Univariate Investment Strategies

FRE7241, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

February 21, 2023

NYU | TANDON SCHOOL OF ENGINEERING

Single Period Binary Gamble

Consider a single investment (gamble) with a binary outcome:
The investor makes no up-front payments, and either wins an amount a (with probability p), or loses an amount b (with probability $q = 1 - p$).

	win	lose
probability	p	q = 1 - p
payout	a	-b
terminal wealth	1 + a	1 - b

The initial wealth is equal to 1 dollar, and the terminal wealth after the gamble is either $1 + a$ (with probability p), or $1 - b$ (with probability $q = 1 - p$).

The amounts a and b are expressed as percentages of the wealth risked in the gamble, and the ratio a/b is called the *betting odds*.

The expected return on the gamble is called the *edge* and is equal to: $\mu = p\,a - q\,b$, and the variance of returns is equal to: $\sigma^2 = p\,q\,(a+b)^2$.

If the investor chooses to risk only a fraction k_f of wealth, then the return on the gamble is either $k_f\,a$ (with probability p), or $-k_f\,b$ (with probability $q = 1 - p$).

The fraction k_f can be greater than 1 (leveraged investing), or it can be negative (shorting).

And the expected return on the gamble is equal to: $p\,k_f\,a - q\,k_f\,b = k_f\,\mu$.

If an investor makes decisions exclusively based on the expected return μ, then they would either invest all their wealth ($k_f = 1$) on the gamble if $\mu > 0$, or choose not to invest at all ($k_f = 0$) if $\mu < 0$.

Without loss of generality we can assume that $p = q = \frac{1}{2}$.

And then $\mu = 0.5\,(a - b)$, and $\sigma^2 = 0.25\,(a+b)^2$.

The *Sharpe ratio* of the gamble is then equal to:

$$S_r = \frac{\mu}{\sigma} = \frac{(a-b)}{(a+b)}$$

Investor Utility and Fractional Betting

The *expected utility* hypothesis states that investors try to maximize the expected *utility* of wealth, not the expected wealth.

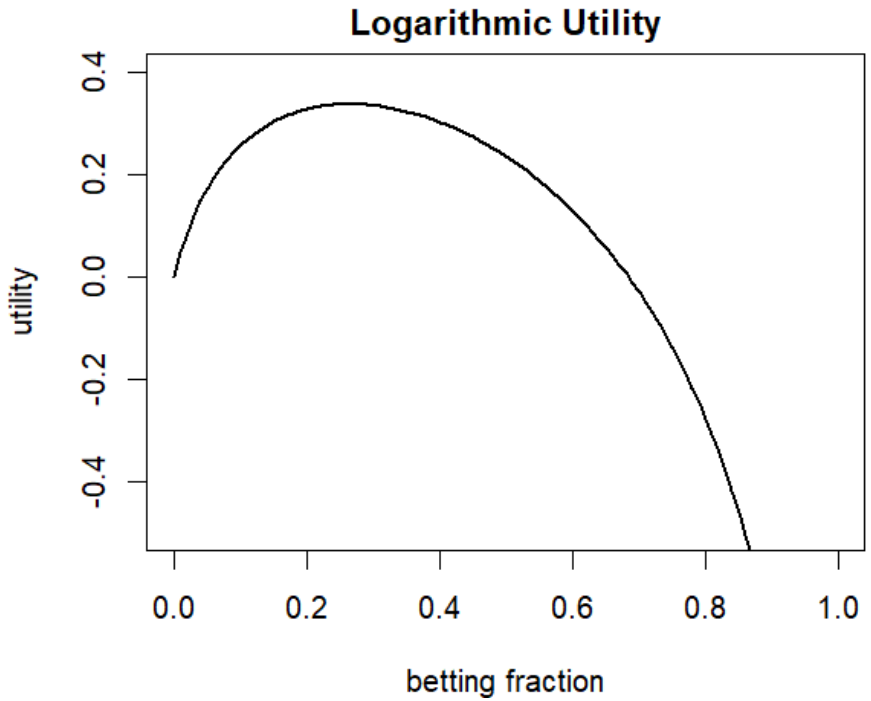
In 1738 Daniel Bernoulli introduced the concept of *logarithmic utility* in his work *"Specimen Theoriae Novae de Mensura Sortis"* (New Theory of the Measurement of Risk).

The *logarithmic utility* function is defined as the logarithm of wealth: $u(w) = \log(w)$.

Under *logarithmic utility* investor preferences depend on the percentage change of wealth, instead of the absolute change of wealth: $\mathrm{d}u(w) = \frac{\mathrm{d}w}{w}$.

An investor with *logarithmic utility* invests only a fraction k_f of their wealth in a gamble, depending on the risk-return of the gamble.

If the initial wealth is equal to 1, then the expected value of *logarithmic utility* for the binary gamble is equal to: $u(k_f) = p\ \log(1 + k_f a) + q\ \log(1 - k_f b)$.




```
> # Define logarithmic utility
> utilfun <- function(frac, p=0.3, a=20, b=1) {
+   p*log(1+frac*a) + (1-p)*log(1-frac*b)
+ }  # end utilfun
> # Plot utility
> curve(expr=utilfun, xlim=c(0, 1),
+ ylim=c(-0.5, 0.4), xlab="betting fraction",
+ ylab="utility", main="", lwd=2)
> title(main="Logarithmic Utility", line=0.5)
```

Optimal Fractional Betting Under Logarithmic Utility

The betting fraction that maximizes the *utility* can be found by equating the derivative of *utility* to zero:

$$\frac{\mathrm{d}u(k_f)}{\mathrm{d}k_f} = \frac{p\,a}{1+k_f\,a} - \frac{q\,b}{1-k_f\,b} = 0$$

$$k_f = \frac{p}{b} - \frac{q}{a} = \frac{p\,a - q\,b}{b\,a} = \frac{\mu}{b\,a}$$

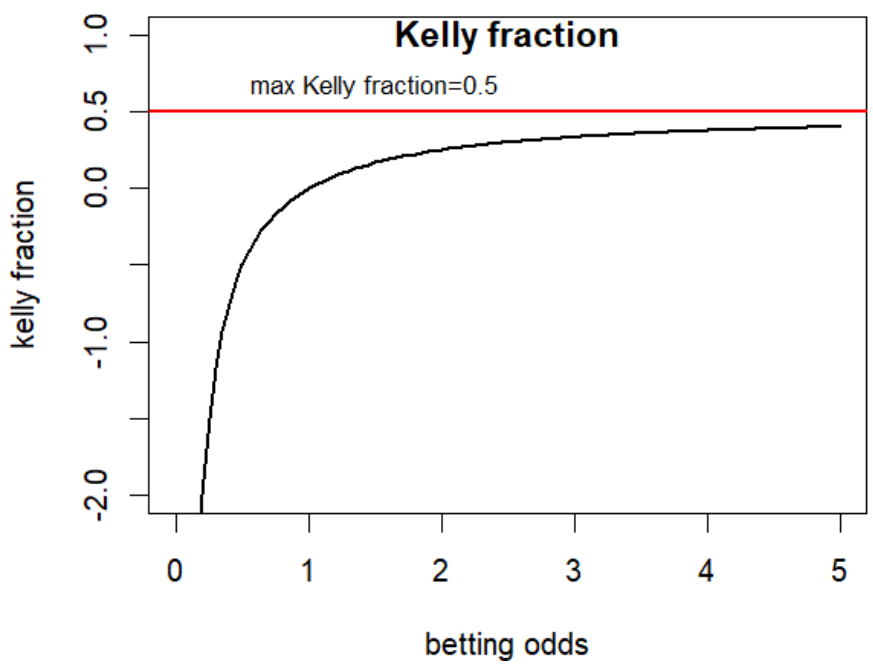
The optimal k_f is called the *Kelly fraction*, and it depends on the parameters of the gamble.

The *Kelly fraction* can be greater than 1 (leveraged investing), or it can be negative (shorting).

If we assume that $b = 1$, then the betting odds are equal to a and the *Kelly fraction* is: $k_f = \frac{p(a+1)-1}{a}$

The *Kelly fraction* is then equal to the expected payout divided by the betting odds.

If the expected payout of the gamble is not positive, then an investor with logarithmic utility should not allocate any capital to the gamble.




```
> # Define and plot Kelly fraction
> kelly_frac <- function(a, p=0.5, b=1) {
+   p/b - (1-p)/a
+ }  # end kelly_frac
> curve(expr=kelly_frac, xlim=c(0, 5),
+ ylim=c(-2, 1), xlab="betting odds",
+ ylab="kelly fraction", main="", lwd=2)
> abline(h=0.5, lwd=2, col="red")
> text(x=1.5, y=0.5, pos=3, cex=0.8, labels="max Kelly fraction=0.5"
> title(main="Kelly fraction", line=-0.8)
```

The Kelly Criterion

The *Kelly criterion* states that investors should bet the optimal *Kelly fraction* of their capital in a gamble.

Investors with concave utility functions (for example logarithmic utility) are sensitive to the risk of ruin (losing all their capital).

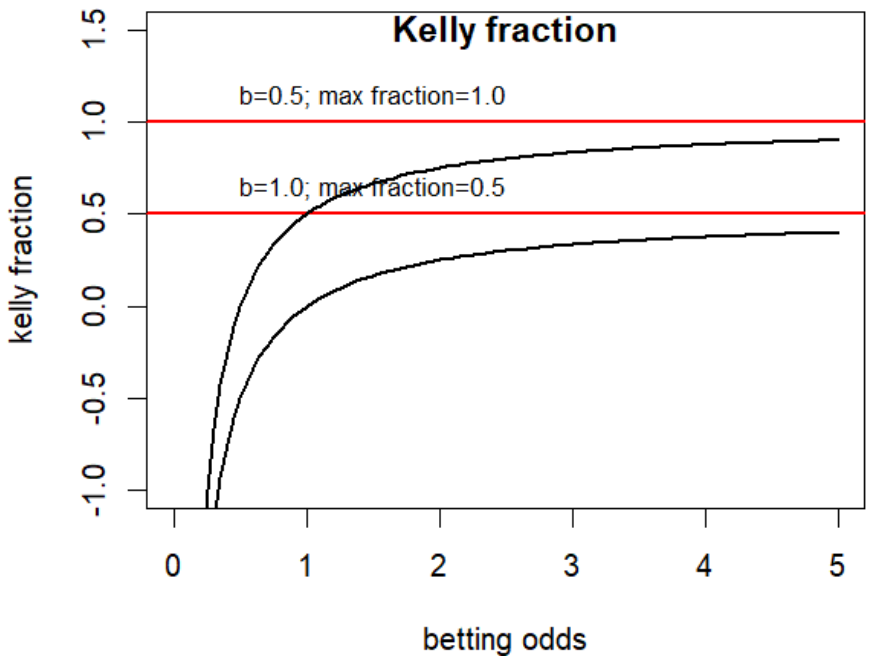
Applying the *Kelly criterion* and betting only a fraction of their capital reduces the risk of ruin (but it doesn't eliminate the risk if prices drop suddenly).

The loss amount b determines the risk of ruin, with larger values of b increasing the risk of ruin.

Therefore investors will choose a smaller betting fraction k_f for larger values of b.

This means that even for huge odds in their favor, investors may not choose to invest all their capital, because of the risk of ruin.

For example, if the betting odds are very large $a \to \infty$, then the *Kelly fraction*: $k_f = \frac{p}{b}$.




```
> # Plot several Kelly curves
> curve(expr=kelly_frac(x, b=1), xlim=c(0, 5),
+ ylim=c(-1, 1.5), xlab="betting odds",
+ ylab="kelly fraction", main="", lwd=2)
> abline(h=0.5, lwd=2, col="red")
> text(x=1.5, y=0.5, pos=3, cex=0.8, labels="b=1.0; max fraction=0.5
> curve(expr=kelly_frac(x, b=0.5), add=TRUE, main="", lwd=2)
> abline(h=1.0, lwd=2, col="red")
> text(x=1.5, y=1.0, pos=3, cex=0.8, labels="b=0.5; max fraction=1.0
> title(main="Kelly fraction", line=-0.8)
```

Utility of Multiperiod Betting

Let r_i be the random return on the gamble in period i, and let $w_i = (1 + k_f r_i)$ be the random wealth increment.

Then the terminal wealth after n rounds is equal to the compounded wealth increments:
$w_n = \prod_{i=1}^{n} w_i = \prod_{i=1}^{n}(1 + k_f r_i)$.

And the utility is equal to the sum of the individual utilities:

$$u_n = \log(w_n) = \sum_{i=1}^{n} \log(w_i) = \sum_{i=1}^{n} \log(1+k_f r_i) = \sum_{i=1}^{n} u_i$$

The individual utilities are all maximized by the same *Kelly fraction* k_f, so the *Kelly fraction* for multiperiod betting is the same as for single period betting:

$$k_f = \frac{p}{b} - \frac{q}{a}$$

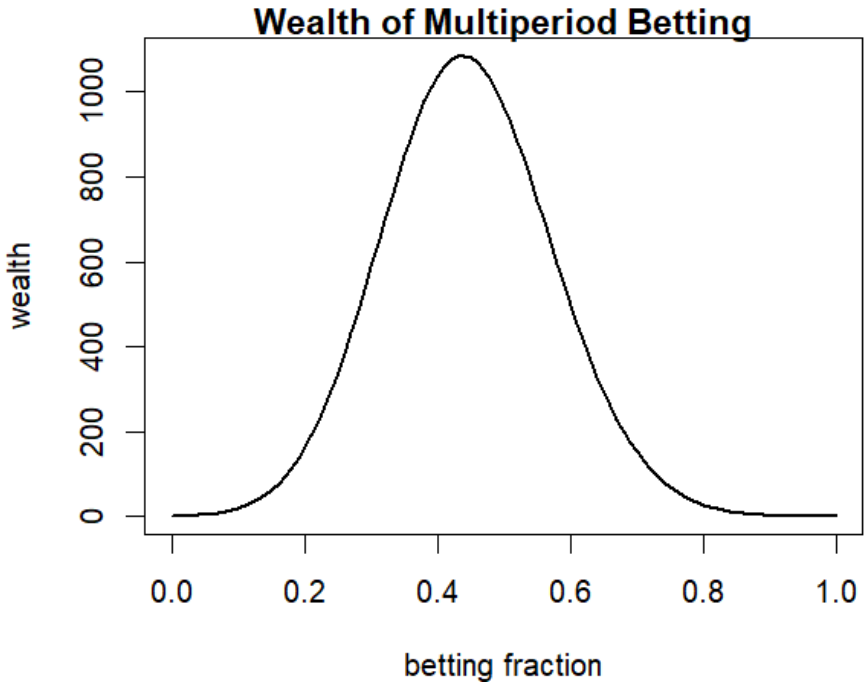
Wealth of Multiperiod Betting

In multiperiod betting the investor participates in n rounds of gambles, and in each round they risk a fixed fraction k_f of their current outstanding wealth.

In each round the wealth is multiplied by either $(1 + k_f a)$ (win) or $(1 - k_f b)$ (loss), so that the current outstanding wealth changes over time.

The terminal wealth after n rounds with m wins is equal to: $w(k_f) = (1 + k_f a)^m (1 - k_f b)^{n-m}$.

If the number of rounds n is very large, then the number of wins is almost always equal to $m = n\,p$, and the terminal wealth is equal to: $w(k_f) = (1 + k_f a)^{np} (1 - k_f b)^{nq}$.




```
> # Wealth of multiperiod binary betting
> wealthv <- function(f, a=0.8, b=0.1, n=1e3, i=150) {
+   (1+f*a)^i * (1-f*b)^(n-i)
+ }  # end wealth
> curve(expr=wealthv, xlim=c(0, 1),
+ xlab="betting fraction",
+ ylab="wealth", main="", lwd=2)
> title(main="Wealth of Multiperiod Betting", line=0.1)
```

Optimal Multiperiod Betting

The betting fraction k_f that maximizes the terminal wealth is found by setting the derivative of $w(k_f)$ to zero:

$$\frac{\mathrm{d}w(k_f)}{\mathrm{d}k_f} = npa(1+k_f a)^{np-1}(1-k_f b)^{nq} - nqb(1+k_f a)^{np}(1-k_f b)^{nq-1}$$

$$= \left(\frac{npa}{1+k_f a} - \frac{nqb}{1-k_f b}\right)(1+k_f a)^{np}(1-k_f b)^{nq} = 0$$

We can then solve for the optimal betting fraction k_f:

$$\frac{pa}{1+k_f a} - \frac{qb}{1-k_f b} = 0$$

$$pa(1-k_f b) - qb(1+k_f a) = 0$$

$$pa - qb - k_f ab = 0$$

$$k_f = \frac{pa-qb}{ab} = \frac{p}{b} - \frac{q}{a}$$

The above is just the *Kelly fraction* k_f that maximizes the utility.

So the *Kelly fraction* k_f that maximizes the utility also maximizes the terminal wealth.

depr: Multiperiod Binary Gambles

The terminal wealth after n repeated gambles with m wins is equal to: $(1 + k_f a)^m (1 - k_f b)^{n-m}$.

And the expected value of the wealth is equal to:

$$w(k_f) = \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} (1 + k_f a)^m (1 - k_f b)^{n-m}$$

We can then find the fraction k_f which maximizes the expected wealth $w(k_f)$:

$$\frac{\mathrm{d}w(k_f)}{\mathrm{d}k_f} = \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} (1 + k_f a)^m (1 - k_f b)^{n-m} \left(\frac{am}{1 + k_f a} - \frac{b(n-m)}{1 - k_f b} \right) =$$

$$\frac{a}{1 + k_f a} \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} m -$$

$$\sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} (1 + k_f a)^m (1 - k_f b)^{n-m} \left(\frac{am}{1 + k_f a} - \frac{b(n-m)}{1 - k_f b} \right)$$

If the investor chooses to risk only a fraction k_f of wealth, then the wealth after the gamble is either $1 + k_f a$ (with probability p), or $1 - k_f b$ (with probability $q = 1 - p$).

(with probability p), or $1 - b$ (with probability $q = 1 - p$). initial wealth is equal to 1, and the The *Kelly fraction* for multiperiod betting can be found by maximizing the expected *utility* of the final wealth distribution:

$$u(k_f) = \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} \log((1 + k_f a)^m (1 - k_f b)^{n-m})$$

depr: Utility of Multiperiod Binary Gambles

The *Kelly fraction* for multiperiod betting can be found by maximizing the expected *utility* of the final wealth distribution:

$$u(k_f) = \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} \log((1 + k_f a)^m (1 - k_f b)^{n-m})$$

$$= \log(1 + k_f a) \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} m +$$

$$\log(1 - k_f b) \sum_{m=0}^{n} \binom{n}{m} p^m q^{n-m} (n - m)$$

$$= n\,p\,\log(1 + k_f a) + n\,q\,\log(1 - k_f b)$$

The above is just the single period *utility* multiplied by the number of rounds of betting n.

The *Kelly fraction* k_f for multiperiod betting is the same as for single period betting:

$$k_f = \frac{p}{b} - \frac{q}{a}$$

Investing With Fixed Margin

Let r_i be the percentage returns on a *risky asset*, so that the asset price p_t at time t is given by:

$$p_t = p_0 \prod_{i=1}^{t} (1 + r_i)$$

The initial investor wealth at time $t = 0$ is equal to 1 dollar, and they also borrow on margin m dollars to invest in the *risky asset*.

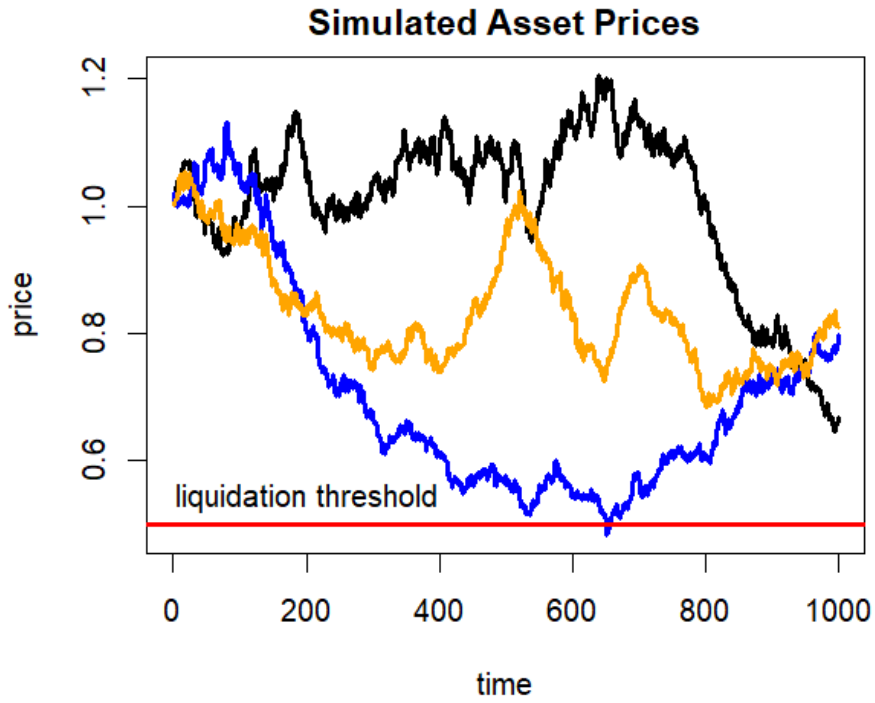
The investor's *wealth* at time t is equal to (the margin borrowing rate is assumed to be zero):

$$w_t = 1 + m \frac{p_t - p_0}{p_0}$$

The *leverage* k_f is equal to the *margin debt* m divided by the total wealth w_t: $k_f = m/w_t$.

If the asset price drops then the *leverage* increases, because the *margin debt* is fixed while the wealth drops.

If the asset price drops enough so that the wealth reaches zero, then the investment is liquidated and the investor is ruined.




```
> # Simulate asset prices
> calc_pricev <- function(x) cumprod(1 + rnorm(1e3, sd=0.01))
> price_paths <- sapply(1:3, calc_pricev)
> plot(price_paths[, 1], type="l", lwd=3,
+      main="Simulated Asset Prices",
+      ylim=range(price_paths),
+      lty="solid", xlab="time", ylab="price")
> lines(price_paths[, 2], col="blue", lwd=3)
> lines(price_paths[, 3], col="orange", lwd=3)
> abline(h=0.5, col="red", lwd=3)
> text(x=200, y=0.5, pos=3, labels="liquidation threshold")
```

Investing With Fixed Leverage

In order to avoid ruin, the investor may choose to maintain a fixed *leverage ratio* equal to k_f, so that the amount invested in the *risky asset* is proportional to the *wealth*: $k_f\, w_t$.

This requires buying the *risky asset* when its price increases, and selling it when it drops.

The return on the *risky asset* in a single period is equal to: $k_f\, w_t\, r_t$, so the *terminal wealth* at time t is equal to the compounded returns:

$$w_t = (1 + k_f\, r_1) \ldots (1 + k_f\, r_t) = \prod_{i=1}^{t} (1 + k_f\, r_i)$$

The utility of the *terminal wealth* is equal to the sum of the utilities of single periods:

$$\mathbb{E}[\log w_t] = \mathbb{E}[\log((1 + k_f\, r_1) \ldots (1 + k_f\, r_t))]$$

$$= \sum_{i=1}^{t} \mathbb{E}[\log\,(1 + k_f\, r_i)] = t\, \mathbb{E}[\log\,(1 + k_f\, r)]$$

The last equality holds because all the utilities of single periods are the same.

Let the returns over a short time period be equal to r, with probability distribution $p(r)$.

The mean return $\bar{r}$, and variance σ^2 are:

$$\bar{r} = \int r\, p(r)\, \mathrm{d}r\ ; \quad \sigma^2 = \int (r - \bar{r})^2\, p(r)\, \mathrm{d}r$$

Since the returns are over a short time period, we have: $r \ll 1$ and $\bar{r} \ll \sigma$, so that we can replace $r - \bar{r}$ with r as follows:

$$\int (r - \bar{r})^2\, p(r)\, \mathrm{d}r \approx \int r^2\, p(r)\, \mathrm{d}r$$

Utility of Leveraged Asset Returns

So the utility of the *terminal wealth* u_t is equal to the utility of a single period times the number of periods:

$$u_t = \mathbb{E}[\log w_t] = t\, \mathbb{E}[\log (1 + k_f\, r)] = t\, u_r$$

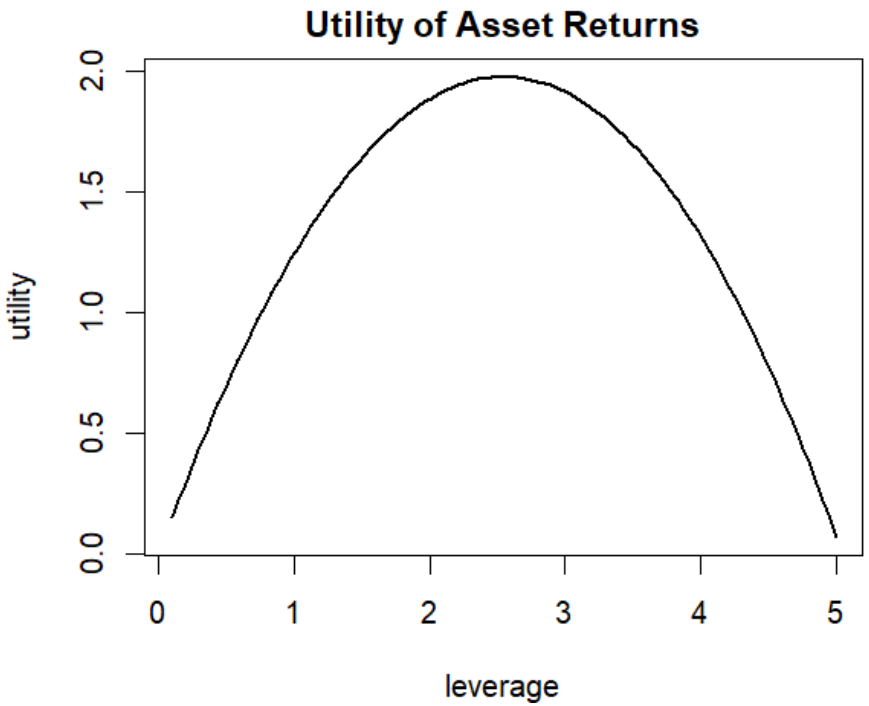
The utility of the asset returns u_r is equal to:

$$u_r = \mathbb{E}[\log (1 + k_f\, r)] = \int \log(1 + k_f\, r)\, p(r)\, \mathrm{d}r$$

The leverage k_f is limited so that $(1 + k_f\, r) > 0$ for all return values r.

If the mean returns are positive, then at first the utility increases with leverage, but only up to a point.

