FRE7241 Algorithmic Portfolio Management

Lecture#5, Spring 2026

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

February 24, 2026

Brownian Motion

Brownian motion B_T is a stochastic process, with the increments $\mathrm{d}B_t$ which are independent and normally distributed, with mean zero and variance equal to $\mathrm{d}t$.

$$\mathrm{d}B_t = \xi_t \sqrt{\mathrm{d}t}$$

Where the ξ_t are random and independent *innovations* following the standard normal distribution $\phi(0, 1)$, with the expected values: $\mathbb{E}[\xi_t] = 0$, $\mathbb{E}[\xi_t^2] = 1$, and $\mathbb{E}[\xi_{t1}\xi_{t2}] = 0$.

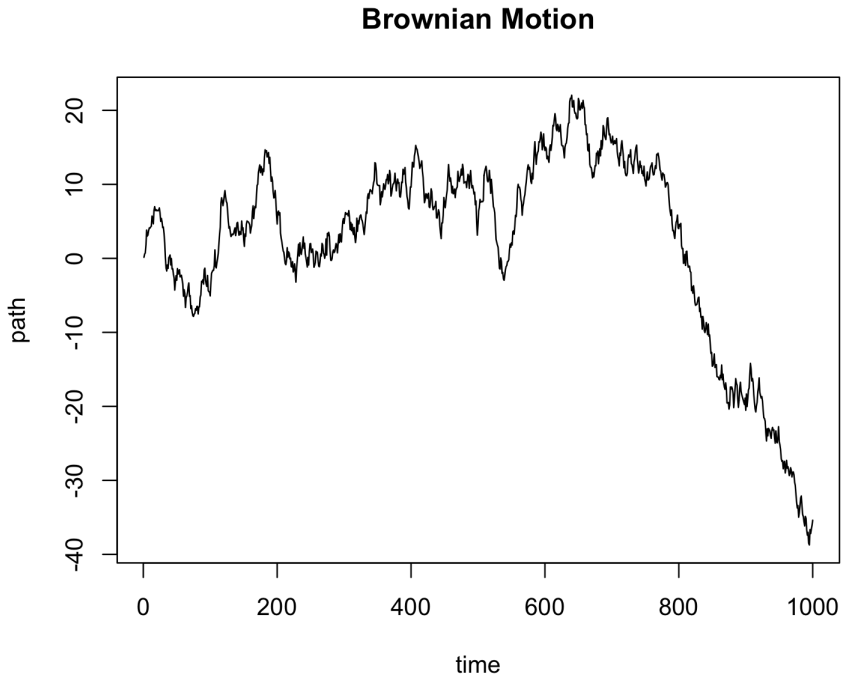
The Brownian motion path B_T is equal to the sum of its increments $\mathrm{d}B_t$:

$$B_T = \sum_{i=1}^{n} \mathrm{d}B_t$$

Where the number of time increments n is equal to the total time of evolution T divided by the increment size $\mathrm{d}t$: $n = T/\mathrm{d}t$.

The variance of Brownian motion is equal to the time of its evolution T:

$$\sigma^2 = \mathbb{E}[(\sum_{i=1}^{n} \xi_t \sqrt{\mathrm{d}t})^2] = \sum_{i=1}^{n} \mathbb{E}[\xi_t^2]\mathrm{d}t = T$$




```
> # Simulate a Brownian motion path
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> pathv <- cumsum(rnorm(nrows))
> plot(pathv, type="l", xlab="time", ylab="path",
+      main="Brownian Motion")
```

Geometric Brownian Motion

If the percentage stock returns $r_t \mathrm{d}t = \mathrm{d} \log p_t$ follow *Brownian motion*:

$$r_t \mathrm{d}t = \mathrm{d} \log p_t = (\mu - \frac{\sigma^2}{2})\mathrm{d}t + \sigma \, \mathrm{d}B_t$$

Then asset prices p_t follow *Geometric Brownian motion* (GBM):

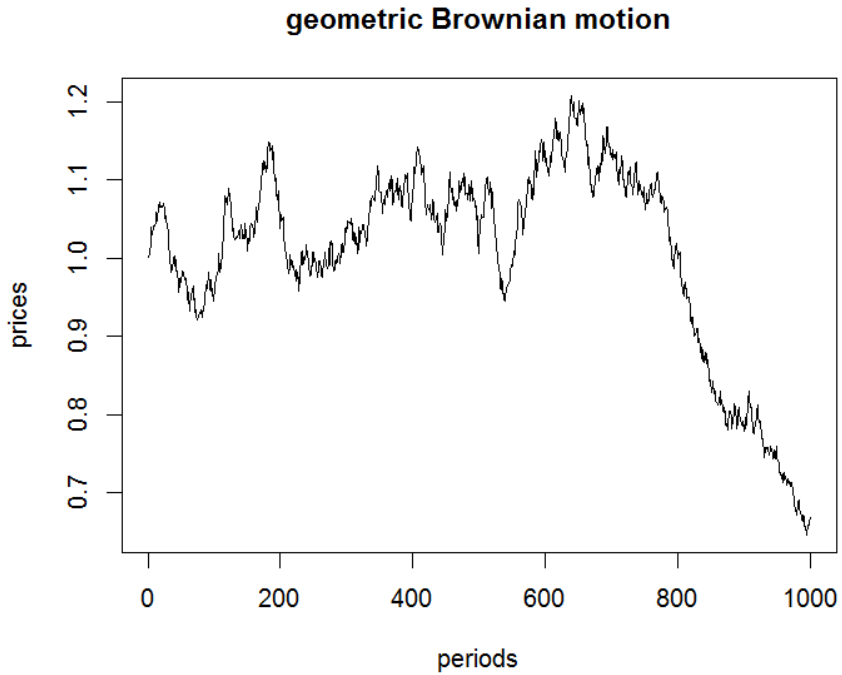
$$\mathrm{d}p_t = \mu p_t \mathrm{d}t + \sigma \, p_t \mathrm{d}B_t$$

Where σ is the volatility of stock returns, and B_t is a *Brownian Motion*, with $\mathrm{d}B_t$ following the normal distribution $\phi(0, \sqrt{\mathrm{d}t})$, with the volatility $\sqrt{\mathrm{d}t}$, equal to the square root of the time increment $\mathrm{d}t$.

The solution of *Geometric Brownian motion* is equal to:

$$p_t = p_0 \exp[(\mu - \frac{\sigma^2}{2})t + \sigma \, B_t]$$

The convexity correction: $-\frac{\sigma^2}{2}$ ensures that the growth rate of prices is equal to μ, (according to Ito's lemma).




```
> # Define the daily volatility and growth rate
> sigmav <- 0.01; drift <- 0.0; nrows <- 1000
> # Simulate geometric Brownian motion
> retp <- sigmav*rnorm(nrows) + drift - sigmav^2/2
> pricev <- exp(cumsum(retp))
> plot(pricev, type="l", xlab="time", ylab="prices",
+      main="Geometric Brownian Motion")
```

The *Log-normal* Probability Distribution

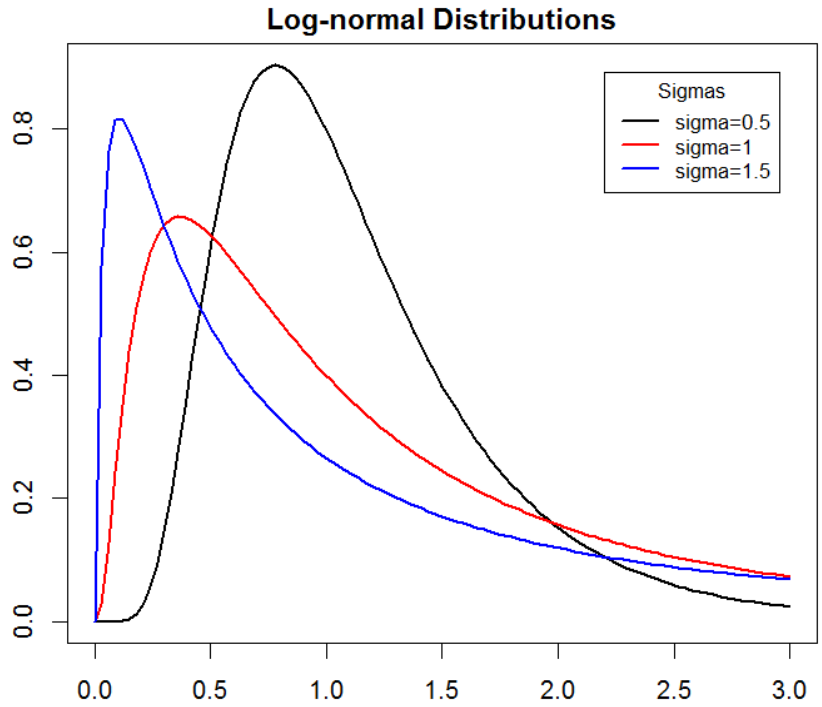
If x follows the *Normal* distribution $\phi(x, \mu, \sigma)$, then the exponential of x: $y = e^x$ follows the *Log-normal* distribution $\log \phi()$:

$$\log \phi(y, \mu, \sigma) = \frac{\exp(-(\log y - \mu)^2/2\sigma^2)}{y\sigma\sqrt{2\pi}}$$

With mean equal to: $\bar{y} = \mathbb{E}[y] = e^{(\mu+\sigma^2/2)}$, and median equal to: $\tilde{y} = e^{\mu}$

With variance equal to: $\sigma_y^2 = (e^{\sigma^2} - 1)e^{(2\mu+\sigma^2)}$, and skewness (third moment) equal to: $\varsigma = \mathbb{E}[(y - \mathbb{E}[y])^3] = (e^{\sigma^2} + 2)\sqrt{e^{\sigma^2} - 1}$

```
> # Standard deviations of log-normal distribution
> sigmavs <- c(0.5, 1, 1.5)
> # Create plot colors
> colorv <- c("black", "red", "blue")
> # Plot all curves
> for (indeks in 1:NROW(sigmavs)) {
+   curve(expr=dlnorm(x, sdlog=sigmavs[indeks]),
+   type="l", lwd=2, xlim=c(0, 3),
+   xlab="", ylab="", col=colorv[indeks],
+   add=as.logical(indeks-1))
+ }  ##  end for
```




```
> # Add title and legend
> title(main="Log-normal Distributions", line=0.5)
> legend("topright", inset=0.05, title="Sigmas",
+  paste("sigma", sigmavs, sep="="),
+  cex=0.8, lwd=2, lty=rep(1, NROW(sigmavs)),
+  col=colorv)
```

The Skewness of *Log-normal* Prices

If percentage stock returns are *normally* distributed and follow *Brownian motion*, then asset prices follow *Geometric Brownian motion*, and they are *Log-normally* distributed at every point in time.

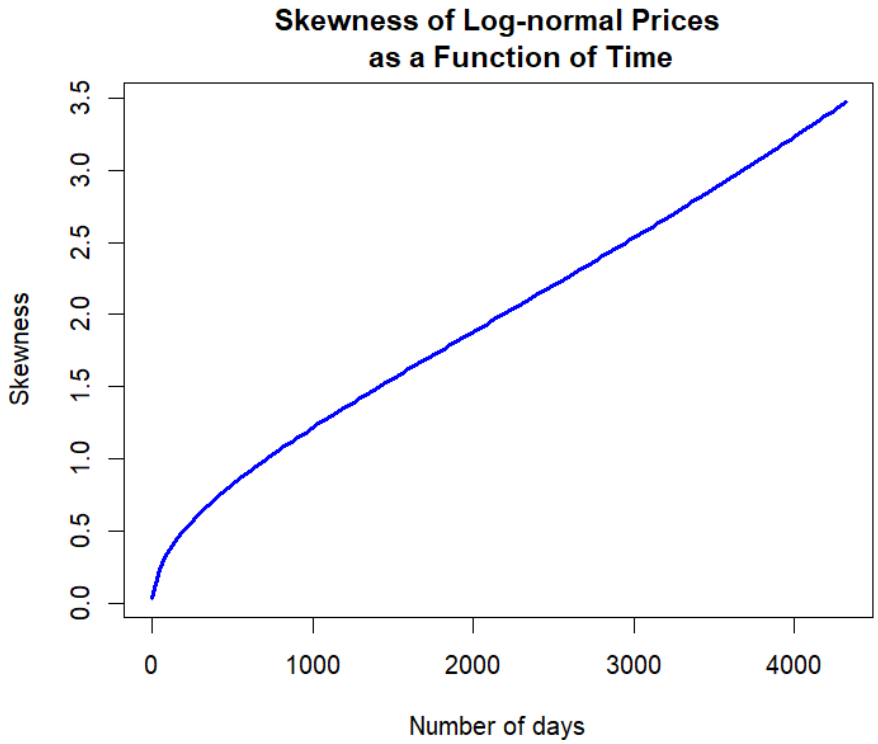
The standard deviation of *log-normal* prices is equal to the return volatility σ_r times the square root of time: $\sigma = \sigma_r \sqrt{t}$.

The *Log-normal* distribution has a strong positive skewness (third moment) equal to:

$$\varsigma = \mathbb{E}[(y - \mathbb{E}[y])^3] = (e^{\sigma^2} + 2)\sqrt{e^{\sigma^2} - 1}$$

For large standard deviation, the skewness increases exponentially with the standard deviation and with time: $\varsigma \propto e^{1.5\sigma^2} = e^{1.5t\sigma_r^2}$

```
> # Return volatility of VTI etf
> sigmav <- sd(rutils::diffit(log(rutils::etfenv$VTI[, 4])))
> sigma2 <- sigmav^2
> nrows <- NROW(rutils::etfenv$VTI)
> # Standard deviation of log-normal prices
> sqrt(nrows)*sigmav
```




```
> # Skewness of log-normal prices
> calcskew <- function(t) {
+   expv <- exp(t*sigma2)
+   (expv + 2)*sqrt(expv - 1)
+ }  ##  end calcskew
> curve(expr=calcskew, xlim=c(1, nrows), lwd=3,
+ xlab="Number of days", ylab="Skewness", col="blue",
+ main="Skewness of Log-normal Prices
+ as a Function of Time")
```

The Mean and Median of *Log-normal* Prices

The mean of the *Log-normal* distribution: $\bar{y} = \mathbb{E}[y] = \exp(\mu + \sigma^2/2)$ is greater than its median, which is equal to: $\tilde{y} = \exp(\mu)$.

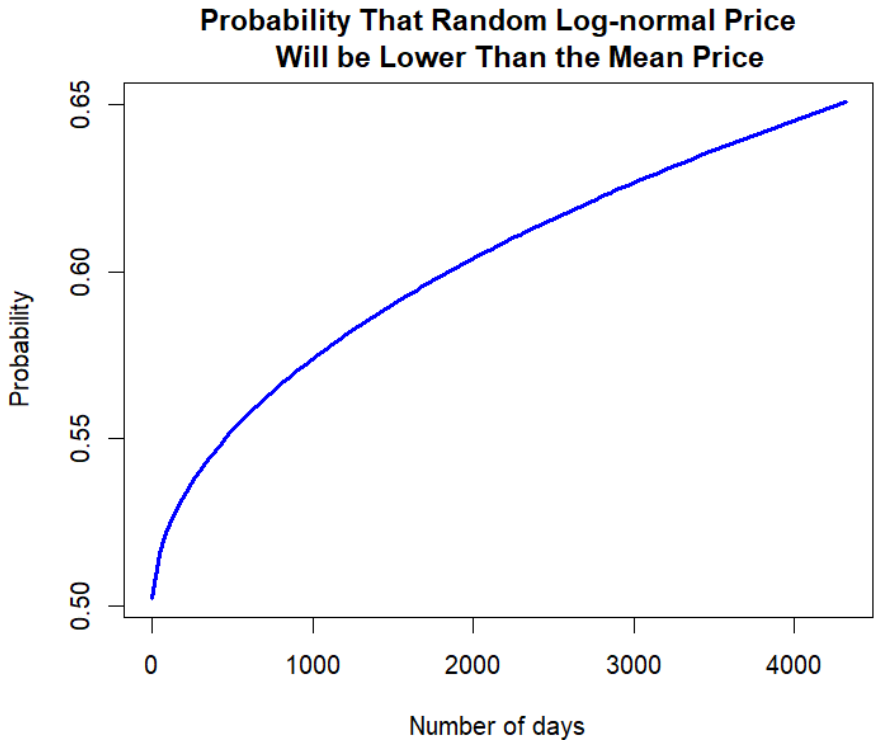
So if stock prices follow *Geometric Brownian motion* and are distributed *log-normally*, then a stock selected at random will have a high probability of havng a lower price than the mean expected price.

The cumulative *Log-normal* probability distribution is equal to $F(x) = \Phi(\frac{\log y - \mu}{\sigma})$, where $\Phi()$ is the cumulative standard normal distribution.

So the probability that the price of a randomly selected stock will be lower than the mean price is equal to $F(\bar{y}) = \Phi(\sigma/2)$.

Therefore an investor without skill, who selects stocks at random, has a high probability of underperforming the index.

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



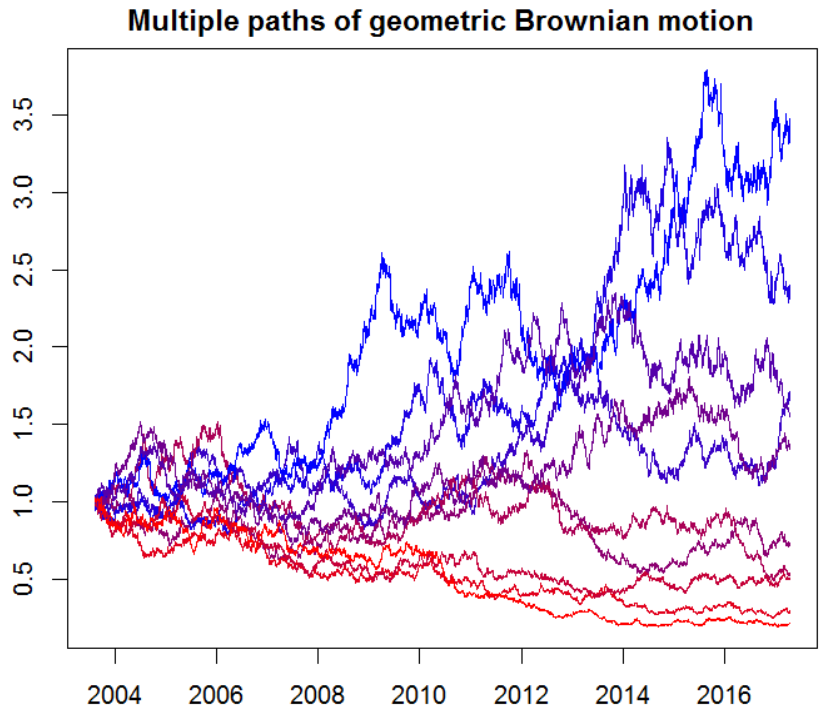

```
> # Probability that random log-normal price will be lower than the
> curve(expr=pnorm(sigmav*sqrt(x)/2),
+ xlim=c(1, nrows), lwd=3,
+ xlab="Number of days", ylab="Probability", col="blue",
+ main="Probability That Random Log-normal Price
+ Will be Lower Than the Mean Price")
```

Paths of Geometric Brownian Motion

The standard deviation of *log-normal* prices σ is equal to the volatility of returns σ_r times the square root of time: $\sigma = \sigma_r \sqrt{t}$.

