we explored IPOs and IPO returns by sector and year since 2004. Today, let’s investigate how portfolios formed with those IPOs have performed. We will need to grab the price histories of the tickers, then form portfolios, then calculate their performance, and then rank those performances in some way.

Since there are several hundred IPOs for which we need to pull returns data, today’s post will be a bit data intensive. If you don’t want to tax your machine too much, you can always subset down the data before running the script. With that, let’s do this!

I won’t go through the code again in detail since we covered it last time, but our first step is to get the tickers and IPO years for a large universe of companies. Here is the code to retrieve the tickers for all companies that went public between 2004 and 2014 – *and have survived* (meaning today’s analysis is going to have serious survivor bias).

nasdaq <-

tq\_exchange("NASDAQ")

amex <-

tq\_exchange("AMEX")

nyse <-

tq\_exchange("NYSE")

company\_ipo\_sector <-

nasdaq %>%

bind\_rows(amex) %>%

bind\_rows(nyse) %>%

select(symbol, company, ipo.year, sector) %>%

filter(![is.na](http://is.na)(ipo.year) & ipo.year < 2015)

And here is what we get:

company\_ipo\_sector %>%

head()

# A tibble: 6 x 4

symbol company ipo.year sector

1 PIH 1347 Property Insurance Holdings, Inc. 2014 Finance

2 FLWS 1-800 [FLOWERS.COM](http://FLOWERS.COM), Inc. 1999 Consumer Services

3 VNET 21Vianet Group, Inc. 2011 Technology

4 TWOU 2U, Inc. 2014 Technology

5 JOBS 51job, Inc. 2004 Technology

6 ABIL Ability Inc. 2014 Capital Goods

There are about 1,830 tickers in this data frame and it conveniently includes a column called ipo.year, which we’ll make us of later in this post. Our next step is to download the price histories of these tickers from tiingo.  
We’ll need to divide our tickers into smaller subsets and pull in the data in pieces so we don’t hit our tiingo API limits.

tickers\_2004\_2006 <-

company\_ipo\_sector %>%

distinct(company, .keep\_all = TRUE) %>%

filter(![is.na](http://is.na)(sector) & between(ipo.year, 2004, 2006)) %>%

group\_by(ipo.year) %>%

pull(symbol)

tickers\_2007\_2009 <-

company\_ipo\_sector %>%

distinct(company, .keep\_all = TRUE) %>%

filter(![is.na](http://is.na)(sector) & between(ipo.year, 2007, 2009)) %>%

group\_by(ipo.year) %>%

pull(symbol)

tickers\_2010\_2012 <-

company\_ipo\_sector %>%

distinct(company, .keep\_all = TRUE) %>%

filter(![is.na](http://is.na)(sector) & between(ipo.year, 2010, 2012)) %>%

group\_by(ipo.year) %>%

pull(symbol)

tickers\_2013\_2014 <-

company\_ipo\_sector %>%

distinct(company, .keep\_all = TRUE) %>%

filter(![is.na](http://is.na)(sector) & between(ipo.year, 2013, 2014)) %>%

group\_by(ipo.year) %>%

pull(symbol)

We now have 4 sets of tickers. Let’s pass them to the riingo function one at a time and set the resample\_frequency to monthly, so we’re pulling in 12 prices per year per company instead of ~252.

prices\_riingo\_2004\_2006 <-

tickers\_2004\_2006 %>%

riingo\_prices(start\_date = "2004-01-01", resample\_frequency = "monthly") %>%

group\_by(ticker)

prices\_riingo\_2007\_2009 <-

tickers\_2007\_2009 %>%

riingo\_prices(start\_date = "2007-01-01", resample\_frequency = "monthly") %>%

group\_by(ticker)

prices\_riingo\_2010\_2012 <-

tickers\_2010\_2012 %>%

riingo\_prices(start\_date = "2010-01-01", resample\_frequency = "monthly") %>%

group\_by(ticker)

prices\_riingo\_2013\_2014 <-

tickers\_2013\_2014 %>%

riingo\_prices(start\_date = "2013-01-01", resample\_frequency = "monthly") %>%

group\_by(ticker)

prices\_riingo\_full <-

prices\_riingo\_2004\_2006 %>%

bind\_rows(prices\_riingo\_2007\_2009) %>%

bind\_rows(prices\_riingo\_2010\_2012) %>%

bind\_rows(prices\_riingo\_2013\_2014)

