Today we will continue our work on Fama French factor models, but more as a vehicle to explore some of the awesome stuff happening in the world of [tidy models](https://www.tidyverse.org/articles/2018/11/tidymodels-update-nov-18/). For new readers who want get familiar with Fama French before diving into this post, see [here](https://rviews.rstudio.com/2018/04/11/introduction-to-fama-french/) where we covered importing and wrangling the data, [here](https://rviews.rstudio.com/2018/05/10/rolling-fama-french/) where we covered rolling models and visualization.

Our goal today is to explore k-fold cross-validation via the rsample package, and a bit of model evaluation via the yardstick package. We started the model evaluation theme last time when we used tidy(), glance() and augment() from the broom package. In this post, we will use the rmse() function from yardstick, but our main focus will be on the vfold\_cv() function from rsample. We are going to explore these tools in the context of linear regression and Fama French, which might seem weird since these tools would typically be employed in the realms of machine learning, classification, and the like. We’ll stay in the world of explanatory models via linear regression world for a few reasons.

First, and this is a personal preference, when getting to know a new package or methodology, I prefer to do so in a context that’s already familiar. I don’t want to learn about rsample whilst also getting to know a new data set and learning the complexities of some crazy machine learning model. Since Fama French is familiar from our previous work, we can focus on the new tools in rsample and yardstick. Second, factor models are important in finance, despite relying on good old linear regression. We won’t regret time spent on factor models, and we might even find creative new ways to deploy or visualize them.

The plan for today is take the same models that we ran in the last post, only this use k-fold cross validation and bootstrapping to try to assess the quality of those models.

For that reason, we’ll be working with the same data as we did previously. I won’t go through the logic again, but in short, we’ll import data for daily prices of five ETFs, convert them to, then import the five Fama French factor data and join it to our five ETF returns data. Here’s the code to make that happen:

library(tidyverse)

library(broom)

library(tidyquant)

library(timetk)

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

# The prices object will hold our daily price data.

prices <-

getSymbols(symbols, src = 'yahoo',

from = "2012-12-31",

to = "2017-12-31",

auto.assign = TRUE,

warnings = FALSE) %>%

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

reduce(merge) %>%

`colnames<-`(symbols)

asset\_returns\_long <-

prices %>%

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

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

group\_by(asset) %>%

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

na.omit()

factors\_data\_address <-

"http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/Global\_5\_Factors\_Daily\_CSV.zip"

factors\_csv\_name <- "Global\_5\_Factors\_Daily.csv"

temp <- tempfile()

download.file(

# location of file to be downloaded

factors\_data\_address,

# where we want R to store that file

temp,

quiet = TRUE)

Global\_5\_Factors <-

read\_csv(unz(temp, factors\_csv\_name), skip = 6 ) %>%

rename(date = X1, MKT = `Mkt-RF`) %>%

mutate(date = ymd(parse\_date\_time(date, "%Y%m%d")))%>%

mutate\_if(is.numeric, funs(. / 100)) %>%

select(-RF)

data\_joined\_tidy <-

asset\_returns\_long %>%

left\_join(Global\_5\_Factors, by = "date") %>%

na.omit()

After running that code, we have an object called data\_joined\_tidy. It holds daily returns for 5 ETFs and the Fama French factors. Here’s a look at the first row for each ETF rows.

data\_joined\_tidy %>%

slice(1)

# A tibble: 5 x 8

# Groups: asset [5]

date asset returns MKT SMB HML RMW CMA

1 2013-01-02 AGG -0.00117 0.0199 -0.0043 0.0028 -0.0028 -0.0023

2 2013-01-02 EEM 0.0194 0.0199 -0.0043 0.0028 -0.0028 -0.0023

3 2013-01-02 EFA 0.0154 0.0199 -0.0043 0.0028 -0.0028 -0.0023

4 2013-01-02 IJS 0.0271 0.0199 -0.0043 0.0028 -0.0028 -0.0023

5 2013-01-02 SPY 0.0253 0.0199 -0.0043 0.0028 -0.0028 -0.0023

Let’s work with just one ETF for today and use filter(asset == "AGG") to shrink our data down to just that ETF.

