FRE7241 Algorithmic Portfolio Management

Lecture#6, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

October 18, 2022

The *Alpha* and *Beta* of Stock Returns

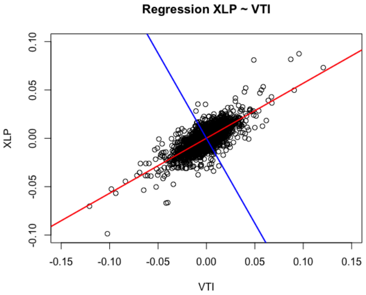
The daily stock returns $r_i - r_f$ in excess of the risk-free rate r_f, can be decomposed into *systematic* returns $\beta(r_m - r_f)$ (where $r_m - r_f$ are the excess market returns) plus *idiosyncratic* returns $\alpha + \varepsilon_i$ (which are uncorrelated to the market returns):

$$r_i - r_f = \alpha + \beta(r_m - r_f) + \varepsilon_i$$

The *alpha* α are the abnormal returns in excess of the risk premium, and ε_i are the regression residuals with zero mean.

The *idiosyncratic* risk (equal to ε_i) is uncorrelated to the *systematic* risk, and can be reduced through portfolio diversification.

```
> # Perform regression using formula
> retsp <- na.omit(rutils::etfenv$returns[, c("XLP", "VTI")])
> riskfree <- 0.03/252
> retsp <- (retsp - riskfree)
> regmod <- lm(XLP ~ VTI, data=retsp)
> regmodsum <- summary(regmod)
> # Get regression coefficients
> coef(regmodsum)
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.71e-05   8.37e-05   0.802     0.423
VTI         5.69e-01   6.80e-03  83.567     0.000
> # Get alpha and beta
> coef(regmodsum)[, 1]
(Intercept)         VTI
   6.71e-05    5.69e-01
```




```
> # Plot scatterplot of returns with aspect ratio 1
> plot(XLP ~ VTI, data=rutils::etfenv$returns,
+      xlim=c(-0.1, 0.1), ylim=c(-0.1, 0.1),
+      asp=1, main="Regression XLP ~ VTI")
> # Add regression line and perpendicular line
> abline(regmod, lwd=2, col="red")
> abline(a=0, b=-1/coef(regmodsum)[2, 1], lwd=2, col="blue")
```

The Statistical Significance of *Alpha* and *Beta*

The stock β is independent of the risk-free rate r_f:

$$\beta = \frac{\mathrm{Cov}(r_i, r_m)}{\mathrm{Var}(r_m)}$$

The *t*-statistic (*t*-value) is the ratio of the estimated value divided by its standard error.

The *p*-value is the probability of obtaining values exceeding the *t*-statistic, assuming the *null hypothesis* is true.

A small *p*-value means that the regression coefficients are very unlikely to be zero (given the data).

The *beta* β values of stock returns are very statistically significant, but the *alpha* α values are mostly not significant.

The *p*-value of the *Durbin-Watson* test is large, which indicates that the regression residuals are not autocorrelated.

In practice, the α, β, and the risk-free rate r_f, depend on the time interval of the data, so they're time dependent.

```
> # Get regression coefficients
> coef(regmodsum)
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.71e-05   8.37e-05   0.802    0.423
VTI         5.69e-01   6.80e-03  83.567    0.000
> # Calculate regression coefficients from scratch
> betav <- drop(cov(retsp$XLP, retsp$VTI)/var(retsp$VTI))
> alpha <- drop(mean(retsp$XLP) - betav*mean(retsp$VTI))
> c(alpha, betav)
[1] 6.71e-05 5.69e-01
> # Calculate the residuals
> residuals <- (retsp$XLP - (alpha + betav*retsp$VTI))
> # Calculate the standard deviation of residuals
> nrows <- NROW(residuals)
> residsd <- sqrt(sum(residuals^2)/(nrows - 2))
> # Calculate the standard errors of beta and alpha
> sum2 <- sum((retsp$VTI - mean(retsp$VTI))^2)
> betasd <- residsd/sqrt(sum2)
> alphasd <- residsd*sqrt(1/nrows + mean(retsp$VTI)^2/sum2)
> c(alphasd, betasd)
[1] 8.37e-05 6.80e-03
> # Perform the Durbin-Watson test of autocorrelation of residuals
> lmtest::dwtest(regmod)

Durbin-Watson test

data:  regmod
DW = 2, p-value = 1
alternative hypothesis: true autocorrelation is greater than 0
```

The *Alpha* and *Beta* of ETF Returns

The *beta* β values of ETF returns are very statistically significant, but the *alpha* α values are mostly not significant.

Some of the ETFs with significant *alpha* α values are the bond ETFs *IEF* and *TLT* (which have performed very well), and the natural resource ETFs *USO* and *DBC* (which have performed very poorly).

```
> retsp <- rutils::etfenv$returns
> symbolv <- colnames(retsp)
> symbolv <- symbolv[symbolv != "VTI"]
> # Perform regressions and collect statistics
> betam <- sapply(symbolv, function(symbol) {
+ # Specify regression formula
+   formulav <- as.formula(paste(symbol, "~ VTI"))
+ # Perform regression
+   regmod <- lm(formulav, data=retsp)
+ # Get regression summary
+   regmodsum <- summary(regmod)
+ # Collect regression statistics
+   with(regmodsum,
+     c(beta=coefficients[2, 1],
+ pbeta=coefficients[2, 4],
+ alpha=coefficients[1, 1],
+ palpha=coefficients[1, 4],
+ pdw=lmtest::dwtest(regmod)$p.value))
+ })  # end sapply
> betam <- t(betam)
> # Sort by palpha
> betam <- betam[order(betam[, "palpha"]), ]
```

```
> betam
        beta     pbeta     alpha  palpha      pdw
IEF  -0.1291 2.26e-170  2.04e-04 0.00020 3.41e-01
VXX  -2.7960  0.00e+00 -1.37e-03 0.00201 3.51e-01
TLT  -0.2778 2.73e-176  3.13e-04 0.00679 3.95e-01
VEU   1.0115  0.00e+00 -2.52e-04 0.01349 1.00e+00
USO   0.7248 7.48e-147 -7.54e-04 0.02847 1.40e-01
XLF   1.2932  0.00e+00 -2.55e-04 0.05233 1.00e+00
GLD   0.0435  1.46e-03  2.79e-04 0.09847 7.69e-01
XLP   0.5686  0.00e+00  1.18e-04 0.15715 1.00e+00
IVE   0.9906  0.00e+00 -6.66e-05 0.18490 1.00e+00
USMV  0.7424  0.00e+00  8.68e-05 0.20998 6.78e-01
XLV   0.7512  0.00e+00  1.05e-04 0.23998 9.00e-01
EEM   1.2352  0.00e+00 -1.66e-04 0.24228 9.99e-01
SVXY  2.2824 9.94e-177 -8.98e-04 0.27259 5.13e-06
VLUE  0.9930  0.00e+00 -1.05e-04 0.30074 9.89e-01
IWD   0.9860  0.00e+00 -4.20e-05 0.38686 1.00e+00
XLY   1.0273  0.00e+00  5.91e-05 0.46899 1.00e+00
IVW   0.9649  0.00e+00  3.10e-05 0.47553 1.00e+00
XLU   0.6498  0.00e+00  8.98e-05 0.48008 1.00e+00
VNQ   1.1879  0.00e+00 -1.25e-04 0.48820 1.00e+00
DBC   0.4162 3.13e-179 -1.21e-04 0.49141 9.91e-01
XLE   1.1203  0.00e+00 -1.03e-04 0.55764 5.78e-01
VTV   0.9687  0.00e+00 -2.44e-05 0.67054 1.00e+00
QUAL  0.9707  0.00e+00  1.89e-05 0.67869 9.93e-01
XLI   1.0092  0.00e+00 -2.76e-05 0.71849 1.00e+00
XLK   1.0868  0.00e+00  2.39e-05 0.79399 9.99e-01
IWB   0.9832  0.00e+00 -5.57e-06 0.79791 1.00e+00
MTUM  1.0291  0.00e+00 -1.66e-05 0.87321 9.42e-02
IWF   0.9944  0.00e+00  1.15e-05 0.87587 1.00e+00
XLB   1.0354  0.00e+00 -8.31e-06 0.93854 1.00e+00
VYM   0.8761  0.00e+00  5.37e-07 0.99434 1.00e+00
```

Capital Asset Pricing Model (*CAPM*)

The CAPM model states that the expected return for stock n: $\mathbb{E}[R_n]$ is proportional to its beta β_n times the expected excess return of the market $\mathbb{E}[R_m] - r_f$:

$$\mathbb{E}[R_n] = r_f + \beta_n(\mathbb{E}[R_m] - r_f)$$

The *CAPM* model states that if a stock has a higher beta then it's also expected to earn higher returns.

According to the *CAPM* model, assets are on average expected to earn only a *systematic* return proportional to their *systematic* risk.

The *CAPM* model is not a regression model.

The *CAPM* model depends on the choice of the risk-free rate r_f.

```
> library(PerformanceAnalytics)
> # Calculate XLP beta
> PerformanceAnalytics::CAPM.beta(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.569
> # Or
> retsxlp <- na.omit(retsp[, c("XLP", "VTI")])
> betav <- drop(cov(retsxlp$XLP, retsxlp$VTI)/var(retsxlp$VTI))
> betav
[1] 0.569
> # Calculate XLP alpha
> PerformanceAnalytics::CAPM.alpha(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.000118
> # Or
> mean(retsp$XLP - betav*retsp$VTI)
[1] NA
> # Calculate XLP bull beta
> PerformanceAnalytics::CAPM.beta.bull(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.583
> # Calculate XLP bear beta
> PerformanceAnalytics::CAPM.beta.bear(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.581
```

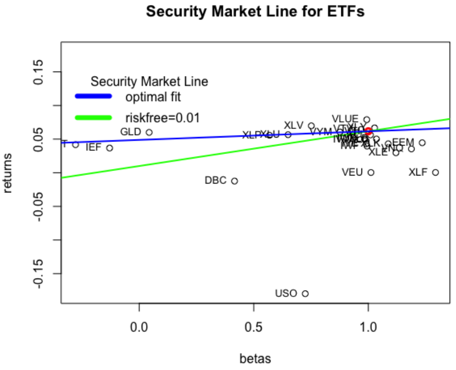
The Security Market Line for ETFs

The *Security Market Line* (SML) represents the linear relationship between expected stock returns and *systematic* risk.

A scatterplot of asset returns versus their β shows which assets earn a positive α, and which don't.

If an asset lies on the *SML*, then its returns are mostly *systematic*, and its α is equal to zero.

