Today we will continue our portfolio fun by calculating the CAPM beta of our portfolio returns. That will entail fitting a linear model and, when we get to visualization next time, considering the meaning of our results from the perspective of asset returns.

By way of brief background, the Capital Asset Pricing Model (CAPM) is a model, created by William Sharpe, that estimates the return of an asset based on the return of the market and the asset’s linear relationship to the return of the market. That linear relationship is the stock’s beta coefficient, or just good ol’ beta.

CAPM was introduced back in 1964, garnered a Nobel for its creator, and, like many ephocally important theories, has been widely used, updated, criticized, debunked, revived, re-debunked, etc. Fama and French have written that CAPM “is the centerpiece of MBA investment courses. Indeed, it is often the only asset pricing model taught in these courses…[u]nfortunately, the empirical record of the model is poor.”[1](https://rviews.rstudio.com/2018/02/08/capm-beta/#fn1)

With that, we will forge ahead with our analysis because calculating CAPM betas can serve as a nice template for more complex models in a team’s work and sometimes it’s a good idea to start with a simple model, even if it hasn’t stood up to empirical rigor. Plus, it might have been questioned by future research, but it’s still an iconic model that we should learn and love.

We are going to focus on one particular aspect of CAPM: beta. Beta, as we noted above, is the beta coefficient of an asset that results from regressing the returns of that asset on market returns. It captures the linear relationship between the asset/portfolio and the market. For our purposes, it’s a good vehicle for exploring reproducible flows for modeling or regressing our portfolio returns on the market returns. Even if your team dislikes CAPM in favor of more nuanced models, these code flows can serve as a good base for the building of those more complex models.

We are going to be calculating beta in several ways: by-hand (for illustrative purposes), in the xts world with PerformanceAnalytics, in the tidyverse with dplyr, and in the tidyquant world. These seem to be the most popular paradigms for doing financial time series work, and even within a team there can be differing preferences. I don’t think everyone needs to grind through their work using each paradigm, but I do think it’s helpful to be fluent, or, at least, conversant, in the various worlds. If you’re a tidyverse type of person but need to collaborate with an xts or tidyquant enthusiast, it will help if each of you is familiar with the three universes (though at some point ya just have to choose a code flow and get stuff done).

We will be working with and calculating beta for our usual portfolio consisting of:

+ SPY (S&P500 fund) weighted 25%

+ EFA (a non-US equities fund) weighted 25%

+ IJS (a small-cap value fund) weighted 20%

+ EEM (an emerging-mkts fund) weighted 20%

+ AGG (a bond fund) weighted 10%

Before we can calculate beta for that portfolio, we need to find portfolio monthly returns, which was covered below:

Calculating Beta For Portfolio:

A brief interlude on portfolios: a portfolio is a weighted collection of assets (its etymology harkens back to the Latin for “to carry sheets of paper”, which I s’pose made its way to the world of investments because securities used to be sheets of paper). The theoretical reason that rational investors prefer a portfolio to a single asset is that a portfolio can offer a better risk/return trade-off due to low or negative covariance amongst portfolio components.

Back to the task at hand: transform a collection of daily ETF prices into an object of portfolio log returns.

Let’s load up our packages.

**library**(tidyverse)

**library**(tidyquant)

**library**(timetk)

First, we import daily prices for the five ETFs, using getSymbols to grab the data, map(~Ad(get(.))) to select adjusted prices only, and reduce(merge) to mash our five prices into one xts object.