With higher leverage, the negative utility of time periods with negative returns becomes significant, forcing the aggregate utility to drop.




```
> # Calculate the VTI returns
> retvti <- rutils::etfenv$returns$VTI
> retvti <- na.omit(retvti)
> c(mean=mean(retvti), std=sd(retvti))
> range(retvti)
```

```
> # Define vectorized logarithmic utility function
> utilfun <- function(kellyfrac, retvti) {
+   sapply(kellyfrac, function(x) sum(log(1 + x*retvti)))
+ }  # end utilfun
> utilfun(1, retvti)
> utilfun(c(1, 4), retvti)
> # Plot the logarithmic utility
> curve(expr=utilfun(x, retvti=retvti),
+ xlim=c(0.1, 5), xlab="leverage", ylab="utility",
+ main="Utility of Asset Returns", lwd=2)
```

Kelly Criterion for Optimal Leverage of Asset Returns

The *logarithmic utility* u_r can be expanded in the moments of the return distribution:

$$u_r = \mathbb{E}[\log(1 + k_f\, r)] = \int \log(1 + k_f\, r)\, p(r)\, \mathrm{d}r$$

$$= \int (k_f\, r - \frac{(k_f\, r)^2}{2} + \frac{(k_f\, r)^3}{3} - \frac{(k_f\, r)^4}{4})\, p(r)\, \mathrm{d}r$$

$$= k_f \bar{r} - \frac{k_f^2 \sigma^2}{2} + \frac{k_f^3 \sigma^3 \varsigma}{3} - \frac{k_f^4 \sigma^4 \kappa}{4}$$

Where $\varsigma = \int \frac{r^3}{\sigma^3}\, p(r)\, \mathrm{d}r$ is the *skewness*, and $\kappa = \int \frac{r^4}{\sigma^4}\, p(r)\, \mathrm{d}r$ is the *kurtosis*.

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

$$\frac{\mathrm{d}u_r}{\mathrm{d}k_f} = \bar{r} - k_f \sigma^2 + k_f^2 \sigma^3 \varsigma - k_f^3 \sigma^4 \kappa = 0$$

This shows that the logarithmic utility has positive odd derivatives and negative even derivatives.

Assuming that the third and fourth moments $\sigma^4 \varsigma$ and $\sigma^4 \kappa$ are small and can be neglected, we get:

$$k_f = \frac{\bar{r}}{\sigma^2} = \frac{S_r}{\sigma}\ ; \quad u_r = \frac{1}{2}\frac{\bar{r}^2}{\sigma^2} = \frac{1}{2} S_r^2$$

The *Kelly leverage* is *approximately* equal to the *Sharpe ratio* divided by the *standard deviation*.

The optimal utility u_r is *approximately* equal to half the *Sharpe ratio* S_r squared.

The *standard deviation* and *Sharpe ratio* are calculated over the same time interval as the returns (not annualized).

```
> # Approximate Kelly leverage
> mean(retvti)/var(retvti)
> PerformanceAnalytics::KellyRatio(R=retvti, method="full")
> # Kelly leverage
> unlist(optimize(
+   f=function(x) -utilfun(x, retvti),
+   interval=c(1, 4)))
```

Kelly Strategy Wealth Path

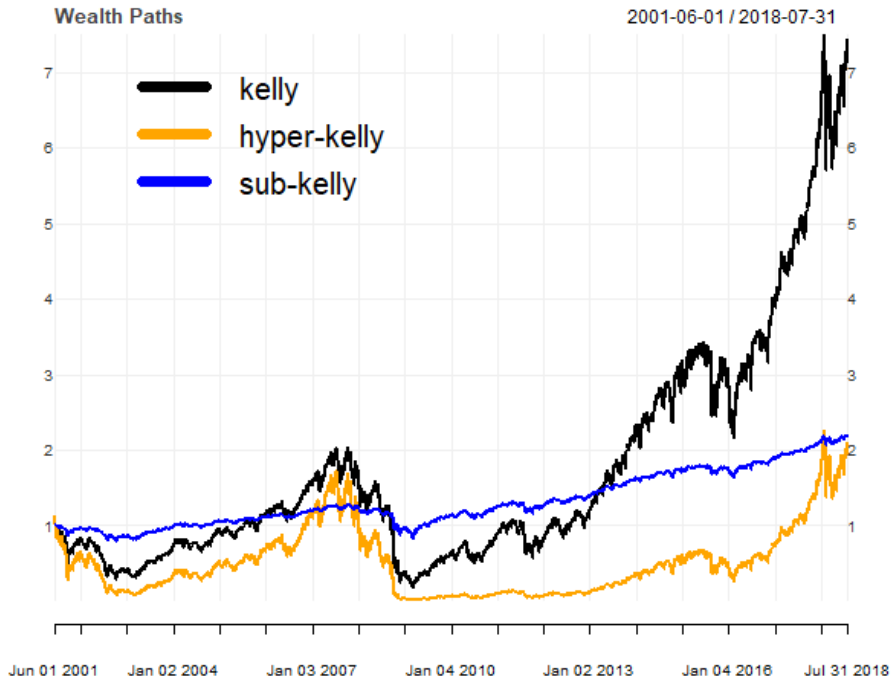
The wealth of a Kelly Strategy with a fixed leverage ratio k_f is equal to:

$$w_t = \prod_{i=1}^{t} (1 + k_f \, r_i)$$

The *Kelly fraction* k_f provides the optimal leverage to maximize the utility of wealth, by balancing the benefit of leveraging higher positive returns, with the risk of ruin due to excessive leverage.

If the mean asset returns are positive, then a higher leverage ratio provides higher returns.

But if the leverage is too high, then the losses in periods with negative returns wipe out most of the wealth, so then it's slow to recover.




```
> retvti <- na.omit(retvti)
> # Calculate wealth paths
> kelly_ratio <- drop(mean(retvti)/var(retvti))
> kelly_wealthv <- cumprod(1 + kelly_ratio*retvti)
> hyper_kelly <- cumprod(1 + (kelly_ratio+2)*retvti)
> sub_kelly <- cumprod(1 + (kelly_ratio-2)*retvti)
> kelly_paths <- cbind(kelly_wealth, hyper_kelly, sub_kelly)
> colnames(kelly_paths) <- c("kelly", "hyper-kelly", "sub-kelly")
```

```
> # Plot wealth paths
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("black", "orange", "blue")
> quantmod::chart_Series(kelly_paths, theme=plot_theme, name="Wealth
> legend("topleft", legend=colnames(kelly_paths),
+  inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+  col=plot_theme$col$line.col, bty="n")
```

Kelly Strategy With Margin Account

The *margin debt* m_t is equal to the dollar amount borrowed to purchase the *risky asset*.

The wealth w_t at time t is equal to the initial wealth $w_0 = 1$ plus the dollar amount of the *risky asset* a_t, minus the *margin debt* m_t: $w_t = 1 + a_t - m_t$.

The dollar amount of the *risky asset* a_t is equal to the wealth w_t times the *leverage* k_f: $a_t = k_f w_t$.

So the *margin debt* m_t is proportional to the wealth w_t: $m_t = (k_f - 1)w_t + 1$.

The wealth changes from w_{t-1} to: $w_t = w_{t-1}(1 + k_f\, r_t)$, while the dollar amount of the *risky asset* changes from $a_{t-1} = k_f w_{t-1}$ to: $a_t = k_f w_{t-1}(1 + r_t)$, so that the leverage changes from k_f to:

$$\frac{k_f w_{t-1}(1 + r_t)}{w_{t-1}(1 + k_f\, r_t)} = \frac{k_f(1 + r_t)}{1 + k_f\, r_t}$$

In order to maintain a fixed *leverage ratio* equal to k_f, the investor must actively trade the *risky asset*, and the *margin debt* m_t changes over time.

The change in margin in a single time period is equal to:

$$\Delta m_t = (k_f - 1)\Delta w_t = k_f(k_f - 1)w_{t-1}r_t$$

The dollar amount of the *risky asset* traded is equal to the change in *margin*.

Therefore the investor must borrow on margin and buy the *risky asset* when its price increases, and sell it when it drops.

Kelly Strategy With Transaction Costs of Trading

The *bid-offer spread* is the percentage difference between the *offer* minus the *bid* price, divided by the *mid* price.

The *bid-offer spread* for liquid stocks can be assumed to be about 10 basis points (bps).

The *transaction costs* c^r due to the *bid-offer spread* are equal to half the *bid-offer spread* δ times the absolute value of the traded dollar amount of the *risky asset*:

$$c^r = \frac{\delta}{2} |\Delta m_t|$$

If the transaction costs are much less than the change in wealth $c^r \ll |\Delta w_t|$, then we can write approximately:

$$c^r = \frac{\delta}{2} k_f (k_f - 1) w_{t-1} |r_t|$$

The wealth of the Kelly Strategy after accounting for the *bid-offer spread* is then equal to:

$$w_t = \prod_{i=1}^{t} (1 + k_f \, r_i - \frac{\delta}{2} k_f (k_f - 1) |r_i|)$$

The effect of the *bid-offer spread* is to reduce the effective asset returns by an amount proportional to the *bid-offer spread*.

```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Calculate wealth paths
> kelly_ratio <- drop(mean(retvti)/var(retvti))
> wealthv <- cumprod(1 + kelly_ratio*retvti)
> wealth_trans <- cumprod(1 + kelly_ratio*retvti -
+   0.5*bid_offer*kelly_ratio*(kelly_ratio-1)*abs(retvti))
> # Calculate compounded wealth from returns
> wealthv <- cbind(wealthv, wealth_trans)
> colnames(wealthv) <- c("Kelly", "Including bid-offer")
> # Plot compounded wealth
> dygraphs::dygraph(wealthv, main="Kelly Strategy With Transaction C
+   dyOptions(colors=c("green", "blue"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

draft: The Half-Kelly Criterion

In reality investors don't know the probability of winning or the odds of the gamble, so they can't accurately calculate the optimal *Kelly fraction*.

The *Kelly fraction*: $k_f = \frac{\bar{r}}{\sigma^2}$ is especially sensitive to the uncertainty of the expected returns $\bar{r}$.

If the expected returns are over-estimated, then it can produce an inflated value of the *Kelly fraction*, leading to ruin.

The risk of applying too much leverage (over-betting) is much greater than the risk of applying too little leverage (under-betting).

Too much leverage (over-betting) not only reduces returns, but it increases the risk of ruin.

So in practice many investors apply only half the theoretical *Kelly fraction* (the Half-Kelly), to reduce the risk of ruin.

Perform bootstrap simulation to obtain the standard error of the *Kelly fraction*.

```
> # Plot several Kelly curves
> curve(expr=kelly_frac(x, b=1), xlim=c(0, 5),
+ ylim=c(-1, 1.5), xlab="betting odds",
+ ylab="kelly fraction", main="", lwd=2)
> abline(h=0.5, lwd=2, col="red")
> text(x=1.5, y=0.5, pos=3, cex=0.8, labels="b=1.0; max fraction=0.5
> curve(expr=kelly_frac(x, b=0.5), add=TRUE, main="", lwd=2)
> abline(h=1.0, lwd=2, col="red")
> text(x=1.5, y=1.0, pos=3, cex=0.8, labels="b=0.5; max fraction=1.0
> title(main="Kelly fraction", line=-0.8)
```

Risk Aversion

Risk aversion is the investor preference to avoid losses more than to seek similar percentage gains in wealth.

For example, for a risk averse investor, a 10% loss of wealth is more important than a 10% gain.

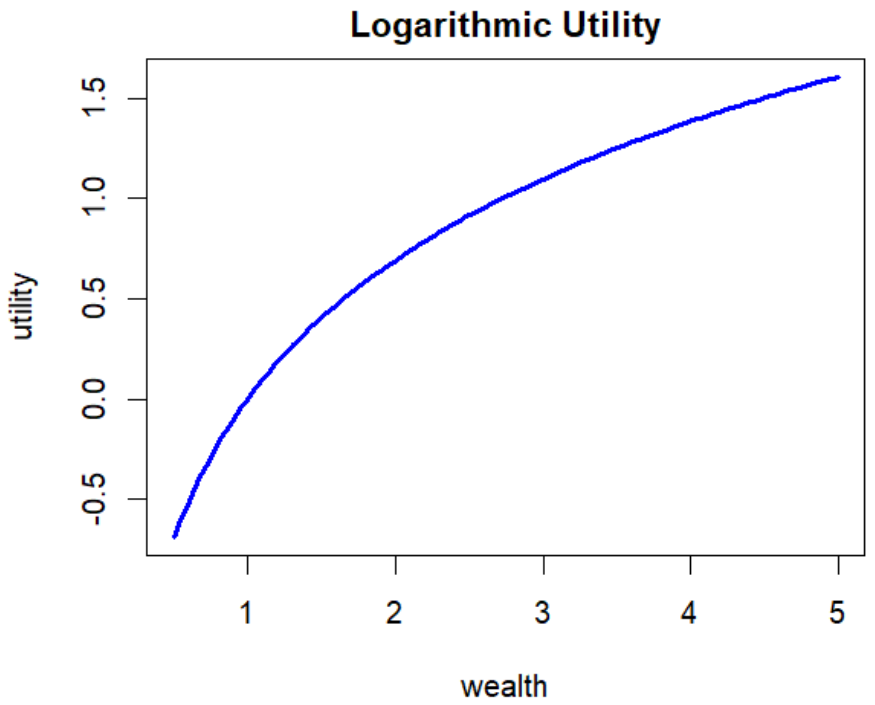
Risk aversion is associated with the *diminishing marginal utility* of the percentage change in wealth Δw.

This manifests itself as a concave utility function, with a negative second derivative $u''(w) < 0$.

For example, the *logarithmic utility* function is concave.

The Arrow-Pratt coefficient of relative risk aversion is proportional to the convexity $u''(w)$ of the utility, and is defined as: $\eta = -\frac{w\, u''(w)}{u'(w)}$.

The relative risk aversion of *logarithmic utility* is equal to one: $\eta = 1$.




```
> # Plot logarithmic utility function
> curve(expr=log, lwd=3, col="blue", xlim=c(0.5, 5),
+ xlab="wealth", ylab="utility",
+ main="Logarithmic Utility")
```

Constant Relative Risk Aversion

It's not a given that all investors have a risk aversion coefficient equal to 1, and other *utility functions* are possible.

The Constant Relative Risk Aversion (*CRRA*) utility function is a generalization of logarithmic utility:

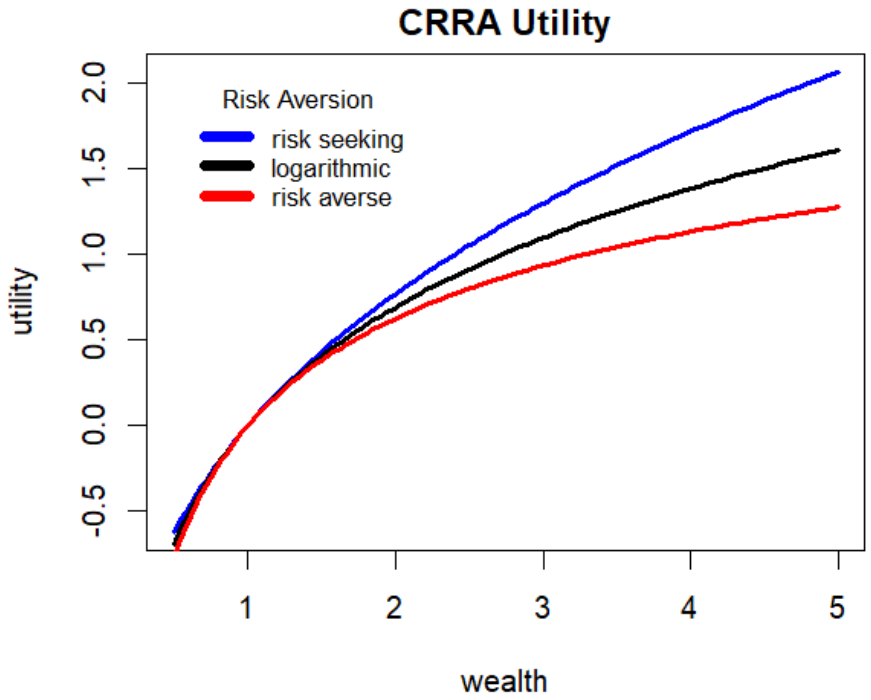
$$u(w) = \frac{w^{1-\eta} - 1}{1 - \eta}$$

Where η is the risk aversion parameter.

The relative risk aversion of the *CRRA* utility function is constant and equal to η.

When the risk aversion parameter is equal to one $\eta = 1$, then the *CRRA* utility function is equal to the logarithmic utility.

In practice, the risk aversion parameter η is not known, and must be estimated through empirical studies.




```
> # Define CRRA utility
> cr_ra <- function(w, ra) {
+   (w^(1-ra) - 1)/(1-ra)
+ }  # end cr_ra
> # Plot utility functions
> curve(expr=cr_ra(x, ra=0.7), xlim=c(0.5, 5), lwd=3,
+ xlab="wealth", ylab="utility", main="", col="blue")
> curve(expr=log, add=TRUE, lwd=3)
> curve(expr=cr_ra(x, ra=1.3), add=TRUE, lwd=3, col="red")
> # Add title and legend
> title(main="CRRA Utility", line=0.5)
> legend(x="topleft", legend=c("risk seeking", "logarithmic", "risk
+  title="Risk Aversion", inset=0.05, cex=0.8, bg="white", y.intersp
+  lwd=6, lty=1, bty="n", col=c("blue", "black", "red"))
```

draft: CRRA Optimal Leverage

It's not a given that all investors have a risk aversion coefficient equal to 1, and other *utility functions* are possible.

The Constant Relative Risk Aversion (*CRRA*) utility function is a generalization of logarithmic utility:

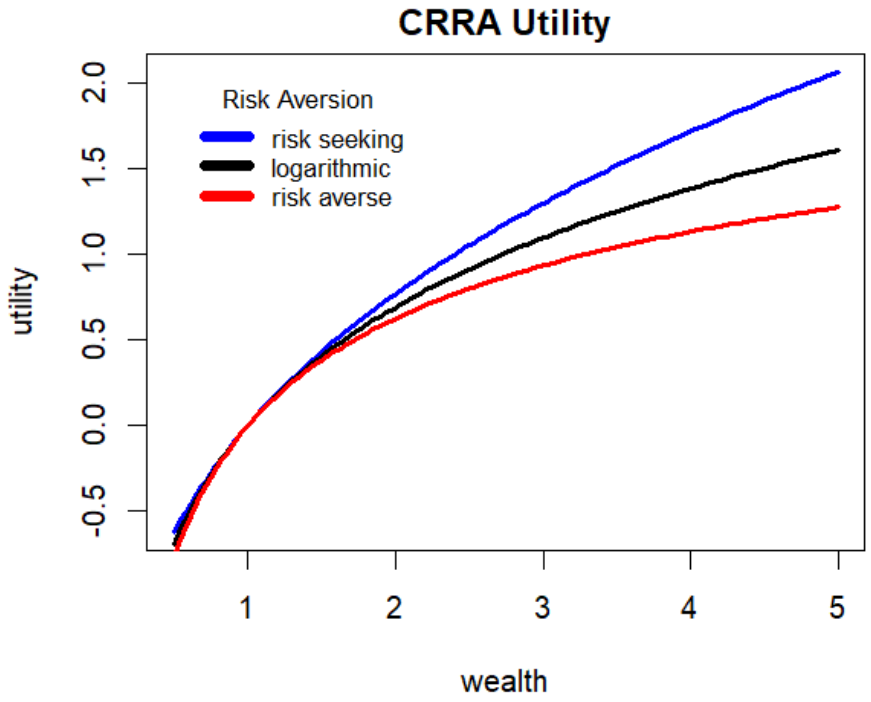
$$u(w) = \frac{w^{1-\eta} - 1}{1 - \eta}$$

Where η is the risk aversion parameter.

The relative risk aversion of the *CRRA* utility function is constant and equal to η.

When the risk aversion parameter is equal to one $\eta = 1$, then the *CRRA* utility function is equal to the logarithmic utility.

In practice, the risk aversion parameter η is not known, and must be estimated through empirical studies.




```
> # Define CRRA utility
> cr_ra <- function(w, ra) {
+   (w^(1-ra) - 1)/(1-ra)
+ }  # end cr_ra
> # Plot utility functions
> curve(expr=cr_ra(x, ra=0.7), xlim=c(0.5, 5), lwd=3,
+ xlab="wealth", ylab="utility", main="", col="blue")
> curve(expr=log, add=TRUE, lwd=3)
> curve(expr=cr_ra(x, ra=1.3), add=TRUE, lwd=3, col="red")
> # Add title and legend
> title(main="CRRA Utility", line=0.5)
> legend(x="topleft", legend=c("risk seeking", "logarithmic", "risk
+  title="Risk Aversion", inset=0.05, cex=0.8, bg="white", y.intersp
+  lwd=6, lty=1, bty="n", col=c("blue", "black", "red"))
```

draft: CRRA Strategy Wealth Path

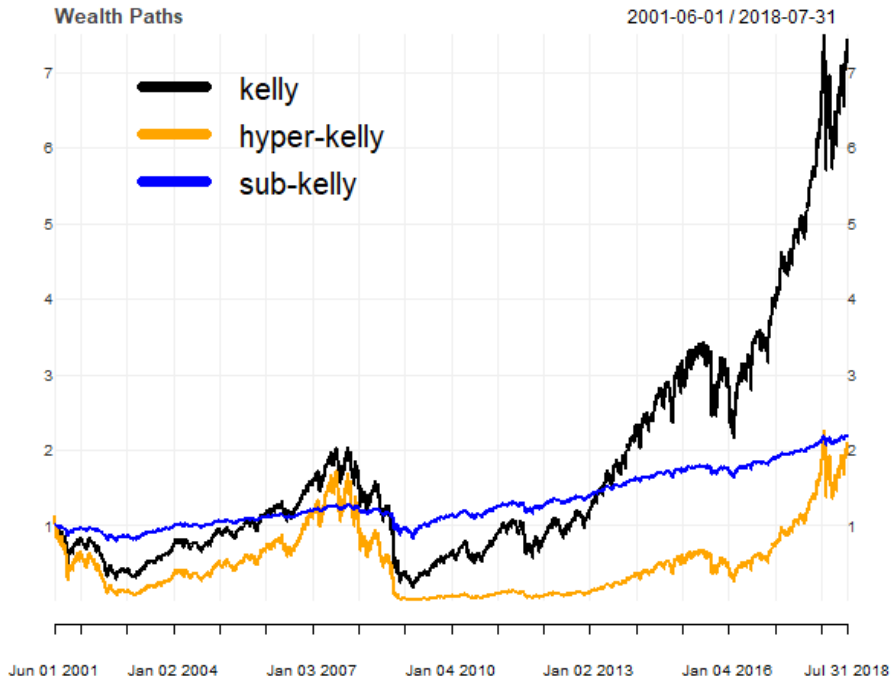
The wealth of a Kelly Strategy with a fixed leverage ratio k_f is equal to:

$$w_t = \prod_{i=1}^{t} (1 + k_f \, r_i)$$

The *Kelly fraction* k_f provides the optimal leverage to maximize the utility of wealth, by balancing the benefit of leveraging higher positive returns, with the risk of ruin due to excessive leverage.

If the mean asset returns are positive, then a higher leverage ratio provides higher returns.

But if the leverage is too high, then the losses in periods with negative returns wipe out most of the wealth, so then it's slow to recover.




```
> retvti <- na.omit(retvti)
> # Calculate wealth paths
> kelly_ratio <- drop(mean(retvti)/var(retvti))
> kelly_wealthv <- cumprod(1 + kelly_ratio*retvti)
> hyper_kelly <- cumprod(1 + (kelly_ratio+2)*retvti)
> sub_kelly <- cumprod(1 + (kelly_ratio-2)*retvti)
> kelly_paths <- cbind(kelly_wealth, hyper_kelly, sub_kelly)
> colnames(kelly_paths) <- c("kelly", "hyper-kelly", "sub-kelly")
```

```
> # Plot wealth paths
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("black", "orange", "blue")
> quantmod::chart_Series(kelly_paths, theme=plot_theme,
+         name="Wealth Paths")
> legend("topleft", legend=colnames(kelly_paths),
+  inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+  col=plot_theme$col$line.col, bty="n")
```

The Utility of Lottery Tickets

Lottery tickets are equivalent to binary gambles with a very small probability of winning p, but a very large winning amount a, and a small loss amount b equal to the ticket price.

The expected payout $\mu = p\,a - q\,b$ of most lottery tickets is negative.

So under *logarithmic utility*, the Kelly fraction k_f for most lottery tickets is also negative, meaning that investors should not be expected to buy these lottery tickets.

But in reality many people do buy lottery tickets with negative expected payouts, which means that their utility functions are not logarithmic.

The demand for lottery tickets can be explained by assuming a strong demand for positive *skewness*, which exceeds the demand for a positive payout.

People buy lottery tickets because they want a small chance of a very large payout, even if the average payout is negative.

Without loss of generality we can assume that the lottery ticket price is one dollar $b = 1$, that it pays out a dollars, and that the expected payout is equal to zero: $\mu = p\,a - q\,b = 0$.

Then the probabilities of winning and losing are equal to: $p = \frac{1}{a+1}$ and $q = \frac{a}{a+1}$.

The variance is equal to: $\sigma^2 = p\,q\,(a+1)^2 = a$.

And the *skewness* is equal to:
$\varsigma = \frac{1}{\sigma^3}\left(\frac{a^3}{a+1} - \frac{a}{a+1}\right) = \frac{a-1}{\sqrt{a}}$.

So the positive *skewness* of a lottery ticket increases as the square root of the *betting odds* a, and it can become very large for large *betting odds*.

Investor *Risk Aversion*, *Prudence* and *Temperance*

Investor risk and return preferences depend on the signs of the derivatives of their *utility* function.

Investors with *logarithmic utility* have positive *odd* derivatives ($u'(w) > 0$ and $u'''(w) > 0$) and negative *even* derivatives ($u''(w) < 0$ and $u''''(w) < 0$), which is typical for most other investors as well.

Risk averse investors have a negative second derivative of utility $u''(w) < 0$.

The demand for lottery tickets shows that investors' utility typically has a positive third derivative $u'''(w) > 0$.

Positive *odd* derivatives imply a preference for larger *odd moments* of the change in the wealth distribution (mean, skewness).

Negative *even* derivatives imply a preference for smaller *even moments* (variance, kurtosis).

The preference for smaller *variance* is called *risk aversion*, for larger *skewness* is called *prudence*, and for smaller *kurtosis* is called *temperance*.

The expected change of the *utility* of wealth $\mathbb{E}[\Delta u(w)]$ can be expanded in the moments of the wealth distribution Δw:

$$\mathbb{E}[\Delta u(w)] = u'(w)\mathbb{E}[\Delta w] + \frac{u''(w)}{2}\sigma^2 + \frac{u'''(w)}{3!}\mu 3 + \frac{u''''(w)}{4!}\mu 3$$

Where $\mathbb{E}[\Delta w]$ is the expected change of wealth, $\sigma^2 = \int \Delta w^2\, p(w)\, \mathrm{d}w$ is the *variance* of The change in wealth, and $\mu 3 = \int \Delta w^3\, p(w)\, \mathrm{d}w = \sigma^3 \varsigma$ and $\mu 4 = \int \Delta w^4\, p(w)\, \mathrm{d}w = \sigma^4 \kappa$ are the third and fourth moments, proportional to the *skewness* ς and the *kurtosis* κ.

Investor Preferences and Empirical Return Distributions

The investor preference for higher *returns* and for lower *volatility* is expressed by maximizing the *Sharpe ratio*.

The third and fourth moments of asset returns are usually much smaller than the *variance*, so they typically have a smaller effect on the investor risk and return preferences.

Nevertheless, there is evidence that investors also have significant preferences for positive *skewness* and lower *kurtosis*.

But stock returns typically have negative *skewness* and excess *kurtosis*, the opposite of what investors prefer.

Many investors may prefer positive *skewness*, even at the expense of lower *returns*, similar to the buyers of lottery tickets.

A paper by Amaya asks if the *Realized Skewness Predicts the Cross-Section of Equity Returns?*

But higher moments are hard to estimate accurately from low frequency (daily) returns, which makes empirical investigations more difficult.