Assets above the *SML* have a positive α, and those below have a negative α.




```
> symbolv <- rownames(betam)
> betav <- betam[-match(c("VXX", "SVXY", "MTUM", "USMV", "QUAL"), sy
> betav <- c(1, betav)
> names(betav)[1] <- "VTI"
> retsann <- sapply(retsp[, names(betav)], PerformanceAnalytics::Ret
> # Plot scatterplot of returns vs betas
> minrets <- min(retsann)
> plot(retsann ~ betav, xlab="betas", ylab="returns",
+      ylim=c(minrets, -minrets), main="Security Market Line for ETI
> retsvti <- retsann["VTI"]
> points(x=1, y=retsvti, col="red", lwd=3, pch=21)
> # Plot Security Market Line
> riskfree <- 0.01
> abline(a=riskfree, b=(retsvti-riskfree), col="green", lwd=2)
> # Add labels
> text(x=betav, y=retsann, labels=names(betav), pos=2, cex=0.8)
```

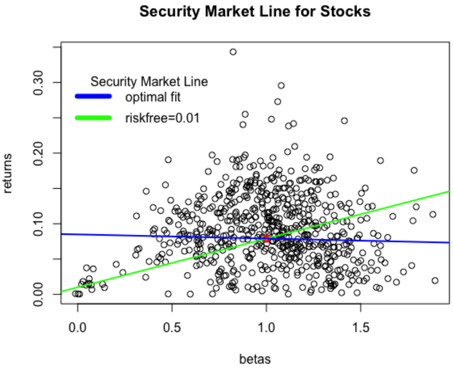
```
> # Find optimal risk-free rate by minimizing residuals
> rss <- function(riskfree) {
+   sum((retsann - riskfree + betav*(retsvti-riskfree))^2)
+ }  # end rss
> optimrss <- optimize(rss, c(-1, 1))
> riskfree <- optimrss$minimum
> abline(a=riskfree, b=(retsvti-riskfree), col="blue", lwd=2)
> legend(x="top", bty="n", title="Security Market Line",
+  legend=c("optimal fit", "riskfree=0.01"),
+  y.intersp=0.5, cex=1.0, lwd=6, lty=1, col=c("blue", "green"))
```

The Security Market Line for Stocks

The best fitting *Security Market Line* (SML) for stocks is almost flat, which shows that stocks with higher β don't earn higher returns.

This is called the *low beta anomaly*.

```
> # Load S&P500 constituent stock returns
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData
> retsvti <- na.omit(rutils::etfenv$returns$VTI)
> retsp <- returns[index(retsvti), ]
> nrows <- NROW(retsp)
> # Calculate stock betas
> betav <- sapply(retsp, function(x) {
+   retsp <- na.omit(cbind(x, retsvti))
+   drop(cov(retsp[, 1], retsp[, 2])/var(retsp[, 2]))
+ })  # end sapply
> mean(betav)
> # Calculate annual stock returns
> retsann <- retsp
> retsann[1, ] <- 0
> retsann <- zoo::na.locf(retsann, na.rm=FALSE)
> retsann <- 252*sapply(retsann, sum)/nrows
> # Remove stocks with zero returns
> sum(retsann == 0)
> betav <- betav[retsann > 0]
> retsann <- retsann[retsann > 0]
> retsvti <- 252*mean(retsvti)
> # Plot scatterplot of returns vs betas
> plot(retsann ~ betav, xlab="betas", ylab="returns",
+      main="Security Market Line for Stocks")
> points(x=1, y=retsvti, col="red", lwd=3, pch=21)
> # Plot Security Market Line
> riskfree <- 0.01
> abline(a=riskfree, b=(retsvti-riskfree), col="green", lwd=2)
```




```
> # Find optimal risk-free rate by minimizing residuals
> rss <- function(riskfree) {
+   sum((retsann - riskfree + betav*(retsvti-riskfree))^2)
+ }  # end rss
> optimrss <- optimize(rss, c(-1, 1))
> riskfree <- optimrss$minimum
> abline(a=riskfree, b=(retsvti-riskfree), col="blue", lwd=2)
> legend(x="top", bty="n", title="Security Market Line",
+  legend=c("optimal fit", "riskfree=0.01"),
+  y.intersp=0.5, cex=1.0, lwd=6, lty=1, col=c("blue", "green"))
```

Beta-adjusted Performance Measurement

The *Treynor* ratio measures the excess returns per unit of the *systematic* risk *beta* β, and is equal to the excess returns (over a risk-free return) divided by the β:

$$T_r = \frac{E[R - r_f]}{\beta}$$

The *Treynor* ratio is similar to the *Sharpe* ratio, with the difference that its denominator represents only *systematic* risk, not total risk.

The *Information* ratio is equal to the excess returns (over a benchmark) divided by the *tracking error* (standard deviation of excess returns):

$$I_r = \frac{E[R - R_b]}{\sqrt{\sum_{i=1}^{n} (R_i - R_{i,b})^2}}$$

The *Information* ratio measures the amount of outperformance versus the benchmark, and the consistency of outperformance.

```
> library(PerformanceAnalytics)
> # Calculate XLP Treynor ratio
> TreynorRatio(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.098
> # Calculate XLP Information ratio
> InformationRatio(Ra=retsp$XLP, Rb=retsp$VTI)
[1] 0.0334
```

CAPM Summary Statistics

PerformanceAnalytics::table.CAPM() calculates the *beta* β and *alpha* α values, the *Treynor* ratio, and other performance statistics.

```
> PerformanceAnalytics::table.CAPM(Ra=retsp[, c("XLP", "XLF")],
+                            Rb=retsp$VTI, scale=252)
                    XLP to VTI XLF to VTI
Alpha                   0.0001    -0.0003
Beta                    0.5686     1.2932
Beta+                   0.5828     1.3695
Beta-                   0.5812     1.3578
R-squared               0.5673     0.7344
Annualized Alpha        0.0303    -0.0621
Correlation             0.7532     0.8570
Correlation p-value     0.0000     0.0000
Tracking Error          0.1285     0.1623
Active Premium          0.0043    -0.0673
Information Ratio       0.0334    -0.4144
Treynor Ratio           0.0980     0.0003
> capmstats <- table.CAPM(Ra=retsp[, symbolv],
+         Rb=retsp$VTI, scale=252)
> colnamev <- strsplit(colnames(capmstats), split=" ")
> colnamev <- do.call(cbind, colnamev)[1, ]
> colnames(capmstats) <- colnamev
> capmstats <- t(capmstats)
> capmstats <- capmstats[, -1]
> colnamev <- colnames(capmstats)
> whichv <- match(c("Annualized Alpha", "Information Ratio", "Treynor Rat
> colnamev[whichv] <- c("Alpha", "Information", "Treynor")
> colnames(capmstats) <- colnamev
> capmstats <- capmstats[order(capmstats[, "Alpha"], decreasing=TRUE), ]
> # Copy capmstats into etfenv and save to .RData file
> etfenv <- rutils::etfenv
> etfenv$capmstats <- capmstats
> save(etfenv, file="/Users/jerzy/Develop/lecture_slides/data/etf_data.RData")
```

```
> rutils::etfenv$capmstats[, c("Beta", "Alpha", "Information"
        Beta   Alpha Information Treynor
TLT  -0.2778  0.0820     -0.1550 -0.1503
GLD   0.0435  0.0729     -0.0567  1.3797
IEF  -0.1291  0.0526     -0.2127 -0.2844
XLP   0.5686  0.0303      0.0334  0.0980
XLV   0.7512  0.0267      0.0913  0.0928
XLU   0.6498  0.0229     -0.0313  0.0870
USMV  0.7424  0.0221     -0.1017  0.1548
XLY   1.0273  0.0150      0.1294  0.0649
IVW   0.9649  0.0078      0.1067  0.0512
XLK   1.0868  0.0061      0.0378  0.0396
QUAL  0.9707  0.0048      0.0566  0.1079
IWF   0.9944  0.0029     -0.0142  0.0398
VYM   0.8761  0.0001     -0.1183  0.0692
VTI   1.0000  0.0000         NaN  0.0617
IWB   0.9832 -0.0014     -0.1009  0.0510
XLB   1.0354 -0.0021     -0.0723  0.0484
MTUM  1.0291 -0.0042     -0.0661  0.1035
VTV   0.9687 -0.0061     -0.1680  0.0663
XLI   1.0092 -0.0069     -0.1254  0.0557
IWD   0.9860 -0.0105     -0.2393  0.0505
IVE   0.9906 -0.0167     -0.3412  0.0434
XLE   1.1203 -0.0256     -0.2150  0.0265
VLUE  0.9930 -0.0260     -0.4243  0.0795
DBC   0.4162 -0.0301     -0.3980 -0.0297
VNQ   1.1879 -0.0311     -0.2188  0.0298
EEM   1.2352 -0.0410     -0.2600  0.0361
VEU   1.0115 -0.0614     -0.6992  0.0004
XLF   1.2932 -0.0621     -0.4144  0.0003
USO   0.7248 -0.1730     -0.7117 -0.2481
SVXY  2.2824 -0.2025         NaN     NaN
VXX  -2.7960 -0.2921     -0.9119  0.2191
```

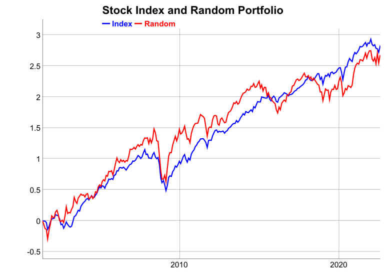
Random Stock Selection

A random portfolio is a sub-portfolio of stocks selected at random.

Random portfolios are used as a benchmark for stock pickers (portfolio managers).

If a portfolio manager outperforms the median of random portfolios, then they may have stock picking skill.

```
> # Load the S&P500 stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_prices.RData'
> # Subset (select) the prices after the start date of VTI
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> prices <- prices[zoo::index(retvti)]
> # Select columns with non-NA prices at start
> prices <- prices[, !is.na(prices[1, ])]
> dim(prices)
> # Copy over NA prices using the function zoo::na.locf()
> prices <- zoo::na.locf(prices, na.rm=FALSE)
> sum(is.na(prices))
> datev <- zoo::index(prices)
> retvti <- retvti[datev]
> nrows <- NROW(prices)
> nstocks <- NCOL(prices)
> # Normalize the prices so that they start at 1
> pricesn <- lapply(prices, function(x) x/as.numeric(x[1]))
> pricesn <- rutils::do_call(cbind, pricesn)
> head(pricesn[, 1:5])
```




```
> # Calculate the equal dollar-weighted average of all stock prices
> indeks <- rowMeans(pricesn)
> indeks <- xts::xts(indeks, order.by=datev)
> colnames(indeks) <- "Index"
> # Select a random, equal dollar-weighted portfolio of 5 stocks
> set.seed(1121)
> samplev <- sample.int(n=nstocks, size=5, replace=FALSE)
> portf <- pricesn[, samplev]
> portf <- rowMeans(portf)
> portf <- xts::xts(portf, order.by=datev)
> colnames(portf) <- "Random"
> # Plot dygraph of stock index and random portfolio
> wealthv <- cbind(indeks, portf)
> colorv <- c("blue", "red")
> endp <- rutils::calc_endpoints(prices, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Stock Index and Ra
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Random Stock Portfolios

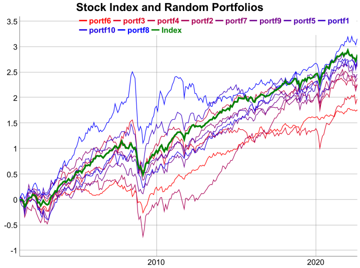
Most random portfolios underperform the index, so picking a portfolio which outperforms the stock index requires great skill.

An investor without skill, who selects stocks at random, has a high probability of underperforming the index, because they will most likely miss selecting the best performing stocks.

Therefore the proper benchmark for a stock picker is the median of random portfolios, not the stock index, which is the mean of all the stock prices.

Performing as well as the index requires *significant* investment skill, while outperforming the index requires *exceptional* investment skill.