For large standard deviation, the skewness ς increases exponentially with the standard deviation and with time: $\varsigma \propto e^{1.5\sigma^2} = e^{1.5t\sigma_r^2}$

```
> # Define the daily volatility and growth rate
> sigmav <- 0.01; drift <- 0.0; nrows <- 5000
> npaths <- 10
> # Simulate multiple paths of geometric Brownian motion
> pricev <- rnorm(npaths*nrows, sd=sigmav) + drift - sigmav^2/2
> pricev <- matrix(pricev, nc=npaths)
> pricev <- exp(matrixStats::colCumsums(pricev))
> # Create xts time series
> pricev <- xts(pricev, order.by=seq.Date(Sys.Date()-nrows+1, Sys.Da
> # Sort the columns according to largest terminal values
> pricev <- pricev[, order(pricev[nrows, ])]
> # Plot xts time series
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pricev))
> par(mar=c(3, 3, 2, 2), oma=c(0, 0, 0, 0))
> plot.zoo(pricev, main="Multiple paths of geometric Brownian motion",
+    xlab=NA, ylab=NA, plot.type="single", col=colorv)
```

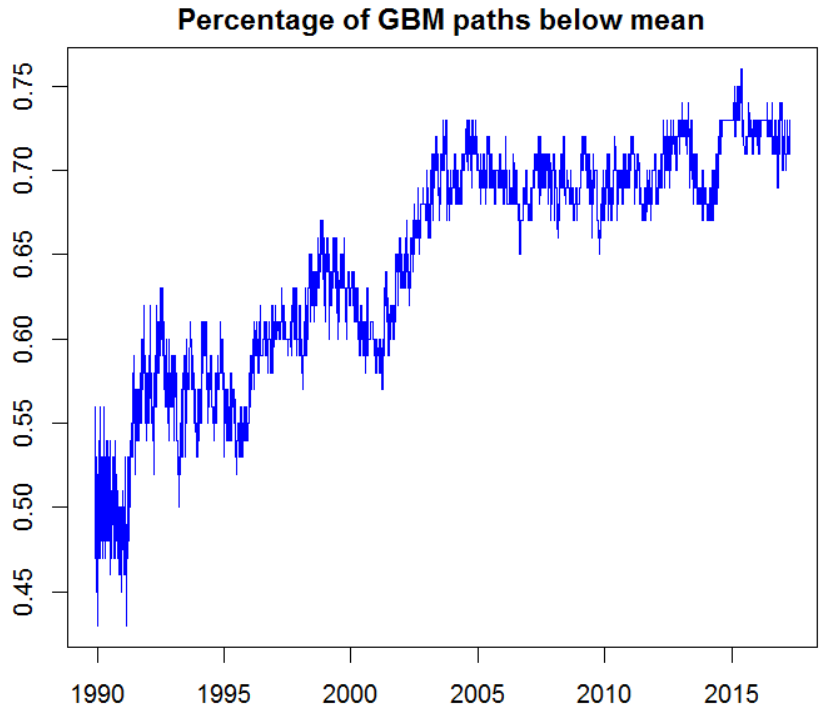


Distribution of Paths of Geometric Brownian Motion

Prices following *Geometric Brownian motion* have a large positive skewness, so that the expected value of prices is skewed by a few paths with very high prices, while the prices of the majority of paths are below their expected value.

For large standard deviation, the skewness ς increases exponentially with the standard deviation and with time: $\varsigma \propto e^{1.5\sigma^2} = e^{1.5t\sigma_r^2}$

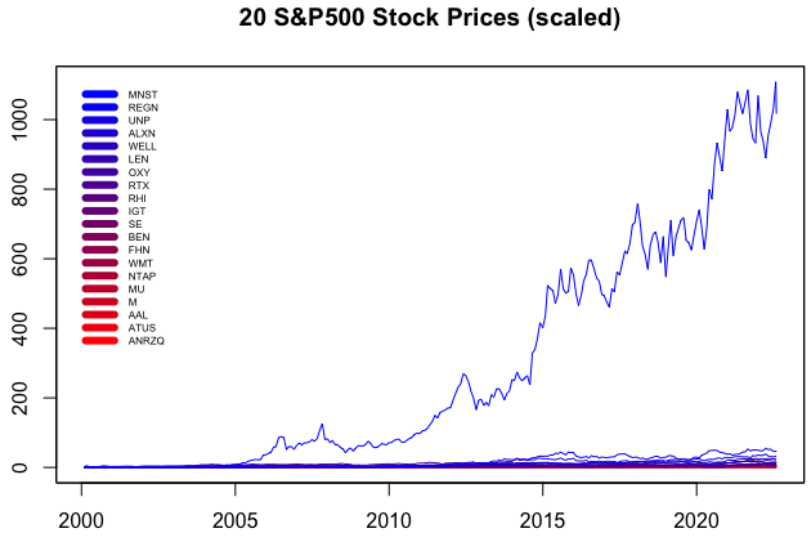
```
> # Define the daily volatility and growth rate
> sigmav <- 0.01; drift <- 0.0; nrows <- 10000
> npaths <- 100
> # Simulate multiple paths of geometric Brownian motion
> pricev <- rnorm(npaths*nrows, sd=sigmav) + drift - sigmav^2/2
> pricev <- matrix(pricev, nc=npaths)
> pricev <- exp(matrixStats::colCumsums(pricev))
> # Calculate fraction of paths below the expected value
> fractv <- rowSums(pricev < 1.0) / npaths
> # Create xts time series of percentage of paths below the expecte
> fractv <- xts(fractv, order.by=seq.Date(Sys.Date()-NROW(fractv)+1, Sys.Date(), by=1))
> # Plot xts time series of percentage of paths below the expected value
> par(mar=c(3, 3, 2, 2), oma=c(0, 0, 0, 0))
> plot.zoo(fractv, main="Percentage of GBM paths below mean",
+    xlab=NA, ylab=NA, col="blue")
```



Time Evolution of Stock Prices

Stock prices evolve over time similar to *Geometric Brownian motion*, and they also exhibit a very skewed distribution of prices.

```
> # Load S&P500 stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500.RData")
> ls(sp500env)
> # Extract the closing prices
> pricev <- eapply(sp500env, quantmod::Cl)
> # Flatten the prices into a single xts series
> pricev <- rutils::do_call(cbind, pricev)
> # Carry forward and backward non-NA prices
> pricev <- zoo::na.locf(pricev, na.rm=FALSE)
> pricev <- zoo::na.locf(pricev, fromLast=TRUE)
> sum(is.na(pricev))
> # Drop ".Close" from column names
> colnames(pricev)
> colnames(pricev) <- rutils::get_name(colnames(pricev))
> # Or
> # colnames(pricev) <- do.call(rbind,
> #   strsplit(colnames(pricev), split="[.]"))[, 1]
> # Select prices after the year 2000
> pricev <- pricev["2000/", ]
> # Scale the columns so that prices start at 1
> pricev <- lapply(pricev, function(x) x/as.numeric(x[1]))
> pricev <- rutils::do_call(cbind, pricev)
> # Sort the columns according to the final prices
> nrows <- NROW(pricev)
> ordern <- order(pricev[nrows, ])
> pricev <- pricev[, ordern]
> # Select 20 symbols
> symbolv <- colnames(pricev)
> symbolv <- symbolv[round(seq.int(from=1, to=NROW(symbolv), length.out=20))]
```



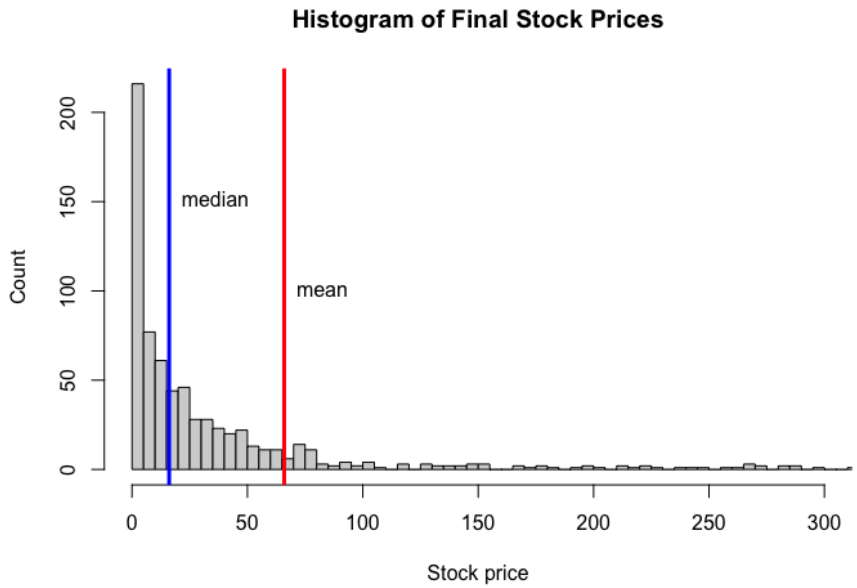

```
> # Plot xts time series of prices
> colorv <- colorRampPalette(c("red", "blue"))(NROW(symbolv))
> endd <- rutils::calc_endpoints(pricev, interval="weeks")
> plot.zoo(pricev[endd, symbolv], main="20 S&P500 Stock Prices (scal
+    xlab=NA, ylab=NA, plot.type="single", col=colorv)
> legend(x="topleft", inset=0.02, cex=0.5, bty="n", y.intersp=0.5,
+  legend=rev(symbolv), col=rev(colorv), lwd=6, lty=1)
```

Distribution of Final Stock Prices

The distribution of the final stock prices is extremely skewed, with over 80% of the *S&P500* constituent stocks from 1990 now below the average price of that portfolio.

The *mean* of the final stock prices is much greater than the *median*.

```
> # Calculate the final stock prices
> pricef <- drop(zoo::coredata(pricev[nrows, ]))
> # Calculate the mean and median stock prices
> max(pricef); min(pricef)
> which.max(pricef)
> which.min(pricef)
> mean(pricef)
> median(pricef)
> # Calculate the percentage of stock prices below the mean
> sum(pricef < mean(pricef))/NROW(pricef)
```




```
> # Plot a histogram of final stock prices
> hist(pricef, breaks=1e3, xlim=c(0, 300),
+      xlab="Stock price", ylab="Count",
+      main="Histogram of Final Stock Prices")
> # Plot a histogram of final stock prices
> abline(v=median(pricef), lwd=3, col="blue")
> text(x=median(pricef), y=150, lab="median", pos=4)
> abline(v=mean(pricef), lwd=3, col="red")
> text(x=mean(pricef), y=100, lab="mean", pos=4)
```

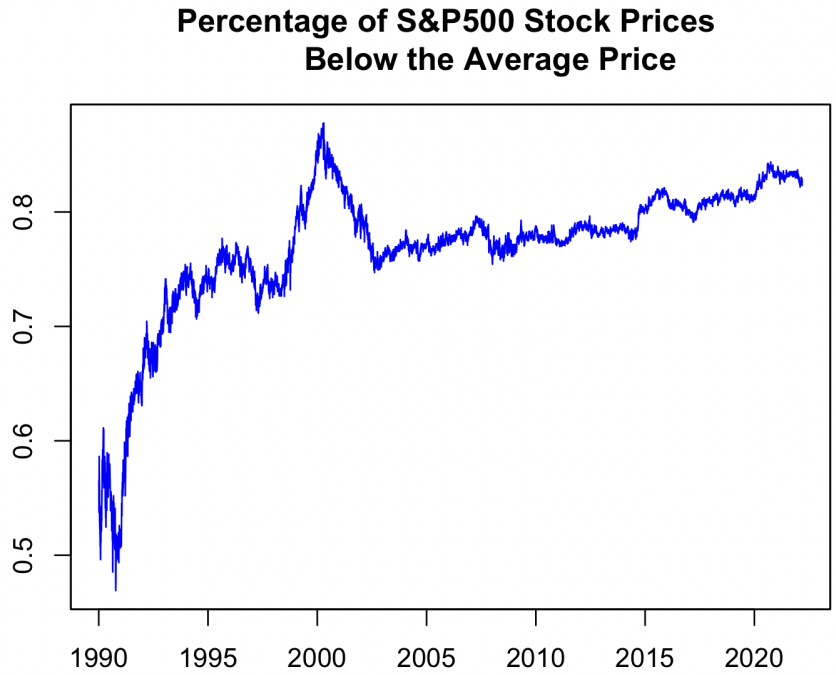
Distribution of Stock Prices Over Time

Usually, a small number of stocks in an index reach very high prices, while the prices of the majority of stocks remain below the index price (the average price of the index portfolio).

For example, the current prices of over 80% of the *S&P500* constituent stocks from 1990 are now below the average price of that portfolio.

Therefore 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.

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




```
> # Calculate average of valid stock prices
> validp <- (pricev != 1)  ##  Valid stocks
> nstocks <- rowSums(validp)
> nstocks[1] <- NCOL(pricev)
> indeks <- rowSums(pricev*validp)/nstocks
> # Calculate fraction of stock prices below the average price
> fractv <- rowSums((pricev < indeks) & validp)/nstocks
> # Create xts time series of average stock prices
> indeks <- xts(indeks, order.by=zoo::index(pricev))
```

```
> dev.new(width=6, height=4, noRStudioGD=TRUE)
> # x11(width=6, height=4)
> # Plot xts time series of average stock prices
> plot.zoo(indeks, main="Average S&P500 Stock Prices (normalized fr
+    xlab=NA, ylab=NA, col="blue")
> # Create xts time series of percentage of stock prices below the a
> fractv <- xts(fractv, order.by=zoo::index(pricev))
> # Plot percentage of stock prices below the average price
> plot.zoo(fractv[-(1:2),],
+    main="Percentage of S&P500 Stock Prices
+    Below the Average Price",
+    xlab=NA, ylab=NA, col="blue")
```

Stock Index Weighting Methods

Stock market indices can be either capitalization-weighted, price-weighted, or equal wealth.

The cap-weighted index is equal to the average of the market capitalizations of all its companies (stock price times number of shares). The *S&P500* index is cap-weighted.

The cap-weighted index is equivalent to owning a fixed number of shares, proportional to the number of shares outstanding. So if company X has twice as many shares outstanding as company Y, then the cap-weighted index will own twice as many shares of company X as company Y.

The price-weighted index is equal to the average of the stock prices. The price-weighted index is equivalent to owning a fixed and equal number of shares. The *DJIA* index is price-weighted.

The equal wealth index invests equal dollar amounts in each stock, and it rebalances its weights as market prices change.

The cap-weighted and price-weighted indices are overweight large-cap stocks, compared to the equal wealth index which has larger weights for small-cap stocks.

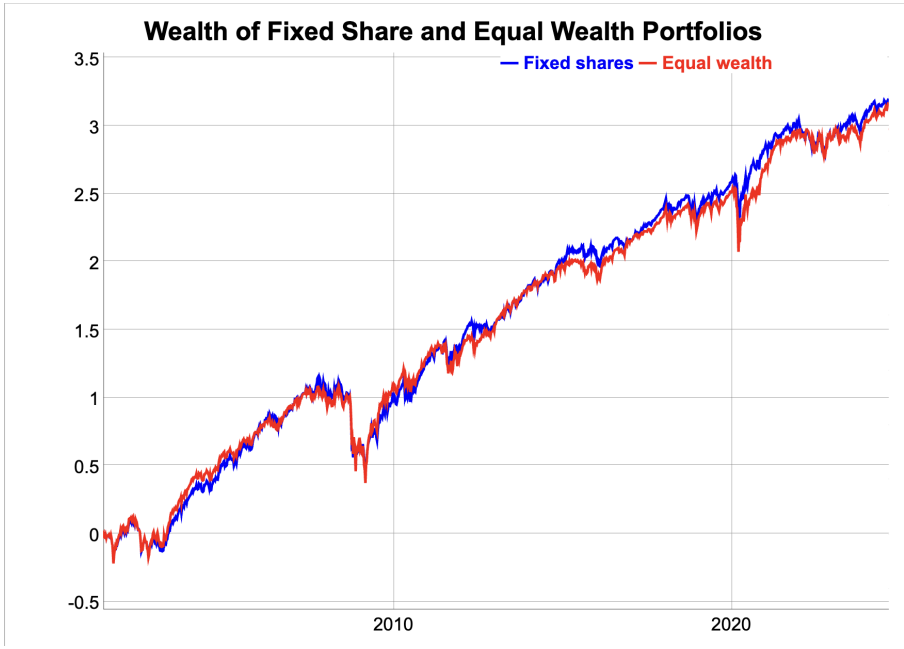
```
> # Calculate the percentage VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> nrows <- NROW(retvti)
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.R
> # Select the stock prices since VTI
> pricestock <- pricestock[datev]
> # Select stocks with no NA values in their prices
> numna <- sapply(pricestock, function(x) sum(is.na(x)))
> pricestock <- pricestock[, numna == 0]
> # Drop penny stocks
> pricel <- last(pricestock)
> pricel <- drop(coredata(pricel))
> pricestock <- pricestock[, pricel > 1]
> # Calculate the dollar and percentage stock returns
> retd <- rutils::diffit(pricestock)
> retp <- retd/rutils::lagit(pricestock)
> retp[1, ] <- 0
> nstocks <- NCOL(retp)
> # Calculate the returns of equal wealth portfolio
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
```

The Equal Wealth Portfolio

The equal wealth portfolio rebalances its weights - it sells the stocks with higher returns and buys stocks with lower returns. So it's a *mean reverting* (contrarian) strategy.

The equal wealth portfolio can often underperform the cap-weighted and fixed share indices because it gradually overweights underperforming stocks, as it rebalances to maintain equal wealth allocations.

In periods when a small number of stocks dominate returns, the cap-weighted and fixed share indices outperform the equal wealth index.




```
> # Wealth of fixed shares portfolio
> wealthfs <- rowMeans(cumprod(1 + retp))
> # Wealth of equal wealth portfolio (with rebalancing)
> wealthew <- cumprod(1 + retew)
```

```
> # Calculate combined log wealth
> wealthv <- cbind(wealthfs, wealthew)
> wealthv <- log(wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Equal wealth")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of combined log wealth
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endw],
+   main="Wealth of Fixed Share and Equal Wealth Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Random Stock Selection

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

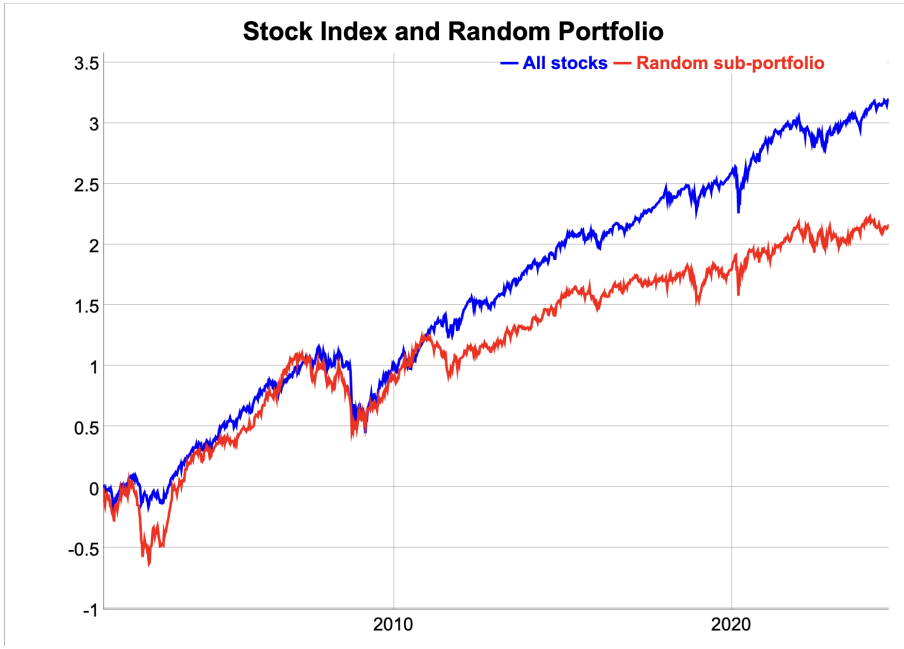
Random portfolios are used as a benchmark for stock pickers (portfolio managers).

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

According to S&P Global, 95% of large-cap actively managed funds have underperformed their benchmark. And the underperformance increases for longer holding periods, because the distribution of stock prices becomes more and more skewed over time.

Charlie Munger (vice chairman of Berkshire Hathaway) said that "Most money managers are little more than fortune tellers or astrologers."

John Bogle (founder of The Vanguard Group) said, "Don't look for the needle in the haystack. Just buy the whole haystack."