We have our prices, now let’s add a column of monthly returns with a call to mutate(monthly\_returns = close/lag(close) - 1).

prices\_riingo\_full <-

prices\_riingo\_full %>%

group\_by(ticker) %>%

mutate(monthly\_returns = close/lag(close) - 1)

One more step, I won’t bore you with the details but after a lot of slogging, I discovered a few errors in my data, where the ipo.year came after the first year in which the ticker was traded. That’s probably because of some equities being relisted on other exchanges or changing ticker names, but suffice it to say, it jacked up my portfolio analysis! I’m going to remove those problematic tickers with the code below. First, we filter(date == min(date)) to get the first day for which we have returns. Then we isolate the year of that first trade with mutate(first\_trade = year(date)). If that year doesn’t match ipo.year, we pull() the ticker.

tickers\_remove\_mismatch\_dates <-

prices\_riingo\_full %>%

left\_join(company\_ipo\_sector, by = c("ticker" = "symbol")) %>%

filter(date == min(date)) %>%

select(ticker, date, ipo.year) %>%

mutate(first\_trade = year(date)) %>%

filter(ipo.year != first\_trade) %>%

pull(ticker)

We now have a vector of the tickers to remove, but we haven’t removed them yet. I separated these two steps so I could take a peak at those tickers first, and maybe come back to later to address this issue in a better way.

Let’s go ahead and remove them (I’m going to remove PGTI also, it was showing up duplicate downstream in my code).

prices\_riingo\_full <-

prices\_riingo\_full %>%

filter(!(ticker %in% tickers\_remove\_mismatch\_dates) & ticker != 'PGTI')

A final step before saving: we’ll left\_join() the company\_ipo\_sector tibble so that we have company name and ipo.year as part of this object.

prices\_riingo\_full <-

prices\_riingo\_full %>%

left\_join(company\_ipo\_sector, by = c("ticker" = "symbol"))

To avoid going through that process each time we revisit this project, we can save this price/returns data as a .RDS file.

write\_rds(prices\_riingo\_full, "prices\_riingo\_full.RDS")

But, another exciting way to save and share data is the new pins package (see intro to pins [here](https://blog.rstudio.com/2019/09/09/pin-discover-and-share-resources/)). Pins is a package that makes it easy to save and share data. I’m going to use [RStudio Connect](https://rstudio.com/products/connect/) for that, but you could use [GitHub](https://github.com/rstudio/pins) or [kaggle](https://rstudio.github.io/pins/articles/boards-kaggle) as well.

First, we register our Connect server.

library(pins)

pins::board\_register(server = "path to your RStudio Connect server here", key = "your API key here", board = "rsconnect")

Now, we’re ready to pin this data up to Connect where we, or any of our colleagues, could pull down this data in the future and take advantage of all the gymnastics we just performed to get this in shape.

pins::pin(

x = prices\_riingo\_full,

name = "ipo\_riingo\_prices\_pins",

description = "Monthly price history of IPOs from 2004 - 2014.",

board = "rsconnect"

)

For the curious, here’s how to retrieve that data from RStudio Connect.

ipo\_riingo\_prices\_pins <-

pins::pin\_get(name = "ipo\_riingo\_prices\_pins",

board = "rsconnect")

Now, back to our ultimate goal: we want to see how portfolios that allocated dollars to IPOs each year would have performed. That is, we’ll build a portfolio for 2004, 2005…through to 2014, 11 portfolios consisting solely of the companies that IPO’d in a specific year. Obviously, we are dealing with selection bias, meaning if we had implemented this strategy, we’d have invested in lots of companies that went to zero and delisted. Those dead companies didn’t make it to today’s project because we pulled the tickers of companies currently listed on the AMEX, NYSE and NASDAQ. So don’t try this at home and nothing in this post is investment advice.