```
> # Select 10 random equal dollar-weighted sub-portfolios
> set.seed(1121)
> nportf <- 10
> portfs <- sapply(1:nportf, function(x) {
+   prices <- pricesn[, sample.int(n=nstocks, size=5, replace=FALSE
+   rowMeans(prices)
+ })  # end sapply
> portfs <- xts::xts(portfs, order.by=datev)
> colnames(portfs) <- paste0("portf", 1:nportf)
> # Sort the sub-portfolios according to perfomance
> portfs <- portfs[, order(portfs[nrows])]
> round(head(portfs), 3)
> round(tail(portfs), 3)
```



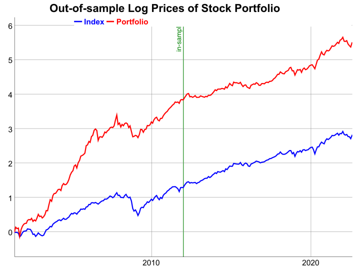

```
> # Plot dygraph of stock index and random portfolios
> colorv <- colorRampPalette(c("red", "blue"))(nportf)
> combined <- cbind(indeks, portfs)
> colnames(combined)[1] <- "Index"
> colnamev <- colnames(combined)
> colorv <- c("green", colorv)
> dygraphs::dygraph(log(combined[endp]), main="Stock Index and Rando
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeWi
+   dyLegend(show="always", width=500)
```

Stock Portfolio Selection Out-of-Sample

The strategy selects the 10 best performing stocks from the in-sample interval, and invests equal dollar amounts in the out-of-sample interval.

The out-of-sample performance of the best performing stocks in-sample, is not any better than the index.

```
> # Define cutoff between in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the 10 best performing stocks in-sample
> perfstat <- sort(drop(coredata(pricesn[cutoff, ])), decreasing=TRU
> symbolv <- names(head(perfstat, 10))
> # Calculate the in-sample portfolio
> pricis <- pricesn[insample, symbolv]
> # Normalize the prices so that they are 1 at cutoff+1
> pricesn <- lapply(prices, function(x) x/as.numeric(x[cutoff+1]))
> pricesn <- rutils::do_call(cbind, pricesn)
> # Calculate the out-of-sample portfolio
> pricos <- pricesn[outsample, symbolv]
> # Scale the prices to preserve the in-sample wealth
> pricos <- sum(pricis[cutoff, ])*pricos/sum(pricos[1, ])
```



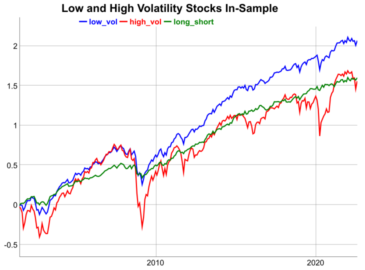

```
> # Combine indeks with out-of-sample stock portfolio returns
> wealthv <- rbind(pricis, pricos)
> wealthv <- xts::xts(rowMeans(wealthv), datev)
> wealthv <- cbind(indeks, wealthv)
> colnames(wealthv)[2] <- "Portfolio"
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv[outsample, ]),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot out-of-sample stock portfolio returns
> dygraphs::dygraph(log(wealthv[endp]), main="Out-of-sample Log Pric
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="in-sample", strokePattern="solid",
+   dyLegend(width=500)
```

Low and High Volatility Stock Portfolios

Research by Robeco, Eric Falkenstein, and others has shown that low volatility stocks have outperformed high volatility stocks.

Betting against volatility is a strategy which invests in low volatility stocks and shorts high volatility stocks.

USMV is an *ETF* that holds low volatility stocks, although it hasn't met expectations.




```
> # Calculate the stock volatilities, betas, and alphas
> retsp <- rutils::diffit(log(prices))
> varvti <- drop(var(retvti))
> meanvti <- mean(retvti)
> riskret <- sapply(retsp, function(rets) {
+   betav <- drop(cov(rets, retvti))/varvti
+   resid <- rets - betav*retvti
+   alphav <- mean(rets) - betav*meanvti
+   c(alpha=alphav, beta=betav, vol=sd(rets), ivol=sd(resid))
+ })  # end sapply
> riskret <- t(riskret)
> tail(riskret)
> # Sort stocks by their volatilities
> riskret <- riskret[order(riskret[, "vol"]), ]
> symbolv <- rownames(riskret)
> # Calculate the cumulative returns of low and high volatility stocks
> volow <- rowMeans(retsp[, symbolv[1:(nstocks %/% 2)]])
> volhigh <- rowMeans(retsp[, symbolv[(nstocks %/% 2):nstocks]])
> wealthv <- cbind(volow, volhigh, volow - 0.3*volhigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("low_vol", "high_vol", "long_short")
```

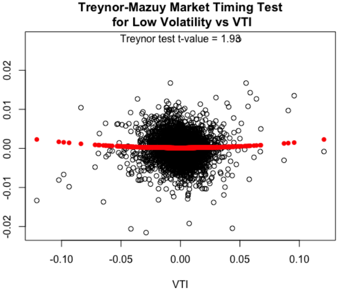
```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high volatility stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Volati
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

Low Volatility Stock Portfolio Market Timing Skill

Market timing skill is the ability to forecast the direction and magnitude of market returns.

The *Treynor-Mazuy* test shows that the *betting against volatility* strategy has some *market timing* skill.

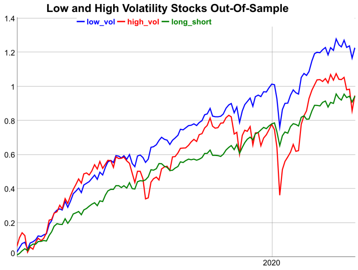
```
> # Merton-Henriksson test
> predictor <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predictor)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predictor); s
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predictor);
> # Plot residual scatterplot
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="Low Volatility"
> title(main="Treynor-Mazuy Market Timing Test\n for Low Volatility
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=max(residv), paste("Treynor test t-value =", tvalue))
```



Low and High Volatility Stock Portfolios Out-Of-Sample

The low volatility stocks selected in-sample also outperform the high volatility stocks in the out-of-sample period.

```
> # Calculate the in-sample stock volatilities, betas, and alphas
> varvti <- drop(var(retvti[insample]))
> meanvti <- mean(retvti[insample])
> riskretis <- sapply(retsp[insample], function(rets) {
+   betav <- drop(cov(rets[insample], retvti[insample]))/varvti
+   resid <- rets - betav*retvti[insample]
+   alphav <- mean(rets[insample]) - betav*meanvti
+   c(alpha=alphav, beta=betav, vol=sd(rets), ivol=sd(resid))
+ })  # end sapply
> riskretis <- t(riskretis)
> tail(riskretis)
> # Sort stocks in-sample by their volatilities
> riskretis <- riskretis[order(riskretis[, "vol"]), ]
> head(riskretis)
> symbolv <- rownames(riskretis)
> # Calculate the out-of-sample returns of low and high volatility
> volow <- rowMeans(retsp[outsample, symbolv[1:(nstocks %/% 2)]])
> volhigh <- rowMeans(retsp[outsample, symbolv[(nstocks %/% 2):nst
> wealthv <- cbind(volow, volhigh, volow - 0.3*volhigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnames(wealthv) <- c("low_vol", "high_vol", "long_short")
```



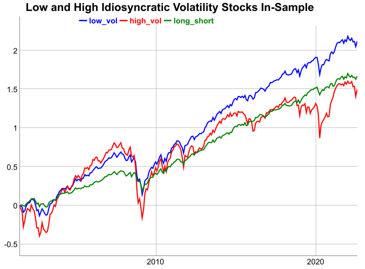

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high volatility stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Volati
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

Low and High Idiosyncratic Volatility Stock Portfolios

Research by Robeco, Eric Falkenstein, and others has shown that low idiosyncratic volatility stocks have outperformed high volatility stocks.

Betting against idiosyncratic volatility is a strategy which invests in low idiosyncratic volatility stocks and shorts high volatility stocks.

```
> # Sort stocks by their idiosyncratic volatilities
> riskret <- riskret[order(riskret[, "ivol"]), ]
> symbolv <- rownames(riskret)
> # Calculate the cumulative returns of low and high volatility stoc
> volow <- rowMeans(retsp[, symbolv[1:(nstocks %/% 2)]])
> volhigh <- rowMeans(retsp[, symbolv[(nstocks %/% 2):nstocks]])
> wealthv <- cbind(volow, volhigh, volow - 0.3*volhigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("low_vol", "high_vol", "long_short")
```

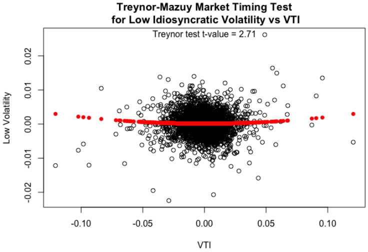



```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high volatility stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Idiosy
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

Low Idiosyncratic Volatility Stock Portfolio Market Timing Skill

Market timing skill is the ability to forecast the direction and magnitude of market returns.

The *Treynor-Mazuy* test shows that the *betting against idiosyncratic volatility* strategy has some *market timing* skill.

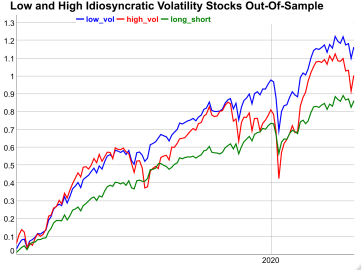



```
> # Merton-Henriksson test
> predictor <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predictor)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predictor); s
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predictor);
> # Plot residual scatterplot
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="Low Volatility",
> title(main="Treynor-Mazuy Market Timing Test\n for Low Idiosyncratic Volatility vs VTI", line=0.5)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=max(residv), paste("Treynor test t-value =", tvalue))
```

Low and High Idiosyncratic Volatility Stock Portfolios Out-Of-Sample

The low idiosyncratic volatility stocks selected in-sample also outperform the high volatility stocks in the out-of-sample period.