```
> # Select a random, fixed share sub-portfolio of 5 stocks
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> samplev <- sample.int(n=nstocks, size=5, replace=FALSE)
> wealthr <- rowMeans(cumprod(1 + retp[, samplev]))
```

```
> # Plot dygraph of index and random sub-portfolio
> wealthv <- cbind(wealthfs, wealthr)
> wealthv <- log(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("Index", "Random portfolio")
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endw], main="Stock Index and Random Port
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

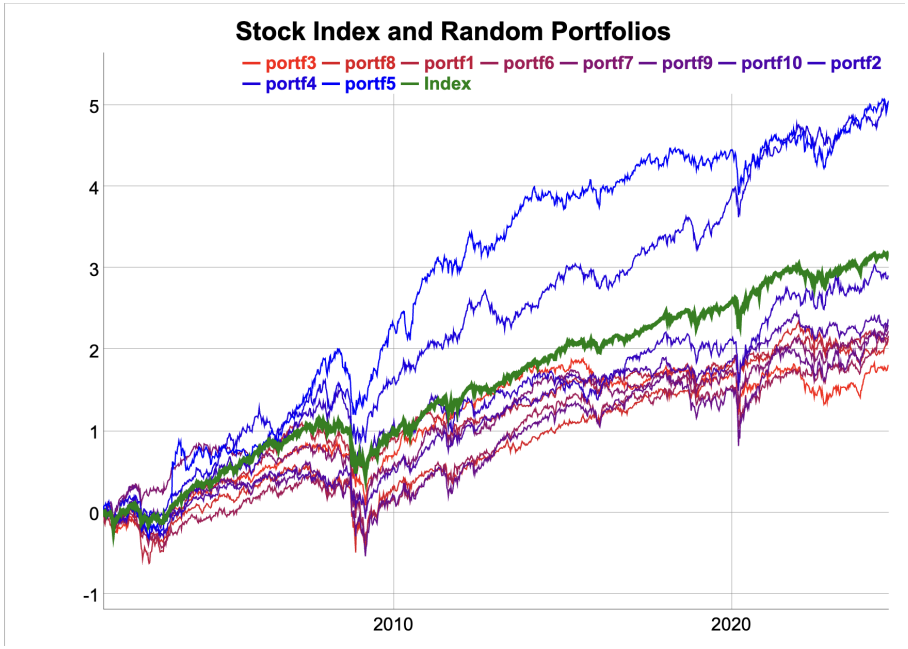
Random Stock Portfolios

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 fixed share sub-portfolios
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> nportf <- 10
> wealthr <- sapply(1:nportf, function(x) {
+   samplev <- sample.int(n=nstocks, size=5, replace=FALSE)
+   rowMeans(cumprod(1 + retp[, samplev]))
+ })  ##  end sapply
> wealthr <- xts::xts(wealthr, order.by=datev)
> colnames(wealthr) <- paste0("portf", 1:nportf)
> # Sort the sub-portfolios according to performance
> wealthr <- wealthr[, order(wealthr[nrows])]
> round(head(wealthr), 3)
> round(tail(wealthr), 3)
```

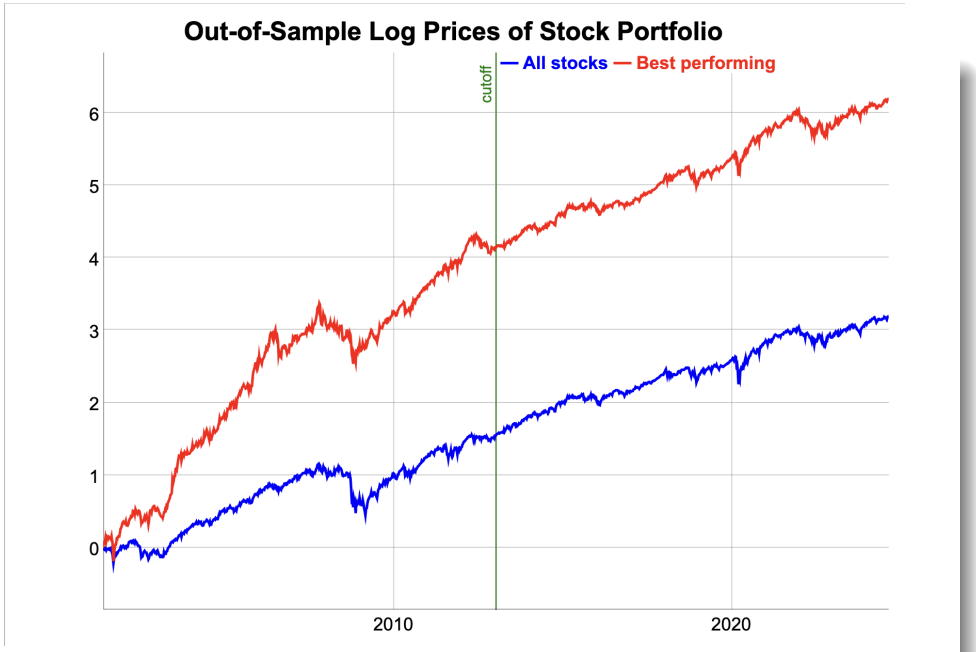
```
> # Plot dygraph of all stock index and random sub-portfolios
> colorv <- colorRampPalette(c("red", "blue"))(nportf)
> colorv <- c("green", colorv)
> wealthv <- cbind(wealthfs, wealthr)
> wealthv <- log(wealthv)
> colnames(wealthv)[1] <- "Index"
> symbolv <- colnames(wealthv)
> dygraphs::dygraph(wealthv[endw], main="Stock Index and Random Port
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[1], strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Stock Portfolio Selection Out-of-Sample

The strategy selects the 10 best performing stocks at the end of the in-sample interval, and invests in them in the out-of-sample interval.

The strategy buys equal and fixed number of shares of stocks, and at the end of the in-sample interval, selects the 10 best performing stocks. It then invests the same number of shares in the out-of-sample interval.

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




```
> # Define 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
> pricev <- cumprod(1 + retp)
> pricet <- pricev[cutoff, ]
> pricet <- drop(coredata(pricet))
> pricet <- sort(pricet, decreasing=TRUE)
> symbolv <- names(head(pricet, 10))
> # Calculate the wealth of the 10 best performing stocks
> wealthb <- rowMeans(pricev[insample, symbolv])
> wealthos <- wealthb[cutoff]*rowMeans(cumprod(1 + retp[outsample,
> wealthb <- c(wealthb, wealthos)
```

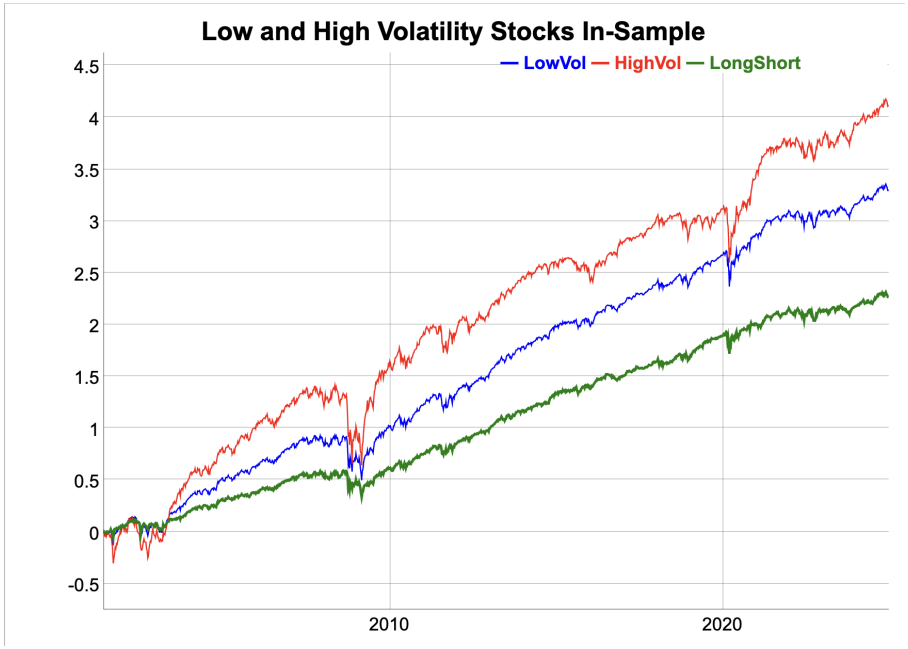
```
> # Combine the fixed share wealth with the 10 best performing stock
> wealthv <- cbind(wealthfs, wealthb)
> wealthv <- xts::xts(log(wealthv), order.by=datev)
> colnames(wealthv) <- c("Index", "Best performing")
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv[insample, ]),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # 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(wealthv[endw], main="Out-of-Sample Log Prices of
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(width=300)
```

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
> # Perform parallel loop under Mac-OSX or Linux
> library(parallel)  ##  Load package parallel
> ncores <- detectCores() - 1
> retvti <- retvti[zoo::index(retp)]
> meanvti <- mean(retvti)
> varvti <- drop(var(retvti))
> riskret <- mclapply(retp, function(rets) {
+   stdev <- sd(rets)
+   betac <- drop(cov(rets, retvti))/varvti
+   resid <- rets - betac*retvti
+   alphac <- mean(rets) - betac*meanvti
+   c(alpha=alphac, beta=betac, stdev=stdev, ivol=sd(resid))
+ }, mc.cores=ncores)  ##  end mclapply
> riskret <- do.call(rbind, riskret)
> tail(riskret)
> # Calculate the median volatility
> riskv <- riskret[, "stdev"]
> medianv <- median(riskv)
> # Calculate the returns of low and high volatility stocks
> retlow <- rowMeans(retp[, (riskv <= medianv)], na.rm=TRUE)
> rethigh <- rowMeans(retp[, (riskv > medianv)], na.rm=TRUE)
```

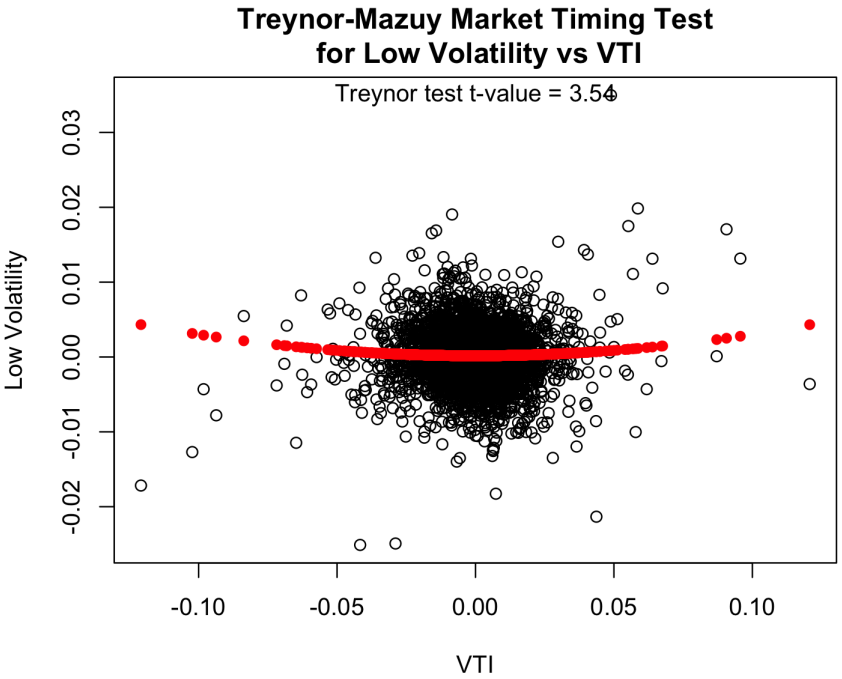
```
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> symbolv <- c("LowVol", "HighVol", "LongShort")
> colnames(wealthv) <- symbolv
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Volati
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

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 very small *market timing* skill.

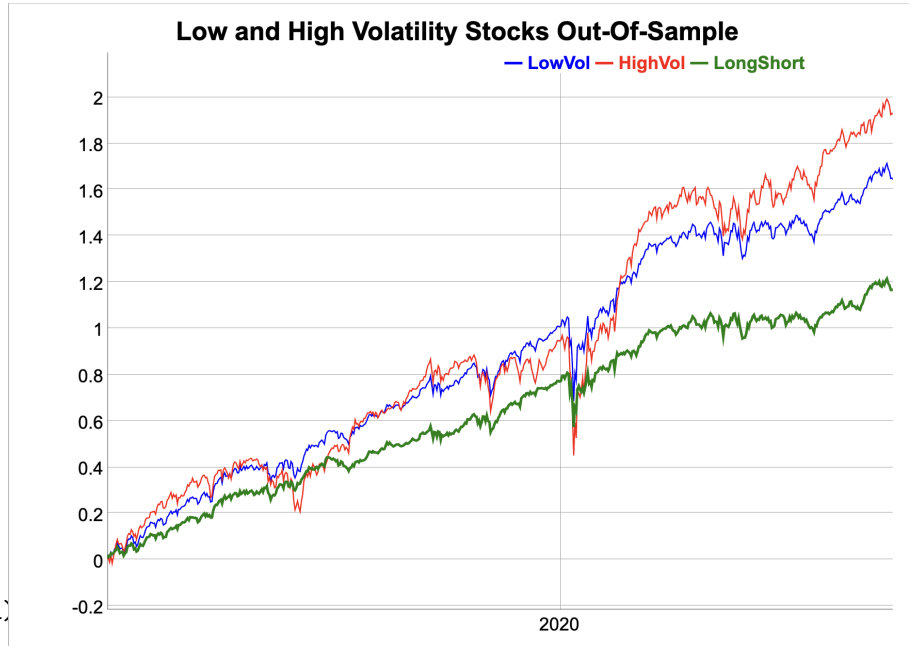
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$LongShort ~ VTI + Merton, data=desm); summary
> # Treynor-Mazuy test
> regmod <- lm(wealthv$LongShort ~ VTI + Treynor, data=desm); summa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



Low and High Volatility Stock Portfolios Out-Of-Sample

The low volatility stocks selected in-sample also have a higher *Sharpe ratio* in the out-of-sample period than the high volatility stocks, although their absolute returns are lower.

```
> # Calculate the in-sample stock volatilities, betas, and alphas
> retvti <- retvti[zoo::index(retp[insample])]
> varvti <- drop(var(retvti))
> meanvti <- mean(retvti)
> riskretis <- mclapply(retp[insample], function(rets) {
+   combv <- na.omit(cbind(rets, retvti))
+   if (NROW(combv) > 11) {
+     stdev <- sd(rets)
+     betac <- drop(cov(rets, retvti))/varvti
+     resid <- rets - betac*retvti
+     alphac <- mean(rets) - betac*meanvti
+     return(c(alpha=alphac, beta=betac, stdev=stdev, ivol=sd(resid)
+   } else {
+     return(c(alpha=0, beta=0, stdev=0, ivol=0))
+   }  ##  end if
+ }, mc.cores=ncores)  ##  end mclapply
> riskretis <- do.call(rbind, riskretis)
> tail(riskretis)
> # Calculate the median volatility
> riskv <- riskretis[, "stdev"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high volatility
> retlow <- rowMeans(retp[outsample, (riskv <= medianv)], na.rm=TRU
> rethigh <- rowMeans(retp[outsample, (riskv > medianv)], na.rm=TRU
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> symbolv <- c("LowVol", "HighVol", "LongShort")
> colnames(wealthv) <- symbolv
```



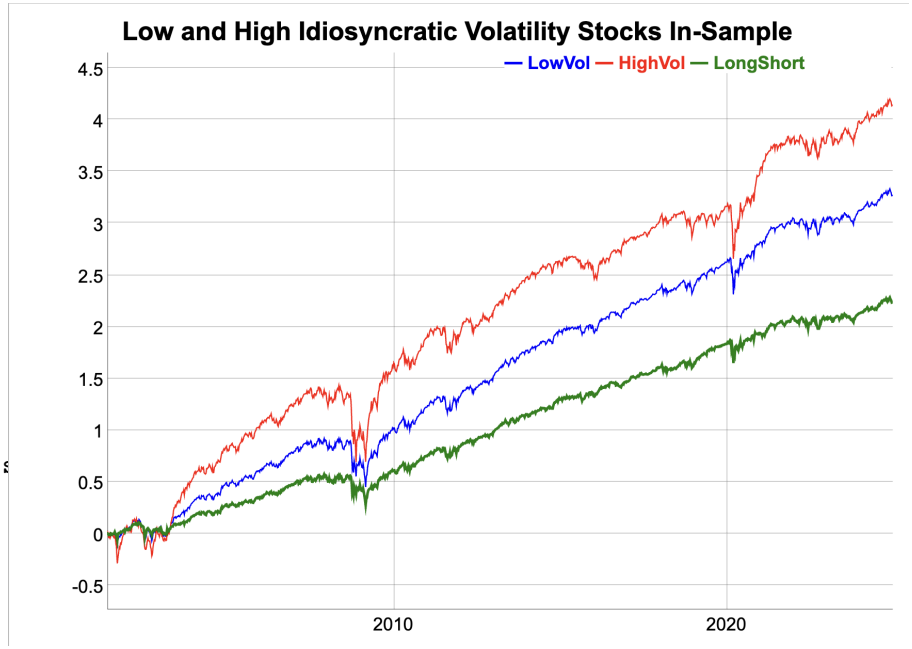

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Volati
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

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.

```
> # Calculate the median idiosyncratic volatility
> riskv <- riskret[, "ivol"]
> medianv <- median(riskv)
> # Calculate the returns of low and high idiosyncratic volatility s
> retlow <- rowMeans(retp[, (riskv <= medianv)], na.rm=TRUE)
> rethigh <- rowMeans(retp[, (riskv > medianv)], na.rm=TRUE)
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> symbolv <- c("LowVol", "HighVol", "LongShort")
> colnames(wealthv) <- symbolv
```



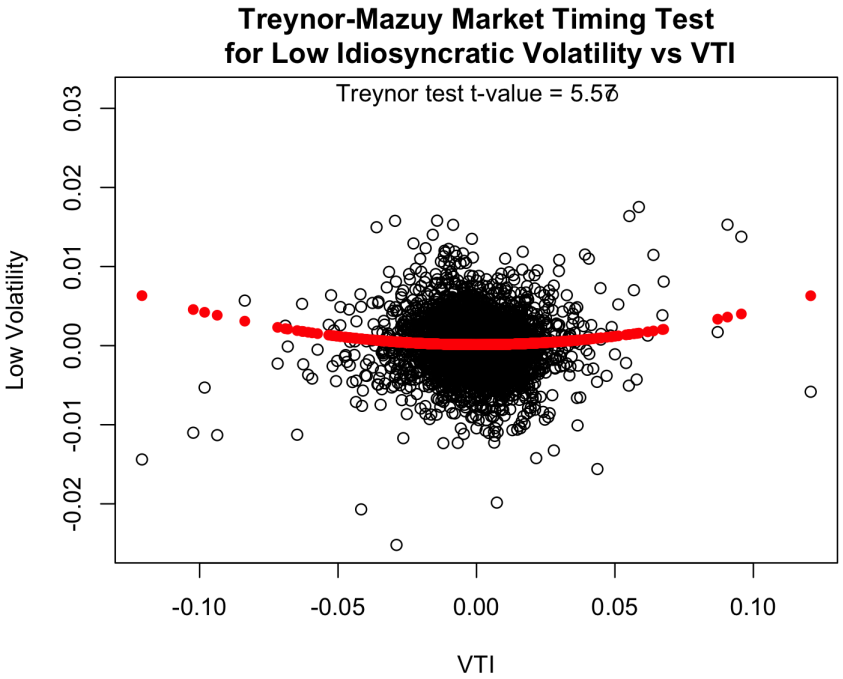

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of returns of low and high idiosyncratic volatility stocks
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Idiosy
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

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.