```
> # Calculate the VTI returns
> retvti <- rutils::etfenv$returns$VTI
> retvti <- na.omit(retvti)
> # Calculate higher moments of VTI returns
> c(mean=sum(retvti),
+   variance=sum(retvti^2),
+   mom3=sum(retvti^3),
+   mom4=sum(retvti^4))/NROW(retvti)
> # Calculate higher moments of minutely SPY returns
> spy <- HighFreq::SPY[, 4]
> spy <- na.omit(spy)
> spy <- rutils::diffit(log(spy))
> c(mean=sum(spy),
+   variance=sum(spy^2),
+   mom3=sum(spy^3),
+   mom4=sum(spy^4))/NROW(spy)
```

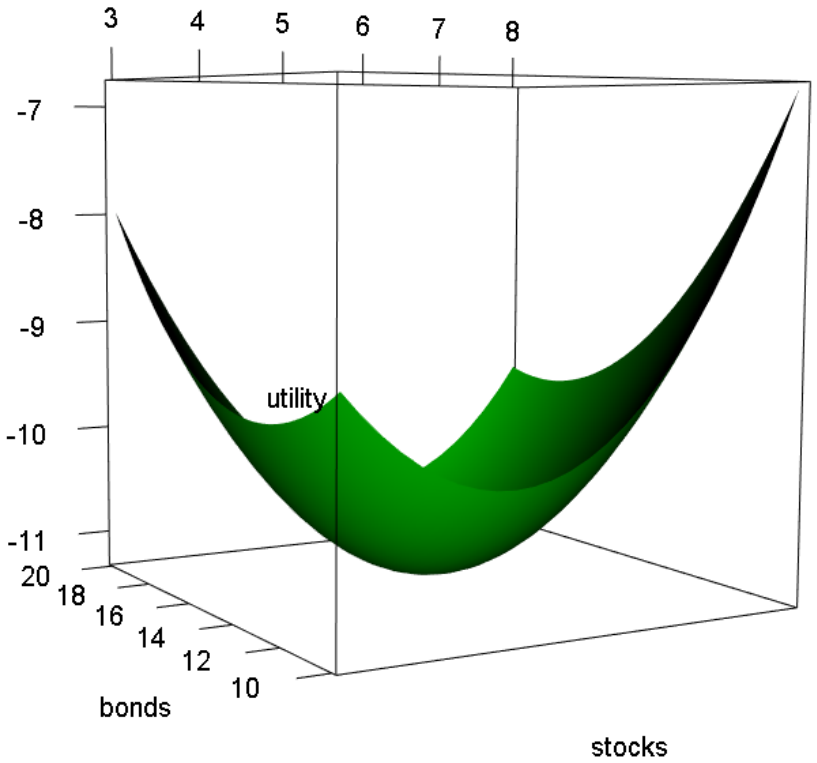
Utility of Stock and Bond Portfolio

The utility u of the stock and bond portfolio with weights *stocku*, *bondu* is equal to:

$$u = \sum_{i=1}^{n} \log(1 + stocku\, r_i^s + bondu\, r_i^b)$$

Where r_i^s, r_i^b are the stock and bond returns.

```
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "IEF")])
> # Logarithmic utility of stock and bond portfolio
> utilfun <- function(stocku, bondu) {
+   -sum(log(1 + stocku*retp$VTI + bondu*retp$IEF))
+ }  # end utilfun
> # Create matrix of utility values
> stocku <- seq(from=3, to=7, by=0.2)
> bondu <- seq(from=12, to=20, by=0.2)
> utilm <- sapply(bondu, function(y) sapply(stocku,
+   function(x) utilfun(x, y)))
> # Set rgl options and load package rgl
> options(rgl.useNULL=TRUE)
> library(rgl)
> # Draw 3d surface plot of utility
> rgl::persp3d(stocku, bondu, utilm, col="green",
+   xlab="stocks", ylab="bonds", zlab="utility")
> # Render the surface plot
> rgl::rglwidget(elementId="plot3drgl")
> # Save the surface plot to png file
> rgl::rgl.snapshot("utility_surface.png")
```



Kelly Optimal Weights

The Kelly optimal stock and bond portfolio weights *stocku*, *bondu* can be calculated by maximizing the utility *u*.

```
> # Approximate Kelly weights
> weightv <- sapply(retp, function(x) mean(x)/var(x))
> # Kelly weight for stocks
> unlist(optimize(f=function(x) utilfun(x, bondu=0), interval=c(1,
> # Kelly weight for bonds
> unlist(optimize(f=function(x) utilfun(x, stocku=0), interval=c(1
> # Vectorized utility of stock and bond portfolio
> utility_vec <- function(weightv) {
+   utilfun(weightv[1], weightv[2])
+ }  # end utility_vec
> # Optimize with respect to vector argument
> optiml <- optim(fn=utility_vec, par=c(3, 10),
+           method="L-BFGS-B",
+           upper=c(8, 20), lower=c(2, 5))
> # Exact Kelly weights
> optiml$par
```

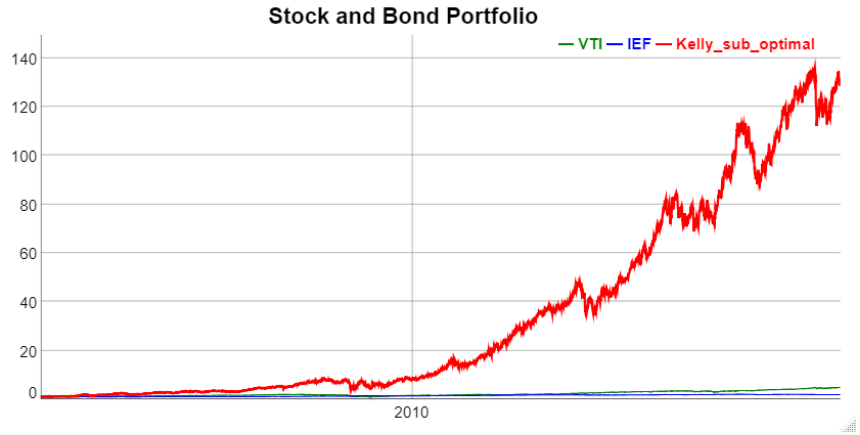
The Kelly optimal weights can be calculated approximately by first calculating the individual stock and bond weights, and then multiplying them by the Kelly weight of the combined portfolio.

```
> # Approximate Kelly weights
> retsport <- (retp %*% weightv)
> drop(mean(retsport)/var(retsport))*weightv
> # Exact Kelly weights
> optiml$par
```

Kelly Optimal Stock and Bond Portfolio

In practice, the Kelly optimal weights under logarithmic utility are too aggressive and they require very active trading, so half-Kelly or even quarter-Kelly weights are used instead.

```
> # Quarter-Kelly sub-optimal weights
> weightv <- optiml$par/4
> # Plot Kelly optimal portfolio
> retp <- cbind(retp, weightv[1]*retp$VTI + weightv[2]*retp$IEF)
> colnames(retp)[3] <- "Kelly_sub_optimal"
> # Calculate compounded wealth from returns
> wealthv <- cumprod(1 + retp)
> # Plot compounded wealth
> dygraphs::dygraph(wealthv, main="Stock and Bond Portfolio") %>%
+   dyOptions(colors=c("green", "blue", "green")) %>%
+   dySeries("Kelly_sub_optimal", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```



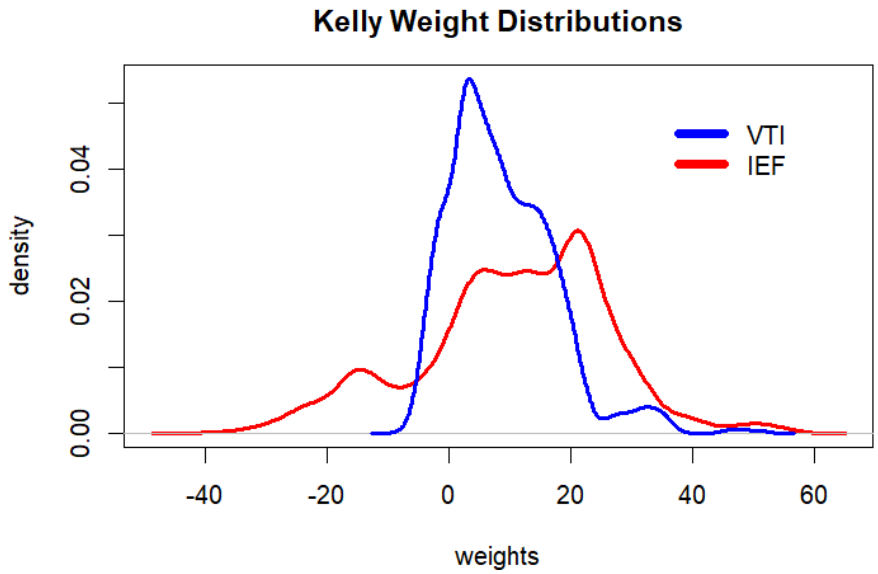
Rolling Kelly Weights

The Kelly weights k_f are calculated daily over a rolling look-back interval:

$$k_f = \frac{\bar{r}_t}{\sigma_t^2}$$

The distribution of the Kelly weights depends on the rolling returns $\bar{r}_t$ and variance σ_t^2.

```
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "IEF")])
> # Calculate rolling returns and variance
> look_back <- 200
> var_rolling <- roll::roll_var(retp, width=look_back)
> weightv <- roll::roll_sum(retp, width=look_back)/look_back
> weightv <- weightv/var_rolling
> weightv[1, ] <- 1/NCOL(weightv)
> weightv <- zoo::na.locf(weightv)
> sum(is.na(weightv))
> range(weightv)
```

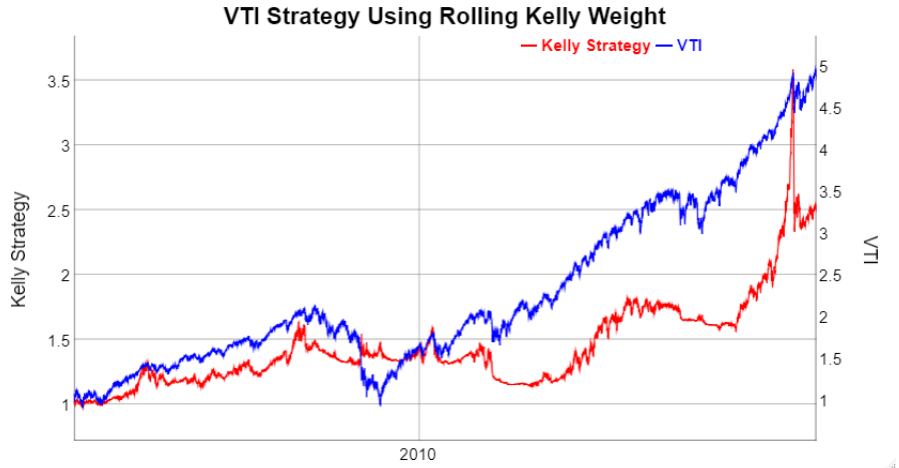



```
> # Plot the weights
> x11(width=6, height=5)
> par(mar=c(4, 4, 3, 1), oma=c(0, 0, 0, 0))
> plot(density(retp$IEF), t="l", lwd=3, col="red",
+      xlab="weights", ylab="density",
+      ylim=c(0, max(density(retp$VTI)$y)),
+      main="Kelly Weight Distributions")
> lines(density(retp$VTI), t="l", col="blue", lwd=3)
> legend("topright", legend=c("VTI", "IEF"),
+  inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+  col=c("blue", "red"), bty="n")
```

Rolling Kelly Strategy For Stocks

In the rolling Kelly strategy, the leverage of the risky asset k_f changes over time.

The leverage is equal to the updated weight from the previous period.




```
> # Scale and lag the Kelly weights
> weightv <- lapply(weightv,
+   function(x) 10*x/sum(abs(range(x))))
> weightv <- do.call(cbind, weightv)
> weightv <- rutils::lagit(weightv)
> # Calculate the compounded Kelly wealth and VTI
> wealthv <- cbind(cumprod(1 + weightv$VTI*retp$VTI), cumprod(1 + re
> colnames(wealthv) <- c("Kelly Strategy", "VTI")
> dygraphs::dygraph(wealthv, main="VTI Strategy Using Rolling Kelly Weight") %>%
+   dyAxis("y", label="Kelly Strategy", independentTicks=TRUE) %>%
+   dyAxis("y2", label="VTI", independentTicks=TRUE) %>%
+   dySeries(name="Kelly Strategy", axis="y", label="Kelly Strategy", strokeWidth=1, col="red") %>%
+   dySeries(name="VTI", axis="y2", label="VTI", strokeWidth=1, col="blue")
```

Rolling Kelly Strategy With Transaction Costs

The *margin debt* m_t is proportional to the wealth w_t: $m_t = (k_f - 1)w_t + 1$.

The dollar amount of the *risky asset* traded is equal to the change in *margin*, equal to: $\Delta m_t = \Delta[(k_f - 1)w_t]$.

If the transaction costs are large, then they will reduce the wealth and reduce the dollar amount of the *risky asset* held by the investor.

The transaction costs depend on the change in wealth, and the wealth is decreased by the transaction costs.

So the transaction costs in each time period must be calculated recursively in a loop from the wealth in the past period.

If the transaction costs are much less than the change in wealth $c^r \ll |\Delta w_t|$, then they can be calculated approximately as the absolute value of the change in *margin* m_t^{nc} for a wealth path with no transaction costs:

$$c^r = \frac{\delta}{2} \left| \Delta m_t^{nc} \right|$$

The transaction costs as a percentage of wealth are equal to: c_t / w_t^{nc}, where w_t^{nc} is the wealth assuming no transaction costs.

The wealth of the Kelly Strategy after accounting for the *bid-offer spread* is then equal to:

$$w_t = \prod_{i=1}^{t} (1 + k_f \, r_i - \frac{\delta}{2} \frac{|\Delta m_i^{nc}|}{w_i^{nc}})$$

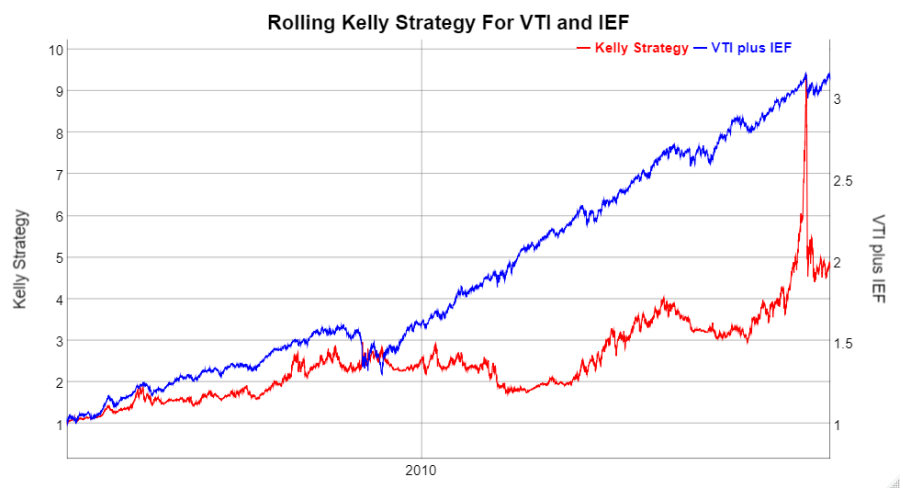
The effect of the *bid-offer spread* is to reduce the effective asset returns by an amount proportional to the *bid-offer spread*.

```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Calculate the compounded Kelly wealth and margin
> wealthv <- cumprod(1 + weightv$VTI*retp$VTI)
> margin <- (retp$VTI - 1)*wealthv + 1
> # Calculate the transaction costs
> costs <- bid_offer*drop(rutils::diffit(margin))/2
> wealth_diff <- drop(rutils::diffit(wealthv))
> costs_rel <- ifelse(wealth_diff>0, costs/wealth_diff, 0)
> range(costs_rel)
> hist(costs_rel, breaks=10000, xlim=c(-0.02, 0.02))
> # Scale and lag the transaction costs
> costs <- rutils::lagit(abs(costs)/wealthv)
> # Recalculate the compounded Kelly wealth
> wealth_trans <- cumprod(1 + retp$VTI*retp$VTI - costs)
> # Plot compounded wealth
> wealthv <- cbind(wealthv, wealth_trans)
> colnames(wealthv) <- c("Kelly", "Including bid-offer")
> dygraphs::dygraph(wealthv, main="Kelly Strategy With Transaction C
+   dyOptions(colors=c("green", "blue"), strokeWidth=2) %>%
```

Rolling Kelly Strategy For Stocks and Bonds

In the rolling Kelly strategy, the leverage of the risky asset k_f changes over time.

The leverage is equal to the updated weight from the previous period.




```
> # Calculate compounded wealth from returns
> wealthv <- cumprod(1 + rowSums(weightv*retp))
> wealthv <- xts::xts(wealthv, zoo::index(retp))
> quantmod::chart_Series(wealthv, name="Rolling Kelly Strategy For V
> # Calculate the compounded Kelly wealth and VTI
> wealthv <- cbind(wealthv, cumprod(1 + 0.6*retp$IEF + 0.4*retp$VTI)
> colnames(wealthv) <- c("Kelly Strategy", "VTI plus IEF")
> dygraphs::dygraph(wealthv, main="Rolling Kelly Strategy For VTI and IEF") %>%
+   dyAxis("y", label="Kelly Strategy", independentTicks=TRUE) %>%
+   dyAxis("y2", label="VTI plus IEF", independentTicks=TRUE) %>%
+   dySeries(name="Kelly Strategy", axis="y", label="Kelly Strategy", strokeWidth=1, col="red") %>%
+   dySeries(name="VTI plus IEF", axis="y2", label="VTI plus IEF", strokeWidth=1, col="blue")
```

Tests for Market Timing Skill

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

The *market timing* skill can be measured by performing a *linear regression* of a strategy's returns against a strategy with perfect *market timing* skill.

The *Merton-Henriksson* market timing test uses a linear *market timing* term:

$$R - R_f = \alpha + \beta(R_m - R_f) + \gamma \max(0, R_m - R_f) + \varepsilon$$

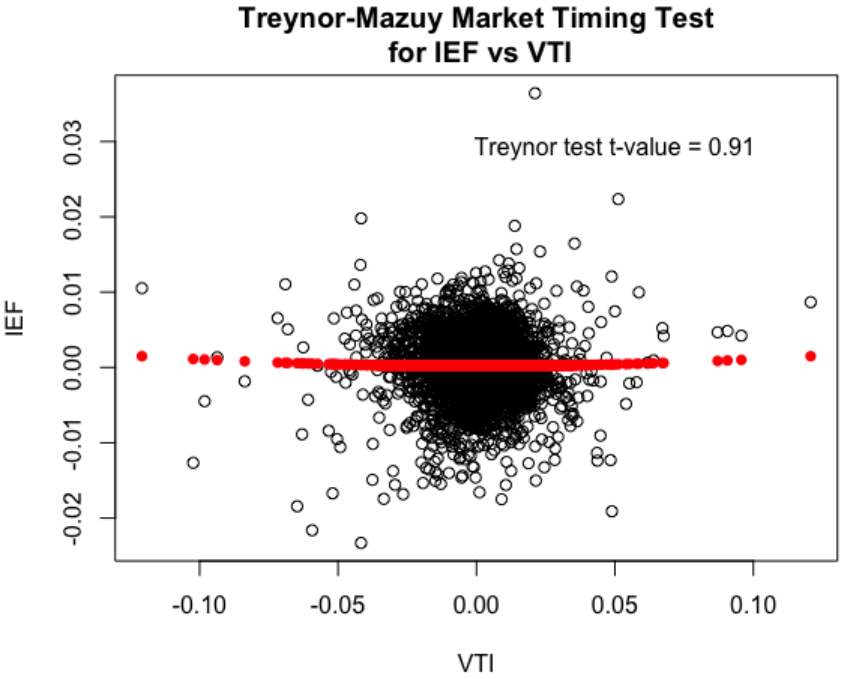
Where R are the strategy returns, R_m are the market returns, and R_f are the risk-free returns.

If the coefficient γ is statistically significant, then it's very likely due to *market timing* skill.

The *market timing* regression is a generalization of the *Capital Asset Pricing Model*.

The *Treynor-Mazuy* test uses a quadratic term, which makes it more sensitive to the magnitude of returns:

$$R - R_f = \alpha + \beta(R_m - R_f) + \gamma(R_m - R_f)^2 + \varepsilon$$




```
> # Test if IEF can time VTI
> retp <- na.omit(rutils::etfenv$returns[, c("IEF", "VTI")])
> retvti <- retp$VTI
> desv <- cbind(retp, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desv)[3:4] <- c("merton", "treynor")
> # Merton-Henriksson test
> regmod <- lm(IEF ~ VTI + merton, data=desv); summary(regmod)
```

```
> # Treynor-Mazuy test
> regmod <- lm(IEF ~ VTI + treynor, data=desv); summary(regmod)
> # Plot residual scatterplot
> x11(width=6, height=5)
> residv <- (desv$IEF - regmod$coeff["VTI"]*retvti)
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="IEF")
> title(main="Treynor-Mazuy Market Timing Test\n for IEF vs VTI", li
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvt
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residv), paste("Treynor test t-value =", tva
```

draft: Identifying Managers With Skill

Consider a binary investment (gamble) with the probability of winning equal to p, the winning amount (gain) equal to a, and the loss equal to b.

The investor makes no up-front payments, and either wins an amount a, or loses an amount b.

Assuming that an investor makes decisions exclusively on the basis of the expected value of future wealth, then they would choose to invest all their wealth on the gamble if its expected value is positive, and choose not to invest at all if its expected value is negative.

	win	lose
probability	p	q = 1 - p
payout	a	-b

The expected value of the gamble is equal to: $m = p\,a - q\,b$.

The variance of the gamble is equal to: $var = p\,q\,(a+b)^2$.

Without loss of generality we can assume that $p = q = \frac{1}{2}$,
$m = 0.5\,(b-a)$,
$var = 0.25\,(a+b)^2$.

The *Sharpe ratio* of the gamble is then equal to:

$$S_r = \frac{m}{sqrt(var)} = \frac{(b-a)}{(a+b)}$$

Calculating Asset Returns

Given a time series of asset prices p_i, the dollar returns r_i^d, the percentage returns r_i^p, and the log returns r_i^l are defined as:

$$r_i^d = p_i - p_{i-1} \quad r_i^p = \frac{p_i - p_{i-1}}{p_{i-1}} \quad r_i^l = \log(\frac{p_i}{p_{i-1}})$$

The initial returns are all equal to zero.

If the log returns are small $r^l \ll 1$, then they are approximately equal to the percentage returns: $r^l \approx r^p$.

```
> library(rutils)
> # Extract ETF prices from rutils::etfenv$prices
> pricev <- rutils::etfenv$prices
> pricev <- zoo::na.locf(pricev, na.rm=FALSE)
> pricev <- zoo::na.locf(pricev, fromLast=TRUE)
> datev <- zoo::index(pricev)
> # Calculate simple dollar returns
> retd <- rutils::diffit(pricev)
> # Or
> # retd <- lapply(pricev, rutils::diffit)
> # retd <- rutils::do_call(cbind, retd)
> # Calculate percentage returns
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate log returns
> retl <- rutils::diffit(log(pricev))
```

Compounding Asset Returns

The sum of the dollar returns: $\sum_{i=1}^{n} r_i^d$ represents the wealth path from owning a *fixed number of shares*.

The compounded percentage returns: $\prod_{i=1}^{n}(1 + r_i^p)$ also represent the wealth path from owning a *fixed number of shares*, initially equal to \$1 dollar.

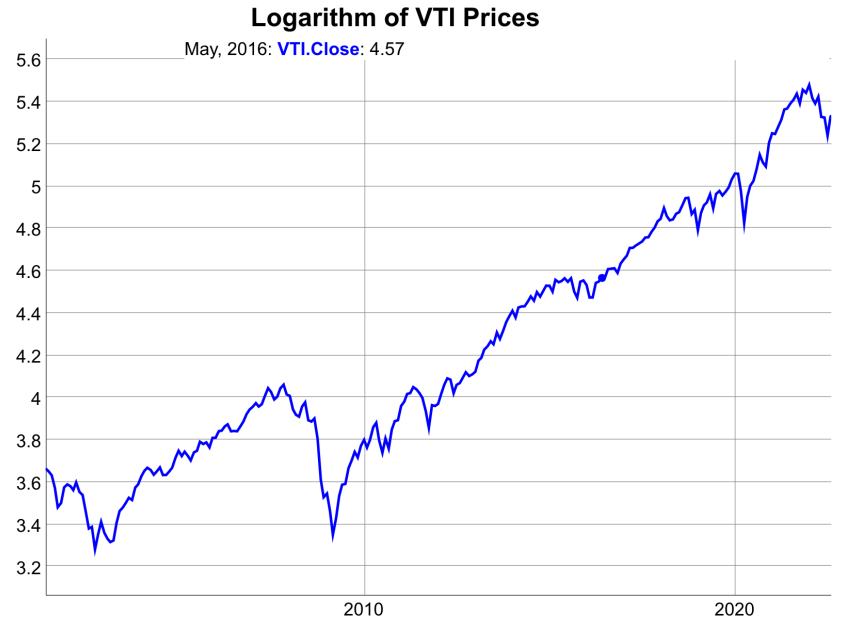
The sum of the percentage returns (without compounding): $\sum_{i=1}^{n} r_i^p$ represents the wealth path from owning a *fixed dollar amount* of stock.

Maintaining a *fixed dollar amount* of stock requires periodic *rebalancing* - selling shares when their price goes up, and vice versa.

This *rebalancing* therefore acts as a mean reverting strategy.

Rebalancing requires borrowing from a *margin account*, and it also incurs trading costs.

The logarithm of the wealth of a *fixed number of shares* is often used to compare investments, and it's approximately equal to the sum of the percentage returns.




```
> # Set the initial dollar returns
> retd[1, ] <- pricev[1, ]
> # Calculate prices from dollar returns
> pricen <- cumsum(retd)
> all.equal(pricen, pricev)
> # Compound the percentage returns
> pricen <- cumprod(1+retp)
> # Set the initial prices
> pricesi <- as.numeric(pricev[1, ])
> pricen <- lapply(1:NCOL(pricen), function (i)
+     pricesi[i]*pricen[, i])
> pricen <- rutils::do_call(cbind, pricen)
> # Or
> # pricen <- t(t(pricen)*pricesi)
> all.equal(pricen, pricev, check.attributes=FALSE)
> # Plot log VTI prices
> endd <- rutils::calc_endpoints(rutils::etfenv$VTI, interval="month
> dygraphs::dygraph(log(quantmod::Cl(rutils::etfenv$VTI)[endd]),
```

Funding Costs of Single Asset Rebalancing

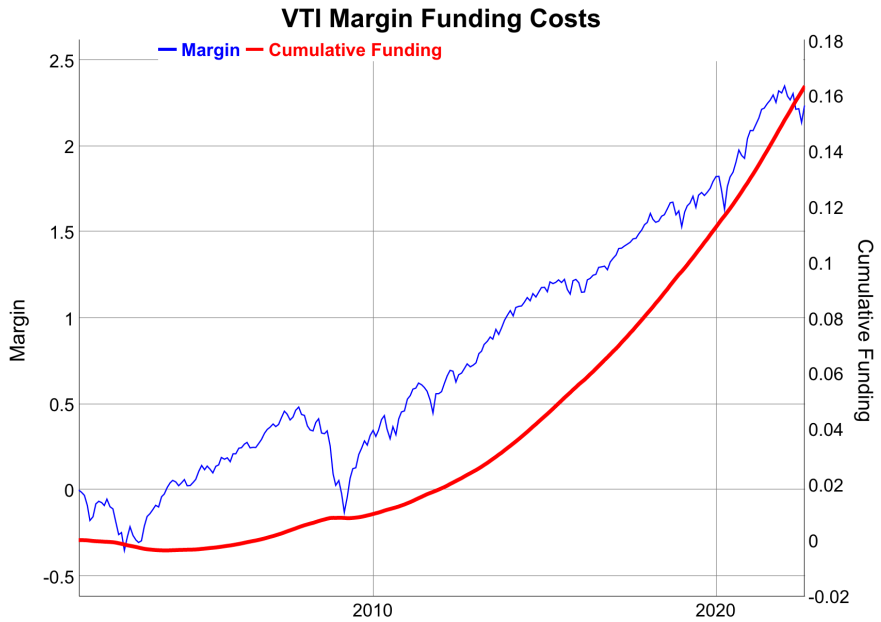
The wealth accumulated from owning a *fixed dollar amount* of stock is equal to the cash earned from rebalancing, which is proportional to the sum of the percentage returns, and it's kept in a *margin account*: $m_t = \sum_{i=1}^{t} r_i^p$.