agg\_ff\_data <-

data\_joined\_tidy %>%

filter(asset == "AGG")

Okay, we’re going to regress the daily returns of AGG on one factor, then three factors, then five factors, and we want to evaluate how well each model explains AGG’s returns. That means we need a way to test the model. Last time, we looked at the adjusted r-squared values when the model was run on the entirety of AGG returns. Today, we’ll evaluate the model using k-fold cross validation. That’s a pretty jargon-heavy phrase that isn’t part of the typical finance lexicon. Let’s start with the second part, cross-validation. Instead of running our model on the entire data set – all the daily returns of AGG – we’ll run it on just part of the data set, then test the results on the part that we did not use. Those different subsets of our original data are often called the training and the testing sets, though rsample calls them the analysis and assessment sets. We validate the model results by applying them to the assessment data and seeing how the model performed.

The k-fold bit refers to the fact that we’re not just dividing our data into training and testing subsets, we’re actually going to divide it into a bunch of groups, a k number of groups, or a k number of folds. One of those folds will be used as the validation set; the model will be fit on the other k - 1 sets, and then tested on the validation set. We’re doing this with a linear model to see how well it explains the data; it’s typically used in machine learning to see how well a model predicts data (we’ll get there in 2019).[1](https://rviews.rstudio.com/2018/12/13/rsampling-fama-french/#fn1)

If you’re like me, it will take a bit of tinkering to really grasp k-fold cross validation, but rsample as a great function for dividing our data into k-folds. If we wish to use five folds (the state of the art seems to be either five or ten folds), we call the vfold\_cv() function, pass it our data object agg\_ff\_data, and set v = 5.

library(rsample)

library(yardstick)

set.seed(752)

cved\_ff<-

vfold\_cv(agg\_ff\_data, v = 5)

cved\_ff

# 5-fold cross-validation

# A tibble: 5 x 2

splits id

1 Fold1

2 Fold2

3 Fold3

4 Fold4

5 Fold5

We have an object called cved\_ff, with a column called splits and a column called id. Let’s peek at the first split.

cved\_ff$splits[[1]]

<1007/252/1259>

Three numbers. The first, 1007, is telling us how many observations are in the analysis. Since we have five folds, we should have 80% (or 4/5) of our data in the analysis set. The second number, 252, is telling us how many observations are in the assessment, which is 20% of our original data. The third number, 1259, is the total number of observations in our original data.

Next, we want to apply a model to the analysis set of this k-folded data and test the results on the assessment set. Let’s start with one factor and run a simple linear model, lm(returns ~ MKT).

We want to run it on analysis(cved\_ff$splits[[1]]) – the analysis set of out first split.

ff\_model\_test <- lm(returns ~ MKT, data = analysis(cved\_ff$splits[[1]]))

ff\_model\_test

Call:

lm(formula = returns ~ MKT, data = analysis(cved\_ff$splits[[1]]))

Coefficients:

(Intercept) MKT

0.0001025 -0.0265516

Nothing too crazy so far. Now we want to test on our assessment data. The first step is to add that data to the original set. We’ll use augment() for that task, and pass it assessment(cved\_ff$splits[[1]])

ff\_model\_test %>%

augment(newdata = assessment(cved\_ff$splits[[1]])) %>%

head() %>%

select(returns, .fitted)

returns .fitted

1 0.0009021065 1.183819e-04

2 0.0011726989 4.934779e-05

3 0.0010815505 1.157267e-04

4 -0.0024385815 -7.544460e-05

5 -0.0021715702 -8.341007e-05

6 0.0028159467 3.865527e-04

We just added our fitted values to the assessment data, the subset of the data on which the model was not fit. How well did our model do when compare the fitted values to the data in the held out set?