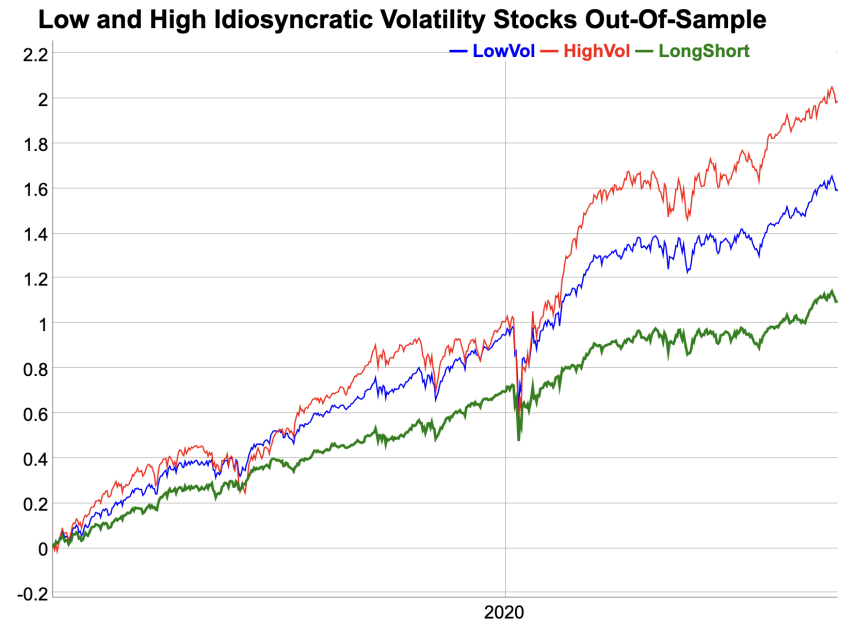
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$LongShort ~ VTI + Merton, data=desm); summary
> # Treynor-Mazuy test
> regmod <- lm(wealthv$LongShort ~ VTI + Treynor, data=desm); summa
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="Low Volatility"
> title(main="Treynor-Mazuy Market Timing Test\n for Low Idiosyncrat
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["Treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), 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 have a higher *Sharpe ratio* in the out-of-sample period than the high idiosyncratic volatility stocks, although their absolute returns are lower.

```
> # Calculate the median in-sample idiosyncratic volatility
> riskv <- riskretis[, "ivol"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high idiosyncrati
> retlow <- rowMeans(retp[outsample, (riskv <= medianv)], na.rm=TRUE
> rethigh <- rowMeans(retp[outsample, (riskv > medianv)], na.rm=TRUE
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> symbolv <- c("LowVol", "HighVol", "LongShort")
> colnames(wealthv) <- symbolv
```



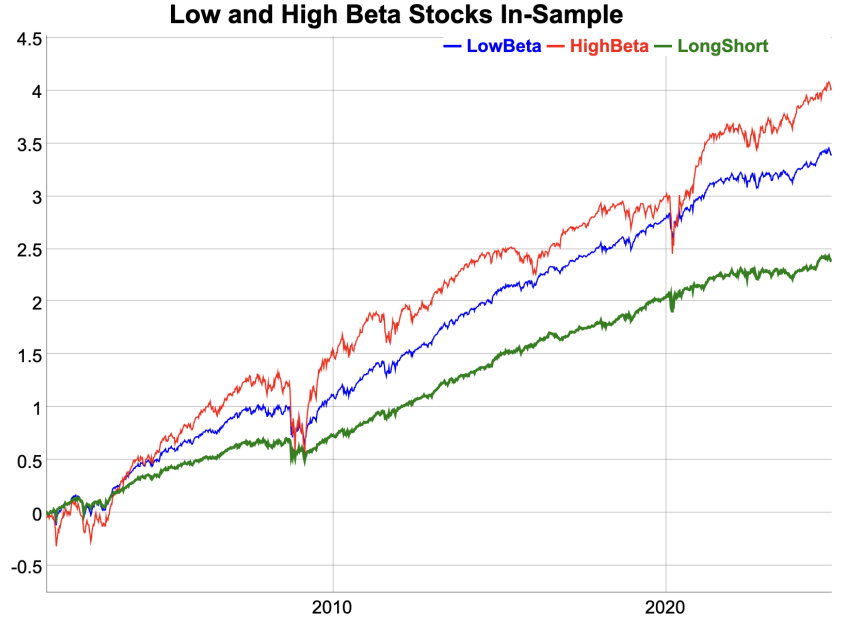

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of out-of-sample returns of low and high volatility stocks
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Idiosy
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low and High Beta Stock Portfolios

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

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

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




```
> # Calculate the median beta
> riskv <- riskret[, "beta"]
> medianv <- median(riskv)
> # Calculate the returns of low and high beta stocks
> betalow <- rowMeans(retp[, names(riskv[riskv <= medianv])], na.rm=TRUE)
> betahigh <- rowMeans(retp[, names(riskv[riskv > medianv])], na.rm
> wealthv <- cbind(betalow, betahigh, betalow - 0.25*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> symbolv <- c("LowBeta", "HighBeta", "LongShort")
> colnames(wealthv) <- symbolv
```

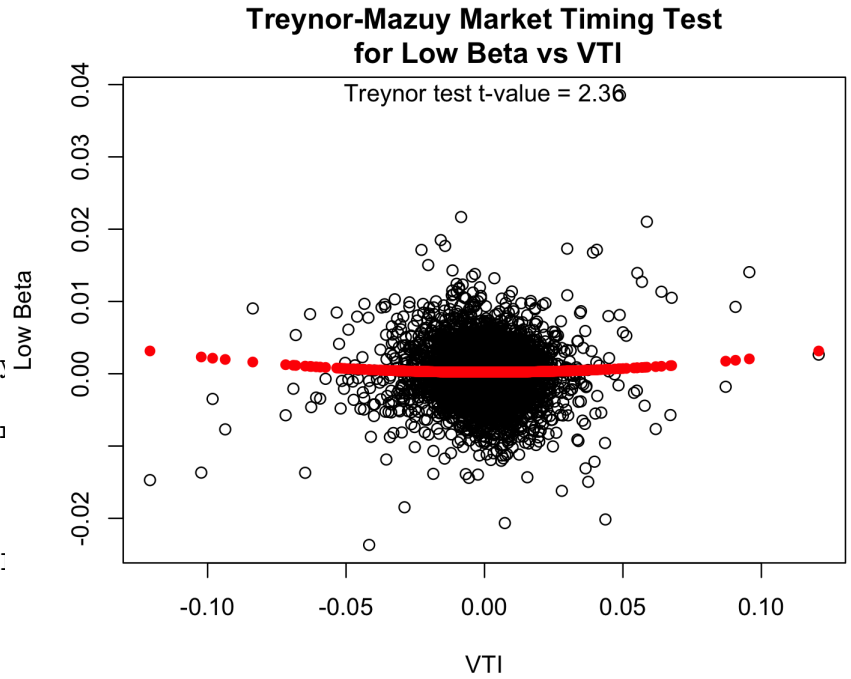
```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high beta stocks
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Beta S
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

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.

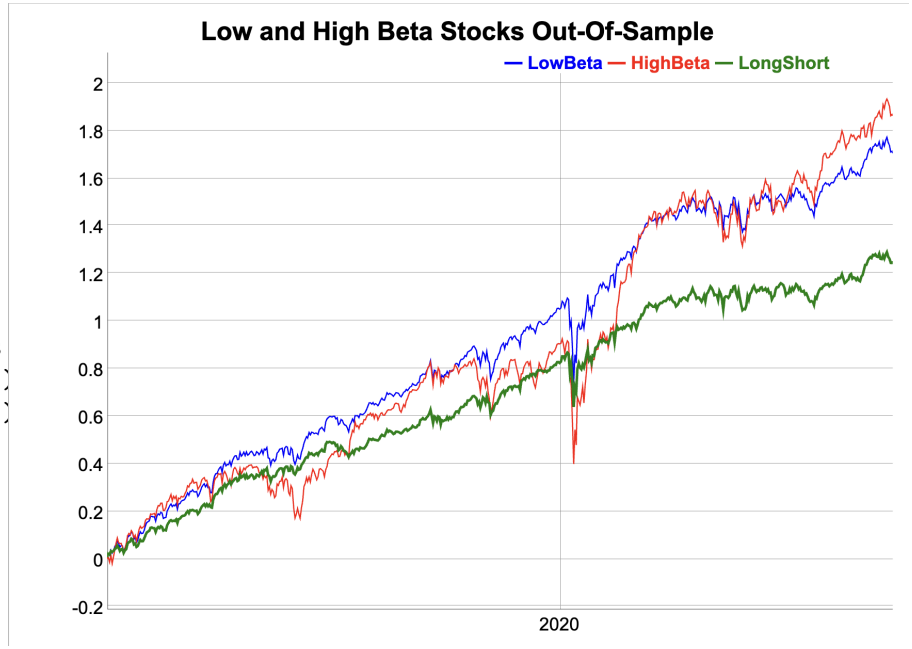
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$LongShort ~ VTI + Merton, data=desm); summary
> # Treynor-Mazuy test
> regmod <- lm(wealthv$LongShort ~ VTI + Treynor, data=desm); summa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



Low and High Beta Stock Portfolios Out-Of-Sample

The low beta stocks selected in-sample also have a higher *Sharpe ratio* in the out-of-sample period than the high beta stocks, although their absolute returns are lower.

```
> # Calculate the median beta
> riskv <- riskretis[, "beta"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high beta stocks
> betalow <- rowMeans(retp[outsample, names(riskv[riskv <= medianv])])
> betahigh <- rowMeans(retp[outsample, names(riskv[riskv > medianv])])
> wealthv <- cbind(betalow, betahigh, betalow - 0.25*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> symbolv <- c("LowBeta", "HighBeta", "LongShort")
> colnames(wealthv) <- symbolv
```




```
> # 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 of out-of-sample returns of low and high beta stocks
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Low and High Beta S
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=symbolv[3], strokeWidth=2) %>%
+   dyLegend(width=300)
```

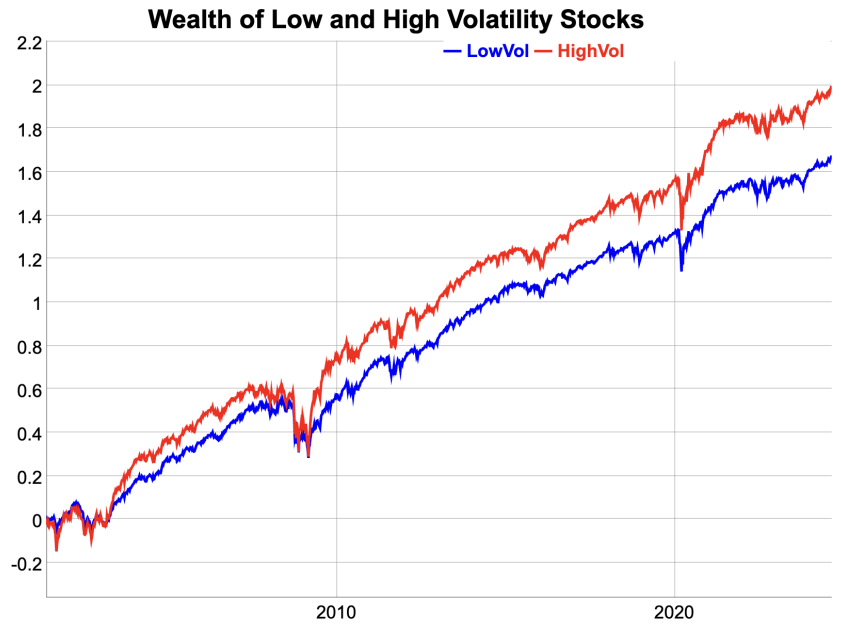
Stocks With Low and High Trailing Volatilities

The trailing volatilities can be used to create low and high volatility portfolios, and test their performance out-of-sample.

The low volatility portfolio consists of stocks with trailing volatilities less than the median, and the high portfolio with trailing volatilities greater than the median.

The portfolios are rebalanced daily, as the volatility changes.

The low volatility portfolio has a higher *Sharpe ratio*, but lower absolute returns than the high volatility portfolio.




```
> # Calculate the trailing percentage volatilities
> volp <- HighFreq::run_var(retp, lambda=0.15)
> volp <- sqrt(volp[, (nstocks+1):(2*nstocks)])
> volp <- rutils::lagit(volp)
> volp[volp == 0] <- 1
> # Calculate the median volatilities
> medianv <- matrixStats::rowMedians(volp)
> # Calculate the wealth of low volatility stocks
> weightv <- (volp <= medianv)
> weightv <- rutils::lagit(weightv)
> retlow <- rowMeans(weightv*retp)
> # Calculate the wealth of high volatility stocks
> weightv <- (volp > medianv)
> weightv <- rutils::lagit(weightv)
> rethigh <- rowMeans(weightv*retp)
```

```
> # Combined wealth
> wealthv <- cbind(retlow, rethigh)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("LowVol", "HighVol")
> # 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 of log wealth
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Wealth of Low and High Volatility Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

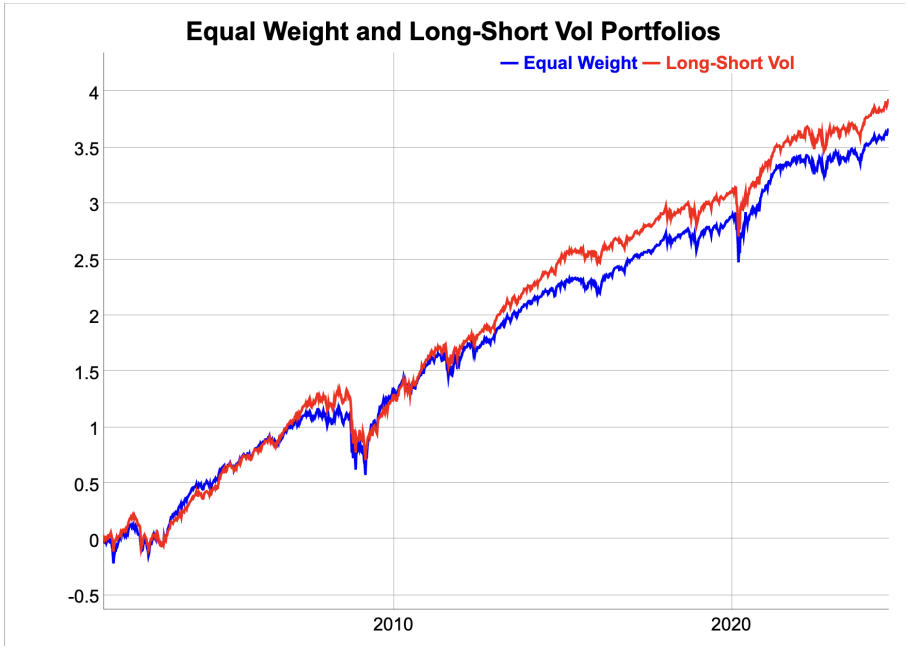
Long-Short Stock Volatility Strategy

The *Long-Short Volatility* strategy buys the low volatility stock portfolio and shorts the high volatility portfolio.

The high volatility portfolio returns are multiplied by a factor to compensate for their higher volatility.

The *Long-Short Volatility* strategy has a higher *Sharpe ratio* and also higher absolute returns than the equal wealth strategy.

```
> # Calculate the long-short volatility returns
> retls <- (retlow - 0.25*rethigh)
> # Scale the PnL volatility to that of wealthew
> retls <- retls*sd(retew)/sd(retls)
> # Combined wealth
> wealthv <- cbind(retew, retls)
> wealthv <- xts::xts(wealthv, datev)
> symbolv <- c("EqualWeight", "Long-Short Vol")
> colnames(wealthv) <- symbolv
> # 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 of log wealth
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Equal Weight and Long-Short Vol Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Risk Parity Weights for Stocks

The dollar amount of a stock with price p_i that has unit dollar volatility is equal to: $\frac{p_i}{\sigma_i^d}$.

Where σ_i^d is the dollar volatility.

So the weights of the risk parity strategy are proportional to the inverse of the stock dollar volatilities: $w_i \propto \frac{1}{\sigma_i^d}$.

The weights are rebalanced daily so that the dollar volatilities of the stock allocations (*dollar amounts*) remain equal.

The stock allocations increase when the volatility is low, and vice versa.

The function `HighFreq::run_var()` calculates the trailing variance of a *time series* of returns r_t, by recursively weighting the past variance estimates σ_{t-1}^2, with the squared differences of the returns minus the trailing means $(r_t - \bar{r}_t)^2$, using the decay factor λ:

$$\bar{r}_t = \lambda \bar{r}_{t-1} + (1 - \lambda) r_t$$

$$\sigma_t^2 = \lambda^2 \sigma_{t-1}^2 + (1 - \lambda^2)(r_t - \bar{r}_t)^2$$

Where σ_t^2 is the trailing variance at time t.

The decay factor λ determines how quickly the variance estimates are updated, with smaller values of λ producing faster updating, giving more weight to recent returns, and vice versa.

```
> # Calculate the trailing dollar volatilities
> lambdav <- 0.99
> vold <- HighFreq::run_var(retd, lambda=lambdav)
> vold <- vold[, (nstocks+1):(2*nstocks)]
>
> vold <- sqrt(vold)
> vold[vold == 0] <- 1
> # Calculate the rolling risk parity weights
> weightv <- 1/vold
> weightv <- weightv/rowSums(weightv)
```

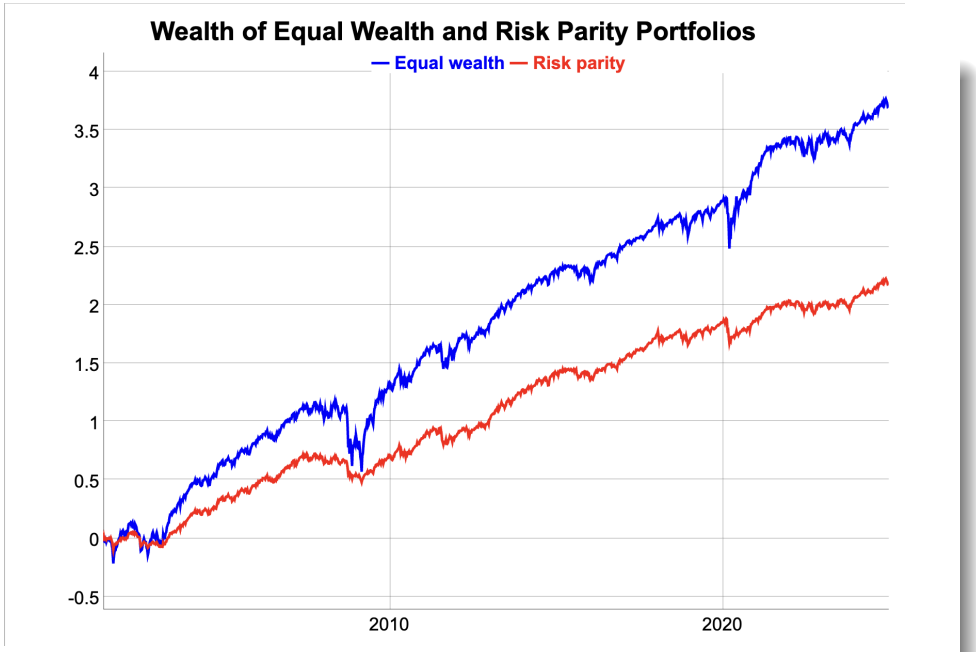
Risk Parity Strategy for Stocks

The risk parity strategy for stocks has a similar *Sharpe ratio* to the equal wealth strategy. But it has lower absolute returns.

The risk parity strategy also has higher transaction costs.

Risk parity for stocks doesn't perform as well as for assets with negative or low correlations, like stocks and bonds, because stock returns have positive correlations.

```
> # Calculate the risk parity allocations
> pricerp <- pricestock*weightv
> # Calculate the dollar returns of risk parity
> retrp <- retp*rutils::lagit(pricerp)
> # Calculate the wealth of risk parity
> retrp <- retrp*sd(retew)/sd(retrp)
> wealthrp <- cumsum(rowMeans(retrp))
> # Wealth of equal wealth portfolio (with rebalancing)
> wealthew <- cumsum(retew)
```




```
> # Combined wealth
> wealthv <- cbind(wealthew, wealthrp)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Equal wealth", "Risk parity")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of log wealth
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endw],
+   main="Wealth of Equal Wealth and Risk Parity Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

Optimal Stock Weights

If an investor's utility of wealth is logarithmic: $u(w) = \log(w)$, then to maximize their utility, they should invest the fraction k_f of their wealth in a stock:

$$k_f = \frac{\bar{r}}{\sigma^2}$$

Where $\bar{r}$ is the expected stock return and σ^2 is the expected variance. The fraction k_f is called the *Kelly fraction* or the *Kelly ratio*.