```
> # Sort stocks in-sample by their volatilities
> riskretis <- riskretis[order(riskretis[, "ivol"]), ]
> head(riskretis)
> symbolv <- rownames(riskretis)
> # Calculate the out-of-sample returns of low and high volatility s
> volow <- rowMeans(retsp[outsample, symbolv[1:(nstocks %/% 2)]])
> volhigh <- rowMeans(retsp[outsample, symbolv[(nstocks %/% 2):nstoc
> wealthv <- cbind(volow, volhigh, volow - 0.3*volhigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnames(wealthv) <- c("low_vol", "high_vol", "long_short")
```




```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high volatility stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Idiosy
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

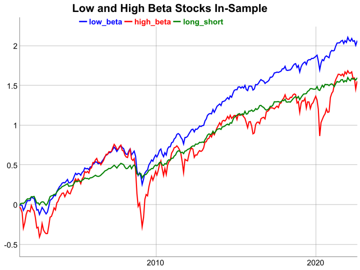
Low and High Beta Stock Portfolios

Research by NYU professors Andrea Frazzini and Lasse Heje Pedersen has shown that contrary to the *CAPM* model, low beta stocks have outperformed high beta stocks.

The low beta stocks are mostly from defensive stock sectors, like consumer staples, healthcare, etc., which investors buy when they fear a market downturn.

The strategy of investing in low beta stocks and shorting high beta stocks is known as betting against beta.

```
> # Sort stocks by their betas
> riskret <- riskret[order(riskret[, "beta"]), ]
> head(riskret)
> symbolv <- rownames(riskret)
> # Calculate the cumulative returns of low and high beta stocks
> betalow <- rowMeans(retsp[, symbolv[1:(nstocks %/% 2)]])
> betahigh <- rowMeans(retsp[, symbolv[(nstocks %/% 2):nstocks]])
> wealthv <- cbind(betalow, betahigh, betalow - 0.3*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("low_beta", "high_beta", "long_short")
```



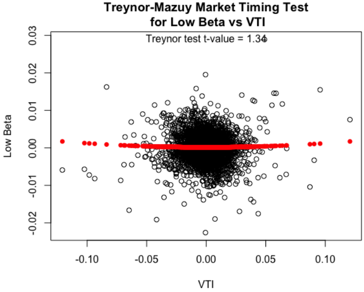

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high beta stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Beta S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

Low Beta Stock Portfolio Market Timing Skill

Market timing skill is the ability to forecast the direction and magnitude of market returns.

The *Treynor-Mazuy* test shows that the *betting against beta* strategy does not have significant *market timing* skill.

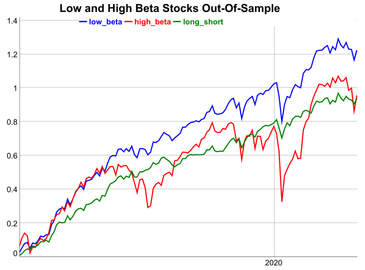
```
> # Merton-Henriksson test
> predictor <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predictor)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predictor); s
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predictor);
> # Plot residual scatterplot
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="Low Beta")
> title(main="Treynor-Mazuy Market Timing Test\n for Low Beta vs VTI
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=max(residv), paste("Treynor test t-value =", tvalue))
```



Low and High Beta Stock Portfolios Out-Of-Sample

The low beta stocks selected in-sample also outperform the high beta stocks in the out-of-sample period.

```
> # Sort stocks in-sample by their betas
> riskretis <- riskretis[order(riskretis[, "beta"]), ]
> head(riskretis)
> symbolv <- rownames(riskretis)
> # Calculate the out-of-sample returns of low and high beta stocks
> betalow <- rowMeans(retsp[outsample, symbolv[1:(nstocks %/% 2)]])
> betahigh <- rowMeans(retsp[outsample, symbolv[(nstocks %/% 2):nsto
> wealthv <- cbind(betalow, betahigh, betalow - 0.3*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnames(wealthv) <- c("low_beta", "high_beta", "long_short")
```




```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative returns of low and high beta stocks
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Low and High Beta S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

Momentum Portfolio Weights

The portfolio weights of *momentum* strategies can be calculated based on the past performance of the assets in many different ways:

- Invest equal dollar amounts in the top `n` best performing stocks and short the `n` worst performing stocks,
- Invest dollar amounts proportional to the past performance - purchase stocks with positive performance, and short stocks with negative performance,
- Subtract the weights mean so that their sum is equal to 0: $\sum_{i=1}^{n} w_i = 0$,
- Scale the weights so that the sum of squares is equal to 1: $\sum_{i=1}^{n} w_i^2 = 1$,

De-meaning the weights reduces the portfolio market *beta*.

Scaling the weights reduces the portfolio *leverage*.

```
> # Calculate the log percentage returns
> retsp <- rutils::diffit(log(prices))
> # Define performance objective function as sum of returns
> objfun <- function(retsp) sum(retsp)
> # Define performance objective function as Sharpe ratio
> objfun <- function(rets) sum(rets)/sd(rets)
> # Calculate performance statistics over look-back intervals
> retsis <- retsp[endp[1]:endp[2]]
> perfstat <- sapply(retsis, objfun)
> perfstat[!is.finite(perfstat)] <- 0
> sum(is.na(perfstat))
> # Calculate the best and worst performing stocks
> perfstat <- sort(perfstat, decreasing=TRUE)
> nstocks <- 10
> symbolb <- names(head(perfstat, nstocks))
> symbolw <- names(tail(perfstat, nstocks))
> # Calculate equal weights for the best and worst performing stocks
> weightv <- numeric(NCOL(retsp))
> names(weightv) <- colnames(retsp)
> weightv[symbolb] <- 1
> weightv[symbolw] <- (-1)
> # Calculate weights proportional to performance statistic
> weightv <- perfstat
> # Center weights so sum is equal to 0
> weightv <- weightv - mean(weightv)
> # Scale weights so sum of squares is equal to 1
> weightv <- weightv/sqrt(sum(weightv^2))
> # Calculate the momentum portfolio returns
> retsportf <- retsp %*% weightv
> # Scale weights so in-sample portfolio volatility is same as equal
> scalef <- sd(rowMeans(retsis))/sd(retsportf)
> weightv <- scalef*weightv
```

Rolling Momentum Strategy

In a *rolling momentum strategy*, the portfolio is rebalanced periodically and held out-of-sample.

Momentum strategies can be *backtested* by specifying the portfolio rebalancing frequency, the formation period, and the holding period:

- Specify a portfolio of stocks and their returns,
- Calculate the *end points* for portfolio rebalancing,
- Define an objective function for calculating the past performance of the stocks,
- Calculate the past performance over the *look-back* formation intervals,
- Calculate the portfolio weights from the past (in-sample) performance,
- Calculate the out-of-sample momentum strategy returns by applying the portfolio weights to the future returns,
- Apply a volatility scaling factor to the out-of-sample returns,
- Calculate the transaction costs and subtract them from the strategy returns.

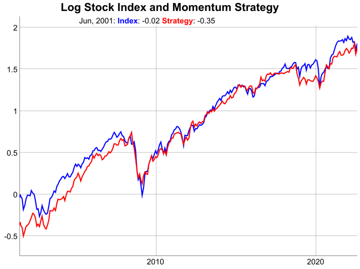
```
> # Calculate a vector of monthly end points
> endp <- rutils::calc_endpoints(retsp, interval="months")
> npts <- NROW(endp)
> # Perform loop over the end points
> nstocks <- 10
> look_back <- 8
> pnls <- lapply(2:(npts-1), function(ep) {
+   # Select the look-back returns
+   startp <- endp[max(1, ep-look_back)]
+   retsis <- retsp[startp:endp[ep], ]
+   # Calculate the best and worst performing stocks in-sample
+   perfstat <- sapply(retsis, objfun)
+   perfstat[!is.finite(perfstat)] <- 0
+   perfstat <- sort(perfstat, decreasing=TRUE)
+   symbolb <- names(head(perfstat, nstocks))
+   symbolw <- names(tail(perfstat, nstocks))
+   # Calculate the momentum weights
+   weightv <- numeric(NCOL(retsp))
+   names(weightv) <- colnames(retsp)
+   weightv[symbolb] <- 1
+   # weightv[symbolw] <- (-1)
+   # Calculate the in-sample portfolio returns
+   retsportf <- retsis %*% weightv
+   # Scale weights so in-sample portfolio volatility is same as equ
+   weightv <- weightv*sd(rowMeans(retsis))/sd(retsportf)
+   # Calculate the momentum portfolio returns
+   retsportf <- retsp[(endp[ep]+1):endp[ep+1], ] %*% weightv
+   rowMeans(retsportf)
+ })  # end lapply
> pnls <- rutils::do_call(c, pnls)
```

Performance of Momentum Strategy for Stocks

The momentum strategy for stocks produces a similar absolute return as the index, and also a similar Sharpe ratio.

The momentum strategy may be improved by a better choice of the model parameters: the length of look-back interval and the number of stocks.

```
> # Calculate the average of all stock returns
> indeks <- rowMeans(retsp)
> indeks <- xts::xts(indeks, order.by=datev)
> colnames(indeks) <- "Index"
> # Add initial startup interval returns
> pnls <- c(rowMeans(retsp[endp[1]:endp[2], ]), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])
```




```
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red")
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Log Stock Index and
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Momentum Strategy Functional

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function btmomtop() simulates (backtests) a *momentum strategy* which buys equal dollar amounts of the best performing stocks.

The function btmomtop() can be used to find the best choice of *momentum strategy* parameters.

```
> btmomtop <- function(rets,
+   objfun=function(rets) (sum(rets)/sd(rets)),
+   look_back=12, rfreq="months", nstocks=10, bid_offer=0.001,
+   endp=rutils::calc_endpoints(rets, interval=rfreq), ...) {
+   # Perform loop over end points
+   npts <- NROW(endp)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endp[max(1, ep-look_back)]
+     retsis <- retsp[startp:endp[ep], ]
+     # Calculate the best and worst performing stocks in-sample
+     perfstat <- sapply(retsis, objfun)
+     perfstat[!is.finite(perfstat)] <- 0
+     perfstat <- sort(perfstat, decreasing=TRUE)
+     symbolb <- names(head(perfstat, nstocks))
+     symbolw <- names(tail(perfstat, nstocks))
+     # Calculate the momentum weights
+     weightv <- numeric(NCOL(retsp))
+     names(weightv) <- colnames(retsp)
+     weightv[symbolb] <- 1
+     # weightv[symbolw] <- (-1)
+     # Calculate the in-sample portfolio returns
+     retsportf <- retsis %*% weightv
+     # Scale weights so in-sample portfolio volatility is same as e
+     weightv <- weightv*sd(rowMeans(retsis))/sd(retsportf)
+     # Calculate the momentum portfolio returns
+     retsportf <- retsp[(endp[ep]+1):endp[ep+1], ] %*% weightv
+     rowMeans(retsportf)
+   })  # end lapply
+   pnls <- rutils::do_call(c, pnls)
+   pnls
+ }  # end btmomtop
```

Optimization of Momentum Strategy Parameters

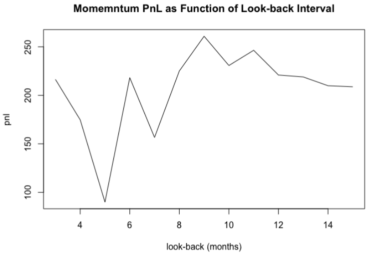
The performance of the *momentum* strategy depends on the length of the *look-back interval* used for calculating the past performance.