We can use the rmse() function from yardstick to measure our model. RMSE stands for root mean-squared error. It’s the sum of the squared differences between our fitted values and the actual values in the assessment data. A lower RMSE is better!

ff\_model\_test %>%

augment(newdata = assessment(cved\_ff$splits[[1]])) %>%

rmse(returns, .fitted)

# A tibble: 1 x 3

.metric .estimator .estimate

1 rmse standard 0.00208

Now that we’ve done that piece by piece, let’s wrap the whole operation into one function. This function takes one argument, a split, and we’re going to use pull() so we can extract the raw number, instead of the entire tibble result.

model\_one <- function(split) {

split\_for\_model <- analysis(split)

ff\_model <- lm(returns ~ MKT, data = split\_for\_model)

holdout <- assessment(split)

rmse <- ff\_model %>%

augment(newdata = holdout) %>%

rmse(returns, .fitted) %>%

pull(.estimate)

}

Now we pass it our first split.

model\_one(cved\_ff$splits[[1]]) %>%

head()

[1] 0.002080324

Now we want to apply that function to each of our five folds that are stored in agg\_cved\_ff. We do that with a combination of mutate() and map\_dbl(). We use map\_dbl() instead of map because we are returning a number here and there’s not a good reason to store that number in a list column.

cved\_ff %>%

mutate(rmse = map\_dbl(cved\_ff$splits, model\_one))

# 5-fold cross-validation

# A tibble: 5 x 3

splits id rmse

\*

1 Fold1 0.00208

2 Fold2 0.00189

3 Fold3 0.00201

4 Fold4 0.00224

5 Fold5 0.00190

OK, we have five RMSE’s since we ran the model on five separate analysis fold sets and tested on five separate assessment fold sets. Let’s find the average RMSE by taking the mean() of the rmse column. That can help reduce noisiness that resulted from our random creation of those five folds.

cved\_ff %>%

mutate(rmse = map\_dbl(cved\_ff$splits, model\_one)) %>%

summarise(mean\_rse = mean(rmse))

# 5-fold cross-validation

# A tibble: 1 x 1

mean\_rse

1 0.00202

We now have the mean RMSE after running on our model, lm(returns ~ MKT), on all five of our folds.

That process for finding the mean RMSE can be applied other models, as well. Let’s suppose we wish to find the mean RMSE for two other models: lm(returns ~ MKT + SMB + HML), the Fama French three-factor model, and lm(returns ~ MKT + SMB + HML + RMW + CMA, the Fama French five-factor model. By comparing the mean RMSE’s, we can evaluate which model explained the returns of AGG better. Since we’re just adding more and more factors, the models can be expected to get more and more accurate but again, we are exploring the rsample machinery and creating a template where we can pop in whatever models we wish to compare.

First, let’s create two new functions, that follow the exact same code pattern as above but house the three-factor and five-factor models.

model\_two <- function(split) {

split\_for\_model <- analysis(split)

ff\_model <- lm(returns ~ MKT + SMB + HML, data = split\_for\_model)

holdout <- assessment(split)

rmse <-

ff\_model %>%

augment(newdata = holdout) %>%

rmse(returns, .fitted) %>%

pull(.estimate)

}

model\_three <- function(split) {

split\_for\_model <- analysis(split)

ff\_model <- lm(returns ~ MKT + SMB + HML + RMW + CMA, data = split\_for\_model)

holdout <- assessment(split)

rmse <-

ff\_model %>%

augment(newdata = holdout) %>%

rmse(returns, .fitted) %>%

pull(.estimate)

}

Now we pass those three models to the same mutate() with map\_dbl() flow that we used with just one model. The result will be three new columns of RMSE’s, one for each of our three models applied to our five folds.

cved\_ff %>%

mutate(

rmse\_model\_1 = map\_dbl(

splits,

model\_one),

rmse\_model\_2 = map\_dbl(

splits,

model\_two),

rmse\_model\_3 = map\_dbl(

splits,

model\_three))