The expected *logarithmic utility* u can be expanded to second order as:

$$u = \mathbb{E}[\log(1 + k_f\, r)] = \mathbb{E}[(k_f\, r - \frac{(k_f\, r)^2}{2}] =$$

$$= k_f \bar{r} - \frac{k_f^2 \sigma^2}{2}$$

The *Kelly ratio* which maximizes the utility is found by equating the derivative of utility to zero:

$$\frac{\mathrm{d}u}{\mathrm{d}k_f} = \bar{r} - k_f \sigma^2 = 0 \;\rightarrow\; k_f = \frac{\bar{r}}{\sigma^2}$$

The *Kelly criterion* favors stocks with lower volatilities. Given two stocks with the same *Sharpe* ratios, the stock with the lower volatility will have a higher *Kelly ratio*.

```
> # Objective function equal to the sum of returns
> objfun <- function(retp) sum(na.omit(retp))
> # Objective function equal to the Sharpe ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     stdev <- sd(retp)
+     if (stdev > 0) mean(retp)/stdev else 0
+   } else 0
+ }  ##  end objfun
> # Objective function equal to the Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     varv <- var(retp)
+     if (varv > 0) mean(retp)/varv else 0
+   } else 0
+ }  ##  end objfun
```

In portfolio optimization, assuming zero correlations, the *maximum Sharpe* portfolio weights are proportional to the *Kelly ratios*:

$$w_i \propto \frac{\bar{r}_i}{\sigma_i^2}$$

This is also a consequence of logarithmic utility.

Portfolio Weight Constraints

Constraints on the portfolio weights are applied to satisfy investment objectives and risk limits.

Let w_i be the portfolio weights produced by a model, which may not satisfy the constraints, so they must be transformed into new weights: w_i'.

For example, the weights can be centered so their sum is equal to 0: $\sum_{i=1}^{n} w_i' = 0$, by shifting them by their mean value:

$$w_i' = w_i - \frac{1}{n}\sum_{i=1}^{n} w_i$$

The advantage of centering is that it produces portfolios that are more risk neutral - less long or short risk.

The disadvantage is that it shifts the mean of the weights, and it allows highly leveraged portfolios, with very large positive and negative weights.

```
> # VTI returns
> retv <- na.omit(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retv) ##  Dates vector
> # Load daily S&P500 percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> retp <- retstock[datev]
> nrows <- NROW(retp) ##  number of rows
> nstocks <- NCOL(retp) ##  number of stocks
> # Objective function equal to the Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     varv <- var(retp)
+     if (varv > 0) mean(retp)/varv else 0
+   } else 0
+ }  ##  end objfun
> # Calculate performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> sum(is.na(perfstat))
> sum(!is.finite(perfstat))
> hist(perfstat, breaks=100, main="Performance Statistics")
> sort(perfstat, decreasing=TRUE)
```

Quadratic Weight Constraint

Another way of satisfying the constraints is by scaling (multiplying) the weights by a factor.

Under the *quadratic* constraint, the sum of the *squared* weights is equal to 1: $\sum_{i=1}^{n} w_i'^2 = 1$, after they are scaled:

$$w_i' = \frac{w_i}{\sqrt{\sum_{i=1}^{n} w_i^2}}$$

Scaling the weights modifies the portfolio *leverage* (the ratio of the portfolio risk divided by the capital), while maintaining the relative weights.

The disadvantage of the *quadratic* constraint is that it can produce portfolios with very low leverage.

```
> # Calculate weights proportional to performance statistic
> # With quadratic constraint
> weightv <- perfstat/sqrt(sum(perfstat^2))
> sum(weightv^2)
> sum(weightv)
> weightv
```

Linear Weight Constraint

A widely used constraint is setting the sum of the weights equal to 1: $\sum_{i=1}^{n} w_i' = 1$, by dividing them by their sum:

$$w_i' = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

The *linear* constraint is equivalent to distributing a unit of capital among a stock portfolio.

The disadvantage of the *linear* constraint is that it has a long risk bias. When the sum of the weights is negative, it switches their sign to positive.

```
> # Calculate weights proportional to performance statistic
> # With linear constraint
> weightv <- perfstat/sum(perfstat)
> sum(weightv^2)
> sum(weightv)
> weightv
```

Volatility Weight Constraint

The weights can be scaled to satisfy a volatility target.

For example, they can be scaled so that the in-sample portfolio volatility σ is the same as the volatility of the equal weight portfolio σ_{ew}:

$$w_i' = \frac{\sigma_{ew}}{\sigma} w_i$$

This produces portfolios with a leverage corresponding to the current market volatility.

Or the weights can be scaled so that the in-sample portfolio volatility σ is equal to a target volatility σ_t:

$$w_i' = \frac{\sigma_t}{\sigma} w_i$$

This produces portfolios with a volatility close to the target, irrespective of the market volatility.

Averaging the stock returns using the function `rowMeans()` with `na.rm=TRUE` is equivalent to rebalancing the portfolio so that stocks with `NA` returns have zero weight.

The function `HighFreq::mult_mat()` multiplies the rows or columns of a *matrix* times a *vector*, element-wise.

```
> # Calculate the weighted returns using transpose
> retw <- t(t(retp)*perfstat)
> # Or using Rcpp
> retf <- HighFreq::mult_mat(perfstat, retp)
> all.equal(retw, retf, check.attributes=FALSE)
> # Calculate the in-sample portfolio volatility
> volis <- sd(rowMeans(retw, na.rm=TRUE))
> # Calculate the equal weight portfolio volatility
> volew <- sd(rowMeans(retp, na.rm=TRUE))
> # Apply the volatility constraint
> weightv <- volew*perfstat/volis
> # Calculate the in-sample portfolio volatility
> retw <- t(t(retp)*weightv)
> all.equal(sd(rowMeans(retw, na.rm=TRUE)), volew)
> # Apply the volatility target constraint
> volt <- 0.01
> weightv <- volt*perfstat/volis
> retw <- t(t(retp)*weightv)
> all.equal(sd(rowMeans(retw, na.rm=TRUE)), volt)
> # Compare speed of R with Rcpp
> library(microbenchmark)
> summary(microbenchmark(
+   trans=t(t(retp)*perfstat),
+   rcpp=HighFreq::mult_mat(perfstat, retp),
+   times=10))[, c(1, 4, 5)]
```

Box Constraints

Box constraints limit the individual weights, for example: $0 \leq w_i \leq 1$.

Box constraints are often applied when constructing long-only portfolios, or when limiting the exposure to certain stocks.

```
> # Box constraints
> weightv[weightv > 1] <- 1
> weightv[weightv < 0] <- 0
> weightv
```

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,
- Apply the weight constraints.

The *momentum* weights can then be applied in the out-of-sample interval.

```
> # Calculate the performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> sum(is.na(perfstat))
> # Calculate the best and worst performing stocks
> perfstat <- sort(perfstat, decreasing=TRUE)
> topstocks <- 10
> symbolb <- names(head(perfstat, topstocks))
> symbolw <- names(tail(perfstat, topstocks))
> # Calculate equal weights for the best and worst performing stocks
> weightv <- numeric(NCOL(retp))
> names(weightv) <- colnames(retp)
> weightv[symbolb] <- 1
> weightv[symbolw] <- (-1)
> # Calculate weights proportional to the 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 in-sample momentum strategy pnls
> pnls <- t(t(retp)*weightv)
> # Or using Rcpp
> pnl2 <- HighFreq::mult_mat(weightv, retp)
> all.equal(pnls, pnl2, check.attributes=FALSE)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls[1] <- 0
> # Scale the pnls so their volatility is the same as equal weight
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
> pnls <- sd(retew)/sd(pnls)*pnls
> wealthv <- xts(cbind(retew, pnls), datev)
> colnames(wealthv) <- c("EqualWeight", "Momentum")
> dygraph(cumsum(wealthv))
```

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 interval, 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.

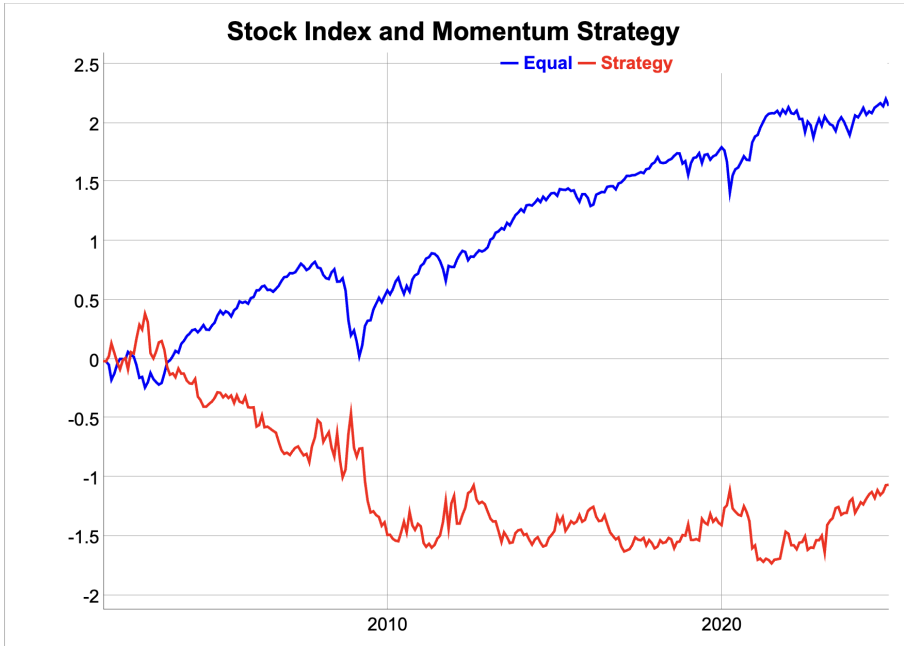
```
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform loop over the end points
> lookb <- 8 ##  Look-back interval in months
> library(parallel)  ##  Load package parallel
> ncores <- detectCores() - 1
> pnls <- mclapply(3:(npts-1), function(tday) {
+   ##  Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   retis <- retp[startp:endd[tday], ]
+   ##  Calculate the best and worst performing stocks in-sample
+   perfstat <- sapply(retis, objfun)
+   perfstat <- sort(perfstat, decreasing=TRUE)
+   symbolb <- names(head(perfstat, topstocks))
+   symbolw <- names(tail(perfstat, topstocks))
+   ##  Calculate the momentum weights
+   weightv <- numeric(NCOL(retp))
+   names(weightv) <- colnames(retp)
+   weightv[symbolb] <- 1
+   weightv[symbolw] <- (-1)
+   ##  Calculate the in-sample momentum PnLs
+   pnlis <- HighFreq::mult_mat(weightv, retis)
+   pnlis <- rowMeans(pnlis, na.rm=TRUE)
+   ##  Scale weights so in-sample pnl volatility is same as equal
+   weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+   ##  Calculate the out-of-sample momentum returns
+   pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd[t
+   pnlos <- rowMeans(pnlos, na.rm=TRUE)
+   drop(pnlos)
+ }, mc.cores=ncores)  ##  end mclapply
> pnls <- rutils::do_call(c, pnls)
```

Performance of Stock Momentum Strategy

The initial stock momentum strategy underperforms the index because of a poor choice of the model parameters.

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
> retew <- rowMeans(retp, na.rm=TRUE)
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("EqualWeight", "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
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Stock Index and Momentum Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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(retp, objfun, lookb=12, rebalf="months", tops
+  bidask=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf),
+   ##  Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(3:(npts-1), function(tday) {
+     ##  Select the look-back returns
+     startp <- endd[max(1, tday-lookb)]
+     retis <- retp[startp:endd[tday], ]
+     ##  Calculate the best and worst performing stocks in-sample
+     perfstat <- sapply(retis, objfun)
+     perfstat <- sort(perfstat, decreasing=TRUE)
+     symbolb <- names(head(perfstat, topstocks))
+     symbolw <- names(tail(perfstat, topstocks))
+     ##  Calculate the momentum weights
+     weightv <- numeric(NCOL(retp))
+     names(weightv) <- colnames(retp)
+     weightv[symbolb] <- 1
+     weightv[symbolw] <- (-1)
+     ##  Calculate the in-sample momentum pnls
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     ##  Scale weights so in-sample pnl volatility is same as equal
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+     ##  Calculate the out-of-sample momentum returns
+     pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     drop(pnlos)
+   })  ##  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.

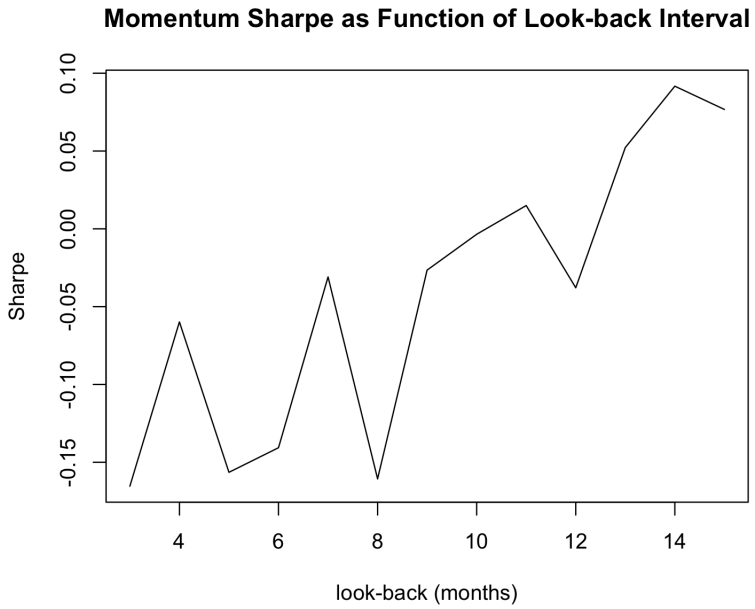
Research indicates that the optimal length of the *look-back interval* for momentum is about 8 to 12 months.

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 long, then the data has large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and estimates will have high *variance*.

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




```
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

```
> # Perform backtests for vector of look-back intervals
> lookbv <- seq(3, 15, by=1)
> endd <- rutils::calc_endpoints(retp, interval="months")
> # Warning - takes very long
> # pnll <- lapply(lookbv, btmomtop, retp=retp, endd=endd, objfun=objfun)
> # Perform parallel loop under Mac-OSX or Linux
> library(parallel)  ##  Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(lookbv, btmomtop, retp=retp, endd=endd, objfun=objfun, mc.cores=ncores)
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
```

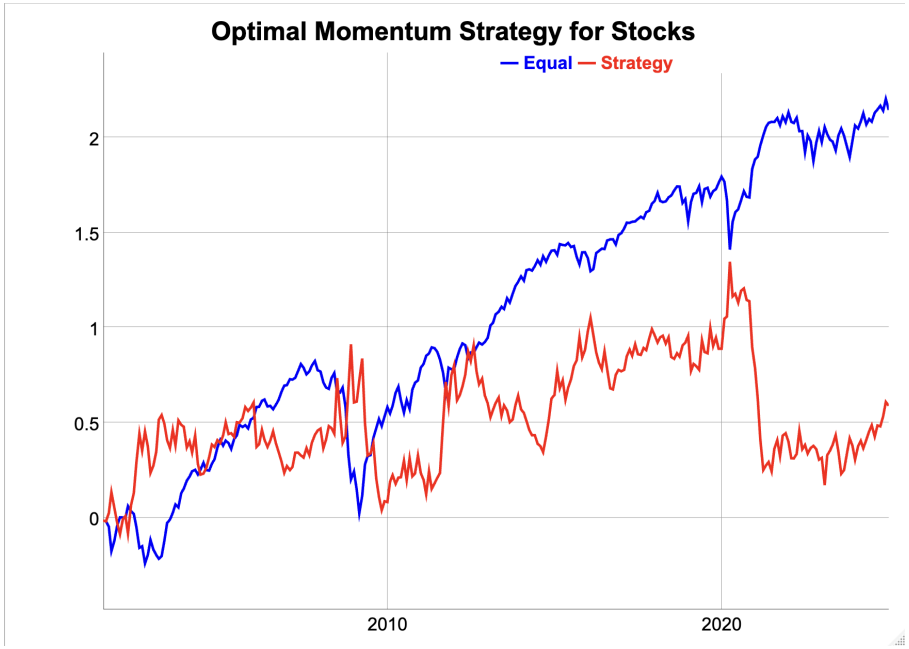
Optimal Stock Momentum Strategy

The best stock momentum strategy underperforms the index because of a poor choice of the model type.

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(sharper)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("EqualWeight", "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
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Optimal Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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(retp, objfun, lookb=12, rebalf="months",
+   bidask=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf),
+   ##  Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(3:(npts-1), function(tday) {
+     ##  Select the look-back returns
+     startp <- endd[max(1, tday-lookb)]
+     retis <- retp[startp:endd[tday], ]
+     ##  Calculate weights proportional to performance
+     perfstat <- sapply(retis, objfun)
+     weightv <- perfstat
+     ##  Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     ##  Scale weights so in-sample pnl volatility is same as equal
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+     ##  Calculate the out-of-sample momentum returns
+     pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     drop(pnlos)
+   })  ##  end lapply
+   rutils::do_call(c, pnls)
+ }  ##  end btmomweight
```

Optimal Weighted Stock Momentum Strategy

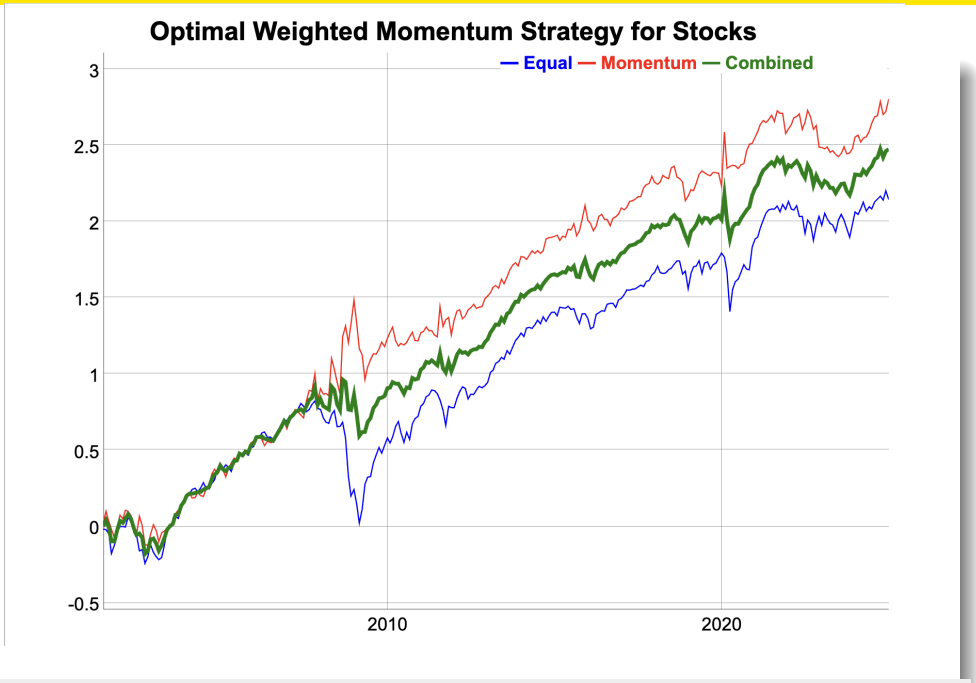
The stock momentum strategy produces a similar absolute return as the index, and also a similar Sharpe ratio.

The advantage of the momentum strategy is that it has a low correlation to stocks, so it can provide significant risk diversification when combined with stocks.