The cash in the *margin account* can be positive (accumulated profits) or negative (losses).

The *funding costs* c_t^f are approximately equal to the *margin account* m_t times the *funding rate* f: $c_t^f = f\, m_t = f \sum_{i=1}^{t} r_i^p$.

Positive *funding costs* represent interest profits earned on the *margin account*, while negative costs represent the interest paid for funding stock purchases.

The *cumulative funding costs* $\sum_{i=1}^{t} c_i^f$ must be added to the *margin account*: $m_t + \sum_{i=1}^{t} c_i^f$.




```
> # Calculate percentage VTI returns
> pricev <- rutils::etfenv$prices$VTI
> pricev <- na.omit(pricev)
> retvti <- rutils::diffit(pricev)/rutils::lagit(pricev, lagg=1, pa
```

```
> # Funding rate per day
> frate <- 0.01/252
> # Margin account
> margin <- cumsum(retvti)
> # Cumulative funding costs
> fcosts <- cumsum(frate*margin)
> # Add funding costs to margin account
> margin <- (margin + fcosts)
> # dygraph plot of margin and funding costs
> datav <- cbind(margin, fcosts)
> colnamev <- c("Margin", "Cumulative Funding")
> colnames(datav) <- colnamev
> endd <- rutils::calc_endpoints(datav, interval="months")
> dygraphs::dygraph(datav[endd], main="VTI Margin Funding Costs") %>
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
```

Transaction Costs of Trading

The total *transaction costs* are the sum of the *broker commissions*, the *bid-offer spread* (for market orders), *lost trades* (for limit orders), and *market impact*.

Broker commissions depend on the broker, the size of the trades, and on the type of investors, with institutional investors usually enjoying smaller commissions.

The *bid-offer spread* is the percentage difference between the *offer* minus the *bid* price, divided by the *mid* price.

Market impact is the effect of large trades pushing the market prices (the limit order book) against the trades, making the filled price worse.

Limit orders are not subject to the bid-offer spread but they are exposed to *lost trades*.

Lost trades are limit orders that don't get executed, resulting in lost potential profits.

Limit orders may receive rebates from some exchanges, which may reduce transaction costs.

The *bid-offer spread* for liquid stocks can be assumed to be about 10 basis points (bps).

In reality the *bid-offer spread* is not static and depends on many factors, such as market liquidity (trading volume), volatility, and the time of day.

The *transaction costs* due to the *bid-offer spread* are equal to the number of traded shares times their price, times half the *bid-offer spread*.

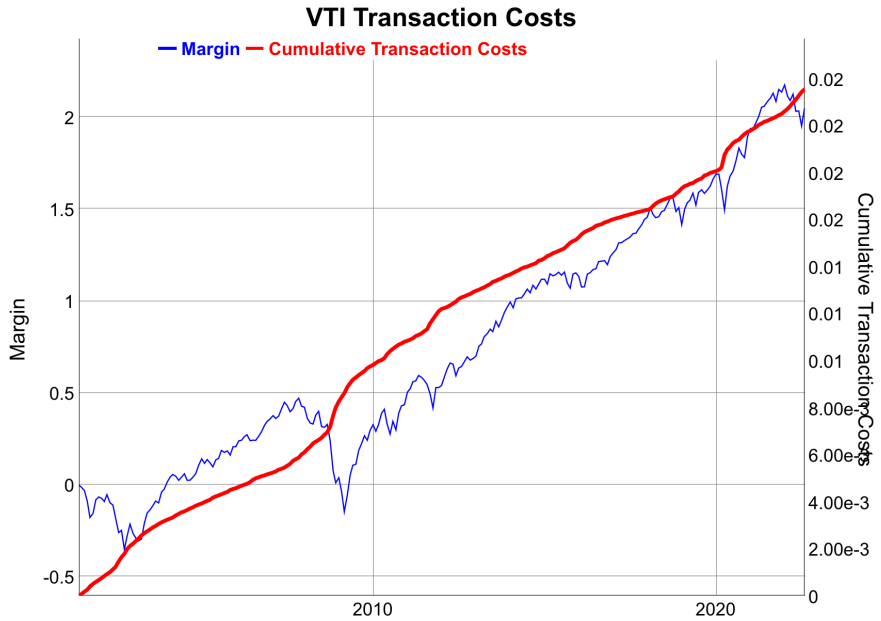
Transaction Costs of Single Asset Rebalancing

Maintaining a *fixed dollar amount* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

The dollar amount of stock that must be traded in a given period is equal to the absolute of the percentage returns: $|r_t|$.

The *transaction costs* c_t^r due to rebalancing are equal to half the *bid-offer spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2} |r_t|$.

The *cumulative transaction costs* $\sum_{i=1}^{t} c_i^r$ must be subtracted from the *margin account* m_t: $m_t - \sum_{i=1}^{t} c_i^r$.




```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(abs(retvti))/2
> # Subtract transaction costs from margin account
> margin <- cumsum(retvti)
> margin <- (margin - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(margin, costs)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endd], main="VTI Transaction Costs") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Combining the Returns of Multiple Assets

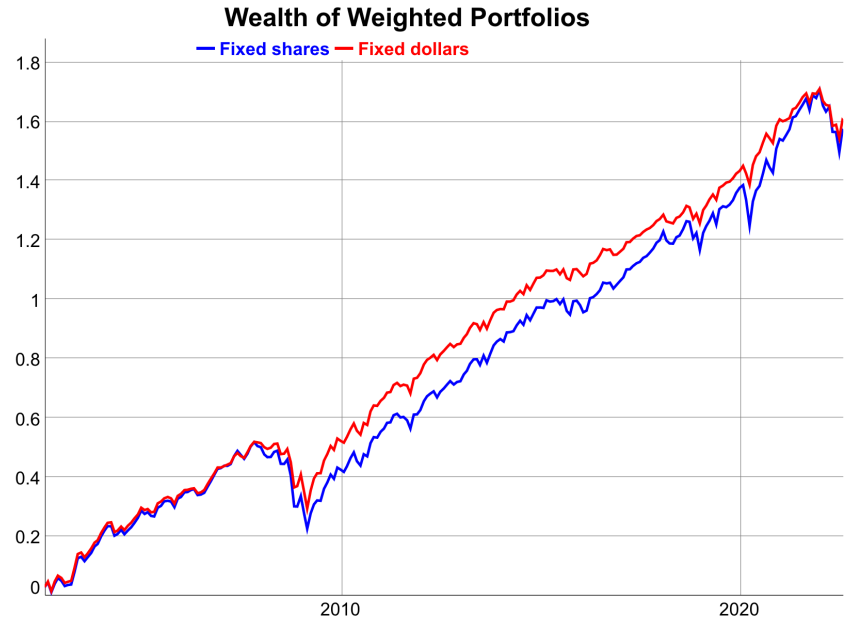
Adding the weighted dollar returns is equivalent to buying a *fixed number of shares* (aka *Fixed Share Allocation* or FSA) proportional to the weights.

Adding the weighted percentage returns is equivalent to investing in *fixed dollar amounts of stock* (aka *Fixed Dollar Amount* or FDA) proportional to the weights.

The portfolio allocations must be periodically rebalanced to keep the dollar amounts of the stocks proportional to the weights.

This *rebalancing* acts as a mean reverting strategy - selling shares when their price goes up, and vice versa.

The portfolio with fixed dollar amounts has a slightly higher Sharpe ratio than the portfolio with a fixed number of shares.




```
> # Calculate VTI and IEF dollar returns
> pricev <- rutils::etfenv$pricev[, c("VTI", "IEF")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> datev <- zoo::index(pricev)
> # Calculate VTI and IEF percentage returns
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Set the initial dollar returns
> retd[1, ] <- pricev[1, ]
> # Wealth of fixed shares equal to $0.5 each (without rebalancing)
> weightv <- c(0.5, 0.5)  # dollar weights
> # Scale the dollar returns using the dollar weights
> pricesi <- as.numeric(pricev[1, ])
> wealth_fsa <- cumsum(retd %*% (weightv/pricesi))
> # Or using percentage returns
> wealth_fsa2 <- cumprod(1+retp) %*% weightv
```

```
> # Wealth of fixed dollars (with rebalancing)
> wealth_fda <- cumsum(retp %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(log(wealth_fsa), wealth_fda)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Fixed dollars")
> sqrt(252)*sapply(rutils::diffit(wealthv),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> colnamev <- colnames(wealthv)
> endd <- rutils::calc_endpoints(retp, interval="months")
> dygraphs::dygraph(wealthv[endd], main="Wealth of Weighted Portfoli
+   dySeries(name=colnamev[1], col="blue", strokeWidth=2) %>%
+   dySeries(name=colnamev[2], col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Transaction Costs of Weighted Portfolio Rebalancing

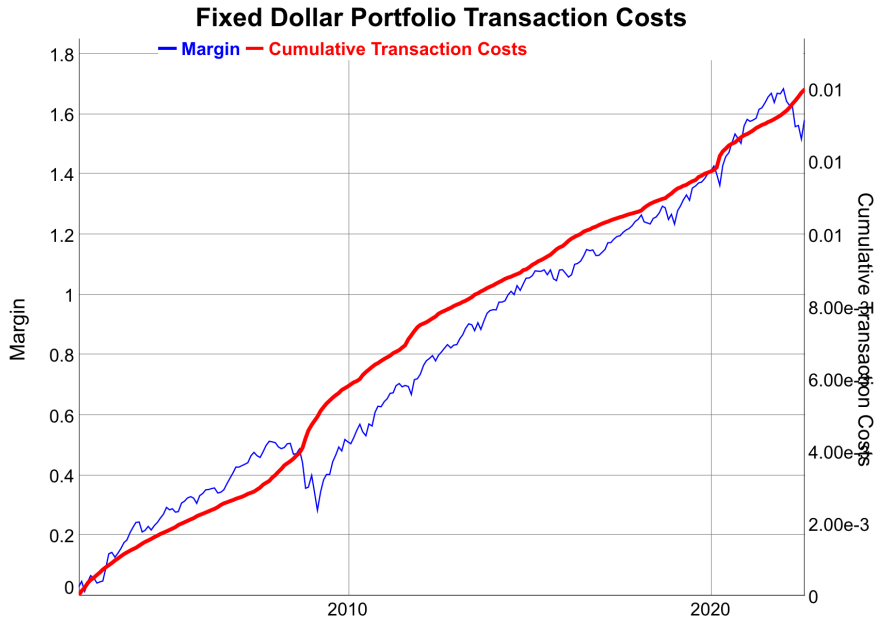
Maintaining a *fixed dollar amount* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

Adding the weighted percentage returns is equivalent to investing in *fixed dollar amounts of stock* proportional to the weights.

The dollar amount of stock that must be traded in a given period is equal to the weighted sum of the absolute percentage returns: $w1 \left|r_t^1\right| + w2 \left|r_t^2\right|$.

The *transaction costs* c_t^r due to rebalancing are equal to half the *bid-offer spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2}(w1 \left|r_t^1\right| + w2 \left|r_t^2\right|)$.

The *cumulative transaction costs* $\sum_{i=1}^{t} c_i^r$ must be subtracted from the *margin account* m_t: $m_t - \sum_{i=1}^{t} c_i^r$.




```
> # Margin account for fixed dollars (with rebalancing)
> margin <- cumsum(retp %*% weightv)
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(abs(retp) %*% weightv)/2
> # Subtract transaction costs from margin account
> margin <- (margin - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(margin, costs)
> datav <- xts::xts(datav, datev)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endd], main="Fixed Dollar Portfolio Transa
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Portfolio With Proportional Dollar Allocations

In the *proportional dollar allocation* strategy (*PDA*), the total wealth w_t is allocated to the assets w_i proportional to the portfolio weights ω_i: $w_i = \omega_i w_t$.

The total wealth w_t is not fixed and is equal to the portfolio market value $w_t = \sum w_i$, so there's no margin account.

The portfolio is rebalanced daily to maintain the dollar allocations w_i equal to the total wealth $w_t = \sum w_i$ times the portfolio weights: ω_i: $w_i = \omega_i w_t$.

Let r_i be the percentage returns, ω_i be the portfolio weights, and $\bar{r}_t = \sum_{i=1}^{n} \omega_i r_i$ be the weighted percentage returns at time t.

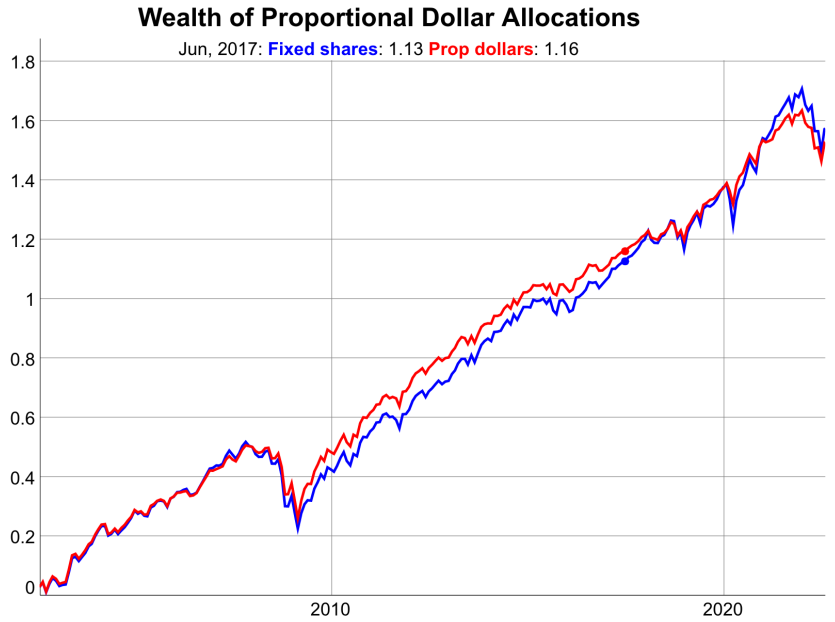
The total portfolio wealth at time t is equal to the wealth at time $t-1$ multiplied by the weighted returns: $w_t = w_{t-1}(1 + \bar{r}_t)$.

The dollar amount of stock i at time t increases by $\omega_i r_i$ so it's equal to $\omega_i w_{t-1}(1 + r_i)$, while the target amount is $\omega_i w_t = \omega_i w_{t-1}(1 + \bar{r}_t)$

The dollar amount of stock i needed to trade to rebalance back to the target weight is equal to:

$$\begin{aligned} \varepsilon_i &= |\omega_i w_{t-1}(1 + \bar{r}_t) - \omega_i w_{t-1}(1 + r_i)| \\ &= \omega_i w_{t-1} |\bar{r}_t - r_i| \end{aligned}$$

If $\bar{r}_t > r_i$ then an amount ε_i of the stock i needs to be bought, and if $\bar{r}_t < r_i$ then it needs to be sold.




```
> # Wealth of fixed shares (without rebalancing)
> wealth_fsa <- cumsum(retd %*% (weightv/pricesi))
> # Or compound the percentage returns
> wealth_fsa <- cumprod(1+retp) %*% weightv
> # Wealth of proportional allocations (with rebalancing)
> wealth_pda <- cumprod(1 + retp %*% weightv)
> wealthv <- cbind(wealth_fsa, wealth_pda)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Prop dollars")
> wealthv <- log(wealthv)
> # 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 the log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Proportional Dollar Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

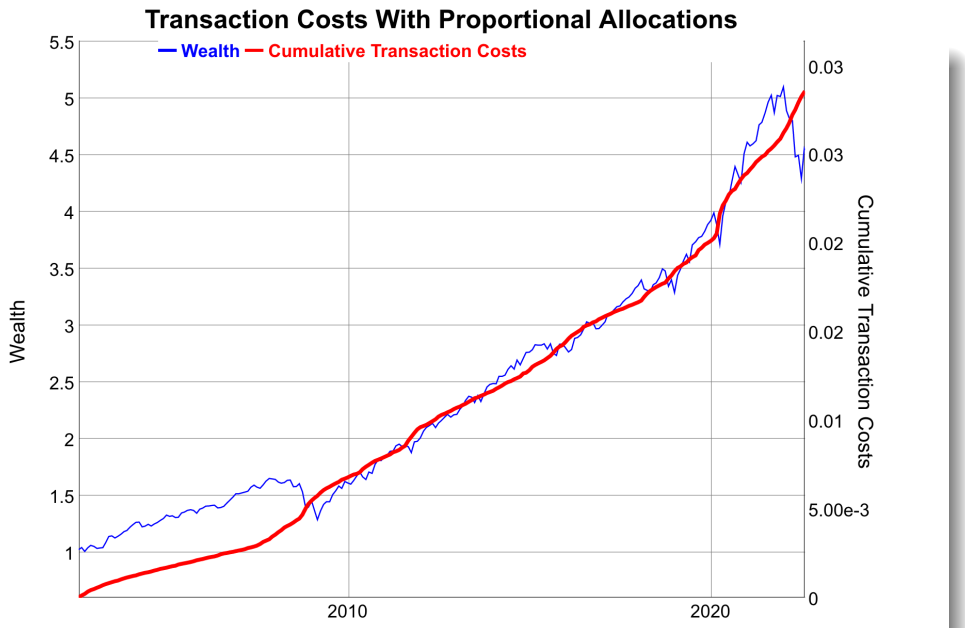
Transaction Costs With Proportional Dollar Allocations

In each period the stocks must be rebalanced to maintain the proportional dollar allocations.

The total dollar amount of stocks that need to be traded to rebalance back to the target weight is equal to: $\sum_{i=1}^{n} \varepsilon_i = w_{t-1} \sum_{i=1}^{n} \omega_i \left| \bar{r}_t - r_i \right|$

The *transaction costs* c_t^r are equal to half the *bid-offer spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2} \sum_{i=1}^{n} \varepsilon_i$.

The *cumulative transaction costs* $\sum_{i=1}^{t} c_i^r$ must be subtracted from the *wealth* w_t: $w_t - \sum_{i=1}^{t} c_i^r$.




```
> # Returns in excess of weighted returns
> retsw <- retp %*% weightv
> retsx <- lapply(retp, function(x) (retsw - x))
> retsx <- do.call(cbind, retsx)
> sum(retsx %*% weightv)
> # Calculate weighted sum of absolute excess returns
> retsx <- abs(retsx) %*% weightv
> # Total dollar amount of stocks that need to be traded
> retsx <- retsx*rutils::lagit(wealth_pda)
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(retsx)/2
> # Subtract transaction costs from wealth
> wealth_pda <- (wealth_pda - costs)
```

```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealth_pda, costs)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Wealth", "Cumulative Transaction Costs")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv[endd],
+   main="Transaction Costs With Proportional Allocations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Proportional Target Allocation Strategy

In the *fixed share strategy* (*FSA*), the number of shares is fixed, with their initial dollar value equal to the portfolio weights.

In the *proportional dollar allocation* strategy (*PDA*), the portfolio is rebalanced daily to maintain the dollar allocations w_i equal to the total wealth $w_t = \sum w_i$ times the portfolio weights: ω_i: $w_i = \omega_i w_t$.

In the *proportional target allocation* strategy (*PTA*), the portfolio is rebalanced only if the dollar allocations w_i differ from their targets $\omega_i w_t$ more than the threshold value τ: $\tau > \frac{\sum |w_i - \omega_i w_t|}{w_t}$.

The *PTA* strategy is path-dependent so it must be simulated using an explicit loop.

The *PTA* strategy is contrarian, since it sells assets that have outperformed, and it buys assets that have underperformed.

If the threshold level is very small then the *PTA* strategy rebalances daily and it's the same as the *PDA*.

If the threshold level is very large then the *PTA* strategy does not rebalance and it's the same as the *FSA*.

```
> # Wealth of fixed shares (without rebalancing)
> wealth_fsa <- drop(apply(retp, 2, function(x) cumprod(1+x)) %*% we
> # Wealth of proportional dollar allocations (with rebalancing)
> wealth_pda <- cumprod(1 + retp %*% weightv) - 1
> # Wealth of proportional target allocation (with rebalancing)
> retp <- zoo::coredata(retp)
> threshold <- 0.05
> wealthv <- matrix(nrow=NROW(retp), ncol=2)
> colnames(wealthv) <- colnames(retp)
> wealthv[1, ] <- weightv
> for (it in 2:NROW(retp)) {
+   # Accrue wealth without rebalancing
+   wealthv[it, ] <- wealthv[it-1, ]*(1 + retp[it, ])
+   # Rebalance if wealth allocations differ from weights
+   if (sum(abs(wealthv[it, ] - sum(wealthv[it, ])*weightv))/sum(wea
+     # cat("Rebalance at:", it, "\n")
+     wealthv[it, ] <- sum(wealthv[it, ])*weightv
+   } # end if
+ } # end for
> wealthv <- rowSums(wealthv) - 1
> wealthv <- cbind(wealth_pda, wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Proportional Allocations", "Proportional T
> dygraphs::dygraph(wealthv, main="Wealth of Proportional Target All
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

draft: Stock Index Weighting Methods

Split this slide to explain equal-weighted indices: https://www.investopedia.com/terms/e/equalweight.asp

Stock market indices can be capitalization-weighted (*S&P500*), price-weighted (*DJIA*), or equal-weighted.

The cap-weighted and price-weighted indices own a fixed number of shares (excluding stock splits).

Equal-weighted indices own the same dollar amount of each stock, so they must be rebalanced as market prices change.

Cap-weighted index = Sum { (Stock Price * Number of shares) / Index Divisor }

Price-weighted index = Sum {Stock Price / Index Divisor }

Equal-weighted index = Sum { (Stock Price * factor) / Index Divisor }

Cap-weighted indices are overweight large-cap stocks, while equal-weighted indices are overweight small-cap stocks.

Cap-weighted indices are *trend following*, while equal-weighted indices are *mean reverting* (contrarian).

```
> # Create name corresponding to "^GSPC" symbol
> setSymbolLookup(
+   SP500=list(name="^GSPC", src="yahoo"))
> getSymbolLookup()
> # view and clear options
> options("getSymbols.sources")
> options(getSymbols.sources=NULL)
> # Download S&P500 prices into etfenv
> quantmod::getSymbols("SP500", env=etfenv,
+     adjust=TRUE, auto.assign=TRUE, from="1990-01-01")
> quantmod::chart_Series(x=etfenv$SP500["2016/"],
+        TA="add_Vo()",
+        name="S&P500 index")
```

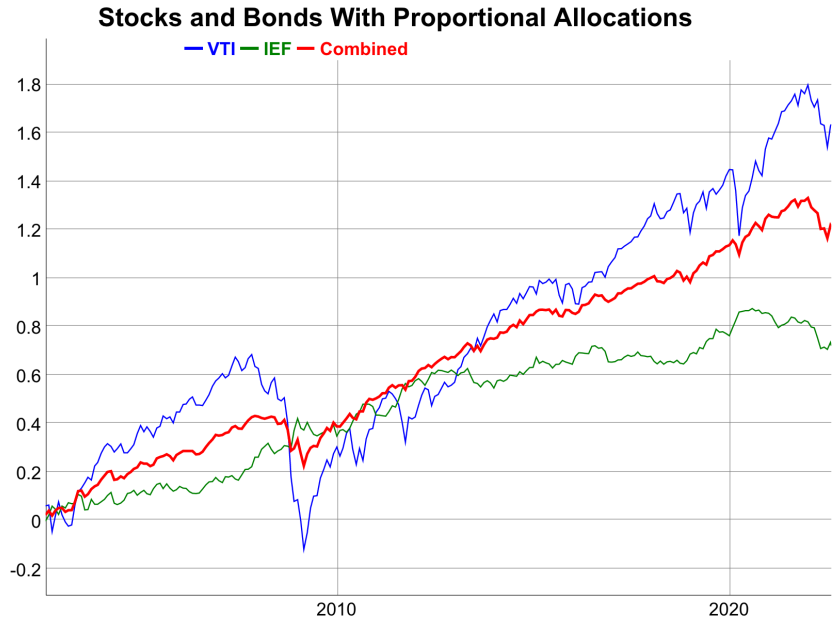
Stock and Bond Portfolio With Proportional Dollar Allocations

Portfolios combining stocks and bonds can provide a much better risk versus return tradeoff than either of the assets separately, because the returns of stocks and bonds are usually negatively correlated, so they are natural hedges of each other.

The fixed portfolio weights represent the percentage dollar allocations to stocks and bonds, while the portfolio wealth grows over time.

The weights depend on the investment horizon, with a greater allocation to bonds for a shorter investment horizon.

Active investment strategies are expected to outperform static stock and bond portfolios.




```
> # Calculate stock and bond returns
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "IEF")])
> weightv <- c(0.4, 0.6)
> retp <- cbind(retp, retp %*% weightv)
> colnames(retp)[3] <- "Combined"
> # Calculate correlations
> cor(retp)
> # Calculate Sharpe ratios
> sqrt(252)*sapply(retp, function(x) mean(x)/sd(x))
> # Calculate standard deviation, skewness, and kurtosis
> sapply(retp, function(x) {
+   # Calculate standard deviation
+   stdev <- sd(x)
+   # Standardize the returns
+   x <- (x - mean(x))/stdev
+   c(stdev=stdev, skew=mean(x^3), kurt=mean(x^4))
+ })  # end sapply
```

```
> # Wealth of proportional allocations
> wealthv <- cumprod(1 + retp)
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> # Plot cumulative log wealth
> dygraphs::dygraph(log(wealthv[endd]),
+   main="Stocks and Bonds With Proportional Allocations") %>%
+   dyOptions(colors=c("blue", "green", "blue", "red")) %>%
+   dySeries("Combined", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

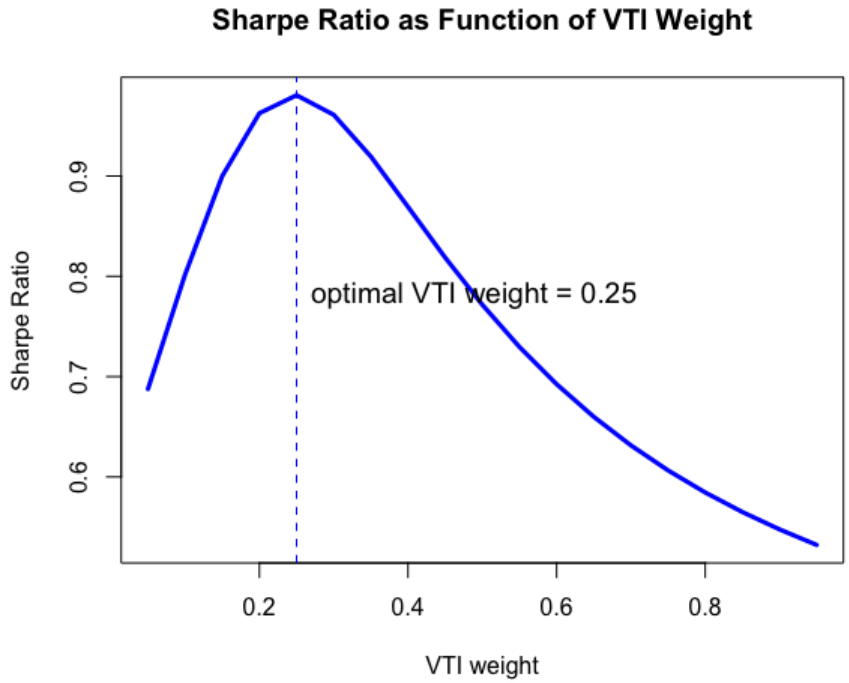
Optimal Stock and Bond Portfolio Allocations

The optimal stock and bond weights can be calculated using optimization.

Using the past 20 years of data, the optimal *VTI* weight is about 0.25.

The comments and conclusions in these slides are based on 20 years of very positive stock and bond returns, when stocks and bonds have been in a secular bull market. The conclusions would not hold if stocks and bonds had suffered from a bear market (losses) over that time.

