An R community blog edited by RStudio
Boston, MA
296 POSTS
261 TAGS
Email
Please check this box if you accept the RStudio privacy policy: * ☐
SUBSCRIBE

Calculating Beta in the Capital Asset Pricing Model

2018-02-08

by Jonathan Regenstein

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]

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 in this post.

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
##       <date>       <dbl>
## 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
##       <date>         <dbl>          <dbl>
## 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
##   <chr>            <list>
## 1   SPY <tibble [59 x 2]>
## 2   EFA <tibble [59 x 2]>
## 3   IJS <tibble [59 x 2]>
## 4   EEM <tibble [59 x 2]>
## 5   AGG <tibble [59 x 2]>
```

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
##   <chr>            <list>   <list>
## 1   SPY <tibble [59 x 2]> <S3: lm>
## 2   EFA <tibble [59 x 2]> <S3: lm>
## 3   IJS <tibble [59 x 2]> <S3: lm>
## 4   EEM <tibble [59 x 2]> <S3: lm>
## 5   AGG <tibble [59 x 2]> <S3: lm>
```

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
##   <chr>            <list>                <list>
## 1   SPY <tibble [59 x 2]> <data.frame [2 x 5]>
## 2   EFA <tibble [59 x 2]> <data.frame [2 x 5]>
## 3   IJS <tibble [59 x 2]> <data.frame [2 x 5]>
## 4   EEM <tibble [59 x 2]> <data.frame [2 x 5]>
## 5   AGG <tibble [59 x 2]> <data.frame [2 x 5]>
```

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
##    <chr>                       <chr>        <dbl>        <dbl>
##  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 <dbl>, p.value <dbl>
```

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
##   <chr>        <dbl>        <dbl>         <dbl>        <dbl>
## 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
##   <chr>    <dbl>
## 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 - see you then!

---

1. The Capital Asset Pricing Model: Theory and Evidence Eugene F. Fama and Kenneth R. French, The Capital Asset Pricing Model: Theory and Evidence, The Journal of Economic Perspectives, Vol. 18, No. 3 (Summer, 2004), pp. 25-46↩

R Language · Finance · Reproducible Finance with R R Language · Finance · CAPM   Comments   Share

NEWER • OLDER
Deep learning at rstudio::conf 2018   Cost-Effective BigQuery with R

You may leave a comment below or discuss the post in the forum community.rstudio.com.

2 Comments   **R Views**   **Disqus' Privacy Policy**   1 **Login**

Recommend 4   Tweet   Share   **Sort by Best**

Join the discussion…

LOG IN WITH   OR SIGN UP WITH DISQUS ?

Name

Kim Dung · 7 months ago

Thank you for your post. But the result for using the above code is different with the results you post.
> beta_byhand
[1] 0.4379543
> ## [1] 0.9010689
> beta_builtin_xts
[1] 0.9009479
> ## [1] 0.9010689
> beta_dplyr_byhand$estimate[2]
[1] 0.9009479
> ## [1] 0.9010689
> beta_builtin_tq$beta_tq
[1] 0.9009479
> ## [1] 0.9010689

^ | ˅ · Reply · Share ›

Prashant Dey · 2 years ago

Interesting Post. But I believe this is for a single portfolio. What if there are multiple portfolios in consideration? Like
Portfolio A = {Stock1, Stock2, Stock3, Stock4, Stock5}
Portfolio B = {Stock1, Stock3, Stock5}
Portfolio c = {Stock3, Stock5}

Stocks in Consideration = {Stock1, Stock2, Stock3, Stock4, Stock5}
How would you proceed in such scenarios?
Weights cannot be assigned the same way as you did for single portfolio!(since there are multiple portfolios now)
Bechmark = S&P500

Thanks in advance

^ | ˅ · Reply · Share ›

Subscribe   Add Disqus to your site   Do Not Sell My Data   DISQUS