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.




```
> # Perform backtests for vector of look-back intervals
> lookbv <- seq(3, 15, by=1)
> # pnll <- lapply(lookbv, btmomweight, retp=retp, endd=endd, objfu
> # Or perform parallel loop under Mac-OSX or Linux
> library(parallel)  ##  Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(lookbv, btmomweight, retp=retp, endd=endd, objfu
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("EqualWeight", "Momentum", "Combined")
> cor(wealthv)
> 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
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Optimal Weighted Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
```

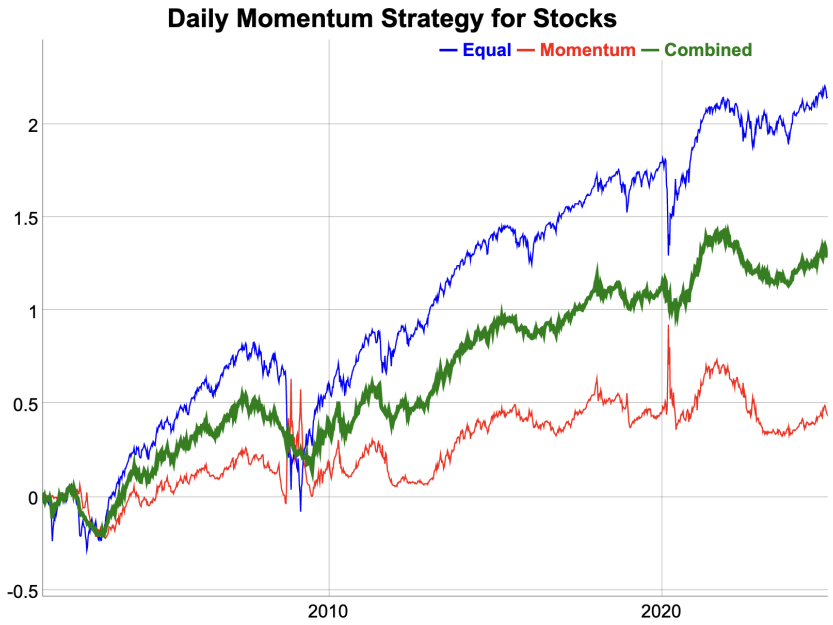
Momentum Strategy With Daily Rebalancing

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

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

A momentum strategy with *daily rebalancing* requires more computations so compiled `C++` functions must be used instead of `apply()` loops.

The functions `HighFreq::run_mean()` and `HighFreq::run_var()` calculate the trailing mean and variance by recursively updating the past estimates with the new values, using the decay factor λ.




```
> # Calculate the trailing average returns and variance using C++ c
> lambdaf <- 0.99
> varm <- HighFreq::run_var(retp, lambda=lambdaf)
> meanm <- varm[, 1:nstocks]
> varm <- varm[, (nstocks+1):(2*nstocks)]
> # Calculate the trailing Kelly ratios
> weightv <- meanm/varm
> weightv <- weightv/sqrt(rowSums(weightv^2, na.rm=TRUE))
> weightv <- rutils::lagit(weightv)
> # Calculate the momentum profits and losses
> pnls <- rowSums(weightv*retp, na.rm=TRUE)
> # Calculate the transaction costs
> bidask <- 0.0
> costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)), na.rm=T
> pnls <- (pnls - costv)
```

```
> # Scale the momentum volatility to the equal weight index
> volew <- sd(retew)
> pnls <- volew*pnls/sd(pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("EqualWeight", "Momentum", "Combined")
> cor(wealthv)
> 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
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Momentum Strategy Functional

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

The decay factor λ determines the rate of decay of the weights applied to the returns, with smaller values of λ producing faster decay, giving more weight to recent returns, and vice versa.

If the argument `trend = -1` then it simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).

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

The function `btmomdaily()` 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(retp, lambdaf=0.9, trend=1, bidask=0.0, ...
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   ##  Calculate the trailing Kelly ratio
+   nstocks <- NCOL(retp)
+   varm <- HighFreq::run_var(retp, lambda=lambdaf)
+   meanm <- varm[, 1:nstocks]
+   varm <- varm[, (nstocks+1):(2*nstocks)]
+   varm[varm < 0.001] <- 0.001 ##  Set volatility floor
+   weightv <- meanm/varm
+   weightv <- weightv/sqrt(rowSums(weightv^2, na.rm=TRUE))
+   weightv <- rutils::lagit(weightv)
+   ##  Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp, na.rm=TRUE)
+   ##  Calculate the transaction costs
+   costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)), na.rm=
+   (pnls - costv)
+ }  ##  end btmomdaily
```

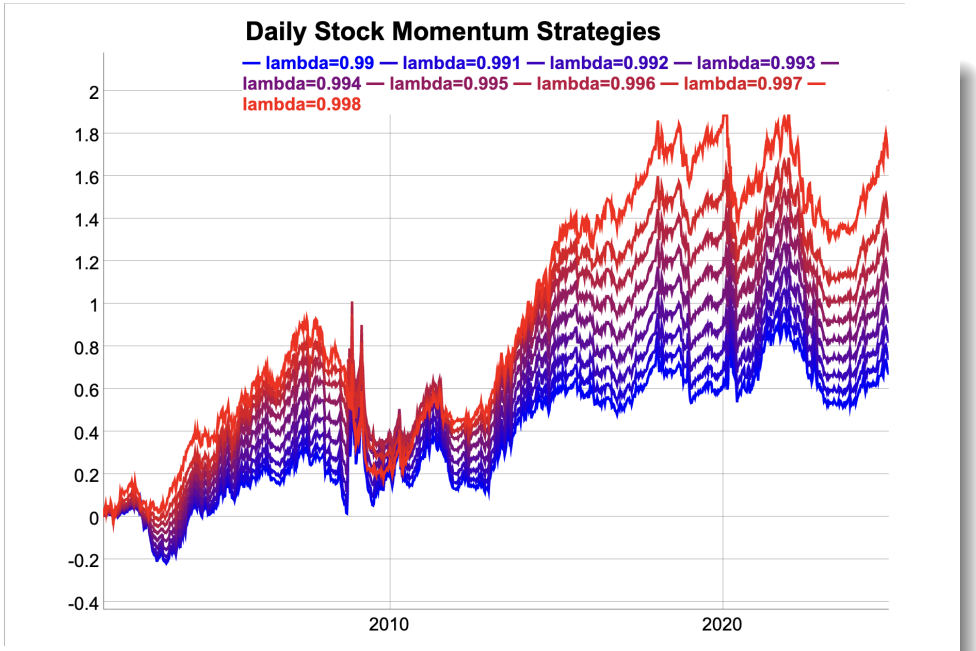
Multiple Daily Stock Momentum Strategies

Multiple daily momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

The best performing momentum strategies with *daily rebalancing* are with λ parameters close to 1.

The momentum strategies with *daily rebalancing* perform worse than with *monthly rebalancing* because of the daily variance of the weights.

```
> # Simulate multiple daily stock momentum strategies
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btmomdaily, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
> tail(pnls)
```




```
> # Plot dygraph of daily stock momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Daily Stock Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot daily stock momentum strategies using quantmod
> themev <- chart_theme()
> themev$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endw],
+   theme=themev, name="Daily Stock Momentum Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=themev$col$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, in order to reduce the variance of the portfolio weights.

Instead of holding the portfolio for only one day, it's held for several days and gradually liquidated. So that many past momentum portfolios are held at the same time.

This is equivalent to averaging the portfolio weights over 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 decay factor λ determines the length of the *holding period*. Smaller values of λ produce a faster decay corresponding to a shorter *holding period*, and vice versa.

The optimal value of the λ parameter 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(retp, lambdaf=0.9, trend=1, bidask=0.0
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   ##  Calculate the trailing Kelly ratio
+   nstocks <- NCOL(retp)
+   varm <- HighFreq::run_var(retp, lambda=lambdaf)
+   meanm <- varm[, 1:nstocks]
+   varm <- varm[, (nstocks+1):(2*nstocks)]
+   varm[varm < 0.001] <- 0.001 ##  Set volatility floor
+   weightv <- meanm/varm
+   weightv <- weightv/sqrt(rowSums(weightv^2, na.rm=TRUE))
+   ##  Average the past weights
+   weightv <- HighFreq::run_mean(weightv, lambda=lambdaf)
+   weightv <- rutils::lagit(weightv)
+   ##  Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp, na.rm=TRUE)
+   ##  Calculate the transaction costs
+   costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)), na.rm=
+   pnls <- (pnls - costv)
+   pnls[1:21] <- 0 ##  Set the warmup PnLs to zero
+   return(pnls)
+ }  ##  end btmomdailyhold
```

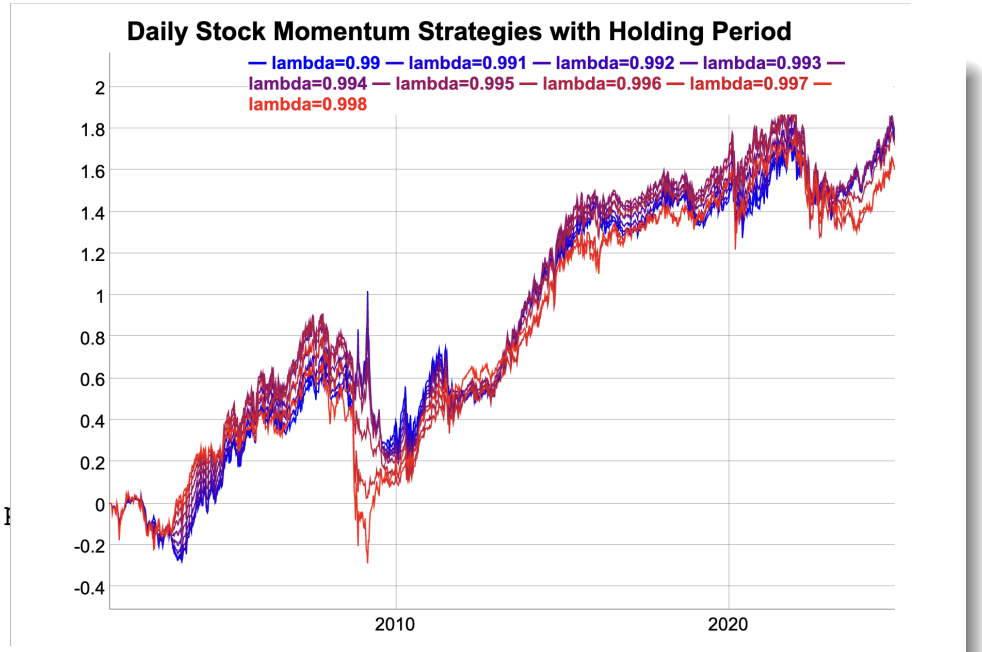
Multiple Daily Momentum Strategies With Holding Period

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

The daily momentum strategies with a holding period perform better than with daily rebalancing.

The reason is that a longer holding period averages the weights and reduces their variance. But this also increases their bias, so there's an optimal holding period for an optimal bias-variance tradeoff.

```
> # Simulate multiple daily stock momentum strategies with holding p
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btmomdailyhold, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
```



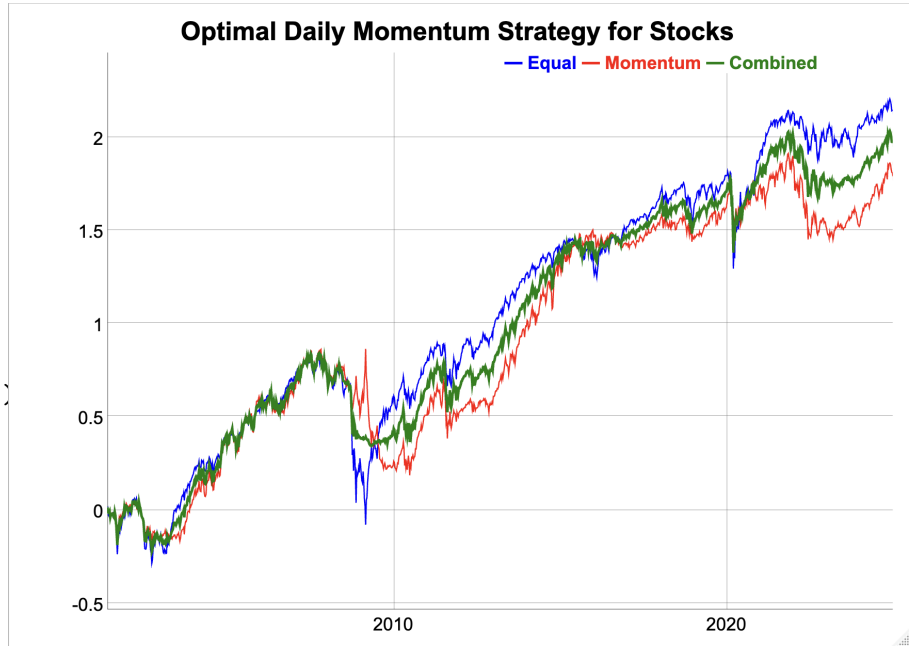

```
> # dygraph of daily stock momentum strategies with holding period
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Daily Stock Momentum Strategies with Holding Period") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot of daily stock momentum strategies with holding period
> themev <- chart_theme()
> themev$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endw],
+   theme=themev, name="Daily Stock Momentum Strategies with Holding
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=themev$col$line.col, bty="n")
```

Optimal Momentum Strategy With Holding Period

The daily momentum strategies with a holding period perform better than with daily rebalancing.

The reason is that a longer holding period averages the weights and reduces their variance. But this also increases their bias, so there's an optimal holding period for an optimal bias-variance tradeoff.

```
> # Calculate best pnls of momentum strategy
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> whichmax <- which.max(sharper)
> lambdav[whichmax]
> pnls <- pnls[, whichmax]
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> colnames(wealthv) <- c("EqualWeight", "Momentum", "Combined")
> cor(wealthv)
> 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
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting Stock Momentum Strategies

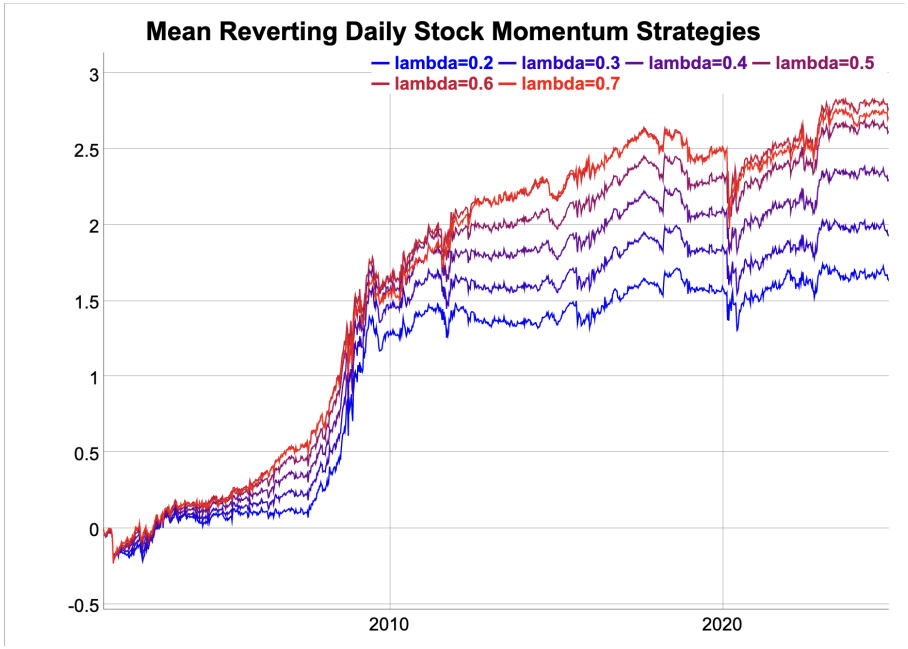
Multiple *mean reverting* stock momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

If the argument `trend = -1` then the function `btmomdaily()` simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).

The *mean reverting* momentum strategies for the stock constituents perform the best for small λ parameters.

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

This simulation doesn't account for transaction costs, which could erase all profits if market orders were used for trade executions. But the strategy could be profitable if limit orders were used for trade executions.




```
> # Perform sapply loop over lambdav
> lambdav <- seq(0.2, 0.7, 0.1)
> pnls <- sapply(lambdav, btmomdaily, retp=retp, trend=(-1))
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
```

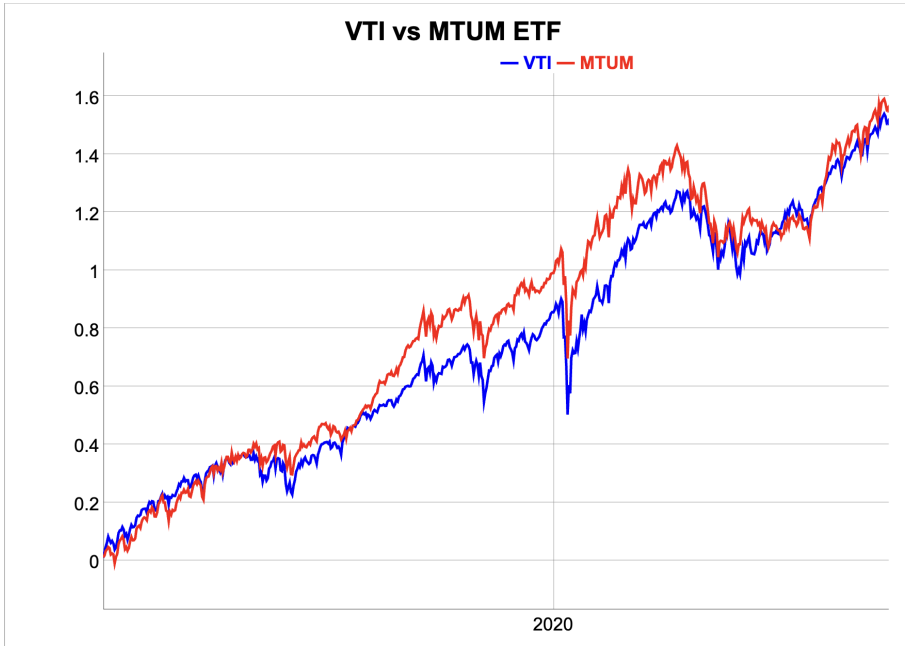
```
> # Plot dygraph of mean reverting daily stock momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Mean Reverting Daily Stock Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
> # Plot mean reverting daily stock momentum strategies using quantm
> themev <- chart_theme()
> themev$col$line.col <- colorRampPalette(c("blue", "red"))(NCOL(pnl
> quantmod::chart_Series(cumsum(pnls)[endw],
+   theme=themev, name="Mean Reverting Daily Stock Momentum Strategi
> legend("topleft", legend=colnames(pnls),
+   inset=0.05, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=themev$col$line.col, bty="n")
```

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.

The weak performance of the *MTUM* ETF demonstrates that it's difficult to implement a successful momentum strategy for individual stocks.




```
> # Calculate the scaled prices of VTI vs MTUM ETF
> wealthv <- na.omit(rutils::etfenv$prices[, c("VTI", "MTUM")])
> wealthv <- rutils::diffit(log(wealthv))
> 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 of scaled prices of VTI vs MTUM ETF
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="VTI vs MTUM ETF") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(width=300)
```

PCA Portfolios

Principal Component Analysis (*PCA*) can be used to construct portfolios of stocks.

The principal components (*PCs*) are portfolios of stocks and can be traded directly as if they were single stocks.

PCA performs better if the stock returns are standardized (mean zero and unit variance).

PCA is equivalent to the *eigen decomposition* of either the correlation or the covariance matrix.

The function `eigen()` calculates the *eigenvectors* and *eigenvalues* of numeric matrices.

Performing *PCA* on returns with `NA` values is more complicated, so they are removed from the data.

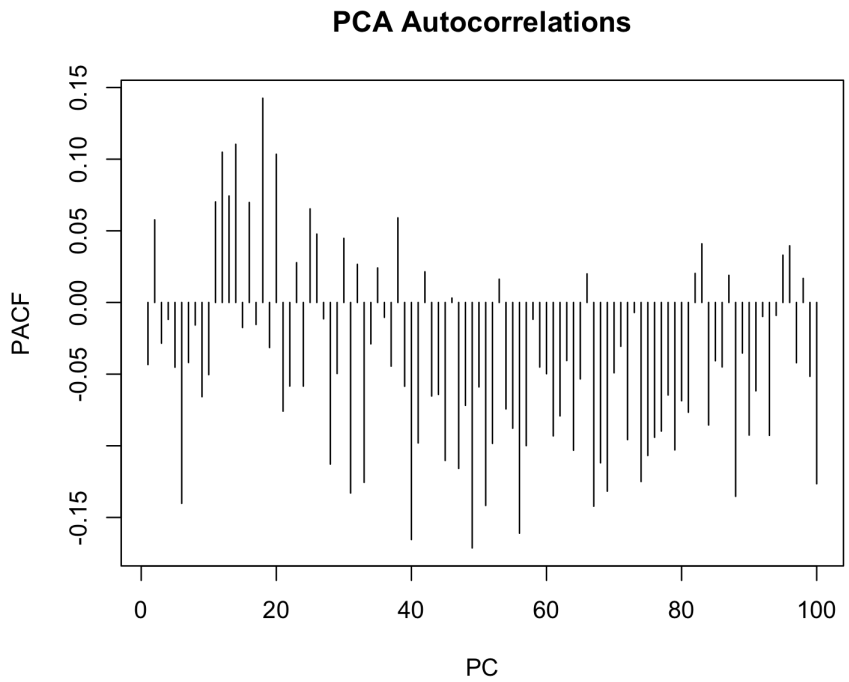
```
> # Select stocks with no NA values in their returns
> numna <- sapply(retp, function(x) sum(is.na(x)))
> retp <- retp[, numna == 0]
> nstocks <- NCOL(retp)
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
> volew <- sd(rowMeans(retp, na.rm=TRUE))
> # Calculate the standardized returns
> retsc <- lapply(retp, function(x) (x - mean(x))/sd(x))
> retsc <- do.call(cbind, retsc)
> # Calculate the PCA loadings
> covmat <- cov(retsc)
> pcad <- eigen(covmat)
> pcal <- pcad$vectors ##  The PCA loadings
> rownames(pcal) <- colnames(retp)
> plot(sort(-pcal[, 1], decreasing=TRUE))
> plot(sort(pcal[, 2], decreasing=TRUE))
> round((t(pcal) %*% pcal)[1:5, 1:5], 4)
> # Calculate the PCA time series from the stock returns and the PCA
> retpca <- retp %*% pcal
> colnames(retpca) <- paste0("PC", 1:nstocks)
> round((t(retpca) %*% retpca)[1:5, 1:5], 4)
```

Autocorrelations of PCA Returns

The *Principal Component* portfolios (*PC*s) have either trending or mean-reverting characteristics.

The lower order *PCs* exhibit greater trending (positive autocorrelations) than individual stocks.

The higher order *PCs* exhibit greater mean reversion (negative autocorrelations) than the lower order ones.