```
> # Calculate the Sharpe ratios
> sqrt(252)*sapply(retp, function(x) mean(x)/sd(x))
> # Calculate the Sharpe ratios for vector of weights
> weightv <- seq(0.05, 0.95, 0.05)
> sharpev <- sqrt(252)*sapply(weightv, function(weight) {
+   weightv <- c(weight, 1-weight)
+   retp <- (retp[, 1:2] %*% weightv)
+   mean(retp)/sd(retp)
+ })  # end sapply
> # Calculate the optimal VTI weight
> weightm <- weightv[which.max(sharpev)]
> # Calculate the optimal weight using optimization
> calc_sharpe <- function(weight) {
+   weightv <- c(weight, 1-weight)
+   retp <- (retp[, 1:2] %*% weightv)
+   -mean(retp)/sd(retp)
+ }  # end calc_sharpe
> optv <- optimize(calc_sharpe, interval=c(0, 1))
> weightm <- optv$minimum
```




```
> # Plot Sharpe ratios
> plot(x=weightv, y=sharpev,
+      main="Sharpe Ratio as Function of VTI Weight",
+      xlab="VTI weight", ylab="Sharpe Ratio",
+      t="l", lwd=3, col="blue")
> abline(v=weightm, lty="dashed", lwd=1, col="blue")
> text(x=weightm, y=0.7*max(sharpev), pos=4, cex=1.2,
+      labels=paste("optimal VTI weight =", round(weightm, 2)))
```

Simulating Wealth Scenarios Using Bootstrap

The past data represents only one possible future scenario. We can generate more scenarios using bootstrap simulation.

The bootstrap data is a list of simulated *VTI* and *IEF* returns, which represent possible realizations of future returns, based on past history.

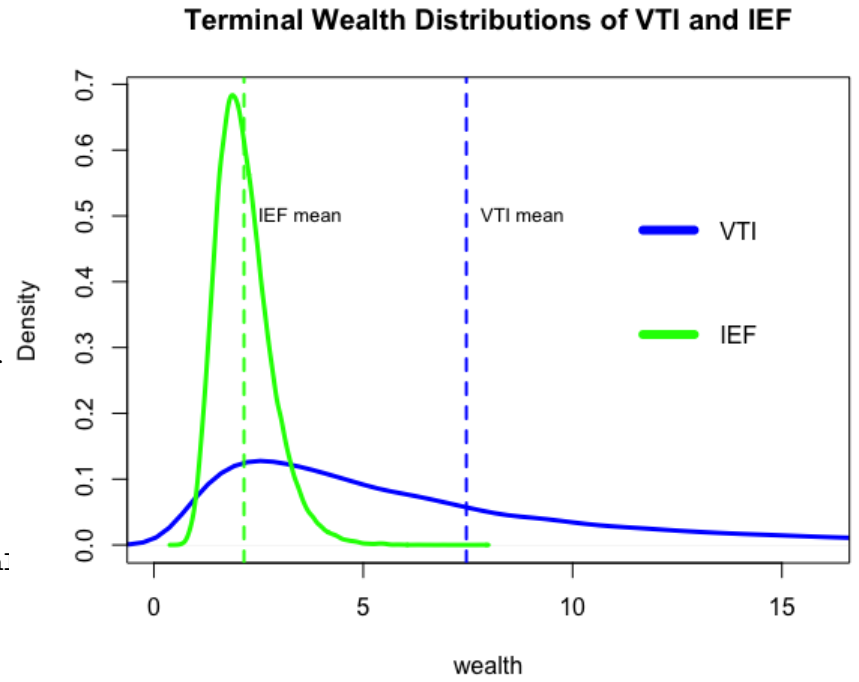
```
> # Coerce the returns from xts time series to matrix
> retp <- zoo::coredata(retp[, 1:2])
> nrows <- NROW(retp)
> # Bootstrap the returns and calculate a list of random returns
> nboot <- 1e4
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> # Perform parallel bootstrap under Windows
> cluster <- makeCluster(ncores)  # Initialize compute cluster under
> clusterSetRNGStream(cluster, 1121)  # Reset random number generato
> clusterExport(cluster, c("retp", "nrows"))
> bootd <- parLapply(cluster, 1:nboot, function(x) {
+   retp[sample.int(nrows, replace=TRUE), ]
+ })  # end parLapply
> # Perform parallel bootstrap under Mac-OSX or Linux
> set.seed(1121)
> bootd <- mclapply(1:nboot, function(x) {
+   retp[sample.int(nrows, replace=TRUE), ]
+ }, mc.cores=ncores)  # end mclapply
> is.list(bootd); NROW(bootd); dim(bootd[[1]])
```

The Distributions of Terminal Wealth From Bootstrap

The distribution of *VTI* and *IEF* wealths can be calculated from the bootstrap data.

The distribution of *VTI* wealth is much wider than *IEF*, but it has a much greater mean value.

```
> # Calculate the distribution of terminal wealths under Windows
> wealthv <- parLapply(cluster, bootd, function(retp) {
+   apply(retp, 2, function(x) prod(1+x))
+ })  # end parLapply
> # Calculate the distribution of terminal wealths under Mac-OSX or
> wealthv <- mclapply(bootd, function(retp) {
+   apply(retp, 2, function(x) prod(1+x))
+ }, mc.cores=ncores)  # end mclapply
> wealthv <- do.call(rbind, wealthv)
> class(wealthv); dim(wealthv); tail(wealthv)
> # Calculate the means and standard deviations of the terminal weal
> apply(wealthv, 2, mean)
> apply(wealthv, 2, sd)
> # Extract the terminal wealths of VTI and IEF
> wealthvti <- wealthv[, "VTI"]
> wealthief <- wealthv[, "IEF"]
```




```
> # Plot the densities of the terminal wealths of VTI and IEF
> meanvti <- mean(wealthvti); meanief <- mean(wealthief)
> densvti <- density(wealthvti); densief <- density(wealthief)
> plot(densvti, col="blue", lwd=3, xlab="wealth",
+      xlim=c(0, 2*max(densief$x)), ylim=c(0, max(densief$y)),
+      main="Terminal Wealth Distributions of VTI and IEF")
> lines(densief, col="green", lwd=3)
> abline(v=meanvti, col="blue", lwd=2, lty="dashed")
> text(x=meanvti, y=0.5, labels="VTI mean", pos=4, cex=0.8)
> abline(v=meanief, col="green", lwd=2, lty="dashed")
> text(x=meanief, y=0.5, labels="IEF mean", pos=4, cex=0.8)
> legend(x="topright", legend=c("VTI", "IEF"),
+  inset=0.1, cex=1.0, bg="white", bty="n", y.intersp=0.5,
+  lwd=6, lty=1, col=c("blue", "green"))
```

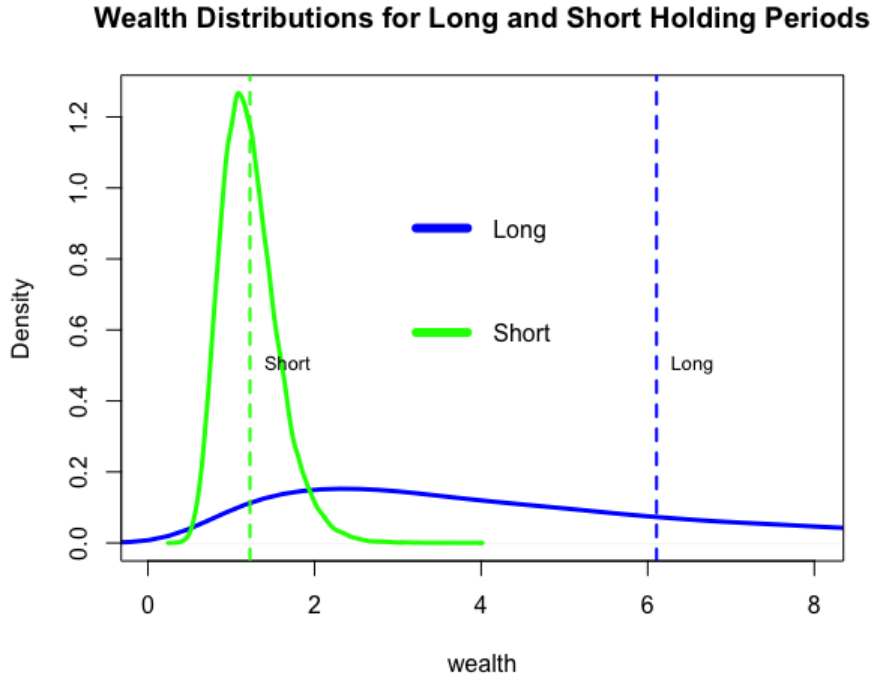
The Distribution of Stock Wealth and Holding Period

The distribution of stock wealth for short holding periods is close to symmetric around par (1).

The distribution for long holding periods is highly positively skewed with a much larger mean.

U.S. stocks in the last 40 years have had higher risk-adjusted wealth for longer holding periods.

The downside risk is equal to the mean of the wealth below par (1).




```
> # Calculate the distributions of stock wealth
> holdv <- nrows*seq(0.1, 1.0, 0.1)
> wealthm <- mclapply(bootd, function(retp) {
+   sapply(holdv, function(holdp) {
+     prod(1 + retp[1:holdp, "VTI"])
+   })  # end sapply
+ }, mc.cores=ncores)  # end mclapply
> wealthm <- do.call(rbind, wealthm)
> dim(wealthm)
> # Define the risk-adjusted wealth measure
> riskretfun <- function(wealthv) {
+   riskv <- 0.01
+   if (min(wealthv) < 1)
+     riskv <- mean((1-wealthv)[wealthv<1])
+   mean(wealthv)/riskv
+ }  # end riskretfun
> # Calculate the stock wealth risk-return ratios
> riskrets <- apply(wealthm, 2, riskretfun)
> # Plot the stock wealth risk-return ratios
> plot(x=holdv, y=riskrets,
+      main="Stock Risk-Return Ratio as Function of Holding Period
+      xlab="Holding Period", ylab="Ratio",
+      t="l", lwd=3, col="blue")
```

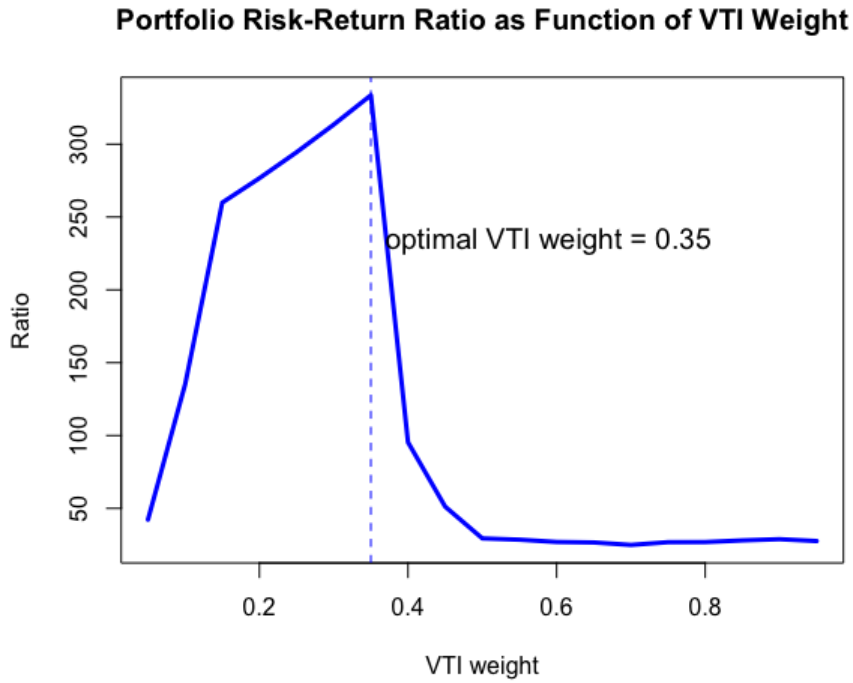
```
> # Plot the stock wealth for long and short holding periods
> wealth1 <- wealthm[, 9]
> wealth2 <- wealthm[, 1]
> mean1 <- mean(wealth1); mean2 <- mean(wealth2)
> dens1 <- density(wealth1); dens2 <- density(wealth2)
> plot(dens1, col="blue", lwd=3, xlab="wealth",
+      xlim=c(0, 2*max(dens2$x)), ylim=c(0, max(dens2$y)),
+      main="Wealth Distributions for Long and Short Holding Periods
> lines(dens2, col="green", lwd=3)
> abline(v=mean1, col="blue", lwd=2, lty="dashed")
> text(x=mean1, y=0.5, labels="Long", pos=4, cex=0.8)
> abline(v=mean2, col="green", lwd=2, lty="dashed")
> text(x=mean2, y=0.5, labels="Short", pos=4, cex=0.8)
> legend(x="top", legend=c("Long", "Short"),
+   inset=0.1, cex=1.0, bg="white", bty="n", y.intersp=0.5,
```

Optimal Stock and Bond Portfolio Allocations From Bootstrap

The optimal stock and bond weights can be calculated using bootstrap simulation.

Bootstrapping the past 20 years of data, the optimal *VTI* weight is about 0.35.

```
> # Calculate the distributions of portfolio wealth
> weightv <- seq(0.05, 0.95, 0.05)
> wealthm <- mclapply(bootd, function(retp) {
+   sapply(weightv, function(weight) {
+     prod(1 + retp %*% c(weight, 1-weight))
+   })  # end sapply
+ }, mc.cores=ncores)  # end mclapply
> wealthm <- do.call(rbind, wealthm)
> dim(wealthm)
> # Calculate the portfolio risk-return ratios
> riskrets <- apply(wealthm, 2, riskretfun)
> # Calculate the optimal VTI weight
> weightm <- weightv[which.max(riskrets)]
```




```
> # Plot the portfolio risk-return ratios
> plot(x=weightv, y=riskrets,
+      main="Portfolio Risk-Return Ratio as Function of VTI Weight",
+      xlab="VTI weight", ylab="Ratio",
+      t="l", lwd=3, col="blue")
> abline(v=weightm, lty="dashed", lwd=1, col="blue")
> text(x=weightm, y=0.7*max(riskrets), pos=4, cex=1.2,
+      labels=paste("optimal VTI weight =", round(weightm, 2)))
```

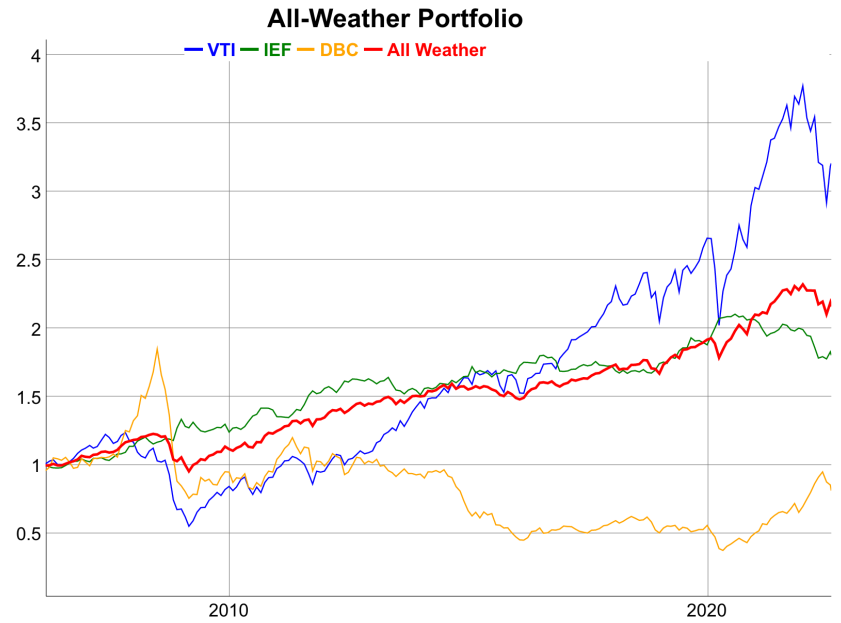
The *All-Weather* Portfolio

The *All-Weather* portfolio is a portfolio with proportional allocations of stocks (30%), bonds (55%), and commodities and precious metals (15%) (approximately).

The *All-Weather* portfolio was designed by Bridgewater Associates, the largest hedge fund in the world:
https://www.bridgewater.com/research-library/the-all-weather-strategy/
http://www.nasdaq.com/article/remember-the-allweather-portfolio-its-having-a-killer-year-cm6855

The three different asset classes (stocks, bonds, commodities) provide positive returns under different economic conditions (recession, expansion, inflation).

The combination of bonds, stocks, and commodities in the *All-Weather* portfolio is designed to provide positive returns under most economic conditions, without the costs of trading.




```
> # Extract ETF returns
> symbolv <- c("VTI", "IEF", "DBC")
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> # Calculate all-weather portfolio wealth
> weightsaw <- c(0.30, 0.55, 0.15)
> retp <- cbind(retp, retp %*% weightsaw)
> colnames(retp)[4] <- "All Weather"
> # Calculate Sharpe ratios
> sqrt(252)*sapply(retp, function(x) mean(x)/sd(x))
```

```
> # Calculate cumulative wealth from returns
> wealthv <- cumprod(1+retp)
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> # dygraph all-weather wealth
> dygraphs::dygraph(wealthv[endd], main="All-Weather Portfolio") %>%
+   dyOptions(colors=c("blue", "green", "orange", "red")) %>%
+   dySeries("All Weather", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot all-weather wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("orange", "blue", "green", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme, lwd=c(2, 2, 2, 4
+        name="All-Weather Portfolio")
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

depr: Combining the Standardized Returns of Multiple Assets

Adding the weighted *standardized dollar returns* is equivalent to buying *fixed number of shares* such that their dollar volatilities are proportional to the weights.

If the asset volatilities change over time then the portfolio allocations must be rebalanced to ensure that the volatilities remain proportional to the target weights.

Adding the weighted *standardized returns* of multiple assets is equivalent to buying stock amounts such that their volatilities are proportional to the weights.

Adding the weighted *standardized percentage returns* is equivalent to buying *fixed dollar amounts of stock* such that their percentage volatilities are proportional to the weights.

```
> # Calculate standardized simple dollar returns
> retsd_std <- lapply(retd, function(x) x/sd(x))
> retsd_std <- do.call(cbind, retsd_std)
> sapply(retsd_std, sd)
> # Wealth of fixed number of shares (without rebalancing)
> weightv <- c(0.5, 0.5)
> wealth_fsa <- cumsum(retd %*% (weightv/pricesi))
> # Calculate standardized percentage returns
> retsp_std <- lapply(retp, function(x) x/sd(x))
> retsp_std <- do.call(cbind, retsp_std)
> sapply(retsp_std, sd)
> # Wealth of fixed dollar amount of shares (with rebalancing)
> wealth_fda <- cumsum(retsp_std %*% weightv)
> # Plot log wealth
> wealthv <- cbind(wealth_fda, log(wealth_fsa))
> # wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("With rebalancing", "Without rebalancing")
> dygraphs::dygraph(wealthv, main="Wealth of Equal Dollar Amount of
+   dyOptions(colors=c("green", "blue"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Constant Proportion Portfolio Insurance Strategy

In the *Constant Proportion Portfolio Insurance* (CPPI) strategy the portfolio is rebalanced between stocks and zero-coupon bonds, to protect against the loss of principal.

A zero-coupon bond pays no coupon, but it's bought at a discount to par (100%), and pays par at maturity. The investor receives capital appreciation instead of coupons.

Let P be the investor principal amount (total initial invested dollar amount), and let F be the zero-coupon *bond floor*. The zero-coupon bond floor F is set so that its value at maturity is equal to the principal P. This guarantees that the investor is paid back at least the full principal P.

The stock investment is levered by the *CPPI multiplier* C. The initial dollar amount invested in stocks is equal to the *cushion* $(P - F)$ times the *multiplier* C: $C * (P - F)$. The remaining amount of the principal is invested in zero-coupon bonds and is equal to: $P - C * (P - F)$.

```
> # Calculate VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI["2008/2009"])
> datev <- zoo::index(retvti)
> nrows <- NROW(retvti)
> retvti <- drop(zoo::coredata(retvti))
> # Bond floor
> bfloor <- 60
> # CPPI multiplier
> coeff <- 2
> # Portfolio market values
> portfv <- numeric(nrows)
> # Initial principal
> portfv[1] <- 100
> # Stock allocation
> stockv <- numeric(nrows)
> stockv[1] <- min(coeff*(portfv[1] - bfloor), portfv[1])
> # Bond allocation
> bondv <- numeric(nrows)
> bondv[1] <- (portfv[1] - stockv[1])
```

CPPI Strategy Dynamics

If the stock price changes and the portfolio value becomes P_t, then the dollar amount invested in stocks must be adjusted to: $C * (P_t - F)$. The amount invested in stocks changes both because the stock price changes and because of rebalancing with the zero-coupon bonds.

The amount invested in zero-coupon bonds is then equal to: $P_t - C * (P_t - F)$. If the portfolio value drops to the *bond floor* $P_t = F$, then all the stocks must be sold, with only the zero-coupon bonds remaining. But if the stock price rises, more stocks must be purchased, and vice versa.

Therefore the *CPPI* strategy is a *trend following* strategy, buying stocks when their prices are rising, and selling when their prices are dropping.

The *CPPI* strategy can be considered a dynamic replication of a portfolio with a zero-coupon bond and a stock call option.

The *CPPI* strategy is exposed to *gap risk*, if stock prices drop suddenly by a large amount. The *gap risk* is exacerbated by high leverage, when the *multiplier* C is large, say greater than 5.


```
> # Simulate CPPI strategy
> for (t in 2:nrows) {
+   portfv[t] <- portfv[t-1] + stockv[t-1]*retvti[t]
+   stockv[t] <- min(coeff*(portfv[t] - bfloor), portfv[t])
+   bondv[t] <- (portfv[t] - stockv[t])
+ }  # end for
> # dygraph plot of CPPI strategy
> pricevti <- 100*cumprod(1+retvti)
> datav <- xts::xts(cbind(stockv, bondv, portfv, pricevti), datev)
> colnames(datav) <- c("stocks", "bonds", "CPPI", "VTI")
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endd], main="CPPI strategy") %>%
+   dyOptions(colors=c("red", "green", "blue", "orange"), strokeWidt
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy

In the *Risk Parity* strategy the dollar portfolio allocations are rebalanced daily so that their dollar volatilities remain equal.

This means that the allocations a_i are proportional to the *standardized prices* ($\frac{p_i}{\sigma_i^d}$ - the dollar amounts of stocks with unit dollar volatilities): $a_i \propto \frac{p_i}{\sigma_i^d}$, where σ_i^d is the dollar volatility.

But the *standardized prices* are equal to the inverse of the percentage volatilities σ_i: $\frac{p_i}{\sigma_i^d} = \frac{1}{\sigma_i}$, so the allocations a_i are proportional to the inverse of the percentage volatilities $a_i \propto \frac{1}{\sigma_i}$.

In general, the dollar allocations a_i may be set proportional to some target weights ω_i:

$$a_i \propto \frac{\omega_i}{\sigma_i}$$

The risk parity strategy is also called the equal risk contributions (ERC) strategy.

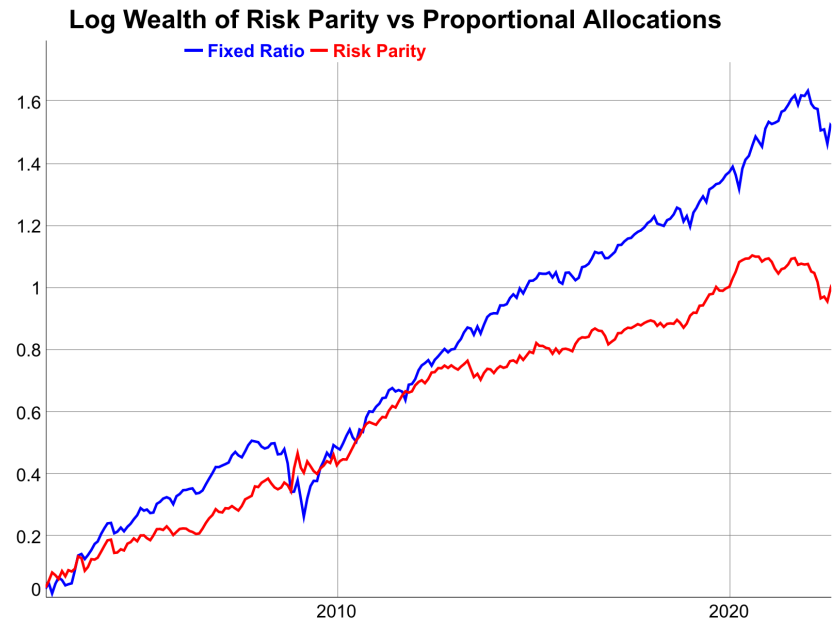
```
> # Calculate dollar and percentage returns for VTI and IEF.
> pricev <- rutils::etfenv$pricev[, c("VTI", "IEF")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate wealth of proportional allocations.
> weightv <- c(0.5, 0.5)
> retsw <- retp %*% weightv
> wealth_pda <- cumprod(1 + retsw)
> # Calculate rolling percentage volatility.
> look_back <- 21
> volat <- HighFreq::roll_var(retp, look_back=look_back)
> iszero <- (rowSums(volat) == 0)
> volat[iszero, ] <- 1
> # Calculate the risk parity portfolio allocations.
> alloc <- lapply(1:NCOL(pricev),
+   function(x) weightv[x]/volat[, x])
> alloc <- do.call(cbind, alloc)
> # Scale allocations to 1 dollar total.
> alloc <- alloc/rowSums(alloc)
> # Lag the allocations
> alloc <- rutils::lagit(alloc)
> # Calculate wealth of risk parity.
> retsw <- rowSums(retp*alloc)
> wealth_risk_parity <- cumprod(1 + retsw)
```

Risk Parity Strategy Performance

The risk parity strategy for *VTI* and *IEF* has a higher *Sharpe ratio* than the fixed ratio strategy because it's more overweight bonds, which is also why it has lower absolute returns.

