition\_states(symbol)



Ah, not quite perfect – notice the chart doesn’t respect the reorder in our aes(), so they appear in alphabetical order and each column disappears as the next one appears. Let’s use shadow\_mark() to keep the previous bar and attempt to reorder the images with arrange().

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(select(barrons\_price, -date), by = "symbol") %>%

select(symbol, total\_div, close) %>%

mutate(yield = total\_div/close) %>%

arrange(yield) %>%

ggplot(aes(x = reorder(symbol, yield), y = yield, fill = symbol)) +

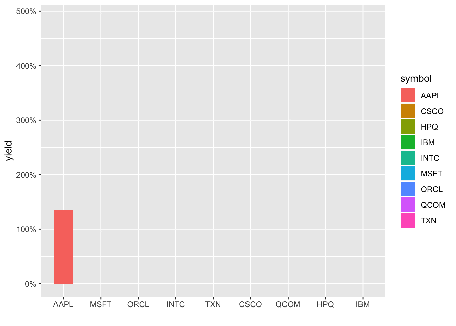
geom\_col(width = .5) +

labs(x = "") +

scale\_y\_continuous(labels = scales::percent) +

transition\_states(symbol, wrap = FALSE) +

shadow\_mark()



It is still not respecting the new order and defaulting to alphabetical. Let’s hard-code that reordering by converting symbol to a factor, ordered by yield. And that means a foray into the forcats package and fct\_reorder(). Note we need to ungroup() first since symbol is our grouping column and then can call symbol\_fct = forcats::as\_factor(symbol) %>% fct\_reorder(yield). I also think it would be a little more dramatic to remove the x-axis labels and have the ticker names appear on the chart.

barrons\_dividends %>%

group\_by(symbol) %>%

slice(n()) %>%

mutate(total\_div = dividends \* 4) %>%

left\_join(select(barrons\_price, -date), by = "symbol") %>%

select(symbol, total\_div, close) %>%

mutate(yield = total\_div/close) %>%

ungroup() %>%

mutate(symbol\_fct = forcats::as\_factor(symbol) %>% fct\_reorder(yield)) %>%

ggplot(aes(x = symbol\_fct, y = yield, fill = symbol\_fct)) +

geom\_col(width = .5) +

geom\_label(aes(label = symbol, y = yield), nudge\_y = .03) +

labs(x = "") +

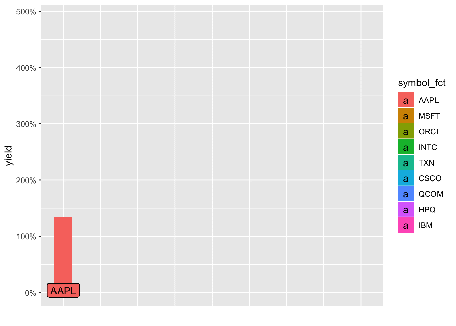
scale\_y\_continuous(labels = scales::percent) +

theme(axis.text.x = element\_blank(),

axis.ticks.x = element\_blank()) +

transition\_states(symbol\_fct, wrap = FALSE) +

shadow\_mark()



Creating and loading those animated gifs takes some time, about 10-30 seconds each on my RStudio Server Pro instance. Plus, it’s totally fair to quibble that these animations haven’t added any new substance to the charts, they just look cool (R plots can be cool, right?). But if you’ve read this far (thanks!), I might as well subject you to my rant about visualization and communication being just-as-if-not-more important than analytical or statistical findings. Most of the consumers of our work are really busy and we’re lucky if they spend two minutes glancing at whatever findings we put in front of them. We don’t have long to grab their attention and communicate our message. If an animation helps us, it’s worth spending the extra time on it, even though we were actually ‘done’ with this job many lines of code ago.

Thanks for reading and see you next time!