```
> # Calculate the autocorrelations of the PCA time series
> pacv <- apply(retpca[, 1:100], 2, function(x)
+   sum(pacf(x, lag=10, plot=FALSE)$acf))
> plot(pacv, type="h", main="PCA Autocorrelations",
+      xlab="PC", ylab="PACF")
```

Momentum Strategy for PCA Portfolios

The momentum strategy performs better for *PCA* portfolios than for individual stocks for two reasons:

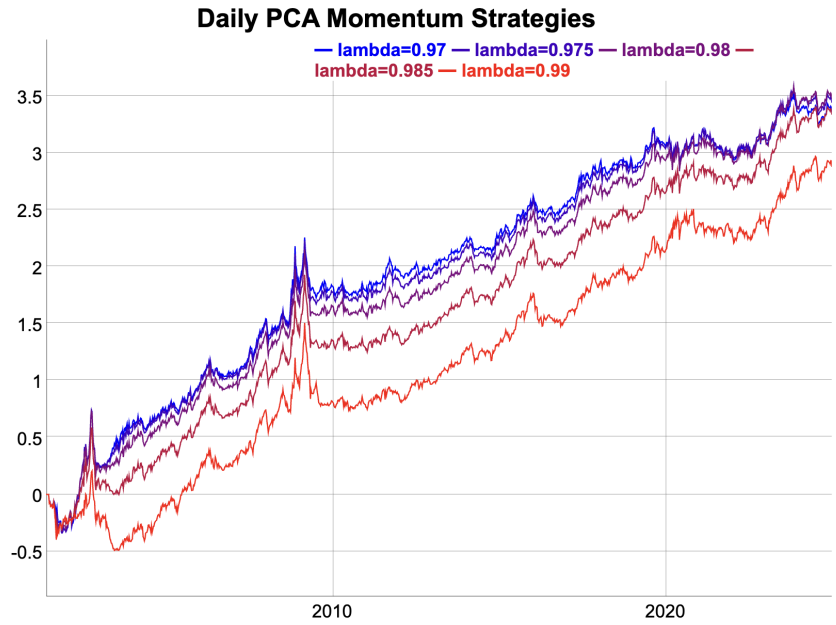
- The *PCA* returns are uncorrelated to each other,
- The lower order *PCA* returns have more positive autocorrelations than individual stocks.
- The higher order *PCA* returns have more negative autocorrelations than individual stocks.

If the stock returns are uncorrelated then the weights of the *maximum Sharpe* portfolio are proportional to the *Kelly ratios* (the returns divided by their variance):

$$w_i = \mathbb{C}^{-1}\bar{r} = \frac{\bar{r}_i}{\sigma_i^2}$$

Where $\mathbb{C}$ is the covariance matrix of returns (which is diagonal in this case).

If the momentum weights are chosen to be the *Kelly ratios*, then the momentum strategy is equivalent to the *maximum Sharpe* optimal portfolio strategy.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> dimax <- 30
> lambdav <- seq(0.97, 0.99, 0.005)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retpca[, 1:dimax]
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
```

```
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambdaf", ylab="Sharpe")
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> endw <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
```

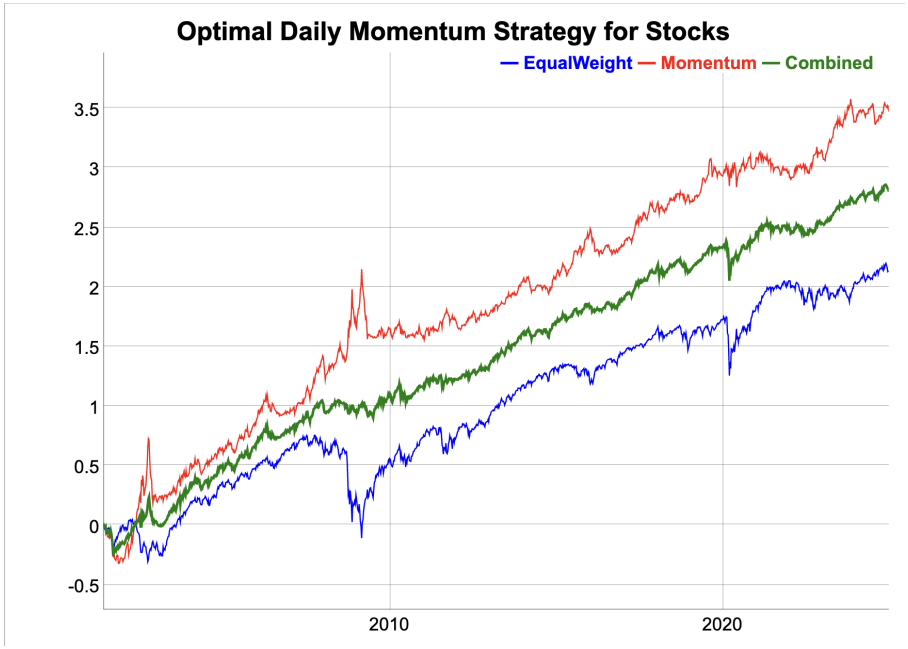
Optimal PCA Momentum Strategy

The *PCA* momentum strategy using only the lowest order *PCs* performs well when combined with the index.

But this is thanks to using the in-sample *PCs*.

The best performing *PCA* momentum strategy has a relatively small decay factor λ, so it's able to quickly adjust to changes in market direction.

```
> # Calculate best pnls of PCA momentum strategy
> whichmax <- which.max(sharper)
> lambdav[whichmax]
> pnls <- pnls[, whichmax]
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> colnames(wealthv) <- c("EqualWeight", "Momentum", "Combined")
> cor(wealthv)
> 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 PCA momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting PCA Momentum Strategy

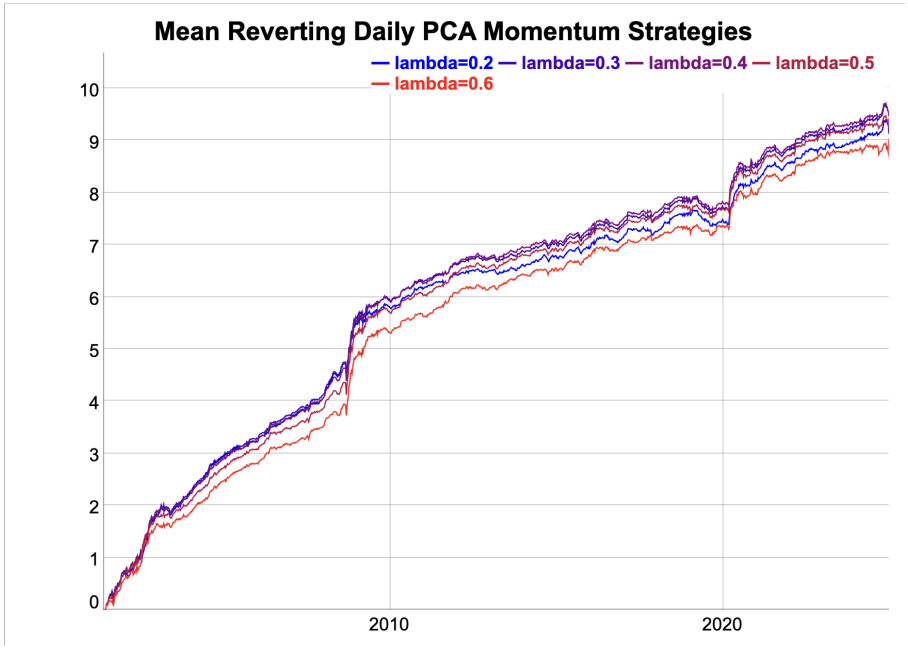
The *mean reverting* momentum strategy buys the worst performing stocks and sells the best.

If the argument `trend` is set to `-1`, then the function `btmomdailyhold()` simulates a mean-reverting strategy.

The *mean reverting* strategy performs well for the higher order *PCs*, but not the highest order, because those have very low volatilities.

The *mean reverting* strategies had their best performance in periods of high volatility, especially during the 2008 financial crisis.

The *mean reverting* strategy trades very frequently, so it's very sensitive to the transaction costs. If the transaction costs are too high then the strategy will lose money.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> lambdav <- seq(0.2, 0.6, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retpca[, (dimax+1
+    trend=(-1), bidask=0.0001, mc.cores=ncores)
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
```

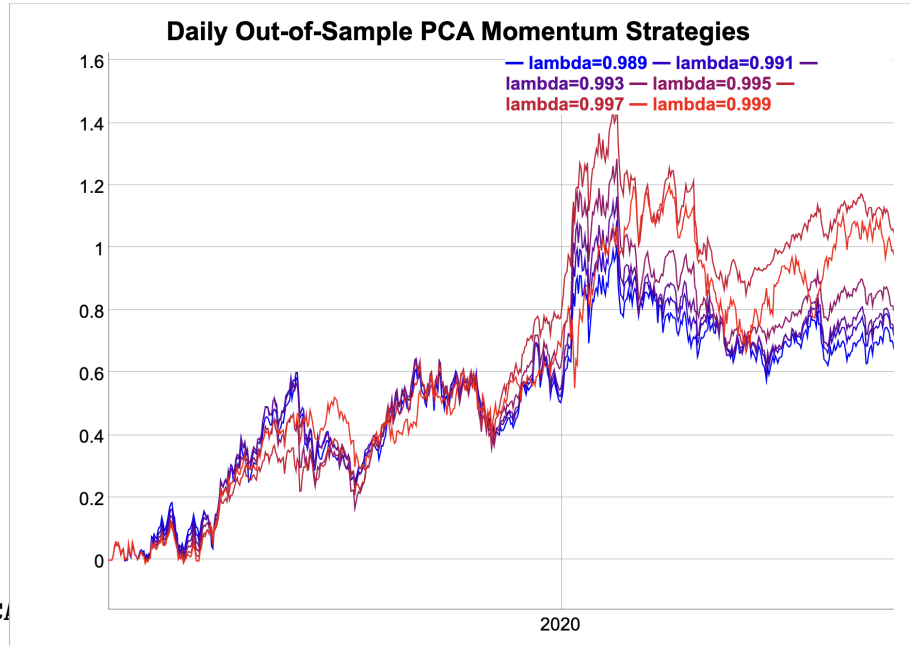
```
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambdaf", ylab="Sharpe")
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
```

PCA Momentum Strategy Out-of-Sample

The principal component weights are calculated in-sample and applied out-of-sample.

The performance is much lower than in-sample, but it's still positive.

```
> # Define in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the PCA loadings in-sample
> covmat <- cov(retsc[insample])
> pcad <- eigen(covmat)
> pcal <- pcad$vectors ##  The PCA loadings
> rownames(pcal) <- colnames(retp)
> # Calculate the PCA time series from the stock returns and the PCA
> retpca <- retp %*% pcal
> colnames(retpca) <- paste0("PC", 1:nstocks)
>
> # Calculate the out-of-sample PCA time series
> retpca <- xts::xts(retpca[outsample, ], order.by=datev[outsample])
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> lambdav <- seq(0.989, 0.999, 0.002)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retpca[, 1:dimax]
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambdaf", ylab="Sharpe")
```



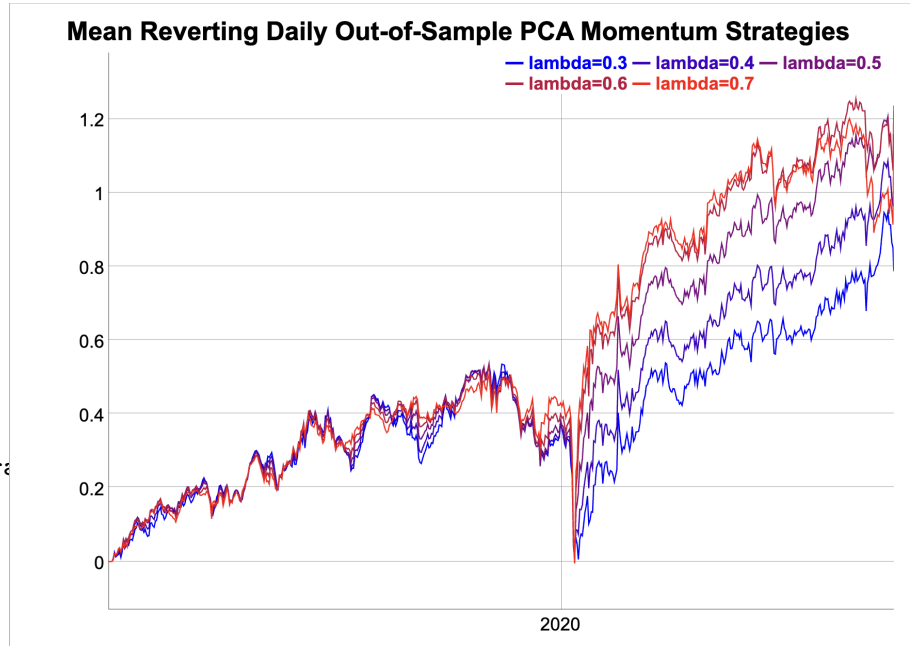

```
> # Calculate a vector of weekly end points
> endw <- rutils::calc_endpoints(pnls, interval="weeks")
> # Plot dygraph of daily out-of-sample PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Daily Out-of-Sample PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting PCA Momentum Strategy Out-of-Sample

The principal component weights are calculated in-sample and applied out-of-sample.

The performance is much lower than in-sample, but it's still positive.

The *mean reverting* strategy trades very frequently, so it's very sensitive to the transaction costs. If the transaction costs are too high then the strategy will lose money.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf para
> lambdav <- seq(0.3, 0.7, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold,
+    retp=retpca[, (dimax+1):(NCOL(retpca)-dimax)],
+    trend=(-1), bidask=0.0001, mc.cores=ncores)
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambdaf", ylab="Sharpe")
```

```
> # Calculate a vector of weekly end points
> endw <- rutils::calc_endpoints(pnls, interval="weeks")
> # Plot dygraph of daily S&P500 momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Mean Reverting Daily Out-of-Sample PCA Momentum Strategies
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=300)
```

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*.

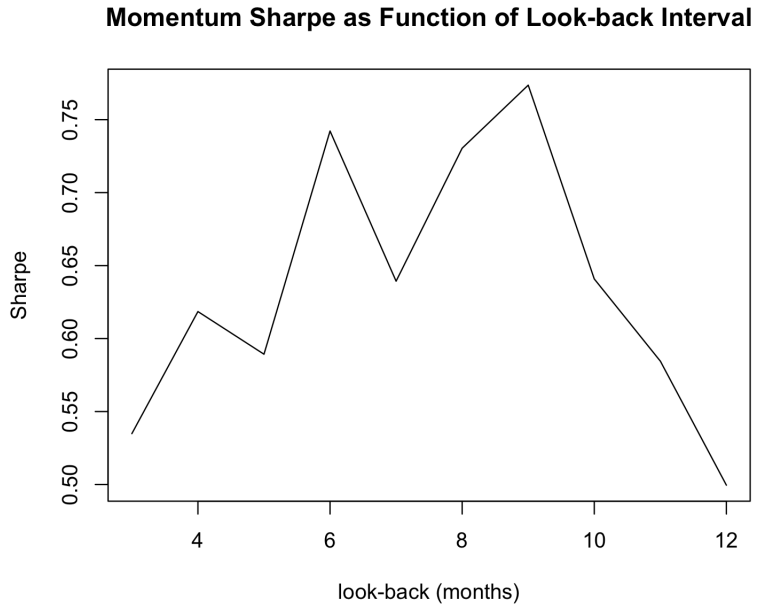
If the *look-back interval* is too long, then the data has large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and estimates will have high *variance*.

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

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")
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> datev <- zoo::index(retp)
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform backtests for vector of look-back intervals
> lookbv <- seq(3, 12, by=1)
> pnll <- lapply(lookbv, btmomweight, retp=retp, endd=endd, objfun=
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

Performance of Momentum Strategy for ETFs

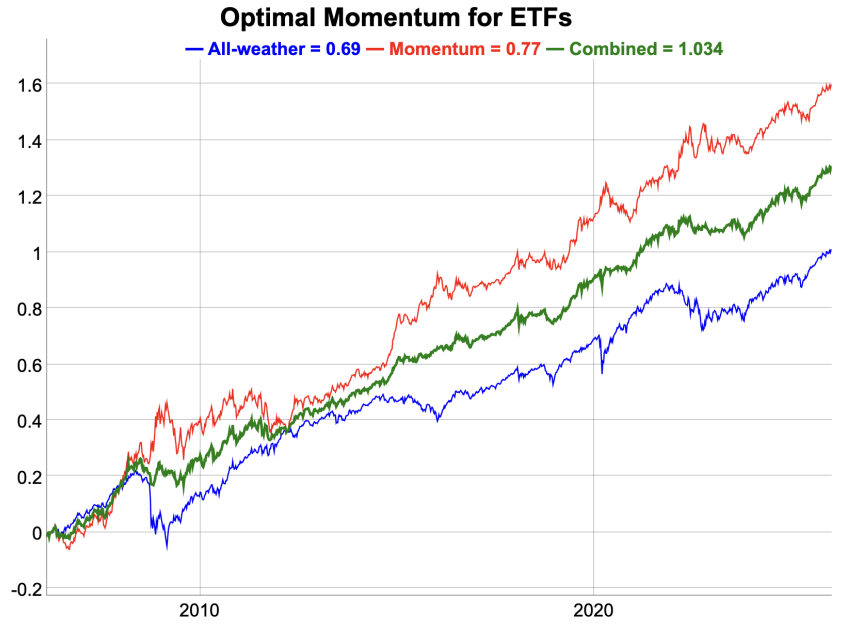
The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

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

The momentum strategy works better for assets that are not correlated or are even anti-correlated.

The momentum strategy also works better for portfolios than for individual stocks because of risk diversification.