Research indicates that the optimal length of the *look-back interval* for momentum is about 8 to 12 months.

The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

Performing many *backtests* on multiple trading strategies risks identifying inherently unprofitable trading strategies as profitable, purely by chance (known as *p-value hacking*).

```
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, by=1)
> endp <- rutils::calc_endpoints(retsp, interval="months")
> pnlsl <- lapply(look_backs, btmomtop, rets=retsp, endp=endp, objfun=objfun)
> # Or perform parallel loop under Mac-OSX or Linux
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnlsl <- mclapply(look_backs, btmomtop, rets=retsp, endp=endp, objfun=objfun, mc.cores=ncores)
> profilev <- sapply(pnlsl, function(pnl) sum(pnl)/sd(pnl))
```




```
> # Plot Momentum profile
> plot(x=look_backs, y=profilev, t="l",
+   main="Momentum PnL as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="pnl")
```

Optimal Momentum Strategy for Stocks

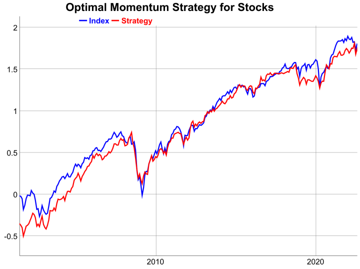
The momentum strategy for stocks produces a similar absolute return as the index, and also a similar Sharpe ratio.

But using a different rebalancing frequency in the *backtest* can produce different values for the optimal trading strategy parameters.

The *backtesting* redefines the problem of finding (tuning) the optimal trading strategy parameters, into the problem of finding the optimal *backtest* (meta-model) parameters.

But the advantage of using the *backtest* meta-model is that it can reduce the number of parameters that need to be optimized.

```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(profilev)
> look_backs[whichmax]
> pnls <- pnlsl[[whichmax]]
> pnls <- c(rowMeans(retsp[endp[1]:endp[2], ]), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Optimal Momentum St
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Weighted Momentum Strategy Functional

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function btmomweight() simulates (backtests) a *momentum strategy* which buys dollar amounts proportional to the past performance of the stocks.

The function btmomweight() can be used to find the best choice of *momentum strategy* parameters.

```
> btmomweight <- function(rets,
+   objfun=function(rets) (sum(rets)/sd(rets)),
+   look_back=12, rfreq="months", bid_offer=0.001,
+   endp=rutils::calc_endpoints(rets, interval=rfreq), ...) {
+   # Perform loop over end points
+   npts <- NROW(endp)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endp[max(1, ep-look_back)]
+     retsis <- rets[startp:endp[ep], ]
+     # Calculate weights proportional to performance
+     perfstat <- sapply(retsis, objfun)
+     perfstat[!is.finite(perfstat)] <- 0
+     weightv <- perfstat
+     # Calculate the in-sample portfolio returns
+     retsportf <- retsis %*% weightv
+     # Scale weights so in-sample portfolio volatility is same as e
+     weightv <- weightv*sd(rowMeans(retsis))/sd(retsportf)
+     # Calculate the momentum portfolio returns
+     rets[(endp[ep]+1):endp[ep+1], ] %*% weightv
+   })  # end lapply
+   rutils::do_call(c, pnls)
+ }  # end btmomweight
```

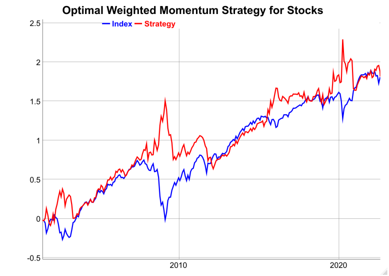
Optimal Momentum Strategy for Stocks

The momentum strategy for stocks produces a similar absolute return as the index, and also a similar Sharpe ratio.

But using a different rebalancing frequency in the *backtest* can produce different values for the optimal trading strategy parameters.

The *backtesting* redefines the problem of finding (tuning) the optimal trading strategy parameters, into the problem of finding the optimal *backtest* (meta-model) parameters.

But the advantage of using the *backtest* meta-model is that it can reduce the number of parameters that need to be optimized.




```
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Optimal Weighted Mo
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

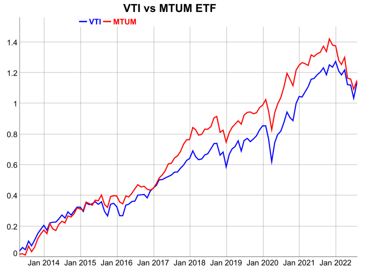
```
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, by=1)
> pnlsl <- lapply(look_backs, btmomweight, rets=retsp, endp=endp, o
> # Or perform parallel loop under Mac-OSX or Linux
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnlsl <- mclapply(look_backs, btmomweight, rets=retsp, endp=endp, objfun=objfun, mc.cores=ncores)
> profilev <- sapply(pnlsl, function(pnl) sum(pnl)/sd(pnl))
> # Plot Momentum profile
> plot(x=look_backs, y=profilev, t="l",
+   main="Momentum PnL as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="pnl")
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(profilev)
> look_backs[whichmax]
> pnls <- pnlsl[[whichmax]]
> pnls <- c(rowMeans(retsp[endp[1]:endp[2], ]), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
```

The MTUM Momentum ETF

The *MTUM* ETF is an actively managed ETF which follows a momentum strategy for stocks.

The *MTUM* ETF has a slightly higher absolute return than the *VTI* ETF, but it has a slightly lower Sharpe ratio.

```
> # Calculate the scaled prices of VTI vs MTUM ETF
> wealthv <- na.omit(rutils::etfenv$returns[, c("VTI", "MTUM")])
> colnames(wealthv) <- c("VTI", "MTUM")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the scaled prices of VTI vs MTUM ETF
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="VTI vs MTUM ETF") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(width=500)
```



Momentum Strategy for an *ETF* Portfolio

The performance of the *momentum* strategy depends on the length of the *look-back interval* used for calculating the past performance.

Research indicates that the optimal length of the *look-back interval* for momentum is about 4 to 10 months.

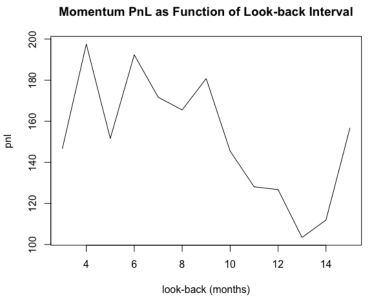
The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

Performing many *backtests* on multiple trading strategies risks identifying inherently unprofitable trading strategies as profitable, purely by chance (known as *p-value hacking*).

But using a different rebalancing frequency in the *backtest* can produce different values for the optimal trading strategy parameters.

So *backtesting* just redefines the problem of finding (tuning) the optimal trading strategy parameters, into the problem of finding the optimal *backtest* (meta-model) parameters.

But the advantage of using the *backtest* meta-model is that it can reduce the number of parameters that need to be optimized.



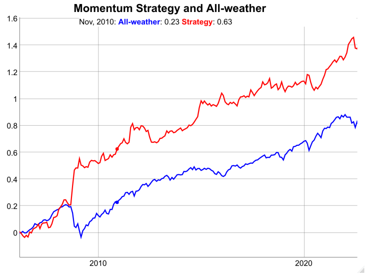

```
> # Extract ETF returns
> symbolv <- c("VTI", "IEF", "DBC")
> retsp <- rutils::etfenv$returns[, symbolv]
> retsp <- na.omit(retsp)
> datev <- zoo::index(retsp)
> # Calculate a vector of monthly end points
> endp <- rutils::calc_endpoints(retsp, interval="months")
> npts <- NROW(endp)
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, by=1)
> objfun <- function(retsp) sum(retsp)/sd(retsp)
> pnlsl <- lapply(look_backs, btmomweight, rets=retsp, endp=endp, ob
> profilev <- sapply(pnlsl, function(pnl) sum(pnl)/sd(pnl))
> # Plot Momentum PnLs
> plot(x=look_backs, y=profilev, t="l",
+   main="Momentum PnL as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="pnl")
```

Performance of Momentum Strategy for ETFs

The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

The momentum strategy for ETFs also has a very low correlation to the static *All-Weather* portfolio.

```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(profilev)
> look_backs[whichmax]
> pnls <- pnlsl[[whichmax]]
> pnls <- c(rowMeans(retsp[endp[1]:endp[2], ]), pnls)
> # Define all-weather benchmark
> weightsaw <- c(0.30, 0.55, 0.15)
> all_weather <- retsp %*% weightsaw
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(all_weather, pnls)
> cor(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("All-weather", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Momentum Strategy a
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

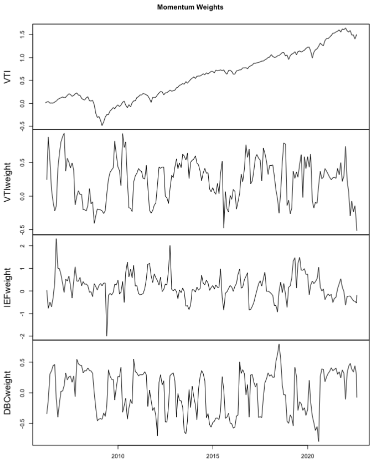
Time Series of Momentum Portfolio Weights

In *momentum* strategies, the portfolio weights are adjusted over time to be proportional to the past performance of the assets.

This way *momentum* strategies switch their weights to the best performing assets.

The weights are scaled to limit the portfolio *leverage* and its market *beta*.

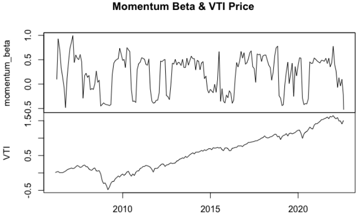
```
> # Calculate the momentum weights
> look_back <- look_backs[whichmax]
> weightv <- lapply(1:(npts-1), function(ep) {
+   # Select the look-back returns
+   startp <- endp[max(1, ep-look_back)]
+   retsis <- retsp[startp:endp[ep], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retsis, objfun)
+   weightv <- drop(perfstat)
+   # Scale weights so in-sample portfolio volatility is same as equ
+   retsportf <- retsis %*% weightv
+   weightv*sd(rowMeans(retsis))/sd(retsportf)
+ })  # end lapply
> weightv <- rutils::do_call(rbind, weightv)
> # Plot the momentum weights
> retvti <- cumsum(retsp$VTI)
> datav <- cbind(retvti[endp], weightv)
> colnames(datav) <- c("VTI", paste0(colnames(retsp), "weight"))
> zoo::plot.zoo(datav, xlab=NULL, main="Momentum Weights")
```



Momentum Strategy Market Beta

The *momentum* strategy market beta can be calculated by multiplying the *ETF* betas by the *ETF* portfolio weights.