*# The symbols vector holds our tickers.*

symbols <- c("SPY","EFA", "IJS", "EEM","AGG")

*# The prices object will hold our raw price data throughout this book.*

prices <-

getSymbols(symbols, src = 'yahoo', from = "2005-01-01",

auto.assign = TRUE, warnings = FALSE) %>%

map(~Ad(get(.))) %>%

reduce(merge) %>%

`colnames<-`(symbols)

Next, we convert those daily adjusted prices to monthly log returns using two methods. For the first method, we stay in the xts world.

prices\_monthly <- to.monthly(prices, indexAt = "last", OHLC = FALSE)

asset\_returns\_xts <- na.omit(Return.calculate(prices\_monthly, method = "log"))

For the second method, we will head to the tidyverse/tidyquant world. We will convert from xts to tibble using a call to tk\_tbl(preserve\_index = TRUE, rename\_index = "date"). We will add a column for log returns using mutate(returns = (log(returns) - log(lag(returns)))).

*# Tidyverse method, to long, tidy format*

asset\_returns\_long <-

prices %>%

to.monthly(indexAt = "last", OHLC = FALSE) %>%

tk\_tbl(preserve\_index = TRUE, rename\_index = "date") %>%

gather(asset, returns, -date) %>%

group\_by(asset) %>%

mutate(returns = (log(returns) - log(lag(returns))))

Have a peek at both asset return objects.

head(asset\_returns\_xts)

## SPY EFA IJS EEM AGG

## 2005-02-28 0.020688126 0.037150948 0.02860922 0.09241734 -0.003721035

## 2005-03-31 -0.018461970 -0.026583514 -0.02388198 -0.08240681 -0.009790455

## 2005-04-29 -0.018913092 -0.016309073 -0.05255677 -0.01255390 0.017081857

## 2005-05-31 0.031716351 -0.008674664 0.05973603 0.03111818 0.008242118

## 2005-06-30 0.001514103 0.014225362 0.03840792 0.03892299 0.008724056

## 2005-07-29 0.037547542 0.029527397 0.05677107 0.07400825 -0.010408874

head(asset\_returns\_long)

## # A tibble: 6 x 3

## # Groups: asset [1]

## date asset returns

## <date> <chr> <dbl>

## 1 2005-01-31 SPY NA

## 2 2005-02-28 SPY 0.020688126

## 3 2005-03-31 SPY -0.018461970

## 4 2005-04-29 SPY -0.018913092

## 5 2005-05-31 SPY 0.031716351

## 6 2005-06-30 SPY 0.001514103

Do we notice any differences?

First, have a look at the left most part of asset\_returns\_xts, where the date is stored. The asset\_returns\_xts has a date index, not a column. It is accessed via index(asset\_returns\_xts). asset\_returns\_long has a column called “date”, accessed via the $date convention, i.e., asset\_returns\_long$date.

Second, notice the first date observation for January of 2005. asset\_returns\_long contains NA, and asset\_returns\_xts excludes the observation completely. Does it matter? It depends. In a few weeks when we get to the Sortino Ratio, we will see that it can matter quite a bit.

Third, asset\_returns\_xts is in wide format, which in this case means there is a column for each of our assets. This is the format that xts likes, and it’s the format that is easier to read as a human. However, asset\_returns\_long is in long, tidy format so that each variable has its own column. It’s a bit harder to read as human, but the tidyverse wants data in this format.

Now on to constructing a portfolio and calculating returns. To turn these five ETFs into a portfolio we need to assign them weights. Let’s first create a weights vector.

w <- c(0.25, 0.25, 0.20, 0.20, 0.10)

Before we use the weights in our calculations, we will run a quick sanity check in the next code chunk. This might not be necessary with five assets as we have today, but it is good practice because if we had 50 assets, it could save us a lot of grief to catch a mistake early.

*# Make sure the weights line up with assets.*

asset\_weights\_sanity\_check <- tibble(w, symbols)

asset\_weights\_sanity\_check

## # A tibble: 5 x 2

## w symbols

## <dbl> <chr>

## 1 0.25 SPY

## 2 0.25 EFA

## 3 0.20 IJS

## 4 0.20 EEM

## 5 0.10 AGG

Make sure that tibble match up with the portfolio we want to create.