Portfolios of stocks can also be selected so that they are more autocorrelated - more trending - they have higher signal-to-noise ratios - larger Hurst exponents.




```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> retew <- rowMeans(retp)
> pnls <- c(retew[endd[1]:endd[3]], pnls)
> # Calculate returns of all-weather benchmark
> weightaw <- c(0.30, 0.55, 0.15)
> retaw <- retp %*% weightaw
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retaw, pnls, 0.5*(retaw+pnls))
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("All-weather", "Momentum", "Combined")
> cor(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> sharper <- sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sharper
```

```
> colnames(wealthv) <- paste(colnames(wealthv), round(sharper[1, ],
> # Plot dygraph of all-weather benchmark and momentum strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw], main="Optimal Momentum fo
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name=colnames(wealthv)[3], strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

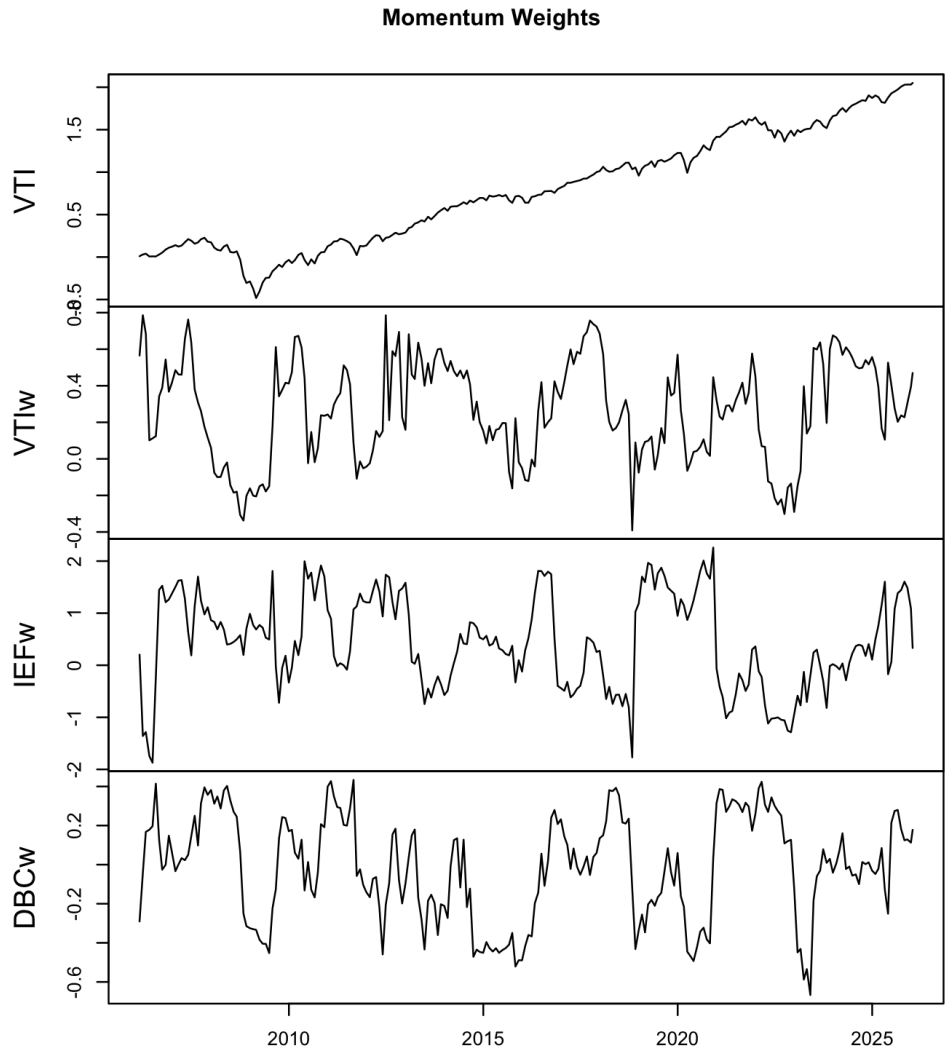
Time Series of Momentum Portfolio Weights

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*.

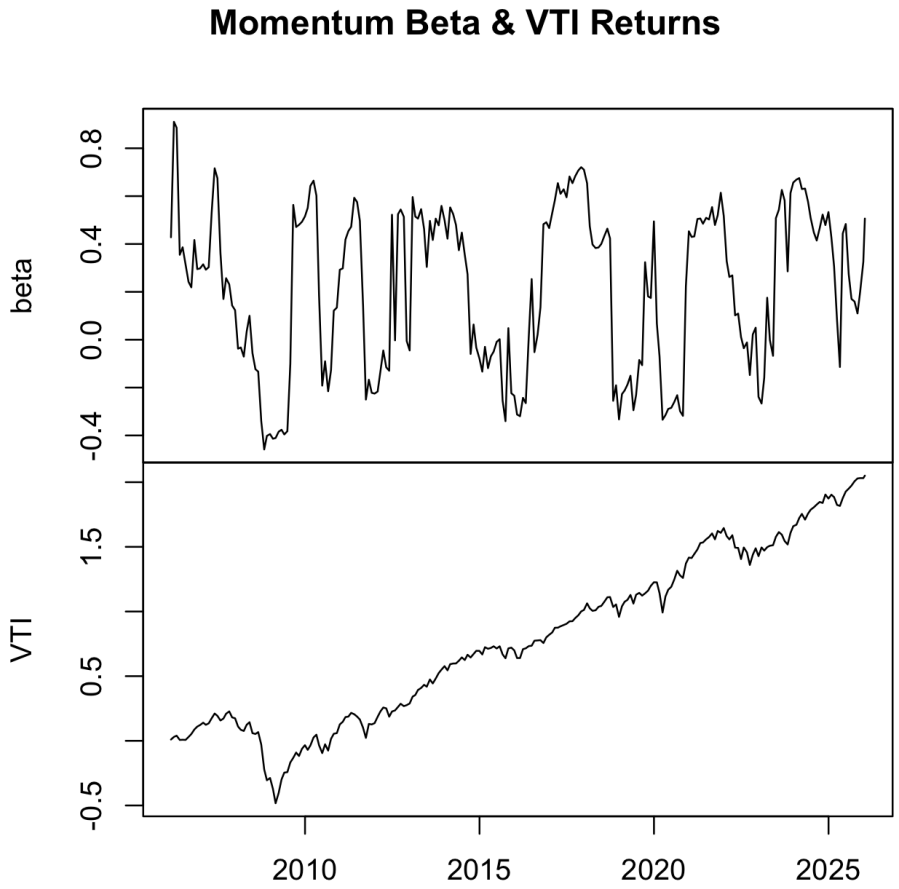
```
> # Calculate the momentum weights
> lookb <- lookbv[whichmax]
> weightv <- lapply(2:npts, function(tday) {
+   ##  Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   retis <- retp[startp:endd[tday], ]
+   ##  Calculate weights proportional to performance
+   perfstat <- sapply(retis, objfun)
+   weightv <- drop(perfstat)
+   ##  Scale weights so in-sample pnl volatility is same as equal
+   pnls <- retis %*% weightv
+   weightv*sd(rowMeans(retis))/sd(pnls)
+ })  ##  end lapply
> weightv <- rutils::do_call(rbind, weightv)
> # Plot of momentum weights
> retvti <- cumsum(retp$VTI)
> endw <- rutils::calc_endpoints(retp, interval="months")
> datav <- cbind(retvti[endw], weightv)
> colnames(datav) <- c("VTI", paste0(colnames(retp), "w"))
> 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
> betetf <- sapply(retp, function(x) cov(retp$VTI, x)/var(retp$VTI))
> # Momentum beta is equal weights times ETF betas
> betam <- weightv %*% betetf
> betam <- xts::xts(betam, order.by=datev[endw])
> colnames(betam) <- "beta"
> datav <- cbind(betam, retvti[endw])
> zoo::plot.zoo(datav, main="Momentum Beta & VTI Returns", xlab="")
```

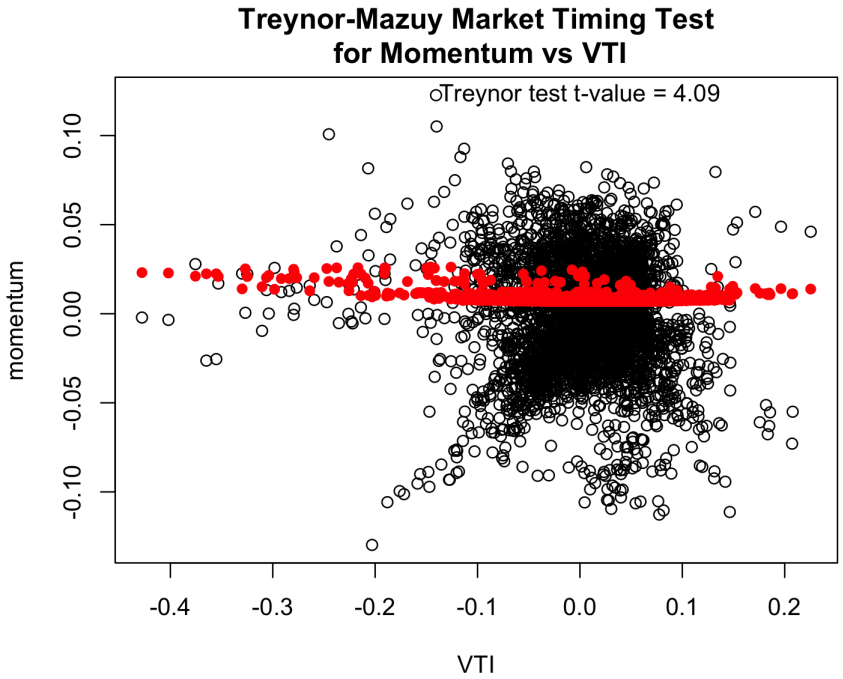


Momentum Strategy Market Timing Skill

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

Since the momentum strategy forecasts over one month intervals, we should also aggregate the returns to one month intervals.

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




```
> # Aggregate the returns to monthly intervals
> retvti <- retp$VTI
> desm <- cbind(pnls, retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> desm <- HighFreq::roll_sumep(desm, lookb=22)
> colnames(desm) <- c("pnls", "VTI", "Merton", "Treynor")
> desm <- as.data.frame(desm)
> # Merton-Henriksson test
> regmod <- lm(pnls ~ VTI + Merton, data=desm); summary(regmod)
> # Treynor-Mazuy test
> regmod <- lm(pnls ~ VTI + Treynor, data=desm); summary(regmod)
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=desm[, "VTI"], y=resids, xlab="VTI", ylab="momentum")
> title(main="Treynor-Mazuy Market Timing Test\n for Momentum vs VTI", line=0.5)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*desm[, "VTI"]
> tvalue <- round(coefreg["Treynor", "t value"], 2)
> points.default(x=desm[, "VTI"], y=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), paste("Treynor test t-value =", tvalue))
```

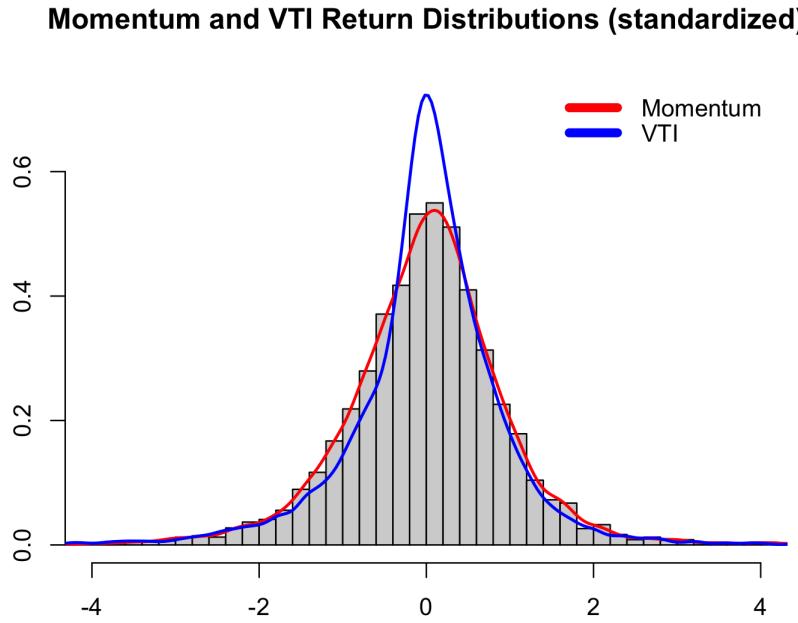
Skewness of Momentum Strategy Returns

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)
```




```
> # Calculate kernel density of VTI
> densvti <- density(retvti)
> # Plot histogram of momentum returns
> hist(pnlsd, breaks=80,
+   main="Momentum and VTI Return Distributions (standardized)",
+   xlim=c(-4, 4), ylim=range(densvti$y), xlab="", ylab="", freq=FAI
> # Draw kernel density of histogram
> lines(density(pnlsd), col='red', lwd=2)
> lines(densvti, col='blue', lwd=2)
> # Add legend
> legend("topright", inset=0.0, cex=1.0, title=NULL,
+  leg=c("Momentum", "VTI"), bty="n", y.intersp=0.7,
+  lwd=6, bg="white", col=c("red", "blue"))
```

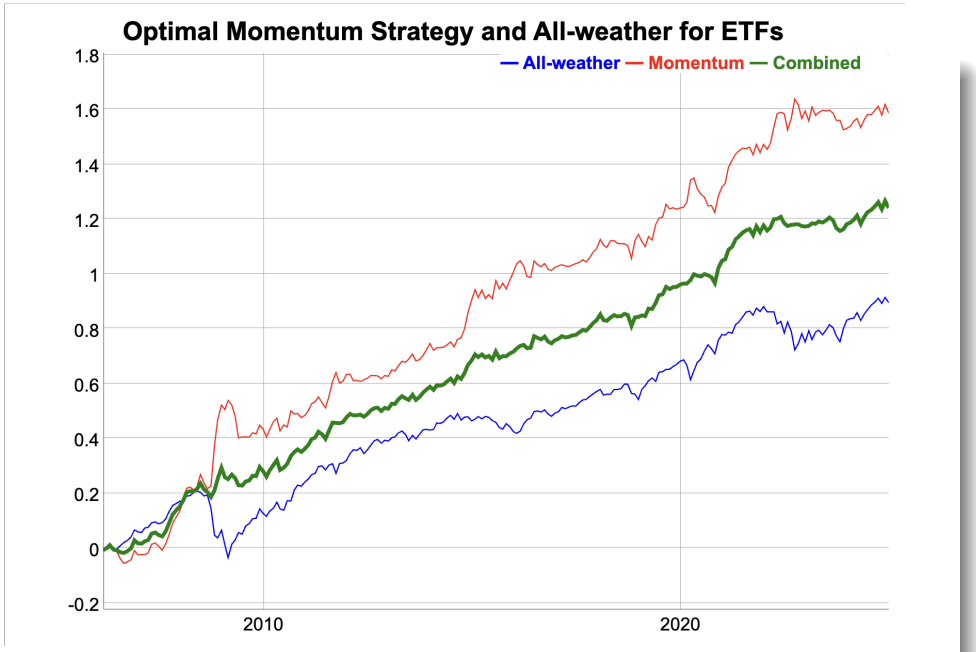
Combining Momentum with the *All-Weather* Portfolio

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(retaw, pnls, 0.5*(pnls + retaw))
> colnames(wealthv) <- c("All-weather", "Momentum", "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)[endw],
+   main="Optimal Momentum Strategy and All-weather for ETFs") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

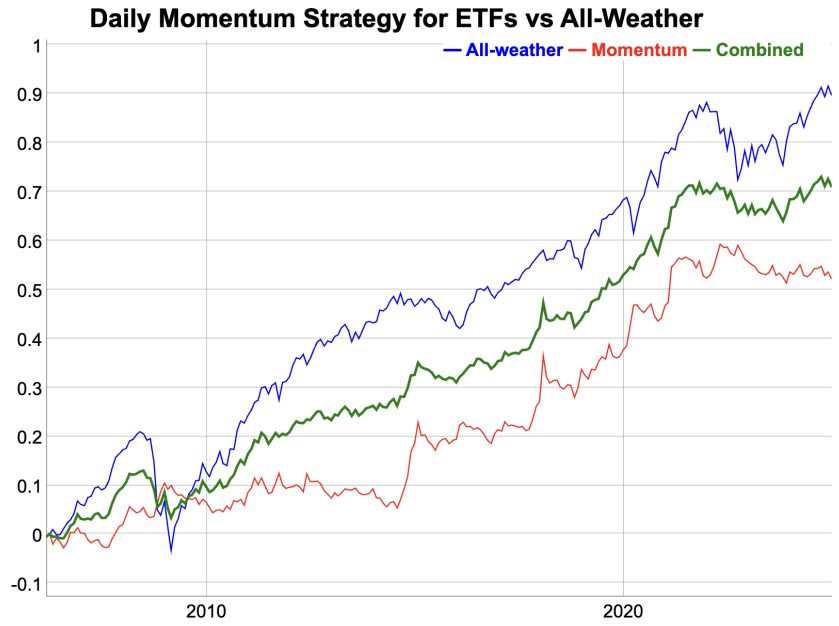
Momentum Strategy for ETFs With Daily Rebalancing

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 must be used instead of `apply()` loops.

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

The advantage of the momentum strategy is that it's negatively correlated to the *All-Weather* returns, so when it's combined with it, it reduces the risk and increases the *Sharpe ratio*.




```
> # Calculate the EMA returns and variance
> varm <- HighFreq::run_var(retp, lambda=0.98)
> meanv <- varm[, 1:3]
> varm <- varm[, 4:6]
> # Calculate the trailing Kelly ratios
> weightv <- meanv/varm
> # weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weightv)
> # Calculate the momentum profits and losses
> pnls <- rowSums(weightv*retp)
> # Calculate the transaction costs
> bidask <- 0.0
> costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costv)
```

```
> # Scale the momentum volatility to all-weather
> pnls <- sd(retaw)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- cbind(retaw, pnls, 0.5*(pnls + retaw))
> colnames(wealthv) <- c("All-weather", "Momentum", "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])))
> cor(wealthv)
> # Plot dygraph of the momentum strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Daily Momentum Strategy for ETFs vs All-Weather") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Backtesting of Strategies

Backtesting is the simulation of a trading or investment strategy on historical data.

Backtesting is performed by *training* the strategy (calibrating the model parameters) on the past in-sample data and *testing* it on future out-of-sample data.

Backtesting is a type of *cross-validation* applied to investment strategies.

Backtesting can be used to rank strategies from the most promising to the least promising.

Backtest simulations can be useful for determining the relative advantages of different strategy features and model parameters.

Backtesting can be used to screen out strategies that have little chance of being profitable.

Because if a strategy is not profitable in backtesting then it has little chance of being profitable in live trading.

Backtesting can be used to identify strategies that have a better chance of being profitable than others.

But backtesting cannot ensure that a strategy will be profitable in live trading, just because it is profitable in simulation.

The *transaction costs* are equal to half the *bid-ask spread* δ times the absolute value of the traded dollar amounts of the *risky assets*.

Backtesting of Momentum Strategies

The *momentum* strategy PnLs are calculated by multiplying the out-of-sample stock returns times the past portfolio weights.

The *momentum* portfolio weights are calculated using the past stock returns, so they depend on the length of the *look-back* interval.

The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*.

The longer the *look-back interval* the less *variance* but the more *bias*, and vice versa.

If the *look-back interval* is too long, then the portfolio weights have large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and the portfolio weights will have high *variance*.

Backtesting can be used to determine the optimal length of the *look-back interval*.

Momentum strategies perform better when applied to portfolios of stocks rather than to individual stocks. Because portfolios diversify the risk.

More frequent portfolio rebalancing doesn't perform better because it increases the variance of the portfolio weights.

Mean reversion momentum performs well with shorter look-back intervals and smaller lambda decay factors.

Limitations of Backtest Simulations

The simulated strategy pnls from backtesting may not be realistic for several reasons:

- Transaction costs (broker commissions, bidask spread, market impact).
- Costs of stock borrowing (stocks must be borrowed in order to short them).
- Limits on stock borrowing and shorting (stocks may not be available for borrow, or they may be prohibited from shorting).

So the backtest simulations should be thought of as experiments to explore which features and parameters have better potential for profit.

But the backtest simulations cannot guarantee future profits.

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

Homework Assignment

Required

Study all the lecture slides in `FRE7241_Lecture_5.pdf`, and run all the code in `FRE7241_Lecture_5.R`

Recommended

- Read about momentum strategies:
  Moskowitz Time Series Momentum.pdf
  Bouchaud Momentum Mean Reversion Equity Returns.pdf
  Hurst Pedersen AQR Momentum Evidence.pdf
- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
- Read about *PCA* in:
  pca-handout.pdf
  pcaTutorial.pdf