```
> # Calculate ETF betas
> betas_etf <- sapply(retsp, function(x)
+   cov(retsp$VTI, x)/var(retsp$VTI))
> # Momentum beta is equal weights times ETF betas
> betas <- weightv %*% betas_etf
> betas <- xts::xts(betas, order.by=datev[endp])
> colnames(betas) <- "momentum_beta"
> datav <- cbind(betas, retvti[endp])
> zoo::plot.zoo(datav, main="Momentum Beta & VTI Price", xlab="")
```

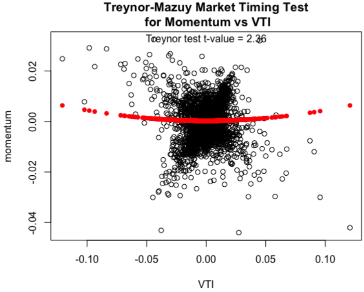


Momentum Strategy Market Timing Skill

Market timing skill is the ability to forecast the direction and magnitude of market returns.

The *Treynor-Mazuy* test shows that the *momentum* strategy has some *market timing* skill.

```
> # Merton-Henriksson test
> retvti <- retsp$VTI
> predictor <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predictor)[2:3] <- c("merton", "treynor")
> regmod <- lm(pnls ~ VTI + merton, data=predictor); summary(regmod)
> # Treynor-Mazuy test
> regmod <- lm(pnls ~ VTI + treynor, data=predictor); summary(regmod
> # Plot residual scatterplot
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="momentum")
> title(main="Treynor-Mazuy Market Timing Test\n for Momentum vs VTI
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=max(residv), paste("Treynor test t-value =", tvalue))
```



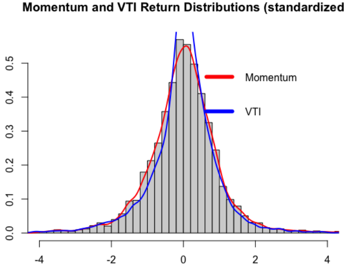
Skewness of Momentum Strategy Returns

Most assets with *positive returns* suffer from *negative skewness*.

The *momentum* strategy returns have more positive skewness compared to the negative skewness of *VTI*.

The *momentum* strategy is a genuine *market anomaly*, because it has both positive returns and positive skewness.

```
> # Standardize the returns
> pnlsd <- (pnls-mean(pnls))/sd(pnls)
> retvti <- (retvti-mean(retvti))/sd(retvti)
> # Calculate skewness and kurtosis
> apply(cbind(pnlsd, retvti), 2, function(x)
+   sapply(c(skew=3, kurt=4),
+     function(e) sum(x^e)))/NROW(retvti)
```




```
> # Plot histogram
> hist(pnlsd, breaks=80,
+   main="Momentum and VTI Return Distributions (standardized",
+   xlim=c(-4, 4), xlab="", ylab="", freq=FALSE)
> # Draw kernel density of histogram
> lines(density(pnlsd), col='red', lwd=2)
> lines(density(retvti), col='blue', lwd=2)
> # Add legend
> legend("topright", inset=0.05, cex=1.0, title=NULL,
+  leg=c("Momentum", "VTI"), bty="n",
+  lwd=6, bg="white", col=c("red", "blue"))
```

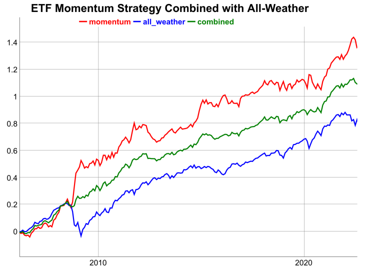
Combining Momentum with the *All-Weather* Portfolio

The *momentum* strategy has attractive returns compared to a static buy-and-hold strategy.

But the *momentum* strategy suffers from draw-downs called *momentum crashes*, especially after the market rallies from a sharp-sell-off.

This suggests that combining the *momentum* strategy with a static buy-and-hold strategy can achieve significant diversification of risk.

```
> # Combine momentum strategy with all-weather
> wealthv <- cbind(pnls, all_weather, 0.5*(pnls + all_weather))
> colnames(wealthv) <- c("momentum", "all_weather", "combined")
> wealthv <- xts::xts(wealthv, datev)
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))
> # Calculate strategy correlations
> cor(wealthv)
```




```
> # Plot ETF momentum strategy combined with All-Weather
> dygraphs::dygraph(cumsum(wealthv)[endp], main="ETF Momentum Strate
+   dyOptions(colors=c("red", "blue", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Or
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("green", "blue", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme,
+        name="ETF Momentum Strategy Combined with All-Weather")
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```

Momentum Strategy With Daily Rebalancing

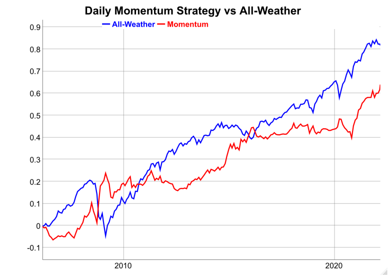
In a momentum strategy with *daily rebalancing*, the weights are updated every day and the portfolio is rebalanced accordingly.

A momentum strategy with *daily rebalancing* requires more computations so compiled C++ functions are preferred to apply() loops.

The package *roll* contains extremely fast functions for calculating rolling aggregations using compiled C++ code.

The momentum strategy with *daily rebalancing* performs worse than the strategy with *monthly rebalancing* because of the daily variance of the weights.

```
> # Calculate rolling variance
> look_back <- 152
> variance <- roll::roll_var(retsp, width=look_back, min_obs=1)
> variance[1, ] <- variance[2, ]
> variance[variance <= 0] <- 0
> # Calculate rolling Sharpe
> perfstat <- roll::roll_mean(retsp, width=look_back, min_obs=1)
> weightv <- perfstat/sqrt(variance)
> weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weightv)
> sum(is.na(weightv))
> # Calculate momentum profits and losses
> pnls <- rowSums(weightv*retsp)
```




```
> # Calculate transaction costs
> bid_offer <- 0.0
> costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costs)
> # Define all-weather benchmark
> weightsaw <- c(0.30, 0.55, 0.15)
> all_weather <- retsp %*% weightsaw
> # Scale the momentum volatility to all_weather
> pnls <- sd(all_weather)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- xts::xts(cbind(all_weather, pnls), order.by=datev)
> colnames(wealthv) <- c("All-Weather", "Momentum")
> cor(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the momentum strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Daily Momentum Stra
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Daily Momentum Strategy Functional

The function `btmomdaily()` simulates a momentum strategy with *daily rebalancing*.

A momentum strategy with *daily rebalancing* requires more computations so compiled `C++` functions are preferred to `apply()` loops.

The package *roll* contains extremely fast functions for calculating rolling aggregations using compiled `C++` code.

The momentum strategy with *daily rebalancing* performs worse than the strategy with *monthly rebalancing* because of the daily variance of the weights.

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function `btmomweight()` can be used to find the best choice of *momentum strategy* parameters.

```
> # Define backtest functional for daily momentum strategy
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdaily <- function(rets, look_back=252, bid_offer=0.001, trend
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate rolling variance
+   variance <- roll::roll_var(rets, width=look_back, min_obs=1)
+   variance[1, ] <- 1
+   variance[variance <= 0] <- 1
+ # Calculate rolling Sharpe
+   perfstat <- roll::roll_mean(rets, width=look_back, min_obs=1)
+   weights <- perfstat/sqrt(variance)
+   weights <- weights/sqrt(rowSums(weights^2))
+   weights <- rutils::lagit(weights)
+   # Calculate momentum profits and losses
+   pnls <- trend*rowSums(weights*rets)
+   # Calculate transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weights)))
+   (pnls - costs)
+ }  # end btmomdaily
```

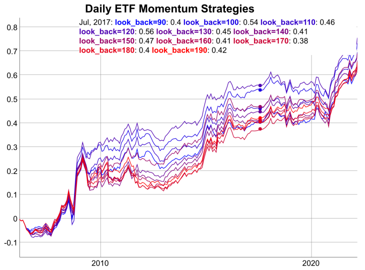
Multiple Daily ETF Momentum Strategies

Multiple daily ETF *momentum* strategies can be backtested by calling the function btmomdaily() in a loop over a vector of *look-back* parameters.

The best performing daily ETF *momentum* strategies are with *look-back* parameters between 100 and 120 days.

The *momentum* strategies do not perform well, especially the ones with a long *look-back* parameter.

```
> # Simulate a daily ETF momentum strategy
> source("/Users/jerzy/Develop/lecture_slides/scripts/back_test.R")
> pnls <- btmomdaily(rets=retsp, look_back=152,
+   bid_offer=bid_offer)
> # Perform sapply loop over look_backs
> look_backs <- seq(90, 190, by=10)
> pnls <- sapply(look_backs, btmomdaily,
+   rets=retsp, bid_offer=bid_offer)
> # Scale the momentum volatility to all_weather
> pnls <- apply(pnls, MARGIN=2,
+   function(pnl) sd(all_weather)*pnl/sd(pnl))
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, zoo::index(retsp))
> tail(pnls)
```




```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Daily ETF Momentum Str
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot daily ETF momentum strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endp],
+   theme=plot_theme, name="Cumulative Returns of Daily ETF Momentu
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retsp)),
+   col=plot_theme$col$line.col, bty="n")
```

Daily Momentum Strategy with Holding Period

The daily ETF momentum strategy can be improved by introducing a *holding period* for the portfolio.

Instead of holding the portfolio for only a day, its held for several days and then liquidated. So several portfolios are held at the same time.

This is equivalent to averaging the portfolio weights over several days from the past.

The best length of the *holding period* depends on the *bias-variance tradeoff*.

If the *holding period* is too short then the weights have too much day-over-day *variance*.

If the *holding period* is too long then the weights have too much *bias* (they are stale).

The optimal length of the *holding period* can be determined by cross-validation (backtesting).

The function `btmomdailyhold()` simulates a momentum strategy with *daily rebalancing* with a holding period.

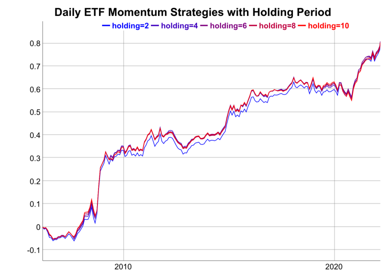
```
> # Define backtest functional for daily momentum strategy
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdailyhold <- function(rets, look_back=252, holdp=5, bid_offer
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate rolling variance
+   variance <- roll::roll_var(rets, width=look_back, min_obs=1)
+   variance[1, ] <- 1
+   variance[variance <= 0] <- 1
+   # Calculate rolling Sharpe
+   perfstat <- roll::roll_mean(rets, width=look_back, min_obs=1)
+   weightv <- perfstat/sqrt(variance)
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   # Average the weights over holding period
+   weightv <- roll::roll_mean(weightv, width=holdp, min_obs=1)
+   weightv <- rutils::lagit(weightv)
+   # Calculate momentum profits and losses
+   pnls <- trend*rowSums(weightv*rets)
+   # Calculate transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
+   (pnls - costs)
+ }  # end btmomdailyhold
```

Daily Momentum Strategy with Holding Period

Multiple daily ETF *momentum* strategies can be backtested by calling the function btmomdaily() in a loop over a vector of holding periods.

The daily *momentum* strategies with a holding period perform much better.