Finally, make sure the weights sum to 100%, or 1. Again, we can eyeball this with five assets, but with 50 assets it would be easier to run the sanity check.

sum(asset\_weights\_sanity\_check$w)

## [1] 1

They sum to 1. Good to go, and on to portfolio returns.

We will start with the textbook equation for the return of a multi-asset portfolio which is:Returnportfolio=W1∗Returnasset1 + W2∗Returnasset2 + W3∗Returnasset3 + W4∗Returnasset4 + W5∗Returnasset5Returnportfolio=W1∗Returnasset1 + W2∗Returnasset2 + W3∗Returnasset3 + W4∗Returnasset4 + W5∗Returnasset5Here’s the LaTeX code for that equation.

*# $$Return\_{portfolio} = W\_{1}\*Return\_{asset1}~+~W\_{2}\*Return\_{asset2}~+~W\_{3}\*Return\_{asset3}~+~W\_{4}\*Return\_{asset4}~+~W\_{5}\*Return\_{asset5}$$*

We ground through the LaTeX; now let’s grind through the R calculation by hand instead of using built-in functions.

First, assign each weight from our w vector to a variable.

Next, assign each asset return stored in asset\_returns\_xts to a variable.

Last, we insert those new variables into the equation.

w\_1 <- w[1]

w\_2 <- w[2]

w\_3 <- w[3]

w\_4 <- w[4]

w\_5 <- w[5]

asset1 <- asset\_returns\_xts[,1]

asset2 <- asset\_returns\_xts[,2]

asset3 <- asset\_returns\_xts[,3]

asset4 <- asset\_returns\_xts[,4]

asset5 <- asset\_returns\_xts[,5]

portfolio\_returns\_byhand <-

(w\_1 \* asset1) +

(w\_2 \* asset2) +

(w\_3 \* asset3) +

(w\_4 \* asset4) +

(w\_5 \* asset5)

names(portfolio\_returns\_byhand) <- "returns"

Our first portfolio returns calculation is now complete and stored as portfolio\_returns\_byhand. From a substantive perspective, we are finished and could head to visualization.

We want to cover more methods, though, so let’s head to to the xts world and the PerformanceAnalytics package. We didn’t explicitly load that package in the setup, because tidyquant imports it for us.

We will use theReturn.portfolio function, which requires two arguments for a portfolio, an xts object of asset returns, and a vector of weights. We have those at hand: asset\_returns\_xts and w. It’s not necessary, but we will set rebalance\_on = "months" so we can confirm it matches our by-hand calculations. Remember, in the by-hand equation, the portfolio weights are fixed, meaning they never change on a month-to-month basis. That is equivalent to re-balancing every month, which in practice would be quite rare.

portfolio\_returns\_xts\_rebalanced\_monthly <-

Return.portfolio(asset\_returns\_xts, weights = w, rebalance\_on = "months") %>%

`colnames<-`("returns")

Next let’s change to a more realistic annual re-balancing and set rebalance\_on = "years". This will change our results so that they no longer match our by-hand calculation, which effectively re-balanced every month (since we hard-coded asset weights to be the same each month).

portfolio\_returns\_xts\_rebalanced\_yearly <-

Return.portfolio(asset\_returns\_xts, weights = w, rebalance\_on = "years") %>%

`colnames<-`("returns")

We can take a peek at our three portfolio objects and see how the annual re-balance made a small but important difference.

head(portfolio\_returns\_byhand)

## returns

## 2005-02-28 0.03829298

## 2005-03-31 -0.03349817

## 2005-04-29 -0.02011949

## 2005-05-31 0.02475548

## 2005-06-30 0.02027345

## 2005-07-29 0.04188371

head(portfolio\_returns\_xts\_rebalanced\_monthly)

## returns

## 2005-02-28 0.03829298

## 2005-03-31 -0.03349817

## 2005-04-29 -0.02011949

## 2005-05-31 0.02475548

## 2005-06-30 0.02027345

## 2005-07-29 0.04188371

head(portfolio\_returns\_xts\_rebalanced\_yearly)

## returns

## 2005-02-28 0.03829298

## 2005-03-31 -0.03418759

## 2005-04-29 -0.02018237

## 2005-05-31 0.02441794

## 2005-06-30 0.02032339

## 2005-07-29 0.04228070

Do you notice where the annual re-balancing starts to show a difference from monthly re-balancing?