Let’s start by building a portfolio of stocks that IPO’d in 2006, assuming we invest equally in each company. We’ll select() a few columns needed for the analysis and then filter(ipo.year == 2006). Next we use the built-in tq\_portfolio() function from tidyquant wherein we specify the assets\_col as our tickers, the returns\_col as monthly\_returns and paste together a name that incorporates 2006. We also set it to rebalance\_on = "months" so that each month we make sure to maintain an equal weighting to each company. We do not need to specify a weights argument because equal weighting is the default.

prices\_riingo\_full %>%

select(ticker, date, monthly\_returns, ipo.year) %>%

filter(ipo.year == 2006) %>%

group\_by(ticker) %>%

filter(![is.na](http://is.na)(monthly\_returns)) %>%

tq\_portfolio(assets\_col = ticker,

returns\_col = monthly\_returns,

col\_rename = paste(2006, "\_port\_returns", sep = ""),

wealth.index = F,

rebalance\_on = "months") %>%

tail()

# A tibble: 6 x 2

date `2006\_port\_returns`

1 2019-05-31 00:00:00 0.261

2 2019-06-28 00:00:00 0.0533

3 2019-07-31 00:00:00 0.0346

4 2019-08-30 00:00:00 -0.00789

5 2019-09-30 00:00:00 -0.00539

6 2019-10-31 00:00:00 0.0233

# slice(1:3, (n()-3):n())

That code works for 2006, but now we want to scale it for more general use. We can turn the code flow into a function that takes year as an argument and then apply the function to all of our IPO returns on a year-by-year basis. I’m also going to add a second argument, show\_growth, that can be set to TRUE or FALSE if we wish output the growth of a dollar instead of monthly returns.

ipo\_by\_year\_portfolios <- function(year, show\_growth = F){

prices\_riingo\_full %>%

select(ticker, date, monthly\_returns, ipo.year) %>%

filter(ipo.year == year) %>%

tq\_portfolio(assets\_col = ticker,

returns\_col = monthly\_returns,

col\_rename = paste(year, "\_port\_returns", sep = ""),

wealth.index = show\_growth,

rebalance\_on = "months")

}

Let’s test that function on 2006 and confirm we get the same result.

ipo\_by\_year\_portfolios(2006) %>%

tail()

# A tibble: 6 x 2

date `2006\_port\_returns`

1 2019-05-31 00:00:00 0.261

2 2019-06-28 00:00:00 0.0533

3 2019-07-31 00:00:00 0.0346

4 2019-08-30 00:00:00 -0.00789

5 2019-09-30 00:00:00 -0.00539

6 2019-10-31 00:00:00 0.0233

And now, we want to apply that function to every year in our data set and mash the results together into one data frame.

We’ll start by creating a vector of years from 2004 to 2014.

years\_numeric <- seq(2004, 2014, by = 1)

And now, we map() our function across that vector of years. If we just ran map(years\_numeric, ipo\_by\_year\_portfolios), it would result in 11 lists of portfolio returns, one for each year of our IPOs. I want a data frame with all the results stored together. Let’s use reduce(left\_join) for that task.

returns\_each\_year\_ipo\_portfolios <-

map(years\_numeric, ipo\_by\_year\_portfolios) %>%

reduce(left\_join)

returns\_each\_year\_ipo\_portfolios %>%

head()