```
> # Perform sapply loop over holding periods
> holdpv <- seq(2, 11, by=2)
> pnls <- sapply(holdpv, btmomdailyhold, look_back=100,
+              rets=retsp, bid_offer=bid_offer)
> # Scale the momentum volatility to all_weather
> pnls <- apply(pnls, MARGIN=2,
+   function(pnl) sd(all_weather)*pnl/sd(pnl))
> colnames(pnls) <- paste0("holding=", holdpv)
> pnls <- xts::xts(pnls, zoo::index(retsp))
```



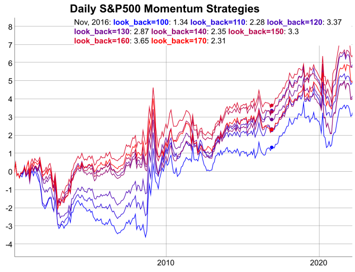

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Daily ETF Momentum Str
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot daily ETF momentum strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endp],
+   theme=plot_theme, name="Cumulative Returns of Daily ETF Momentu
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retsp)),
+   col=plot_theme$col$line.col, bty="n")
```

Backtesting Multiple S&P500 Momentum Strategies

Multiple *S&P500 momentum* strategies can be backtested by calling the function btmomdaily() in a loop over a vector of *look-back* parameters.

The best performing daily *S&P500 momentum* strategies are with *look-back* parameters between 120 and 160 days.

The *momentum* strategies do not perform well, especially the ones with a short *look-back* parameter.




```
> # Load daily S&P500 percentage stock returns.
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> # Overwrite NA values in returns100
> retsp <- returns100["2000/"]
> retsp[1, is.na(retsp[1, ])] <- 0
> retsp <- zoo::na.locf(retsp, na.rm=FALSE)
> # Simulate a daily S&P500 momentum strategy.
> # Perform sapply loop over look_backs
> look_backs <- seq(100, 170, by=10)
> pnls <- sapply(look_backs, btmomdailyhold,
+   holdp=5, rets=retsp, bid_offer=0)
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, zoo::index(retsp))
```

```
> # Calculate a vector of monthly end points
> endp <- rutils::calc_endpoints(retsp, interval="months")
> # Plot dygraph of daily S&P500 momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Daily S&P500 Momentum
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot daily S&P500 momentum strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorRampPalette(c("blue", "red"))(NCOI
> quantmod::chart_Series(cumsum(pnls)[endp],
+   theme=plot_theme, name="Daily S&P500 Momentum Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retsp)),
+   col=plot_theme$col$line.col, bty="n")
```

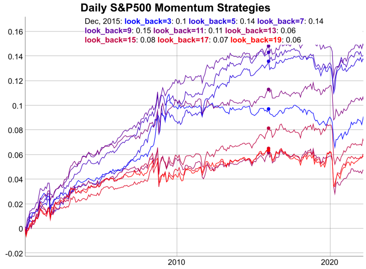
Backtesting Multiple S&P500 *Mean Reverting* Strategies

Multiple *S&P500 mean reverting* strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of *look-back* parameters.

The *mean reverting* strategies for the *S&P500* constituents perform the best for short *look-back* parameters.

The *mean reverting* strategies had their best performance prior to the 2008 financial crisis.

```
> # Perform sapply loop over look_backs
> look_backs <- seq(3, 20, by=2)
> pnls <- sapply(look_backs, btmomdaily,
+   rets=retsp, bid_offer=0, trend=(-1))
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, zoo::index(retsp))
```



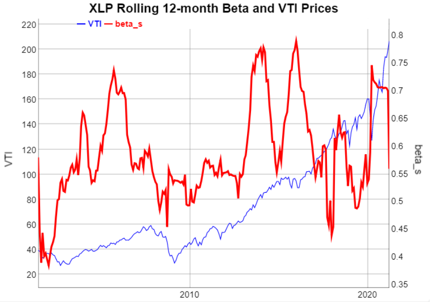

```
> # Plot dygraph of daily S&P500 momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Daily S&P500 Momentum
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot daily S&P500 momentum strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorRampPalette(c("blue", "red"))(NCOI
> quantmod::chart_Series(cumsum(pnls)[endp],
+   theme=plot_theme, name="Cumulative Returns of S&P500 Mean Revert
> legend("topleft", legend=colnames(pnls),
+   inset=0.05, bg="white", cex=0.7, lwd=rep(6, NCOL(retsp)),
+   col=plot_theme$col$line.col, bty="n")
```

Rolling Beta Regressions Over Time

The rolling beta of *XLP* versus *VTI* changes over time, with lower beta in periods of *VTI* selloffs.

The function `roll_reg()` from package *HighFreq* performs rolling regressions in `C++` (*RcppArmadillo*), so it's therefore much faster than equivalent `R` code.

```
> # Calculate XLP and VTI returns
> retsp <- na.omit(rutils::etfenv$returns[, c("XLP", "VTI")])
> # Calculate monthly end points
> endp <- xts::endpoints(retsp, on="months")[-1]
> # Calculate start points from look-back interval
> look_back <- 12  # Look back 12 months
> startp <- c(rep(1, look_back), endp[1:(NROW(endp)-look_back)])
> head(cbind(endp, startp), look_back+2)
> # Calculate rolling beta regressions every month in R
> formulav <- XLP ~ VTI  # Specify regression formula
> betar <- sapply(1:NROW(endp), FUN=function(ep) {
+     datav <- retsp[startp[ep]:endp[ep], ]
+     # coef(lm(formulav, data=datav))[2]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   })  # end sapply
> # Calculate rolling betas using RcppArmadillo
> reg_stats <- HighFreq::roll_reg(response=retsp$XLP, retsp=retsp$V
> betas <- reg_stats$VTI
> all.equal(betas, betar)
> # Compare the speed of RcppArmadillo with R code
> library(microbenchmark)
> summary(microbenchmark(
+   Rcpp=HighFreq::roll_reg(response=retsp$XLP, retsp=retsp$VTI, en
+   Rcode=sapply(1:NROW(endp), FUN=function(ep) {
+     datav <- retsp[startp[ep]:endp[ep], ]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   }),
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```




```
> # dygraph plot of rolling XLP beta and VTI prices
> dates <- zoo::index(retsp[endp, ])
> pricev <- rutils::etfenv$prices$VTI[dates]
> datav <- cbind(pricev, betas)
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav, main="XLP Rolling 12-month Beta and VTI P
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

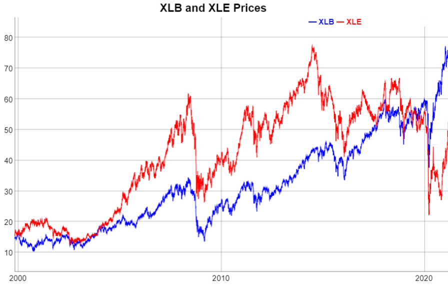
Engle-Granger Two-step Procedure for Cointegration

The *ADF* test can be applied to test for the cointegration of time series of prices.

The Engle-Granger two-step procedure for two time series consists of:

- Performing a regression to calculate the cointegrating factor β,
- Applying the *ADF* test to the residuals of the regression to determine that they don't have a unit root (they are mean reverting).

The regression of prices is not statistically valid because they are not normally distributed.




```
> # Calculate XLB and XLE prices
> pricev <- na.omit(rutils::etfenv$prices[, c("XLB", "XLE")])
> cor(rutils::diffit(log(pricev)))
> xlb <- drop(zoo::coredata(pricev$XLB))
> xle <- drop(zoo::coredata(pricev$XLE))
> # Calculate regression coefficients of XLB ~ XLE
> betav <- cov(xlb, xle)/var(xle)
> alpha <- (mean(xlb) - betav*mean(xle))
> # Calculate regression residuals
> fittedv <- (alpha + betav*xle)
> residuals <- (xlb - fittedv)
> # Perform ADF test on residuals
> tseries::adf.test(residuals, k=1)
```

```
> # Plot prices
> dygraphs::dygraph(pricev, main="XLB and XLE Prices") %>%
+   dyOptions(colors=c("blue", "red"))
> # Plot cointegration residuals
> residuals <- xts::xts(residuals, zoo::index(pricev))
> dygraphs::dygraph(residuals, main="XLB and XLE Cointegration Resic
```

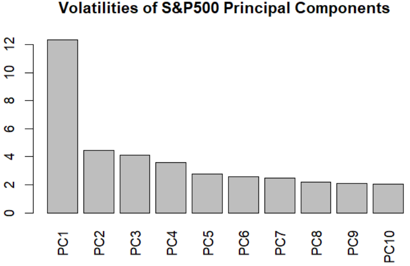
Principal Components of *S&P500* Stock Constituents

The *PCA* standard deviations are the volatilities of the *principal component* time series.

The original time series of returns can be calculated approximately from the first few *principal components* with the largest standard deviations.

The *Kaiser-Guttman* rule uses only *principal components* with *variance* greater than 1.

Another rule of thumb is to use the *principal components* with the largest standard deviations which sum up to 80% of the total variance of returns.




```
> # Load S&P500 constituent stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_prices.RData'
> # Calculate stock prices and percentage returns
> pricets <- zoo::na.locf(pricets, na.rm=FALSE)
> pricets <- zoo::na.locf(pricets, fromLast=TRUE)
> retsp <- rutils::diffit(log(pricev))
> # Standardize (de-mean and scale) the returns
> retsp <- lapply(retsp, function(x) {(x - mean(x))/sd(x)})
> retsp <- rutils::do_call(cbind, retsp)
> # Perform principal component analysis PCA
> pcad <- prcomp(retsp, scale=TRUE)
> # Find number of components with variance greater than 2
> ncomp <- which(pcad$sdev^2 < 2)[1]
```

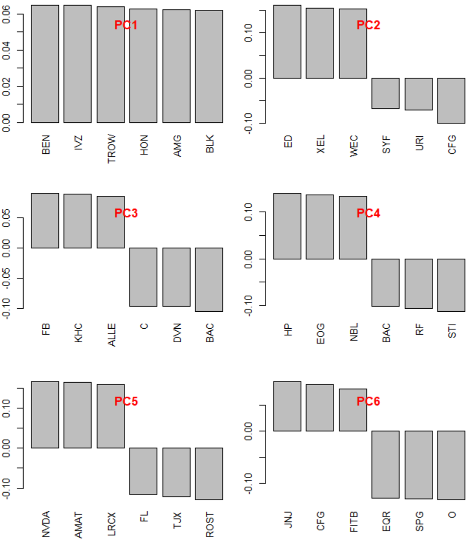
```
> # Plot standard deviations of principal components
> barplot(pcad$sdev[1:ncomp],
+   names.arg=colnames(pcad$rotation[, 1:ncomp]),
+   las=3, xlab="", ylab="",
+   main="Volatilities of S&P500 Principal Components")
```

S&P500 Principal Component Loadings (Weights)

Principal component loadings are the weights of *principal component* portfolios.