As before, we could stop here and have accomplished our substantive task (twice already - by hand and using the built-in function from PerformanceAnalytics), but we want to explore alternate methods in the world of tidyverse/tidyquant. We will use our long, tidy-formatted asset\_returns\_long and convert to portfolio returns using the tq\_portfolio function from tidyquant.

The tq\_portfolio function takes a tibble and then asks for an assets column to group by, a returns column to find return data, and a weights column. It’s a wrapper for Return.portfolio, and thus also accepts the argument rebalance\_on = "months". Since we are re-balancing by months, we should again get a portfolio returns object that matches our two existing objects portfolio\_returns\_byhand and portfolio\_returns\_xts\_rebalanced\_monthly.

portfolio\_returns\_tq\_rebalanced\_monthly <-

asset\_returns\_long %>%

tq\_portfolio(assets\_col = asset,

returns\_col = returns,

weights = w,

col\_rename = "returns",

rebalance\_on = "months")

If we want to re-balance annually, it’s the same code as above, except we set rebalance\_on = "years".

portfolio\_returns\_tq\_rebalanced\_yearly <-

asset\_returns\_long %>%

tq\_portfolio(assets\_col = asset,

returns\_col = returns,

weights = w,

col\_rename = "returns",

rebalance\_on = "years")

We now have two more portfolio returns objects and they are both tidy tibbles. Let’s take a quick look and compare how a tidy tibble of portfolio returns compares to an xts object of portfolio returns.

head(portfolio\_returns\_tq\_rebalanced\_yearly)

## # A tibble: 6 x 2

## date returns

## <date> <dbl>

## 1 2005-01-31 0.00000000

## 2 2005-02-28 0.03829298

## 3 2005-03-31 -0.03418759

## 4 2005-04-29 -0.02018237

## 5 2005-05-31 0.02441794

## 6 2005-06-30 0.02032339

head(portfolio\_returns\_xts\_rebalanced\_yearly)

## returns

## 2005-02-28 0.03829298

## 2005-03-31 -0.03418759

## 2005-04-29 -0.02018237

## 2005-05-31 0.02441794

## 2005-06-30 0.02032339

## 2005-07-29 0.04228070

Again, we can see a discrepancy for January of 2005. Our xts object elides that date completely, while our tibble records it as a 0.00.

Since there is only one column of returns, there is no wide versus long format for the tibble, and it looks almost identical to the xts object. The only difference is the date: the tibble has a column that holds the date that can be accessed with the $ operator, whereas the xts object has a date index, accessed with index.

That’s all for today. The xts and tidyquant object have their own uses and advantages depending on our end goal. Next time we will think about how to visualize portfolio returns, and how the different objects fit into different visualization paradigms.

Calculate Beta For the capital Asset Pricing Model

I won’t go through the logic again but the code is here:

library(tidyquant)

library(tidyverse)

library(timetk)

library(tibbletime)

library(broom)

symbols <- c("SPY","EFA", "IJS", "EEM","AGG")

prices <-

getSymbols(symbols, src = 'yahoo',

from = "2013-01-01",

to = "2017-12-31",

auto.assign = TRUE, warnings = FALSE) %>%

map(~Ad(get(.))) %>%

reduce(merge) %>%

`colnames<-`(symbols)

prices\_monthly <- to.monthly(prices, indexAt = "last", OHLC = FALSE)

asset\_returns\_xts <- na.omit(Return.calculate(prices\_monthly, method = "log"))