Risk parity works better for assets with low correlations and very different volatilities, like stocks and bonds.

The shiny app `app_risk_parity_strat.R` allows users to study the performance of the risk parity strategy as a function of its weight parameters.



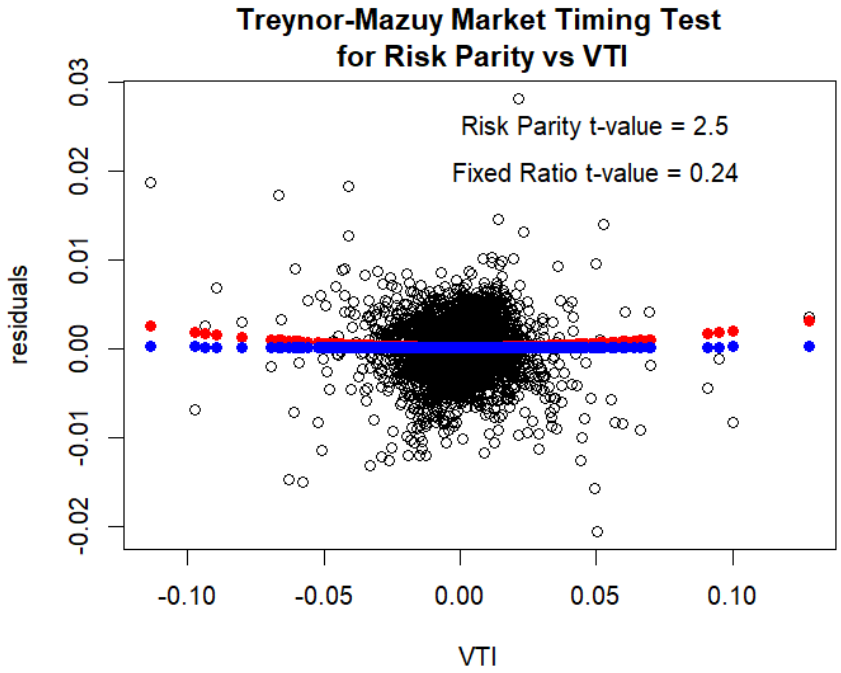

```
> # Calculate the log wealths.
> wealthv <- log(cbind(wealth_pda, wealth_risk_parity))
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed Ratio", "Risk Parity")
> # Calculate the Sharpe ratios.
> sqrt(252)*sapply(rutils::diffit(wealthv), function (x) mean(x)/sd
> # Plot a dygraph of the log wealths.
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(wealthv[endd],
+   main="Log Wealth of Risk Parity vs Proportional Allocations") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Risk Parity Strategy Market Timing Skill

The risk parity strategy reduces allocations to assets with rising volatilities, which is often accompanied by negative returns.

This allows the risk parity strategy to better time the markets - selling when prices are about to drop and buying when prices are rising.

The t-value of the *Treynor-Mazuy* test is slightly significant, indicating some market timing skill of the risk parity strategy for *VTI* and *IEF*.




```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> retsrp <- rutils::diffit(wealthv)
> retvti <- retp$VTI
> desv <- cbind(retsrp, retvti, retvti^2)
> desv <- na.omit(desv)
> colnames(desv)[1:2] <- c("fixed", "risk_parity")
> colnames(desv)[4] <- "treynor"
> regmod <- lm(risk_parity ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot residual scatterplot
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Risk Parity vs
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retv
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residv), paste("Treynor test t-value =", tvalue))
```

```
> # Test for fixed ratio market timing of VTI using Treynor-Mazuy te
> regmod <- lm(fixed ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvt
> points.default(x=retvti, y=fittedv, pch=16, col="blue")
> text(x=0.05, y=0.6*max(residv), paste("Fixed Ratio t-value =", rou
```

draft: Transaction Costs of Risk Parity Strategy

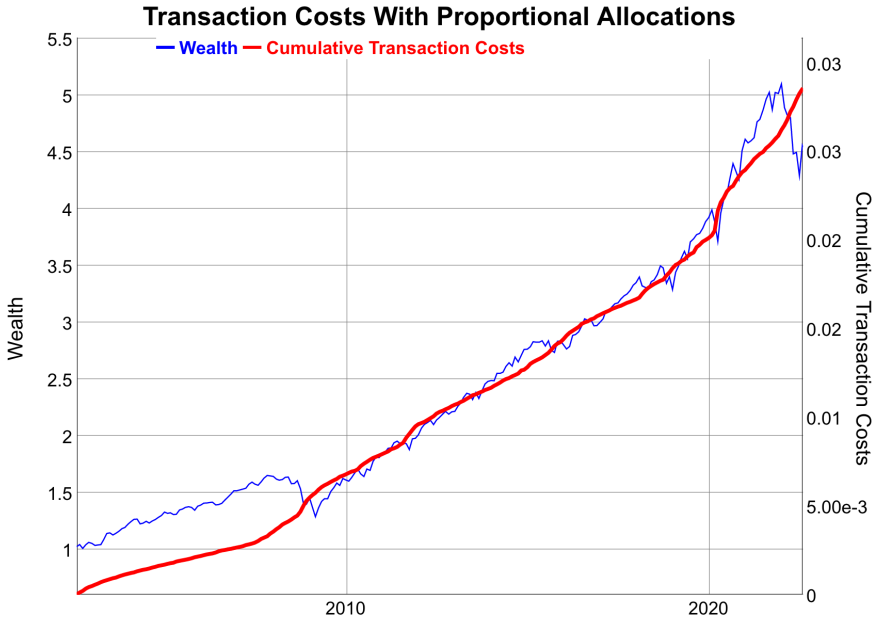
In each period the stocks must be rebalanced to maintain the proportional allocations.

The total dollar amount of stocks that need to be traded to rebalance back to the target weight is equal to: $\sum_{i=1}^{n} \varepsilon_i = w_{t-1} \sum_{i=1}^{n} \omega_i \left| \bar{r}_t - r_i \right|$

The *transaction costs* c_t^r are equal to half the *bid-offer spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2} \sum_{i=1}^{n} \varepsilon_i$.

The *cumulative transaction costs* $\sum_{i=1}^{t} c_i^r$ must be subtracted from the *wealth* w_t: $w_t - \sum_{i=1}^{t} c_i^r$.

```
> # Returns in excess of weighted returns
> retsx <- lapply(retp, function(x) (retsw - x))
> retsx <- do.call(cbind, retsx)
> sum(retsx %*% weightv)
> # Calculate weighted sum of absolute excess returns
> retsx <- abs(retsx) %*% weightv
> # Total dollar amount of stocks that need to be traded
> retsx <- retsx*rutils::lagit(wealth_pda)
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(retsx)/2
> # Subtract transaction costs from wealth
> wealth_pda <- (wealth_pda - costs)
```

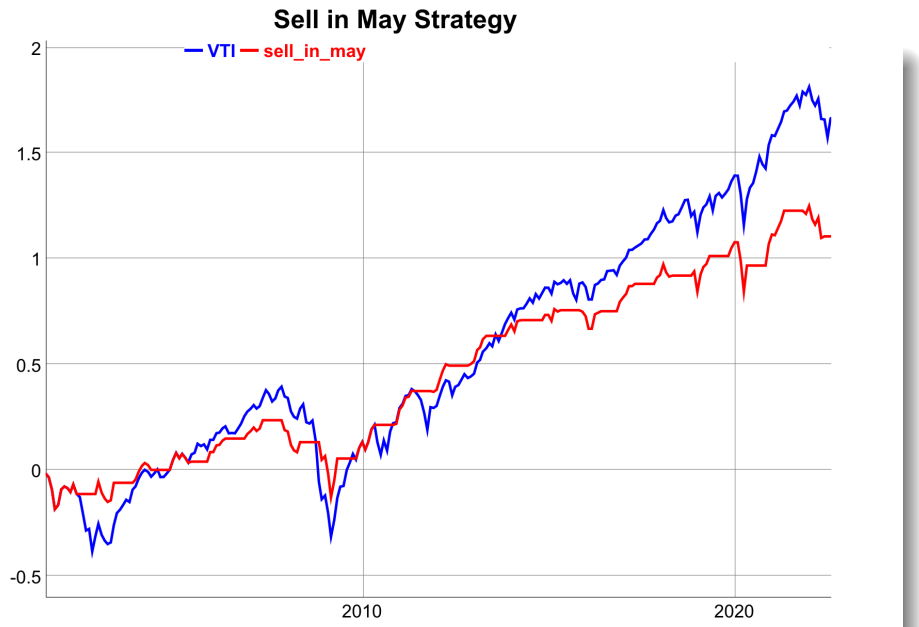



```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealth_pda, costs)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Wealth", "Cumulative Transaction Costs")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv, main="Transaction Costs With Proportion
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Sell in May Calendar Strategy

Sell in May is a *market timing calendar strategy*, in which stocks are sold at the beginning of May, and then bought back at the beginning of November.

```
> # Calculate positions
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> posit <- rep(NA_integer_, NROW(retvti))
> datev <- zoo::index(retvti)
> datev <- format(datev, "%m-%d")
> posit[datev == "05-01"] <- 0
> posit[datev == "05-03"] <- 0
> posit[datev == "11-01"] <- 1
> posit[datev == "11-03"] <- 1
> # Carry forward and backward non-NA posit
> posit <- zoo::na.locf(posit, na.rm=FALSE)
> posit <- zoo::na.locf(posit, fromLast=TRUE)
> # Calculate strategy returns
> sell_inmay <- posit*retvti
> wealthv <- cbind(retvti, sell_inmay)
> colnames(wealthv) <- c("VTI", "sell_in_may")
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])
```

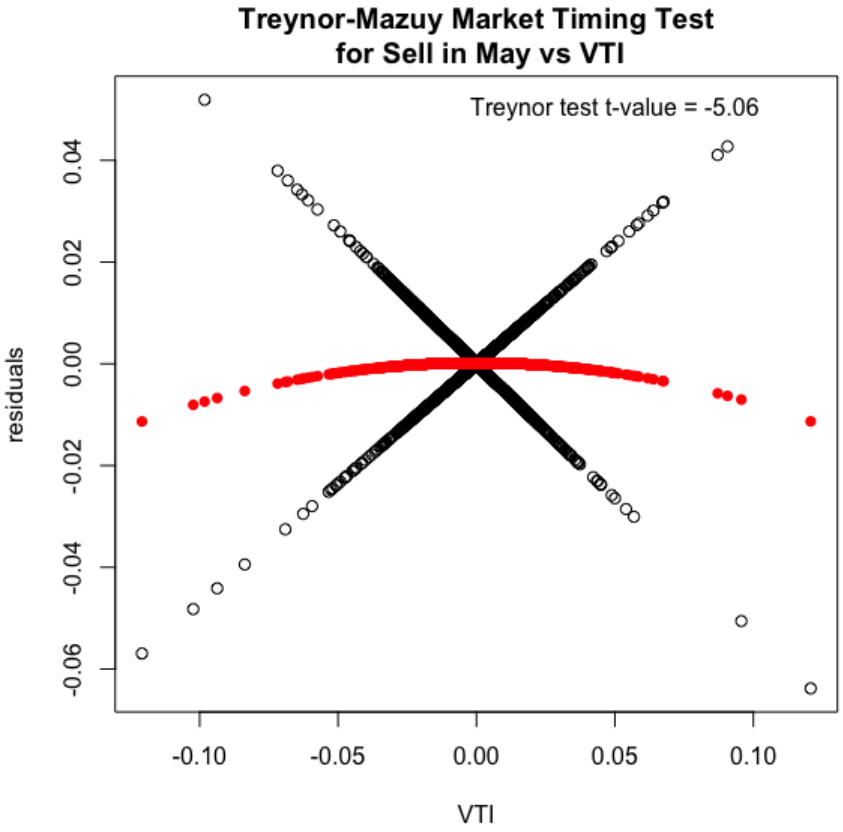



```
> # Plot wealth of Sell in May strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Sell in May Strateg
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # OR: Open x11 for plotting
> x11(width=6, height=5)
> par(mar=c(4, 4, 3, 1), oma=c(0, 0, 0, 0))
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("blue", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme, name="Sell in Ma
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

Sell in May Strategy Market Timing

The *Sell in May* strategy doesn't demonstrate any ability of *timing* the *VTI* ETF.

```
> # Test if Sell in May strategy can time VTI
> desv <- cbind(wealth$sell_in_may, 0.5*(retvti+abs(retvti)), retvti
> colnames(desv) <- c("VTI", "merton", "treynor")
> # Perform Merton-Henriksson test
> regmod <- lm(sell_inmay ~ VTI + merton, data=desv)
> summary(regmod)
> # Perform Treynor-Mazuy test
> regmod <- lm(sell_inmay ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot Treynor-Mazuy residual scatterplot
> residv <- (sell_inmay - regmod$coeff["VTI"]*retvti)
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Sell in May vs
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvt
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residv), paste("Treynor test t-value =", tva
```



Seasonal Overnight Market Anomaly

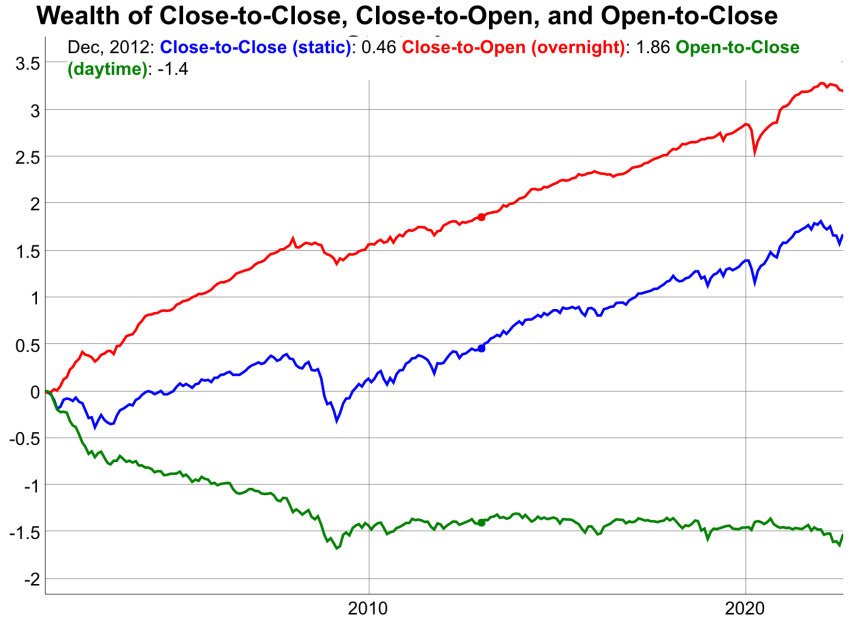
The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the daytime returns.

The Overnight Strategy consists of holding a long position only overnight (buying at the close and selling at the open the next day).

The Daytime Strategy consists of holding a long position only during the daytime (buying at the open and selling at the close the same day).

The *Overnight Market Anomaly* has been observed for many decades for most stock market indices, but not always for all stock sectors.

The *Overnight Market Anomaly* has mostly disappeared after the 2008–2009 financial crisis.




```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> # Calculate the close-to-close log returns, the intraday
> # open-to-close returns and the overnight close-to-open returns.
> close_close <- rutils::diffit(closep)
> colnames(close_close) <- "close_close"
> open_close <- (closep - openp)
> colnames(open_close) <- "open_close"
> close_open <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(close_open) <- "close_open"
```

```
> # Calculate Sharpe and Sortino ratios
> wealthv <- cbind(close_close, close_open, open_close)
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Close-to-Close, Close-to-Open, and Open-to-Close
+   dySeries(name="close_close", label="Close-to-Close (static)", st
+   dySeries(name="close_open", label="Close-to-Open (overnight)", s
+   dySeries(name="open_close", label="Open-to-Close (daytime)", str
+   dyLegend(width=600)
```

Turn of the Month Effect

The *Turn of the Month* (TOM) effect is the outperformance of stocks on the last trading day of the month and on the first three days of the following month.

The *TOM* effect was observed for the period from 1928 to 1975, but it has been less pronounced since the year 2000.

The *TOM* effect has been attributed to the investment of funds deposited at the end of the month.

This would explain why the *TOM* effect has been more pronounced for less liquid small-cap stocks.


```
> # Calculate the VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retvti)
> # Calculate first business day of every month
> dayv <- as.numeric(format(datev, "%d"))
> indeks <- which(rutils::diffit(dayv) < 0)
> datev[head(indeks)]
> # Calculate Turn of the Month dates
> indeks <- lapply((-1):2, function(x) indeks + x)
> indeks <- do.call(c, indeks)
> sum(indeks > NROW(datev))
> indeks <- sort(indeks)
> datev[head(indeks, 11)]
> # Calculate Turn of the Month pnls
> pnls <- numeric(NROW(retvti))
> pnls[indeks] <- retvti[indeks, ]
```

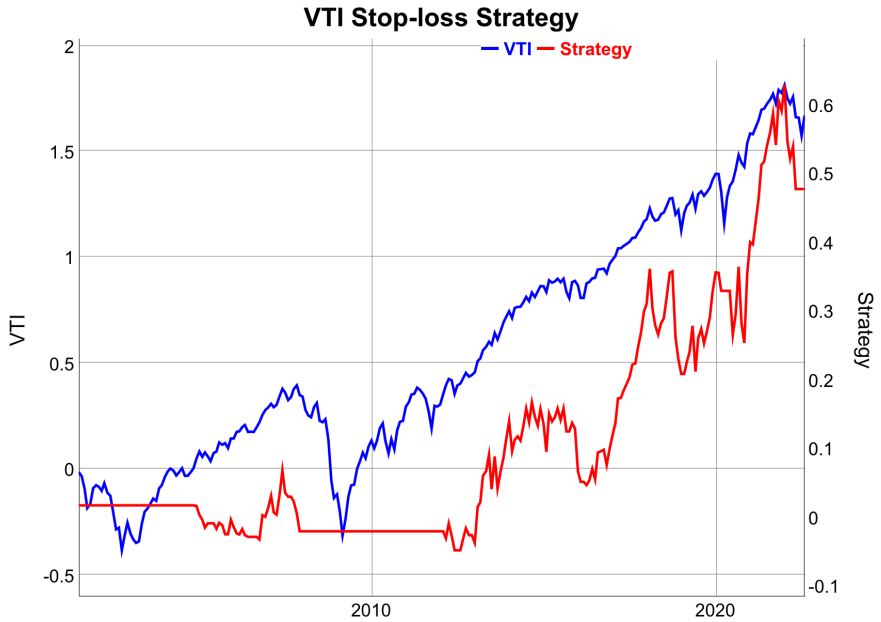
```
> # Combine data
> wealthv <- cbind(retvti, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI Turn of the Month strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Turn of the Month Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", strokeWidth=2, col="blue")
+   dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="red")
```

Stop-loss Rules

Stop-loss rules are used to reduce losses in case of a significant drawdown in returns.

For example, a simple stop-loss rule is to sell the stock if its price drops by 5% below the recent maximum price, and buy it back when the price recovers.

```
> # Calculate the VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retvti)
> retvti <- drop(coredata(retvti))
> nrows <- NROW(retvti)
> # Simulate stop-loss strategy
> stopl <- 0.05
> maxp <- 0.0
> retc <- 0.0
> pnls <- retvti
> for (i in 1:(nrows-1)) {
+ # Calculate drawdown
+   retc <- retc + retvti[i]
+   maxp <- max(maxp, retc)
+   dd <- (retc - maxp)
+ # Check for stop-loss
+   if (dd < -stopl*maxp)
+     pnls[i+1] <- 0
+ }  # end for
> # Same but without using explicit loops
> cumsumv <- cumsum(retvti)
> cummaxv <- cummax(cumsumv)
> dd <- (cumsumv - cummaxv)
> pnls2 <- retvti
> isdd <- rutils::lagit(dd < -stopl*cummaxv)
> pnls2 <- ifelse(isdd, 0, pnls2)
> all.equal(pnls, pnls2)
```

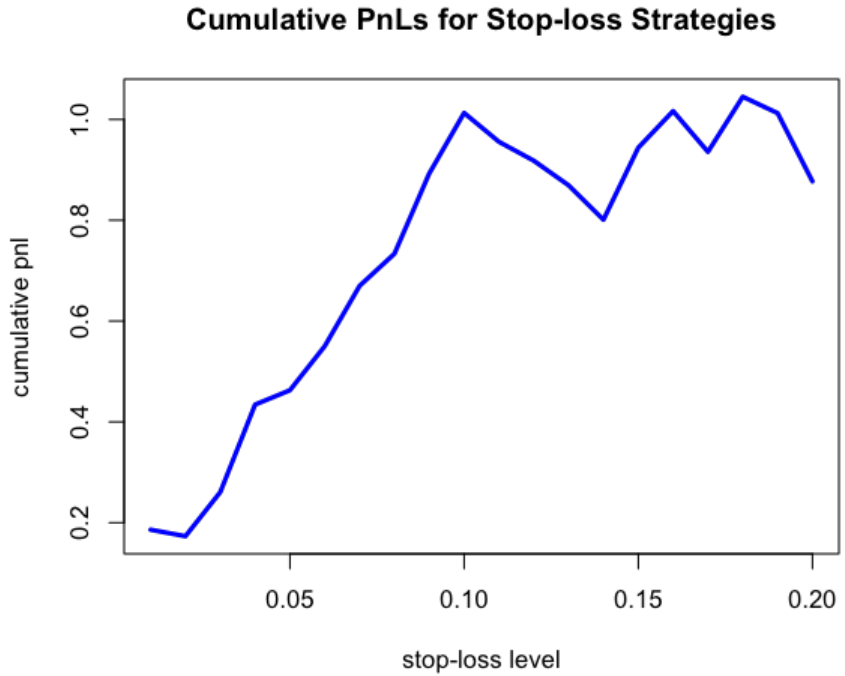



```
> # Combine data
> wealthv <- xts::xts(cbind(retvti, pnls), datev)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI stop-loss strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Stop-loss Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", strokeWidth=2, col="blue")
+   dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="red")
```

Optimal Stop-loss Rules

Stop-loss rules can reduce the largest drawdowns but they also tend to reduce cumulative returns.

```
> # Simulate multiple stop-loss strategies
> cumsumv <- cumsum(retvti)
> cummaxv <- cummax(cumsumv)
> dd <- (cumsumv - cummaxv)
> cum_pnls <- sapply(0.01*(1:20), function(stopl) {
+   pnls <- retvti
+   isdd <- rutils::lagit(dd < -stopl*cummaxv)
+   pnls <- ifelse(isdd, 0, pnls)
+   sum(pnls)
+ })  # end sapply
```



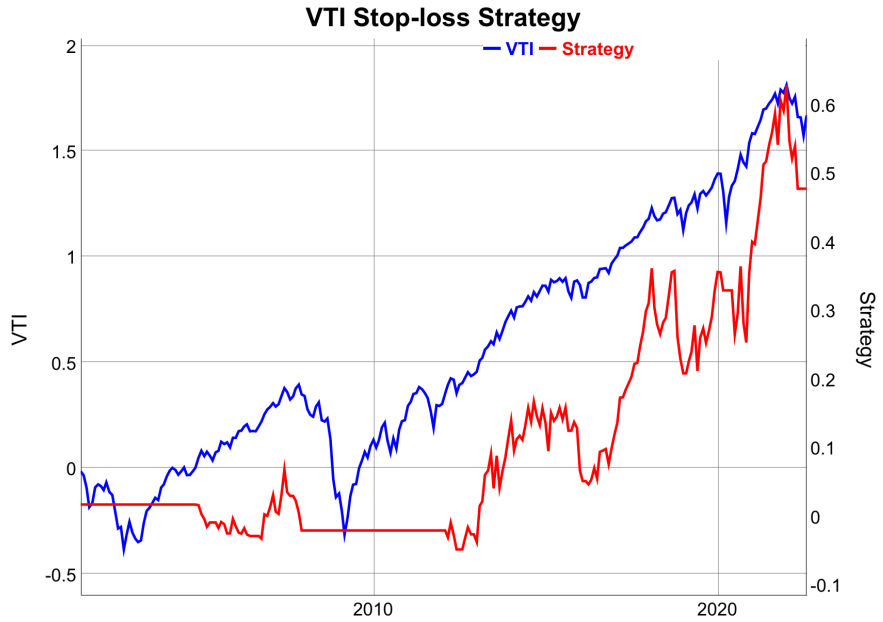

```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=0.01*(1:20), y=cum_pnls,
+      main="Cumulative PnLs for Stop-loss Strategies",
+      xlab="stop-loss level", ylab="cumulative pnl",
+      t="l", lwd=3, col="blue")
```

draft: Stop-loss and Gain Rules

Stop-loss rules are used to reduce losses in case of a significant drawdown in returns.

For example, a simple stop-loss rule is to sell the stock if its price drops by 5% below the recent maximum price.

```
> # Simulate stop-loss strategy
> stopl <- 0.05
> maxp <- 0.0
> minp <- 0.0
> retc <- 0.0
> pnls <- retvti
> for (i in 1:(nrows-1)) {
+ # Calculate drawdown
+   retc <- retc + retvti[i]
+   maxp <- max(maxp, retc)
+   dd <- (retc - maxp)
+ # Check for stop-loss
+   if (dd < -stopl*maxp) {
+     pnls[i+1] <- 0
+     minp <- min(minp, retc)
+     du <- (retc - minp)
+ # Check for gain
+     if (du > stopl*minp) {
+ pnls[i+1] <- retvti[i+1]
+     }  # end if
+   } else {
+     minp <- retc
+   }  # end if
+ }  # end for
```




```
> # Combine data
> wealthv <- xts::xts(cbind(retvti, pnls), datev)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI stop-loss strategy
> dygraphs::dygraph(cumsum(wealthv), main="VTI Stop-loss Strategy")
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", strokeWidth=2, col="blue")
+   dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="red")
```

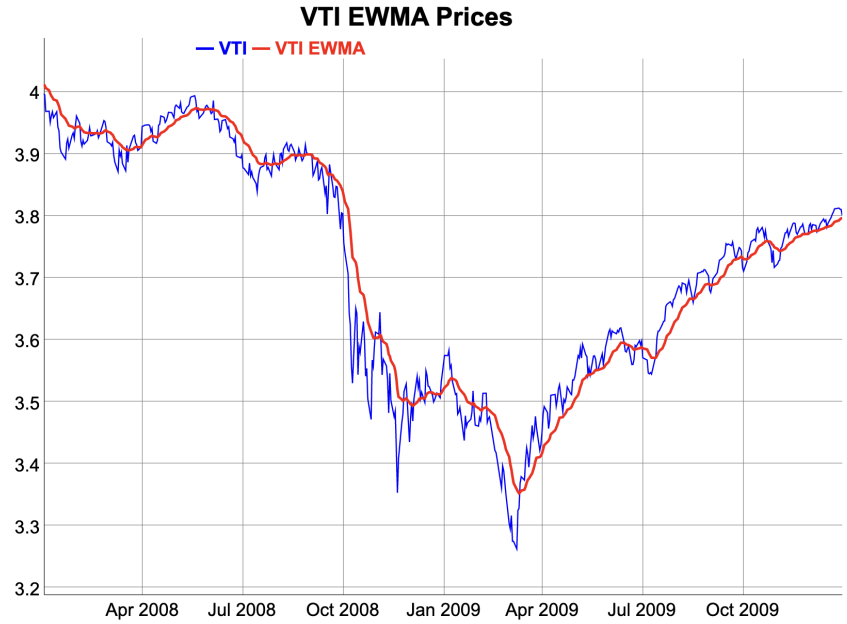
EWMA Price Technical Indicator

The *Exponentially Weighted Moving Average Price* (*EWMA*) is defined as the weighted average of prices over a rolling interval:

$$p_i^{EWMA} = (1 - \lambda) \sum_{j=0}^{\infty} \lambda^j p_{i-j}$$

Where the decay parameter λ determines the rate of decay of the *EWMA* weights, with smaller values of λ producing faster decay, giving more weight to recent pricev, and vice versa.

The function `HighFreq::roll_wsum()` calculates the convolution of a time series with a vector of weights.