# 5-fold cross-validation

# A tibble: 5 x 5

splits id rmse\_model\_1 rmse\_model\_2 rmse\_model\_3

\*

1 Fold1 0.00208 0.00211 0.00201

2 Fold2 0.00189 0.00184 0.00178

3 Fold3 0.00201 0.00195 0.00194

4 Fold4 0.00224 0.00221 0.00213

5 Fold5 0.00190 0.00183 0.00177

We can also find the mean RMSE for each model.

cved\_ff %>%

mutate(

rmse\_model\_1 = map\_dbl(

splits,

model\_one),

rmse\_model\_2 = map\_dbl(

splits,

model\_two),

rmse\_model\_3 = map\_dbl(

splits,

model\_three)) %>%

summarise(mean\_rmse\_model\_1 = mean(rmse\_model\_1),

mean\_rmse\_model\_2 = mean(rmse\_model\_2),

mean\_rmse\_model\_3 = mean(rmse\_model\_3))

# 5-fold cross-validation

# A tibble: 1 x 3

mean\_rmse\_model\_1 mean\_rmse\_model\_2 mean\_rmse\_model\_3

1 0.00202 0.00199 0.00192

That code flow worked just fine, but we had to repeat ourselves when creating the functions for each model. Let’s toggle to a flow where we define three models – the ones that we wish to test with via cross-validation and RMSE – then pass those models to one function.

First, we use as.formula() to define our three models.

mod\_form\_1 <- as.formula(returns ~ MKT)

mod\_form\_2 <- as.formula(returns ~ MKT + SMB + HML)

mod\_form\_3 <- as.formula(returns ~ MKT + SMB + HML + RMW + CMA)

Now we write one function that takes split as an argument, same as above, but also takes formula as an argument, so we can pass it different models. This gives us the flexibility to more easily define new models and pass them to map, so I’ll append \_flex to the name of this function.

ff\_rmse\_models\_flex <- function(split, formula) {

split\_for\_data <- analysis(split)

ff\_model <- lm(formula, data = split\_for\_data)

holdout <- assessment(split)

rmse <-

ff\_model %>%

augment(newdata = holdout) %>%

rmse(returns, .fitted) %>%

pull(.estimate)

}

Now we use the same code flow as before, except we call map\_dbl(), pass it our cved\_ff$splits object, our new flex function called ff\_rmse\_models\_flex(), and the model we wish to pass as the formula argument. First we pass it mod\_form\_1.

cved\_ff %>%

mutate(rmse\_model\_1 = map\_dbl(cved\_ff$splits,

ff\_rmse\_models\_flex,

formula = mod\_form\_1))

# 5-fold cross-validation

# A tibble: 5 x 3

splits id rmse\_model\_1

\*

1 Fold1 0.00208

2 Fold2 0.00189

3 Fold3 0.00201

4 Fold4 0.00224

5 Fold5 0.00190

Now let’s pass it all three models and find the mean RMSE.

cved\_ff %>%

mutate(rmse\_model\_1 = map\_dbl(cved\_ff$splits,

ff\_rmse\_models\_flex,

formula = mod\_form\_1),

rmse\_model\_2 = map\_dbl(cved\_ff$splits,

ff\_rmse\_models\_flex,

formula = mod\_form\_2),

rmse\_model\_3 = map\_dbl(cved\_ff$splits,

ff\_rmse\_models\_flex,

formula = mod\_form\_3)) %>%

summarise(mean\_rmse\_model\_1 = mean(rmse\_model\_1),

mean\_rmse\_model\_2 = mean(rmse\_model\_2),

mean\_rmse\_model\_3 = mean(rmse\_model\_3))

# 5-fold cross-validation

# A tibble: 1 x 3

mean\_rmse\_model\_1 mean\_rmse\_model\_2 mean\_rmse\_model\_3

1 0.00202 0.00199 0.00192

Alright, that code flow seems a bit more flexible than our original method of writing a function to assess each model. We didn’t do much hard thinking about functional form here, but hopefully this provides a template where you could assess more nuanced models. We’ll get into bootstrapping and time series work next week, then head to Shiny to ring in the New Year!

And, finally, a couple of public service announcements.