w <- c(0.25, 0.25, 0.20, 0.20, 0.10)

portfolio\_returns\_xts\_rebalanced\_monthly <-

Return.portfolio(asset\_returns\_xts, weights = w, rebalance\_on = "months") %>%

`colnames<-`("returns")

asset\_returns\_long <-

prices %>%

to.monthly(indexAt = "last", OHLC = FALSE) %>%

tk\_tbl(preserve\_index = TRUE, rename\_index = "date") %>%

gather(asset, returns, -date) %>%

group\_by(asset) %>%

mutate(returns = (log(returns) - log(lag(returns)))) %>%

na.omit()

portfolio\_returns\_tq\_rebalanced\_monthly <-

asset\_returns\_long %>%

tq\_portfolio(assets\_col = asset,

returns\_col = returns,

weights = w,

col\_rename = "returns",

rebalance\_on = "months")

We will be working with two objects of portfolio returns and one object of our individual asset returns:

+ portfolio\_returns\_xts\_rebalanced\_monthly (an xts of monthly returns)

+ portfolio\_returns\_tq\_rebalanced\_monthly (a tibble of monthly returns)

+ asset\_returns\_long (a tidy tibble of monthly returns for those 5 assets above)

Let’s get to it.

**CAPM and Market Returns**

Our first step is to make a choice about which asset to use as a proxy for the market return, and we will go with the SPY ETF, effectively treating the S&P 500 as the market. That’s going to make our calculations substantively uninteresting because (1) SPY is 25% of our portfolio and (2) we have chosen assets and a time period (2013 – 2017) in which correlations with SPY have been high. It will offer one benefit in the way of a sanity check, which I’ll note below. With those caveats in mind, feel free to choose a different asset for the market return and try to reproduce this work, or construct a different portfolio that does not include SPY.

Let’s calculate our market return for SPY and save it as market\_return\_xts. Note the start date is “2013-01-01” and the end date is “2017-12-31”, so we will be working with five years of returns.

spy\_monthly\_xts <-

getSymbols("SPY",

src = 'yahoo',

from = "2013-01-01",

to = "2017-12-31",

auto.assign = TRUE,

warnings = FALSE) %>%

map(~Ad(get(.))) %>%

reduce(merge) %>%

`colnames<-`("SPY") %>%

to.monthly(indexAt = "last", OHLC = FALSE)

market\_returns\_xts <-

Return.calculate(spy\_monthly\_xts, method = "log") %>%

na.omit()

We will also want a data.frame object of market returns, and will convert the xts object using tk\_tbl(preserve\_index = TRUE, rename\_index = "date") from the timetk package.

market\_returns\_tidy <-

market\_returns\_xts %>%

tk\_tbl(preserve\_index = TRUE, rename\_index = "date") %>%

na.omit() %>%

select(date, returns = SPY)

head(market\_returns\_tidy)

## # A tibble: 6 x 2

## date returns

##

## 1 2013-02-28 0.01267837

## 2 2013-03-28 0.03726809

## 3 2013-04-30 0.01903021

## 4 2013-05-31 0.02333503

## 5 2013-06-28 -0.01343411

## 6 2013-07-31 0.05038580

We have a market\_returns\_tidy object. Let’s make sure it’s periodicity aligns perfectly with our portfolio returns periodicity

portfolio\_returns\_tq\_rebalanced\_monthly %>%

mutate(market\_returns = market\_returns\_tidy$returns) %>%

head()

## # A tibble: 6 x 3

## date returns market\_returns

##

## 1 2013-02-28 -0.0008696129 0.01267837

## 2 2013-03-28 0.0186624381 0.03726809

## 3 2013-04-30 0.0206248856 0.01903021

## 4 2013-05-31 -0.0053529694 0.02333503

## 5 2013-06-28 -0.0229487618 -0.01343411

## 6 2013-07-31 0.0411705818 0.05038580

Note that if the periodicities did not align, mutate() would have thrown an error in the code chunk above.