# A tibble: 6 x 12

date `2004\_port\_retu… `2005\_port\_retu… `2006\_port\_retu…

1 2004-02-27 00:00:00 0 NA NA

2 2004-03-31 00:00:00 -0.00296 NA NA

3 2004-04-30 00:00:00 -0.00141 NA NA

4 2004-05-31 00:00:00 0.00703 NA NA

5 2004-06-30 00:00:00 -0.0194 NA NA

6 2004-07-30 00:00:00 -0.0472 NA NA

# … with 8 more variables: `2007\_port\_returns` ,

# `2008\_port\_returns` , `2009\_port\_returns` ,

# `2010\_port\_returns` , `2011\_port\_returns` ,

# `2012\_port\_returns` , `2013\_port\_returns` ,

# `2014\_port\_returns`

We now have one data frame with a date column that runs from the beginning of 2004, the first year for which we pulled in IPO tickers, to the today’s date, the last date for which we have returns for these companies. Then we have one column with the returns for each yearly portfolio constructed by equal weighting each company that IPO’d from 2004 to 2014. Since each column represents a year, we start out with values for the 2004 column but the rest have NA for all of 2004. Then in 2005, the 2005 portfolio springs into existence, but the rest of the columns have NA. And then each year another column gets values as new companies IPO in those years.

We constructed this so that each year we allocated fresh money to the new IPOs, equal weighted, with no regard to the past or any other market conditions. We didn’t have to do any of that, of course. We could use a very similar construction to tweak our existing portfolio based on what happened last year, or look at the past performance of basket of funds in a sector and then allocate to certain tickers going forward. In that case, we would map our function to first create a signal year-by-year, then run our allocation algorithm based on the signal from the previous year. Also note that we’re doing something a bit unusual by constructing each portfolio by year and then doing nothing beyond a monthly rebalance: not quite set it and forget it, but pretty close.

Let’s dig in to our new portfolio returns object by calculating the Sharpe Ratio for each of these IPO portfolios. We currently have our data in wide, decidedly un-tidy format so let’s pivot\_longer() this data to make it tidy (long time readers or dplyers might note that we would have used gather() for this a few months ago – pivot\_longer() has replaced gather() in the latest release of dplyr). We are treating each yearly portfolio as a discrete return stream so we can goup\_by() the portfolio\_by\_year column. From there we call tq\_performance(), same as if these return streams were from ETFs or any other asset. Our result should be one Sharpe ratio for each year, or 11 Sharpe Ratios.

port\_sharpes <-

returns\_each\_year\_ipo\_portfolios %>%

pivot\_longer(-date, names\_to = "portfolio\_by\_year", values\_to = "monthly\_return") %>%

group\_by(portfolio\_by\_year) %>%

arrange(portfolio\_by\_year, date) %>%

filter(![is.na](http://is.na)(monthly\_return)) %>%

tq\_performance(Ra = monthly\_return,

performance\_fun = SharpeRatio,

Rf = 0,

FUN= "StdDev") %>%

`colnames<-`(c("portfolio\_by\_year", "port\_sharpe"))%>%

add\_column(year = years\_numeric)

port\_sharpes

# A tibble: 11 x 3

# Groups: portfolio\_by\_year [11]

portfolio\_by\_year port\_sharpe year

1 2004\_port\_returns 0.234 2004

2 2005\_port\_returns 0.192 2005

3 2006\_port\_returns 0.249 2006

4 2007\_port\_returns 0.190 2007

5 2008\_port\_returns 0.142 2008

6 2009\_port\_returns 0.220 2009

7 2010\_port\_returns 0.279 2010

8 2011\_port\_returns 0.152 2011

9 2012\_port\_returns 0.309 2012

10 2013\_port\_returns 0.182 2013

11 2014\_port\_returns 0.218 2014

Let’s port this data to plotly and create a bar chart, where the height of each bar is the portfolio’s Sharpe Ratio.

(

port\_sharpes %>%

ggplot(aes(x = year, y = port\_sharpe, fill = portfolio\_by\_year)) +

geom\_col(width = .4) +

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

labs(x = "", y = "sharpe by year", title = "IPO Portfolio Sharpe Ratios") +

theme\_minimal()

) %>%

ggplotly()

Here is the exact same data, except displayed as a point chart. Notice how defaults to a slightly different scale, with the y-axis not extending all the way to 0. That makes the poor performance of 2008 seem even more dramatic. Were the companies that IPO’d in 2008 all dogs, or is it just a bad idea to invest in IPOs when the market is in a sharp correction because of a financial crisis?