```
> # Extract log VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> closep <- quantmod::Cl(ohlc)
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> # Calculate EWMA weights
> look_back <- 333
> lambda <- 0.984
> weightv <- lambda^(0:look_back)
> weightv <- weightv/sum(weightv)
> # Calculate EWMA prices as the convolution
> ewmacpp <- HighFreq::roll_wsum(closep, weightv=weightv)
> pricev <- cbind(closep, ewmacpp)
> colnames(pricev) <- c("VTI", "VTI EWMA")
```

```
> # Dygraphs plot with custom line colors
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2008/2009"], main="VTI EWMA Prices") %>%
+   dySeries(name=colnamev[1], label=colnamev[1], strokeWidth=1, col
+   dySeries(name=colnamev[2], label=colnamev[2], strokeWidth=2, col
+   dyLegend(show="always", width=500)
> # Standard plot of  EWMA prices with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> colorv <- c("blue", "red")
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI EWMA Prices")
> legend("topleft", legend=colnames(pricev), y.intersp=0.5,
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

Recursive EWMA Price Indicator

The *EWMA* prices can be calculated recursively as follows:

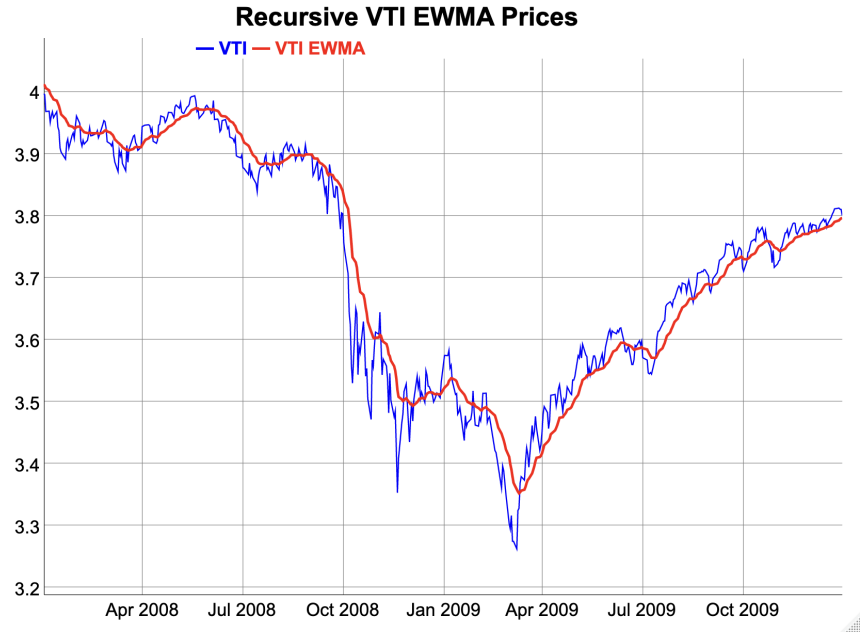
$$p_i^{EWMA} = (1 - \lambda)p_i + \lambda p_{i-1}^{EWMA}$$

Where the decay parameter λ determines the rate of decay of the *EWMA* weights, with smaller values of λ producing faster decay, giving more weight to recent pricev, and vice versa.

The recursive *EWMA* prices are slightly different from those calculated as a convolution, because the convolution uses a fixed look-back interval.

The compiled `C++` function `stats:::C_rfilter()` calculates the exponentially weighted moving average prices recursively.

The function `HighFreq::run_mean()` calculates the exponentially weighted moving average prices recursively.




```
> # Calculate EWMA prices recursively using C++ code
> ewmar <- .Call(stats:::C_rfilter, closep, lambda, c(as.numeric(cl
> # Or R code
> # ewmar <- filter(closep, filter=lambda, init=as.numeric(closep[1
> ewmar <- (1-lambda)*ewmar
> # Calculate EWMA prices recursively using RcppArmadillo
> ewmacpp <- HighFreq::run_mean(closep, lambda=lambda)
> all.equal(drop(ewmacpp), ewmar)
> # Compare the speed of C++ code with RcppArmadillo
> library(microbenchmark)
> summary(microbenchmark(
+   Rcpp=HighFreq::run_mean(closep, lambda=lambda),
+   rfilter=.Call(stats:::C_rfilter, closep, lambda, c(as.numeric(c
```

```
> # Dygraphs plot with custom line colors
> pricev <- cbind(closep, ewmacpp)
> colnames(pricev) <- c("VTI", "VTI EWMA")
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2008/2009"], main="Recursive VTI EWMA Pr
+   dySeries(name=colnamev[1], label=colnamev[1], strokeWidth=1, col
+   dySeries(name=colnamev[2], label=colnamev[2], strokeWidth=2, col
+   dyLegend(show="always", width=500)
> # Standard plot of  EWMA prices with custom line colors
> plot_theme <- chart_theme()
> colorv <- c("blue", "red")
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI EWMA Prices")
> legend("topleft", legend=colnames(pricev), y.intersp=0.5,
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

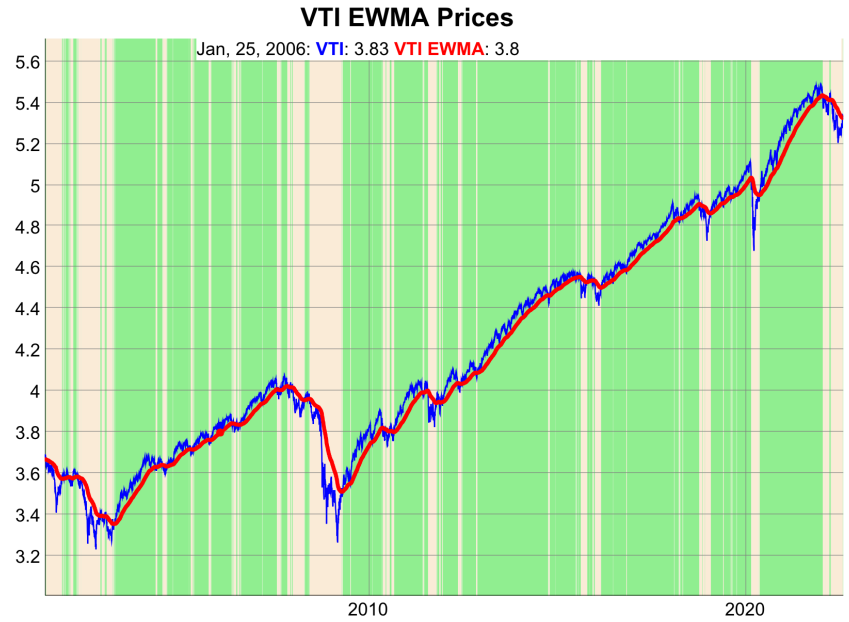
Simulating the EWMA Crossover Strategy

The trend following *EWMA Crossover* strategy switches its risk position depending if the current price is above or below the *EWMA*.

If the stock price is above the *EWMA* price, then the strategy switches to long \$1 dollar risk position, and if it is below, to short risk \$1 dollar.

The strategy holds the same position until the *EWMA* crosses over the current price (either from above or below), and then it switches its position.

The strategy is therefore always either long \$1 dollar risk, or short \$1 dollar risk.




```
> # Calculate positions, either: -1, 0, or 1
> indic <- sign(closep - ewmacpp)
> posit <- rutils::lagit(indic, lagg=1)
> # Create colors for background shading
> crossd <- (rutils::diffit(posit) != 0)
> shadev <- posit[crossd]
> crossd <- c(zoo::index(shadev), end(posit))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen",
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(pricev, main="VTI EWMA Prices") %>%
+   dySeries(name=colnamev[1], label=colnamev[1], strokeWidth=1, c
+   dySeries(name=colnamev[2], label=colnamev[2], strokeWidth=3, col="red") %>%
+   dyLegend(show="always", width=500)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], color=shadev[i])
+ }  # end for
> # Plot the dygraph object
> dyplot
```

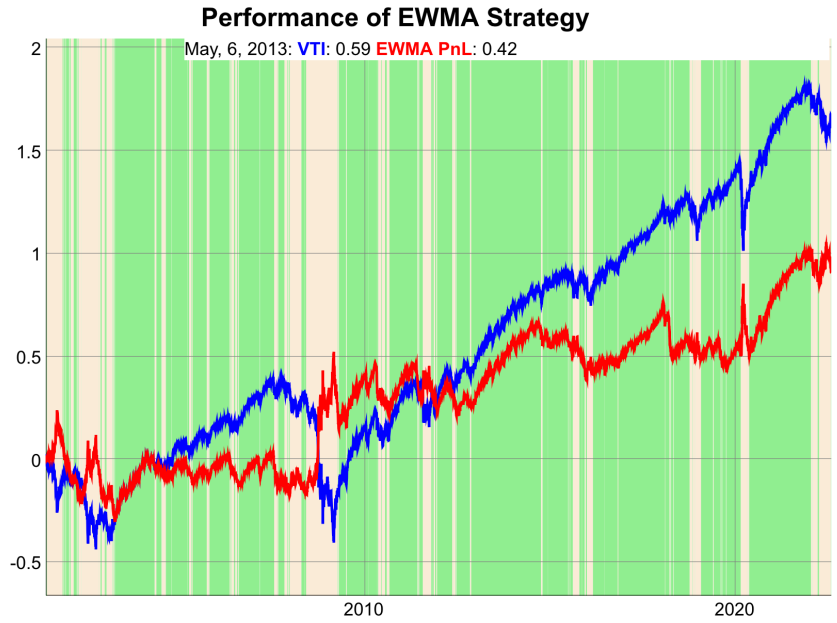
```
> # Standard plot of EWMA prices with position shading
> quantmod::chart_Series(pricev, theme=plot_theme,
+        lwd=2, name="VTI EWMA Prices")
> add_TA(posit > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posit < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("topleft", legend=colnames(pricev),
+  inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+  col=plot_theme$col$line.col, bty="n")
```

Performance of the EWMA Crossover Strategy

The crossover strategy trades at the *Close* price on the same day that prices cross the *EWMA*, which may be difficult in practice.

The crossover strategy performance is worse than the underlying asset (*VTI*), but it has a negative correlation to it, which is very valuable when building a portfolio.

```
> # Calculate daily profits and losses of EWMA strategy
> retvti <- rutils::diffit(closep)  # VTI returns
> pnls <- retvti*posit
> colnames(pnls) <- "EWMA"
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "EWMA PnL")
> # Annualized Sharpe ratio of EWMA strategy
> sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
> # Plot dygraph of EWMA strategy wealth
> # Create dygraph object without plotting it
> colorv <- c("blue", "red")
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main="Performance o
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>%
+ dyShading(from=crossd[i], to=crossd[i+1], color=shadev[i])
+ }  # end for
> # Plot the dygraph object
> dyplot
```



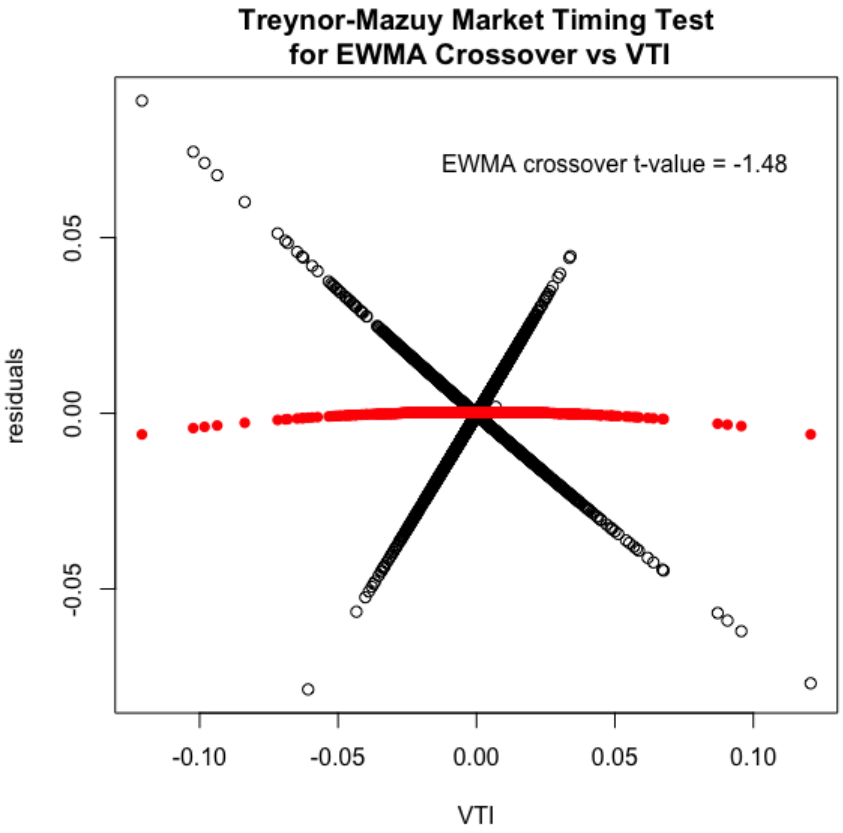

```
> # Standard plot of EWMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+        name="Performance of EWMA Strategy")
> add_TA(posit > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posit < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv), y.intersp=0.5,
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

EWMA Crossover Strategy Market Timing Skill

The EWMA crossover strategy shorts the market during significant selloffs, but otherwise doesn't display market timing skill.

The t-value of the *Treynor-Mazuy* test is negative, but not statistically significant.

```
> # Test EWMA crossover market timing of VTI using Treynor-Mazuy tes
> desv <- cbind(pnls, retvti, retvti^2)
> desv <- na.omit(desv)
> colnames(desv) <- c("EWMA", "VTI", "treynor")
> regmod <- lm(EWMA ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot residual scatterplot
> residv <- (desv$EWMA - regmod$coeff["VTI"]*retvti)
> residv <- regmod$residuals
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for EWMA Crossover
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvt
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residv), paste("Treynor test t-value =", tva
```



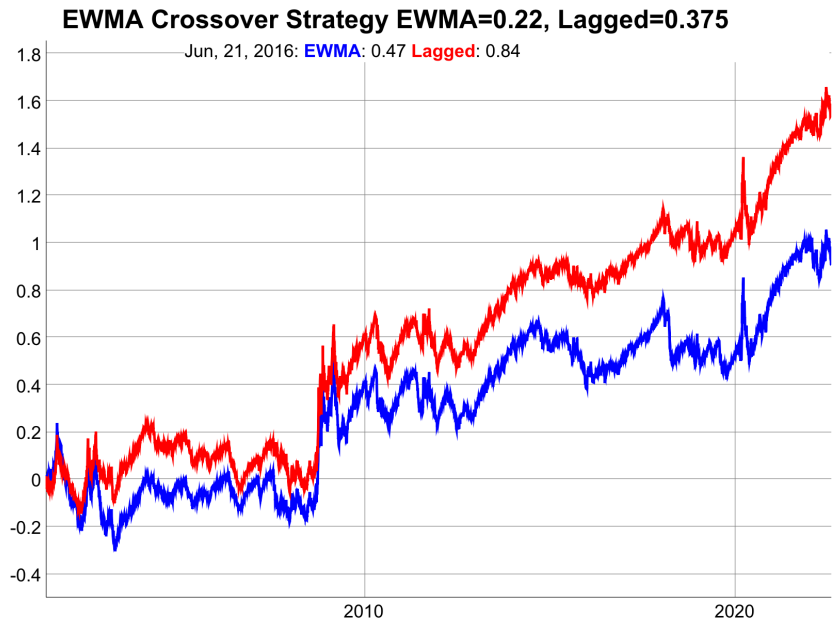
EWMA Crossover Strategy With Lag

The crossover strategy suffers losses when prices are range-bound without a trend, because whenever it switches position the prices soon change direction. (This is called a *"whipsaw"*.)

To prevent whipsaws and over-trading, the crossover strategy may choose to delay switching positions until the indicator repeats the same value for several periods.

There's a tradeoff between switching positions too early and risking a whipsaw, and waiting too long and missing an emerging trend.

```
> # Determine trade dates right after EWMA has crossed prices
> indic <- sign(closep - ewmacpp)
> # Calculate positions from lagged indicator
> lagg <- 2
> indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
> # Calculate positions, either: -1, 0, or 1
> posit <- rep(NA_integer_, nrows)
> posit[1] <- 0
> posit <- ifelse(indic == lagg, 1, posit)
> posit <- ifelse(indic == (-lagg), -1, posit)
> posit <- zoo::na.locf(posit, na.rm=FALSE)
> posit <- xts::xts(posit, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posit <- rutils::lagit(posit, lagg=1)
> # Calculate PnLs of lagged strategy
> pnlslag <- retvti*posit
> colnames(pnlslag) <- "Lagged Strategy"
```




```
> wealthv <- cbind(pnls, pnlslag)
> colnames(wealthv) <- c("EWMA", "Lagged")
> # Annualized Sharpe ratios of EWMA strategies
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # Plot both strategies
> dygraphs::dygraph(cumsum(wealthv), main=paste("EWMA Crossover Stra
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

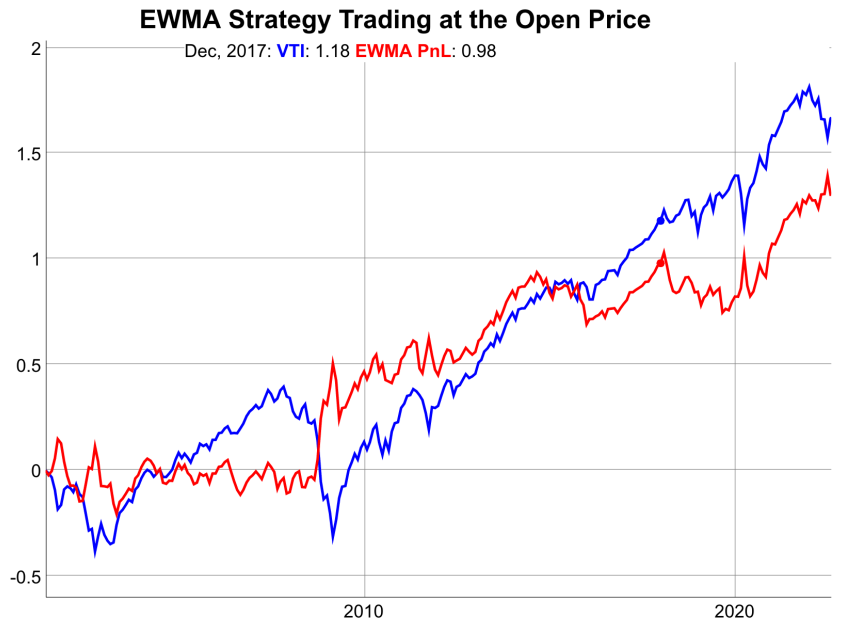
EWMA Strategy Trading at the Open Price

In practice it may not be possible to trade immediately at the *Close* price on the same day that prices cross the *EWMA*.

Then the strategy may trade at the *Open* price on the next day.

The Profit and Loss (*PnL*) on a trade date is the sum of the realized *PnL* from closing the old position, plus the unrealized *PnL* after opening the new position.

```
> # Calculate positions, either: -1, 0, or 1
> indic <- sign(closep - ewmacpp)
> posit <- rutils::lagit(indic, lagg=1)
> # Calculate daily pnl for days without trades
> pnls <- retvti*posit
> # Determine trade dates right after EWMA has crossed prices
> crossd <- which(rutils::diffit(posit) != 0)
> # Calculate realized pnl for days with trades
> openp <- quantmod::Op(ohlc)
> closelag <- rutils::lagit(closep)
> poslag <- rutils::lagit(posit)
> pnls[crossd] <- poslag[crossd]*(openp[crossd] - closelag[crossd])
> # Calculate unrealized pnl for days with trades
> pnls[crossd] <- pnls[crossd] +
+   posit[crossd]*(closep[crossd] - openp[crossd])
> # Calculate the wealth
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "EWMA PnL")
> # Annualized Sharpe ratio of EWMA strategy
> sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```




```
> # Plot dygraph of EWMA strategy wealth
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="EWMA Strategy Trad
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Standard plot of EWMA strategy wealth
> quantmod::chart_Series(cumsum(wealthv)[endd], theme=plot_theme,
+        name="EWMA Strategy Trading at the Open Price")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

EWMA Crossover Strategy With Transaction Costs

The *bid-offer spread* is the percentage difference between the *offer* minus the *bid* price, divided by the *mid* price.

The *bid-offer spread* for liquid stocks can be assumed to be about 10 basis points (bps).

Let n_t be the number of shares of the stock owned at time t, and let p_t be their price.

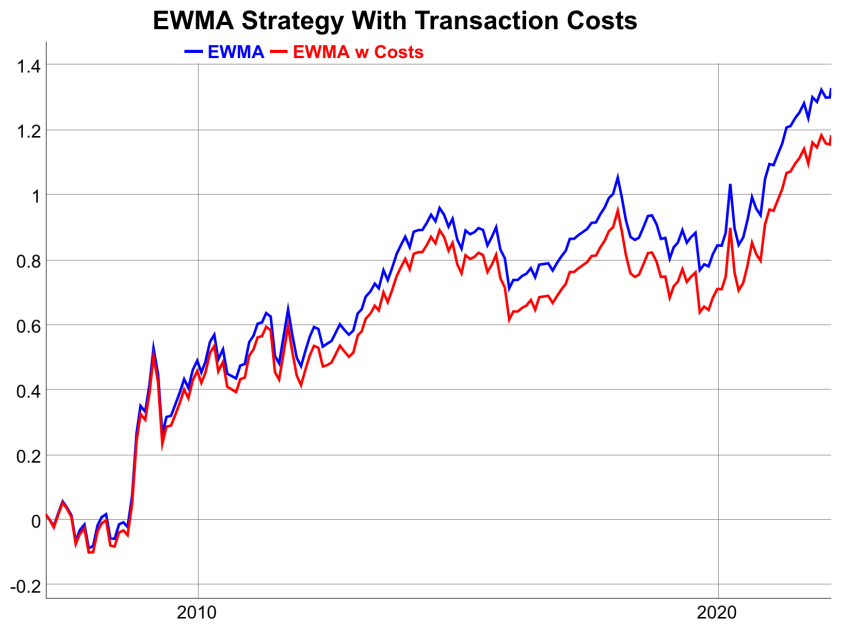
Then the traded dollar amount of the stock is equal to the change in the number of shares times the stock price: $\Delta n_t p_t$.

The the *transaction costs* c^r due to the *bid-offer spread* are equal to half the *bid-offer spread* δ times the absolute value of the traded dollar amount of the stock:

$$c^r = \frac{\delta}{2} |\Delta n_t| \, p_t$$

If d_t is the dollar amount of the stock owned at time t then the *transaction costs* c^r are equal to:

$$c^r = \frac{\delta}{2} |\Delta d_t|$$




```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Calculate transaction costs
> costs <- 0.5*bid_offer*abs(poslag - posit)
> # Plot strategy with transaction costs
> wealthv <- cbind(pnls, pnls - costs)
> colnames(wealthv) <- c("EWMA", "EWMA w Costs")
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="EWMA Strategy With
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Simulation Function for EWMA Crossover Strategy

The *EWMA* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_ewma()` performs a simulation of the *EWMA* strategy, given an *OHLC* time series of pricev, and a decay parameter λ.

The function `sim_ewma()` returns the *EWMA* strategy positions and returns, in a two-column *xts* time series.

```
> sim_ewma <- function(ohlc, lambda=0.9, look_back=333, bid_offer=0.
+                 trend=1, lagg=1) {
+   closep <- quantmod::Cl(ohlc)
+   retvti <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate EWMA prices
+   ewmacpp <- HighFreq::run_mean(closep, lambda=lambda)
+   # Calculate the indicator
+   indic <- trend*sign(closep - ewmacpp)
+   if (lagg > 1) {
+     indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
+     indic[1:lagg] <- 0
+   }  # end if
+   # Calculate positions, either: -1, 0, or 1
+   posit <- rep(NA_integer_, nrows)
+   posit[1] <- 0
+   posit <- ifelse(indic == lagg, 1, posit)
+   posit <- ifelse(indic == (-lagg), -1, posit)
+   posit <- zoo::na.locf(posit, na.rm=FALSE)
+   posit <- xts::xts(posit, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posit <- rutils::lagit(posit, lagg=1)
+   # Calculate PnLs of strategy
+   pnls <- retvti*posit
+   costs <- 0.5*bid_offer*abs(rutils::diffit(posit))
+   pnls <- (pnls - costs)
+   # Calculate strategy returns
+   pnls <- cbind(posit, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_ewma
```

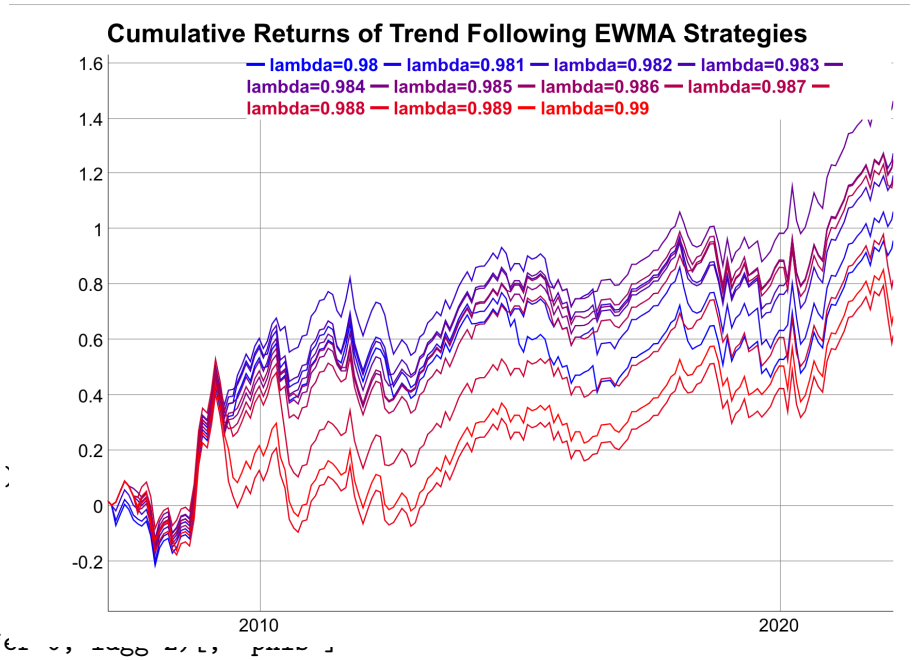
Simulating Multiple Trend Following EWMA Strategies

Multiple *EWMA* strategies can be simulated by calling the function `sim_ewma()` in a loop over a vector of λ parameters.

But `sim_ewma()` returns an *xts* time series, and `sapply()` cannot merge *xts* time series together.