**Calculating CAPM Beta**

There are several R code flows to calculate portfolio beta but first let’s have a look at the equation.

$${\beta}\_{portfolio} = cov(R\_p, R\_m)/\sigma\_m $$

\[{\beta}\_{portfolio} = cov(R\_p, R\_m)/\sigma\_m \]

Portfolio beta is equal to the covariance of the portfolio returns and market returns, divided by the variance of market returns.

We can calculate the numerator, or covariance of portfolio and market returns, with cov(portfolio\_returns\_xts\_rebalanced\_monthly, market\_returns\_tidy$returns) and the denominator with var(market\_return$returns).

Our portfolio beta is equal to:

cov(portfolio\_returns\_xts\_rebalanced\_monthly,market\_returns\_tidy$returns)/var(market\_returns\_tidy$returns)

## [,1]

## returns 0.9010689

That beta is quite near to 1 as we were expecting – after all, SPY is a big part of this portfolio.

We can also calculate portfolio beta by finding the beta of each of our assets and then multiplying by asset weights. That is, another equation for portfolio beta is the weighted sum of the asset betas:

$${\beta}\_{portfolio} ={\sum\_{i=1}^n}W \_i~{\beta}\_i $$

\[{\beta}\_{portfolio} ={\sum\_{i=1}^n}W \_i~{\beta}\_i \]

To use that method with R, we first find the beta for each of our assets, and this gives us an opportunity to introduce a code flow for running regression analysis.

We need to regress each of our individual asset returns on the market return. We could do that for asset 1 with lm(asset\_return\_1 ~ market\_returns\_tidy$returns), and then again for asset 2 with lm(asset\_return\_2 ~ market\_returns\_tidy$returns), etc. for all five of our assets. But if we had a 50-asset portfolio, that would be impractical. Instead let’s write a code flow and use map() to regress all of our assets and calculate betas with one call.

We will start with our asset\_returns\_long tidy data frame and will then run nest(-asset).

beta\_assets <-

asset\_returns\_long %>%

na.omit() %>%

nest(-asset)

beta\_assets

## # A tibble: 5 x 2

## asset data

##

## 1 SPY

## 2 EFA

## 3 IJS

## 4 EEM

## 5 AGG

That nest(-asset) changed our data frame so that there are two columns: one called asset that holds our asset name and one called data that holds a list of returns for each asset. We have now ‘nested’ a list of returns within a column.

Now we can use map() to apply a function to each of those nested lists and store the results in a new column via the mutate() function. The whole piped command is mutate(model = map(data, ~ lm(returns ~ market\_returns\_tidy$returns, data = .)))

beta\_assets <-

asset\_returns\_long %>%

na.omit() %>%

nest(-asset) %>%

mutate(model = map(data, ~ lm(returns ~ market\_returns\_tidy$returns, data = .)))

beta\_assets

## # A tibble: 5 x 3

## asset data model

##

## 1 SPY

## 2 EFA

## 3 IJS

## 4 EEM

## 5 AGG

We now have three columns: asset which we had before, data which we had before, and model which we just added. The model column holds the results of the regression lm(returns ~ market\_returns\_tidy$returns, data = .) that we ran for each of our assets. Those results are a beta and an intercept for each of our assets, but they are not in a great format for presentation to others, or even readability by ourselves.