The *principal component* portfolios have mutually orthogonal returns represent the different orthogonal modes of the return variance.

```
> # Calculate principal component loadings (weights)
> # Plot barplots with PCA weights in multiple panels
> ncomps <- 6
> par(mfrow=c(ncomps/2, 2))
> par(mar=c(4, 2, 2, 1), oma=c(0, 0, 0, 0))
> # First principal component weights
> weights <- sort(pcad$rotation[, 1], decreasing=TRUE)
> barplot(weights[1:6], las=3, xlab="", ylab="", main="")
> title(paste0("PC", 1), line=-2.0, col.main="red")
> for (ordern in 2:ncomps) {
+   weights <- sort(pcad$rotation[, ordern], decreasing=TRUE)
+   barplot(weights[c(1:3, 498:500)], las=3, xlab="", ylab="", main=
+   title(paste0("PC", ordern), line=-2.0, col.main="red")
+ }  # end for
```

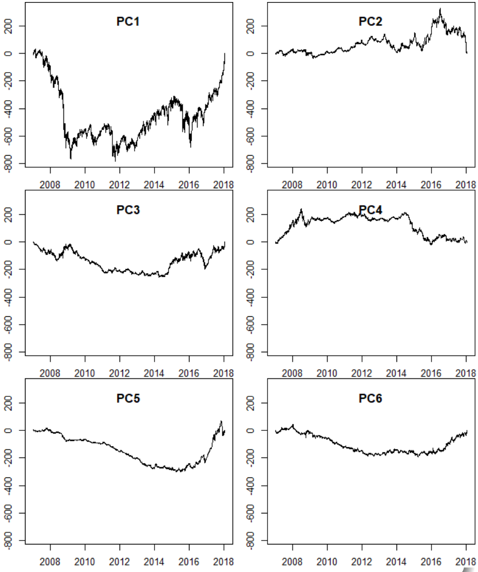


S&P500 Principal Component Time Series

The time series of the *principal components* can be calculated by multiplying the loadings (weights) times the original data.

Higher order *principal components* are gradually less volatile.

```
> # Calculate principal component time series
> retspca <- xts(retsp %*% pcad$rotation[, 1:ncomps],
+           order.by=dates)
> round(cov(retspca), 3)
> pcacum <- cumsum(retspca)
> # Plot principal component time series in multiple panels
> par(mfrow=c(ncomps/2, 2))
> par(mar=c(2, 2, 0, 1), oma=c(0, 0, 0, 0))
> rangev <- range(pcacum)
> for (ordern in 1:ncomps) {
+   plot.zoo(pcacum[, ordern], ylim=rangev, xlab="", ylab="")
+   title(paste0("PC", ordern), line=-2.0)
+ }  # end for
```



S&P500 Factor Model From *Principal Components*

By inverting the *PCA* analysis, the *S&P500* constituent returns can be calculated from the first k *principal components* under a *factor model*:

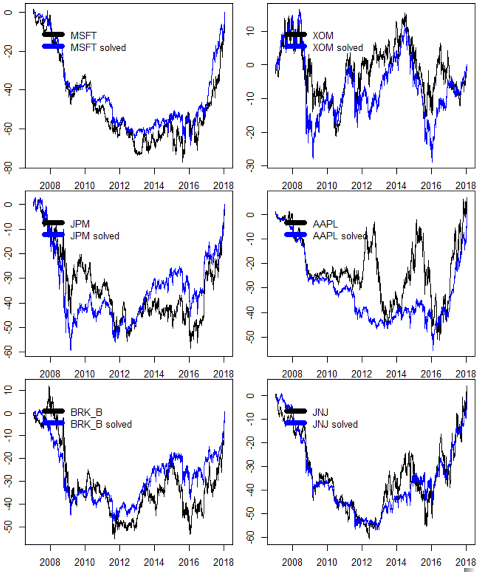
$$\mathbf{r}_i = \alpha_i + \sum_{j=1}^{k} \beta_{ji} \, \mathbf{F}_j + \varepsilon_i$$

The *principal components* are interpreted as *market factors*: $\mathbf{F}_j = \mathbf{pc}_j$.

The *market betas* are the inverse of the *principal component loadings*: $\beta_{ji} = w_{ij}$.

The ε_i are the *idiosyncratic* returns, which should be mutually independent and uncorrelated to the *market factor* returns.

```
> # Invert principal component time series
> invmat <- solve(pcad$rotation)
> all.equal(invmat, t(pcad$rotation))
> solved <- retspca %*% invmat[1:ncomps, ]
> solved <- xts::xts(solved, dates)
> solved <- cumsum(solved)
> retc <- cumsum(retsp)
> # Plot the solved returns
> symbolv <- c("MSFT", "XOM", "JPM", "AAPL", "BRK_B", "JNJ")
> for (symbol in symbolv) {
+   plot.zoo(cbind(retc[, symbol], solved[, symbol]),
+     plot.type="single", col=c("black", "blue"), xlab="", ylab="")
+   legend(x="topleft", bty="n",
+    legend=paste0(symbol, c("", " solved")),
+    title=NULL, inset=0.05, cex=1.0, lwd=6,
+    lty=1, col=c("black", "blue"))
+ }  # end for
```



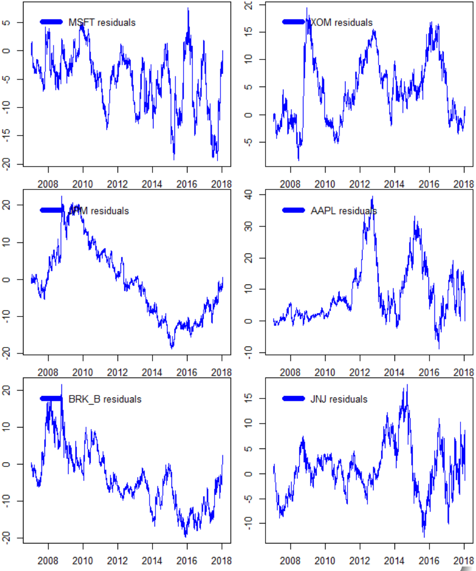
S&P500 Factor Model Residuals

The original time series of returns can be calculated exactly from the time series of all the *principal components*, by inverting the loadings matrix.

The original time series of returns can be calculated approximately from just the first few *principal components*, which demonstrates that *PCA* is a form of *dimension reduction*.

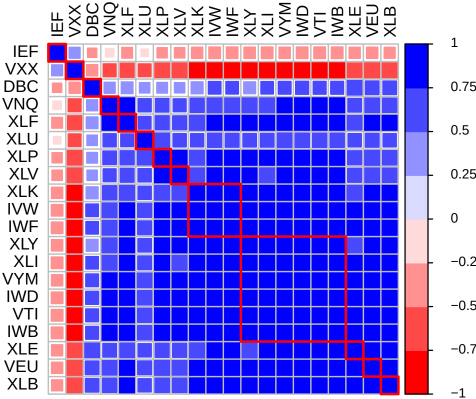
The function `solve()` solves systems of linear equations, and also inverts square matrices.

```
> # Perform ADF unit root tests on original series and residuals
> sapply(symbolv, function(symbol) {
+   c(series=tseries::adf.test(retc[, symbol])$p.value,
+     resid=tseries::adf.test(retc[, symbol] - solved[, symbol])$p.v
+ })  # end sapply
> # Plot the residuals
> for (symbol in symbolv) {
+   plot.zoo(retc[, symbol] - solved[, symbol],
+     plot.type="single", col="blue", xlab="", ylab="")
+   legend(x="topleft", bty="n", legend=paste(symbol, "residuals"),
+    title=NULL, inset=0.05, cex=1.0, lwd=6, lty=1, col="blue")
+ }  # end for
> # Perform ADF unit root test on principal component time series
> retspca <- xts(retsp %*% pcad$rotation, order.by=dates)
> pcacum <- cumsum(retspca)
> adf_pvalues <- sapply(1:NCOL(pcacum), function(ordern)
+   tseries::adf.test(pcacum[, ordern])$p.value)
> # AdF unit root test on stationary time series
> tseries::adf.test(rnorm(1e5))
```



Correlation and Factor Analysis

```
> ### Perform pair-wise correlation analysis
> # Calculate correlation matrix
> cormat <- cor(retsp)
> colnames(cormat) <- colnames(retsp)
> rownames(cormat) <- colnames(retsp)
> # Reorder correlation matrix based on clusters
> # Calculate permutation vector
> library(corrplot)
> ordern <- corrMatOrder(cormat, order="hclust",
+         hclust.method="complete")
> # Apply permutation vector
> cormat <- cormat[ordern, ordern]
> # Plot the correlation matrix
> colorv <- colorRampPalette(c("red", "white", "blue"))
> corrplot(cormat, tl.col="black", tl.cex=0.8,
+     method="square", col=colorv(8),
+     cl.offset=0.75, cl.cex=0.7,
+     cl.align.text="l", cl.ratio=0.25)
> # draw rectangles on the correlation matrix plot
> corrRect.hclust(cormat, k=NROW(cormat) %/% 2,
+           method="complete", col="red")
```



Hierarchical Clustering Analysis

The function `as.dist()` converts a matrix representing the *distance* (dissimilarity) between elements, into a list of class `"dist"`.

For example, `as.dist()` converts `(1-correlation)` to distance.

The function `hclust()` recursively combines elements into clusters based on their mutual *distance*.

First `hclust()` combines individual elements that are closest to each other.

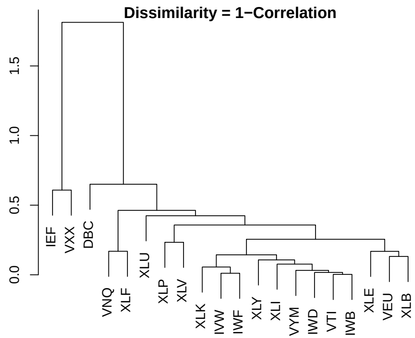
Then it combines elements to the closest clusters, then clusters with other clusters, until all elements are combined into one cluster.

This process of recursive clustering can be represented as a *dendrogram* (tree diagram).

Branches of a *dendrogram* represent clusters.

Neighboring branches contain elements that are close to each other (have small distance).

Neighboring branches combine into larger branches, that then combine with their closest branches, etc.




```
> # Convert correlation matrix into distance object
> distancev <- as.dist(1-cormat)
> # Perform hierarchical clustering analysis
> cluster <- hclust(distancev)
> plot(cluster, ann=FALSE, xlab="", ylab="")
> title("Dendrogram representing hierarchical clustering
+ \nwith dissimilarity = 1-correlation", line=-0.5)
```

Homework Assignment

Required

Study all the lecture slides in `FRE7241_Lecture_6.pdf`, and run all the code in `FRE7241_Lecture_6.R`

Recommended

- Read about *estimator shrinkage*:
  Aswani Regression Shrinkage Bias Variance Tradeoff.pdf
  Blei Regression Lasso Shrinkage Bias Variance Tradeoff.pdf
- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  DEoptim Introduction.pdf
  Ardia DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
  Mullen Package DEoptim.pdf
- Read about *momentum*:
  Bouchaud Momentum Mean Reversion Equity Returns.pdf