(

port\_sharpes %>%

ggplot(aes(x = year, y = port\_sharpe, fill = portfolio\_by\_year)) +

geom\_point(size = 2) +

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

labs(x = "", y = "sharpe by year", title = "IPO Portfolio Sharpe Ratios") +

theme\_minimal()

) %>%

ggplotly()

If you’re an avid reader of these posts, it won’t surprise you to learn that we’re going to calculate the rolling Sharpe Ratio as well. We can use the same code as last time and the blazing fast roll package.

library(roll)

returns\_each\_year\_ipo\_portfolios %>%

pivot\_longer(-date, names\_to = "portfolio\_by\_year", values\_to = "monthly\_return") %>%

group\_by(portfolio\_by\_year) %>%

arrange(portfolio\_by\_year, date) %>%

mutate( rolling\_mean = roll\_mean(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sd = roll\_sd(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sharpe = rolling\_mean/rolling\_sd) %>%

na.omit() %>%

head()

# A tibble: 6 x 6

# Groups: portfolio\_by\_year [1]

date portfolio\_by\_ye… monthly\_return rolling\_mean

1 2004-07-30 00:00:00 2004\_port\_retur… -0.0472 -0.0107

2 2004-08-31 00:00:00 2004\_port\_retur… 0.00956 -0.00906

3 2004-09-30 00:00:00 2004\_port\_retur… 0.0493 -0.000354

4 2004-10-29 00:00:00 2004\_port\_retur… 0.00395 0.000539

5 2004-11-30 00:00:00 2004\_port\_retur… 0.131 0.0211

6 2004-12-31 00:00:00 2004\_port\_retur… 0.0737 0.0367

# … with 2 more variables: rolling\_sd , rolling\_sharpe

And let’s port these rolling calculations straight to plotly.

(

returns\_each\_year\_ipo\_portfolios %>%

pivot\_longer(-date, names\_to = "portfolio\_by\_year", values\_to = "monthly\_return") %>%

group\_by(portfolio\_by\_year) %>%

arrange(portfolio\_by\_year, date) %>%

mutate( rolling\_mean = roll\_mean(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sd = roll\_sd(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sharpe = rolling\_mean/rolling\_sd,

date = ymd(date)) %>%

na.omit() %>%

ggplot(aes(x = date, y = rolling\_sharpe, color = portfolio\_by\_year)) +

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

geom\_line() +

theme\_minimal()

) %>%

ggplotly()

Try double clicking on 2004\_port\_returns and then clicking on 2008\_port\_returns to isolate just those two. Here’s what you should see:

(

returns\_each\_year\_ipo\_portfolios %>%

pivot\_longer(-date, names\_to = "portfolio\_by\_year", values\_to = "monthly\_return") %>%

group\_by(portfolio\_by\_year) %>%

arrange(portfolio\_by\_year, date) %>%

mutate( rolling\_mean = roll\_mean(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sd = roll\_sd(as.matrix(monthly\_return), 6, complete\_obs = T),

rolling\_sharpe = rolling\_mean/rolling\_sd,

date = ymd(date)) %>%

na.omit() %>%

filter(portfolio\_by\_year %in% c("2004\_port\_returns", "2008\_port\_returns")) %>%

ggplot(aes(x = date, y = rolling\_sharpe, color = portfolio\_by\_year)) +

geom\_line() +

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

geom\_line() +

theme\_minimal()

) %>%

ggplotly()

Recall that the 2004 portfolio has a Sharpe Ratio that is ~2x the Sharpe Ratio of the 2008 portfolio. But, look at how dramatically the 2004 portfolio plummeted through March of 2008. Would you have been able to ride out that storm? I dunno if I would have.

And with that…we’re done!

Next week we’ll add a short post on how to add a benchmark to today’s work, since the performance of these or any portfolio isn’t of much use without a benchmark. Thanks for reading and see you next time!