Let’s tidy up our results with the tidy() function from the broom package. We want to apply that function to our model column and will use the mutate() and map() combination again. The complete call is to mutate(model = map(model, tidy)).

beta\_assets <-

asset\_returns\_long %>%

na.omit() %>%

nest(-asset) %>%

mutate(model = map(data, ~ lm(returns ~ market\_returns\_tidy$returns, data = .))) %>%

mutate(model = map(model, tidy))

beta\_assets

## # A tibble: 5 x 3

## asset data model

##

## 1 SPY

## 2 EFA

## 3 IJS

## 4 EEM

## 5 AGG

We are getting close now, but the model column holds nested data frames. Have a look and see that they are nicely formatted data frames:

beta\_assets$model

## [[1]]

## term estimate std.error statistic

## 1 (Intercept) 1.806734e-18 1.136381e-18 1.589902e+00

## 2 market\_returns\_tidy$returns 1.000000e+00 3.899949e-17 2.564136e+16

## p.value

## 1 0.1173886

## 2 0.0000000

##

## [[2]]

## term estimate std.error statistic

## 1 (Intercept) -0.005427739 0.002908978 -1.865858

## 2 market\_returns\_tidy$returns 0.945476441 0.099833320 9.470550

## p.value

## 1 6.720983e-02

## 2 2.656258e-13

##

## [[3]]

## term estimate std.error statistic

## 1 (Intercept) -0.001693293 0.003639218 -0.4652905

## 2 market\_returns\_tidy$returns 1.120583127 0.124894444 8.9722416

## p.value

## 1 6.434963e-01

## 2 1.713903e-12

##

## [[4]]

## term estimate std.error statistic

## 1 (Intercept) -0.00811518 0.004785237 -1.695878

## 2 market\_returns\_tidy$returns 0.95562574 0.164224722 5.819013

## p.value

## 1 9.536495e-02

## 2 2.841106e-07

##

## [[5]]

## term estimate std.error statistic

## 1 (Intercept) 0.001888304 0.001230331 1.5347933

## 2 market\_returns\_tidy$returns -0.005419543 0.042223776 -0.1283529

## p.value

## 1 0.1303671

## 2 0.8983215

Still, I don’t like to end up with nested data frames, so let’s unnest() that model column.

beta\_assets <-

asset\_returns\_long %>%

na.omit() %>%

nest(-asset) %>%

mutate(model = map(data, ~ lm(returns ~ market\_returns\_tidy$returns, data = .))) %>%

mutate(model = map(model, tidy)) %>%

unnest(model)

beta\_assets

## # A tibble: 10 x 6

## asset term estimate std.error

##

## 1 SPY (Intercept) 1.806734e-18 1.136381e-18

## 2 SPY market\_returns\_tidy$returns 1.000000e+00 3.899949e-17

## 3 EFA (Intercept) -5.427739e-03 2.908978e-03

## 4 EFA market\_returns\_tidy$returns 9.454764e-01 9.983332e-02

## 5 IJS (Intercept) -1.693293e-03 3.639218e-03

## 6 IJS market\_returns\_tidy$returns 1.120583e+00 1.248944e-01

## 7 EEM (Intercept) -8.115180e-03 4.785237e-03

## 8 EEM market\_returns\_tidy$returns 9.556257e-01 1.642247e-01

## 9 AGG (Intercept) 1.888304e-03 1.230331e-03

## 10 AGG market\_returns\_tidy$returns -5.419543e-03 4.222378e-02

## # ... with 2 more variables: statistic , p.value

Now that looks human-readable and presentable. We will do one further cleanup and get rid of the intercept results, since we are isolating the betas.

beta\_assets <-

asset\_returns\_long %>%

na.omit() %>%

nest(-asset) %>%

mutate(model = map(data, ~ lm(returns ~ market\_returns\_tidy$returns, data = .))) %>%

unnest(model %>% map(tidy)) %>%

filter(term == "market\_returns\_tidy$returns") %>%

select(-term)

beta\_assets

## # A tibble: 5 x 5

## asset estimate std.error statistic p.value

##

## 1 SPY 1.000000000 3.899949e-17 2.564136e+16 0.000000e+00

## 2 EFA 0.945476441 9.983332e-02 9.470550e+00 2.656258e-13

## 3 IJS 1.120583127 1.248944e-01 8.972242e+00 1.713903e-12

## 4 EEM 0.955625743 1.642247e-01 5.819013e+00 2.841106e-07

## 5 AGG -0.005419543 4.222378e-02 -1.283529e-01 8.983215e-01

A quick sanity check on those asset betas should reveal that SPY has beta of 1 with itself.

beta\_assets %>% select(asset, estimate) %>% filter(asset == "SPY")

## # A tibble: 1 x 2

## asset estimate

##

## 1 SPY 1

Now let’s see how our combination of these assets leads to a portfolio beta.

Let’s assign portfolio weights as we chose above.

w <- c(0.25, 0.25, 0.20, 0.20, 0.10)

Now we can use those weights to get our portfolio beta, based on the betas of the individual assets.

beta\_byhand <-

w[1] \* beta\_assets$estimate[1] +

w[2] \* beta\_assets$estimate[2] +

w[3] \* beta\_assets$estimate[3] +

w[4] \* beta\_assets$estimate[4] +

w[5] \* beta\_assets$estimate[5]

beta\_byhand

## [1] 0.9010689

That beta is the same as we calculated above using the covariance/variance method, and now we know the the covariance of portfolio returns and market returns divided by the variance of market returns is equal to the weighted estimates we got by regressing each asset’s return on market returns.

**Calculating CAPM Beta in the xts World**

We can make things even more efficient, of course, with built-in functions. Let’s go to the xts world and use the built-in CAPM.beta() function from PerformanceAnalytics. That function takes two arguments: the returns for the portfolio (or any asset) whose beta we wish to calculate, and the market returns. Our function will look like CAPM.beta(portfolio\_returns\_xts\_rebalanced\_monthly, mkt\_return\_xts).

beta\_builtin\_xts <- CAPM.beta(portfolio\_returns\_xts\_rebalanced\_monthly, market\_returns\_xts)

beta\_builtin\_xts

## [1] 0.9010689

**Calculating CAPM Beta in the Tidyverse**

We will run that same function through a dplyr and tidyquant code flow to stay in the tidy world.

First we’ll use dplyr to grab our portfolio beta. We’ll return to this flow later for some visualization, but for now will extract the portfolio beta.

To calculate the beta, we call do(model = lm(returns ~ market\_returns\_tidy$returns, data = .)). Then we head back to the broom package and use the tidy() function to make our model results a little easier on the eyes.

beta\_dplyr\_byhand <-

portfolio\_returns\_tq\_rebalanced\_monthly %>%

do(model = lm(returns ~ market\_returns\_tidy$returns, data = .)) %>%

tidy(model) %>%

mutate(term = c("alpha", "beta"))

beta\_dplyr\_byhand

## term estimate std.error statistic p.value

## 1 alpha -0.003129799 0.00155617 -2.011219 4.903980e-02

## 2 beta 0.901068930 0.05340627 16.871969 7.855042e-24

**Calculating CAPM Beta in the Tidyquant World**

Let’s use one more flow with built-in functions, this time using tidyquant and the tq\_performance() function. This will allow us to apply the CAPM.beta() function from PerformanceAnalytics to a data frame.

beta\_builtin\_tq <-

portfolio\_returns\_tq\_rebalanced\_monthly %>%

mutate(market\_return = market\_returns\_tidy$returns) %>%

na.omit() %>%

tq\_performance(Ra = returns,

Rb = market\_return,

performance\_fun = CAPM.beta) %>%

`colnames<-`("beta\_tq")

Let’s take a quick look at our four beta calculations.

beta\_byhand

## [1] 0.9010689

beta\_builtin\_xts

## [1] 0.9010689

beta\_dplyr\_byhand$estimate[2]

## [1] 0.9010689

beta\_builtin\_tq$beta\_tq

## [1] 0.9010689

Consistent results and a beta near 1 as we were expecting, since our portfolio has a 25% allocation to the S&P 500. We’re less concerned with numbers than we are with the various code flows used to get here. Next time we’ll do some visualizing