So instead the loop is performed using `lapply()` which returns a list of *xts*, and the list is merged into a single *xts* using the functions `do.call()` and `cbind()`.

```
> source("/Users/jerzy/Develop/lecture_slides/scripts/ewma_model.R")
> lambdav <- seq(from=0.98, to=0.99, by=0.001)
> # Perform lapply() loop over lambdav
> pnltrend <- lapply(lambdav, function(lambda) {
+   # Simulate EWMA strategy and calculate returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, look_back=look_back, bid_offe
+ })  # end lapply
> pnltrend <- do.call(cbind, pnltrend)
> colnames(pnltrend) <- paste0("lambda=", lambdav)
```




```
> # Plot dygraph of multiple EWMA strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnltrend))
> endd <- rutils::calc_endpoints(pnltrend, interval="months")
> dygraphs::dygraph(cumsum(pnltrend)[endd], main="Cumulative Returns
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot EWMA strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(pnltrend), theme=plot_theme,
+   name="Cumulative Returns of EWMA Strategies")
> legend("topleft", legend=colnames(pnltrend), inset=0.1,
+   bg="white", cex=0.8, lwd=rep(6, NCOL(pnltrend)),
+   col=plot_theme$col$line.col, bty="n")
```

Simulating EWMA Strategies Using Parallel Computing

Simulating *EWMA* strategies naturally lends itself to parallel computing, since the simulations are independent from each other.

The function `parLapply()` is similar to `lapply()`, and performs loops under *Windows* using parallel computing on several CPU cores.

The resulting list of time series can then be collapsed into a single *xts* series using the functions `rutils::do_call()` and `cbind()`.

```
> # Initialize compute cluster under Windows
> library(parallel)
> ncores <- detectCores() - 1  # Number of cores
> cluster <- makeCluster(detectCores()-1)
> clusterExport(cluster,
+   varlist=c("ohlc", "look_back", "sim_ewma"))
> # Perform parallel loop over lambdav under Windows
> pnltrend <- parLapply(cluster, lambdav, function(lambda) {
+   library(quantmod)
+   # Simulate EWMA strategy and calculate returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, look_back=look_back)[, "pnls"
+ })  # end parLapply
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel loop over lambdav under Mac-OSX or Linux
> pnltrend <- mclapply(lambdav, function(lambda) {
+   library(quantmod)
+   # Simulate EWMA strategy and calculate returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, look_back=look_back)[, "pnls"
+ }, mc.cores=ncores)  # end mclapply
> pnltrend <- do.call(cbind, pnltrend)
> colnames(pnltrend) <- paste0("lambda=", lambdav)
```

Optimal Decay Parameter of Trend Following EWMA Strategies

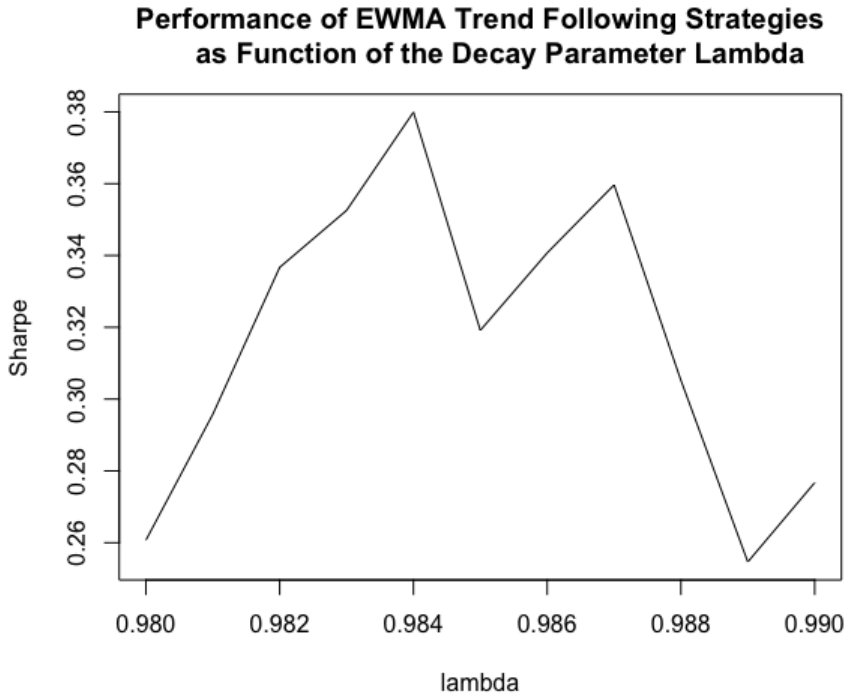
The performance of trend following *EWMA* strategies depends on the λ decay parameter, with smaller λ parameters performing better than larger ones.

The optimal λ parameter applies significant weight to returns 8 - 12 months in the past, which is consistent with research on trend following strategies.

The *Sharpe ratios* of *EWMA* strategies with different λ parameters can be calculated by performing an `sapply()` loop over the *columns* of returns.

`sapply()` treats the columns of *xts* time series as list elements, and loops over the columns.

Performing loops in `R` over the *columns* of returns is acceptable, but `R` loops over the *rows* of returns should be avoided.

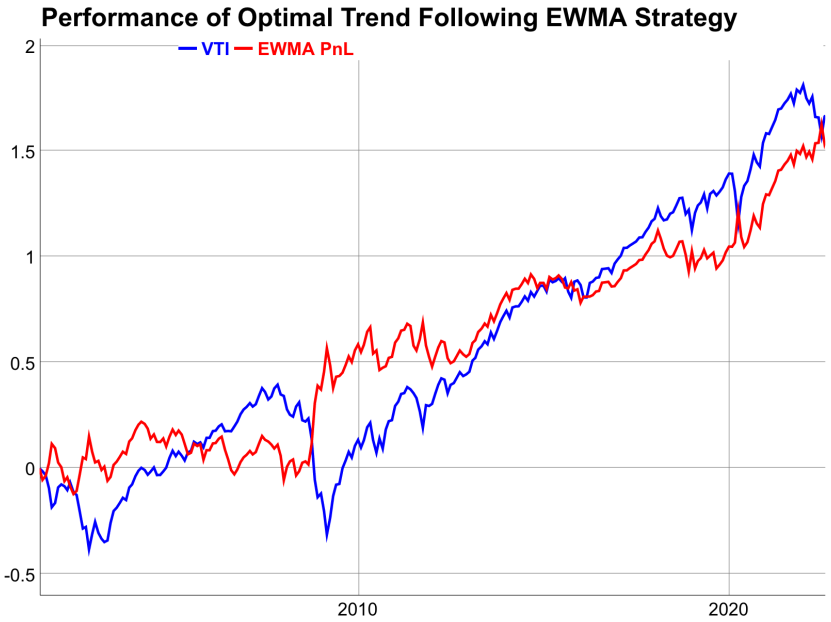



```
> # Calculate annualized Sharpe ratios of strategy returns
> sharpetrend <- sqrt(252)*sapply(pnltrend, function(xtsv) {
+   mean(xtsv)/sd(xtsv)
+ })  # end sapply
> # Plot Sharpe ratios
> dev.new(width=6, height=5, noRStudioGD=TRUE)
> plot(x=lambdav, y=sharpetrend, t="l",
+      xlab="lambda", ylab="Sharpe",
+      main="Performance of EWMA Trend Following Strategies
+      as Function of the Decay Parameter Lambda")
```

Optimal Trend Following EWMA Strategy

The best performing trend following *EWMA* strategy has a relatively small λ parameter, corresponding to slower weight decay (giving more weight to past pricev), and producing less frequent trading.

```
> # Calculate optimal lambda
> lambda <- lambdav[which.max(sharpetrend)]
> # Simulate best performing strategy
> ewmatrend <- sim_ewma(ohlc=ohlc, lambda=lambda, bid_offer=0, lagg=
> posit <- ewmatrend[, "positions"]
> trendopt <- ewmatrend[, "pnls"]
> wealthv <- cbind(retvti, trendopt)
> colnames(wealthv) <- c("VTI", "EWMA PnL")
> # 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])))
> cor(wealthv)[1, 2]
> # Plot dygraph of EWMA strategy wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Performance of Opt
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```



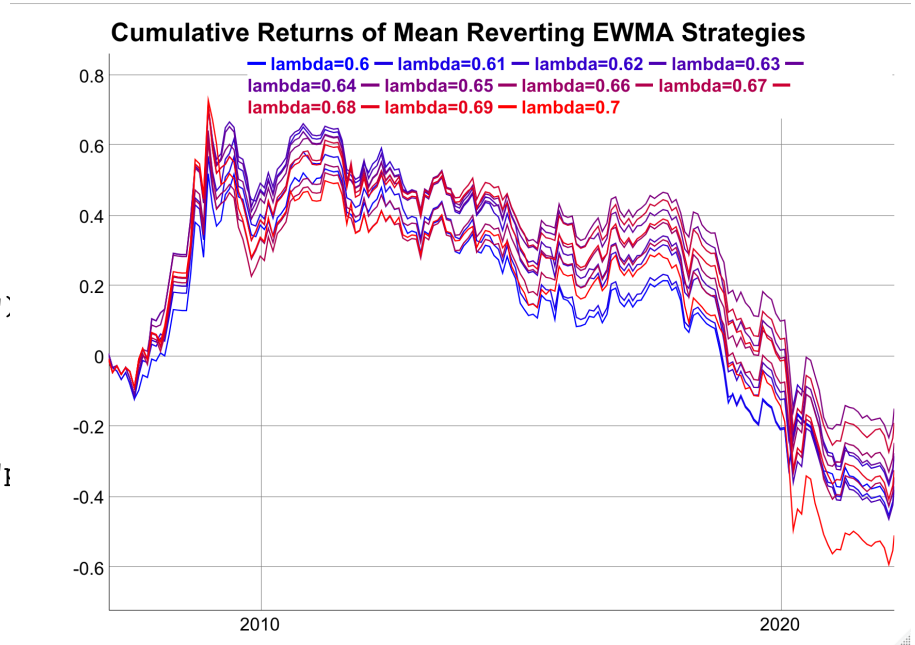

```
> # Plot EWMA PnL with position shading
> # Standard plot of EWMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+        name="Performance of EWMA Strategy")
> add_TA(posit > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posit < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

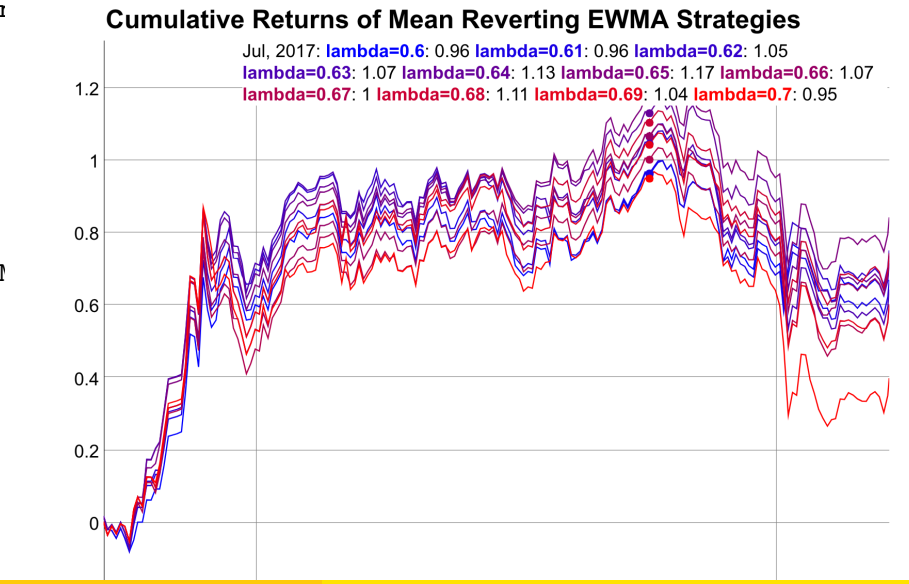
Mean Reverting EWMA Crossover Strategies

Mean reverting EWMA crossover strategies can be simulated using function `sim_ewma()` with argument `trend=(-1)`.

The profitability of mean reverting strategies can be significantly improved by using limit orders, to reduce transaction costs.

```
> source("/Users/jerzy/Develop/lecture_slides/scripts/ewma_model.R")
> lambdav <- seq(0.6, 0.7, 0.01)
> # Perform lapply() loop over lambdav
> pnlrevert <- lapply(lambdav, function(lambda) {
+   # Simulate EWMA strategy and calculate returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, bid_offer=0, trend=(-1))[, "I
+ })  # end lapply
> pnlrevert <- do.call(cbind, pnlrevert)
> colnames(pnlrevert) <- paste0("lambda=", lambdav)
> # Plot dygraph of mean reverting EWMA strategies
> colorv <- colorRampPalette(c("blue", "red"))(NROW(lambdav))
> dygraphs::dygraph(cumsum(pnlrevert)[endd], main="Cumulative Returr
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot EWMA strategies with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pnlrevert,
+   theme=plot_theme, name="Cumulative Returns of Mean Reverting EWI
> legend("topleft", legend=colnames(pnlrevert),
+   inset=0.1, bg="white", cex=0.8, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```





Performance of Mean Reverting EWMA Strategies

The *Sharpe ratios* of *EWMA* strategies with different λ parameters can be calculated by performing an `sapply()` loop over the *columns* of returns.

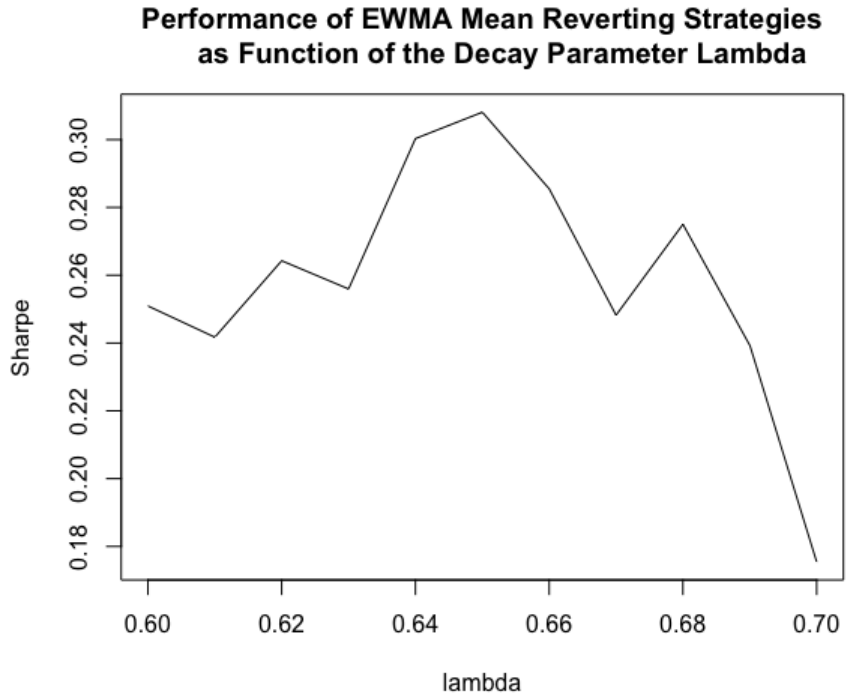
`sapply()` treats the columns of *xts* time series as list elements, and loops over the columns.

Performing loops in `R` over the *columns* of returns is acceptable, but `R` loops over the *rows* of returns should be avoided.

The performance of mean reverting *EWMA* strategies depends on the λ parameter, with performance decreasing for very small or very large λ parameters.

For too large λ parameters, the trading frequency is too high, causing high transaction costs.

For too small λ parameters, the trading frequency is too low, causing the strategy to miss profitable trades.



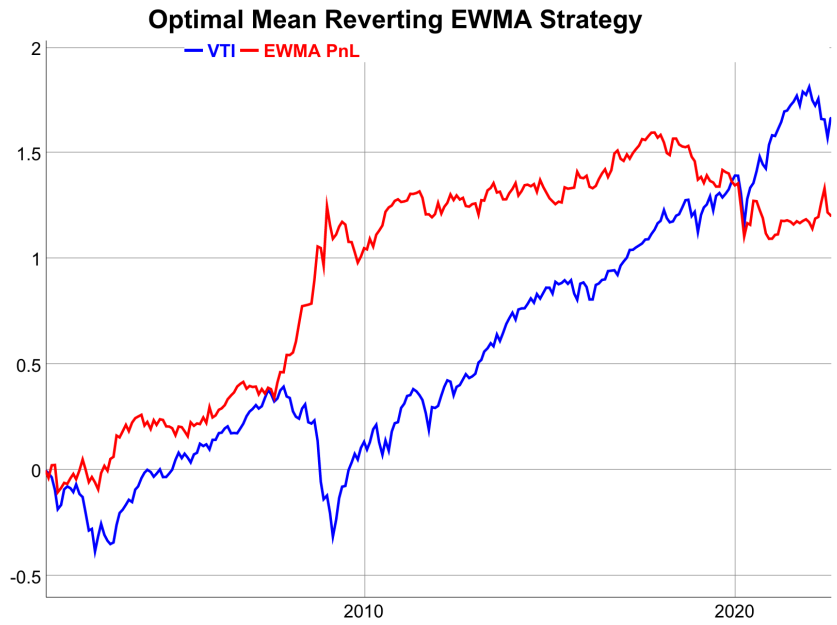

```
> # Calculate Sharpe ratios of strategy returns
> sharperevert <- sqrt(252)*sapply(pnlrevert, function(xtsv) {
+   mean(xtsv)/sd(xtsv)
+ })  # end sapply
> plot(x=lambdav, y=sharperevert, t="l",
+      xlab="lambda", ylab="Sharpe",
+      main="Performance of EWMA Mean Reverting Strategies
+      as Function of the Decay Parameter Lambda")
```

Optimal Mean Reverting EWMA Strategy

Reverting the direction of the trend following *EWMA* strategy creates a mean reverting strategy.

The best performing mean reverting *EWMA* strategy has a relatively large λ parameter, corresponding to faster weight decay (giving more weight to recent pricev), and producing more frequent trading.

But a too large λ parameter also causes very high trading frequency, and high transaction costs.



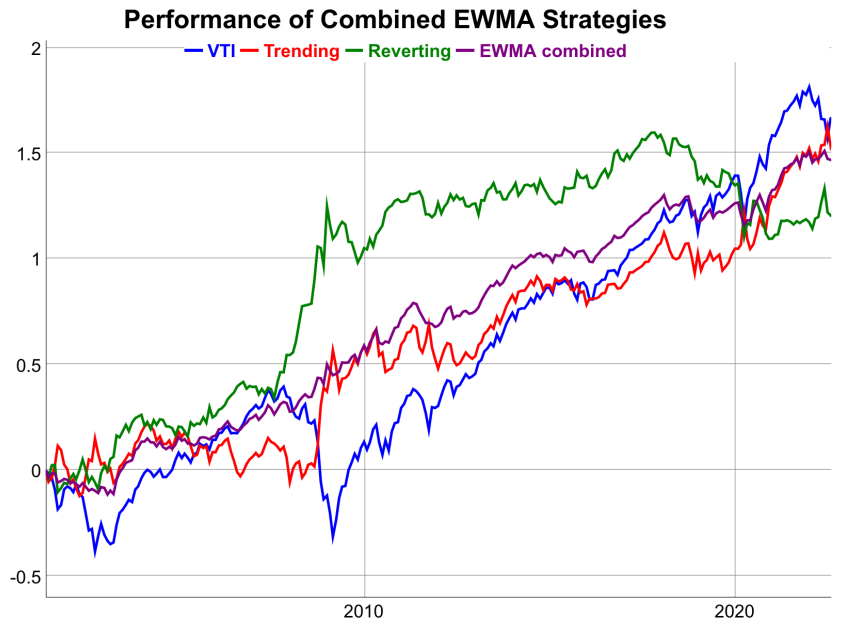

```
> # Calculate optimal lambda
> lambda <- lambdav[which.max(sharperevert)]
> # Simulate best performing strategy
> ewmarevert <- sim_ewma(ohlc=ohlc, bid_offer=0.0,
+   lambda=lambda, trend=(-1))
> posit <- ewmarevert[, "positions"]
> revertopt <- ewmarevert[, "pnls"]
> wealthv <- cbind(retvti, revertopt)
> colnames(wealthv) <- c("VTI", "EWMA PnL")
> # Plot dygraph of EWMA strategy wealth
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Optimal Mean Reve
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

```
> # Standard plot of EWMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+        name="Optimal Mean Reverting EWMA Strategy")
> add_TA(posit > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posit < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

Combining Trend Following and Mean Reverting Strategies

The returns of trend following and mean reverting strategies are usually negatively correlated to each other, so combining them can achieve significant diversification of risk.

The main advantage of EWMA crossover strategies is that they provide positive returns and a diversification of risk with respect to static stock portfolios.




```
> # Calculate correlation between trend following and mean revertin
> trendopt <- ewmatrend[, "pnls"]
> colnames(trendopt) <- "trend"
> revertopt <- ewmarevert[, "pnls"]
> colnames(revertopt) <- "revert"
> cor(cbind(retvti, trendopt, revertopt))
> # Calculate combined strategy
> combstrat <- (retvti + trendopt + revertopt)/3
> colnames(combstrat) <- "combined"
> # Calculate annualized Sharpe ratio of strategy returns
> retp <- cbind(retvti, trendopt, revertopt, combstrat)
> colnames(retp) <- c("VTI", "Trending", "Reverting", "EWMA combin
> sqrt(252)*sapply(retp, function(xtsv) mean(xtsv)/sd(xtsv))
```

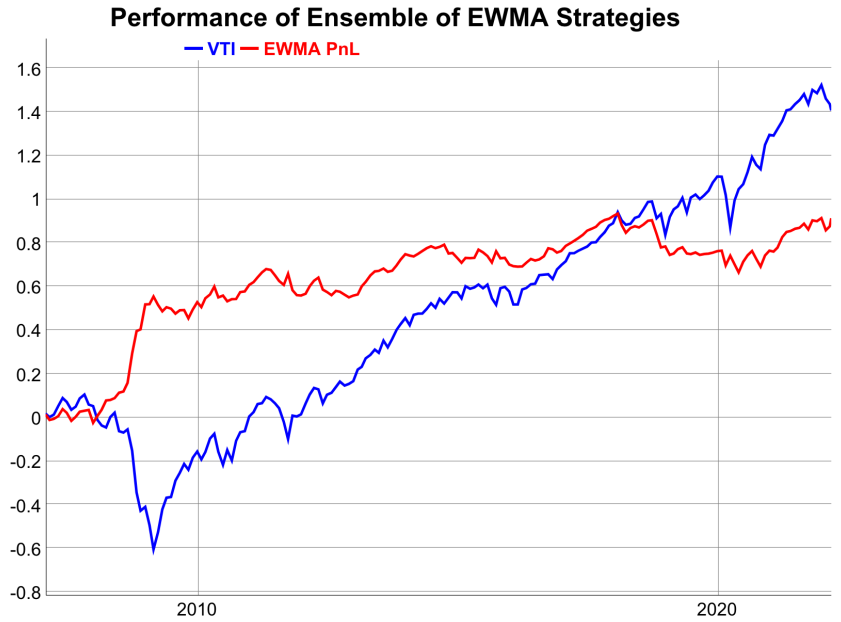
```
> # Plot dygraph of EWMA strategy wealth
> colorv <- c("blue", "red", "green", "purple")
> dygraphs::dygraph(cumsum(retp)[endd], main="Performance of Combine
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Standard plot of EWMA strategy wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pnls, theme=plot_theme,
+        name="Performance of Combined EWMA Strategies")
> legend("topleft", legend=colnames(pnls),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

Ensemble of EWMA Strategies

Instead of selecting the best performing *EWMA* strategy, one can choose a weighted average of strategies (ensemble), which corresponds to allocating positions according to the weights.

The weights can be chosen to be proportional to the Sharpe ratios of the *EWMA* strategies.

```
> # Calculate weights proportional to Sharpe ratios
> weightv <- c(sharpetrend, sharperevert)
> weightv[weightv<0] <- 0
> weightv <- weightv/sum(weightv)
> retp <- cbind(pnltrend, pnlrevert)
> retp <- retp %*% weightv
> retp <- xts::xts(retp, order.by=zoo::index(retp))
> retp <- cbind(retp, retp)
> colnames(retp) <- c("VTI", "EWMA PnL")
> # Plot dygraph of EWMA strategy wealth
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(retp)[endd], main="Performance of Ensemble of EWMA Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Standard plot of EWMA strategy wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(retp), theme=plot_theme,
+        name="Performance of Ensemble of EWMA Strategies")
> legend("topleft", legend=colnames(pnls),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

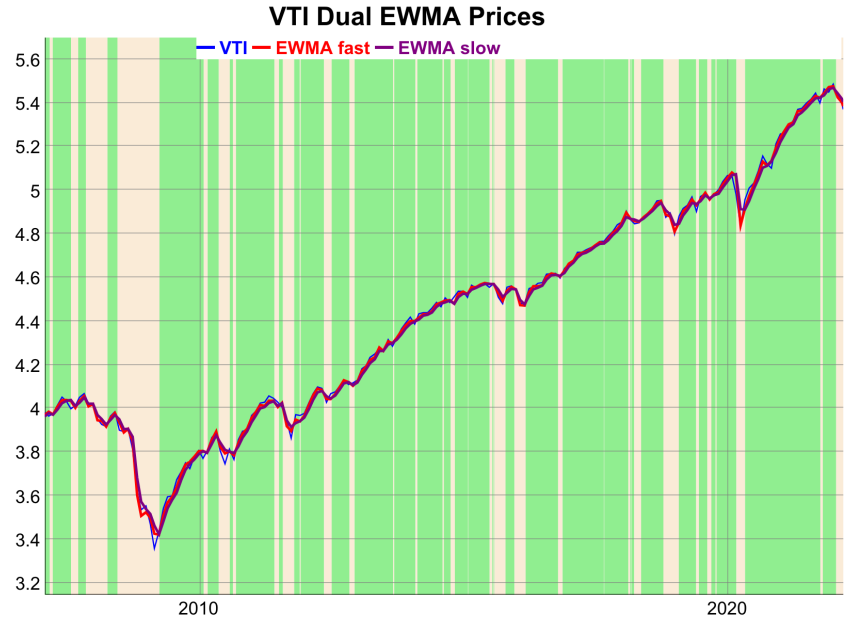


Simulating the Dual EWMA Crossover Strategy

In the *Dual EWMA Crossover* strategy, the risk position depends on the difference between two moving averages.

The risk position flips when the fast moving *EWMA* crosses the slow moving *EWMA*.

```
> # Calculate fast and slow EWMAs
> look_back <- 333
> lambda1 <- 0.89
> lambda2 <- 0.95
> # Calculate EWMA prices
> ewma1 <- HighFreq::run_mean(closep, lambda=lambda1)
> ewma2 <- HighFreq::run_mean(closep, lambda=lambda2)
> # Calculate EWMA prices
> pricev <- cbind(closep, ewma1, ewma2)
> colnames(pricev) <- c("VTI", "EWMA fast", "EWMA slow")
> # Calculate positions, either: -1, 0, or 1
> indic <- sign(ewma1 - ewma2)
> lagg <- 2
> indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
> posit <- rep(NA_integer_, nrows)
> posit[1] <- 0
> posit <- ifelse(indic == lagg, 1, posit)
> posit <- ifelse(indic == (-lagg), -1, posit)
> posit <- zoo::na.locf(posit, na.rm=FALSE)
> posit <- xts::xts(posit, order.by=zoo::index(closep))
> posit <- rutils::lagit(posit, lagg=1)
```



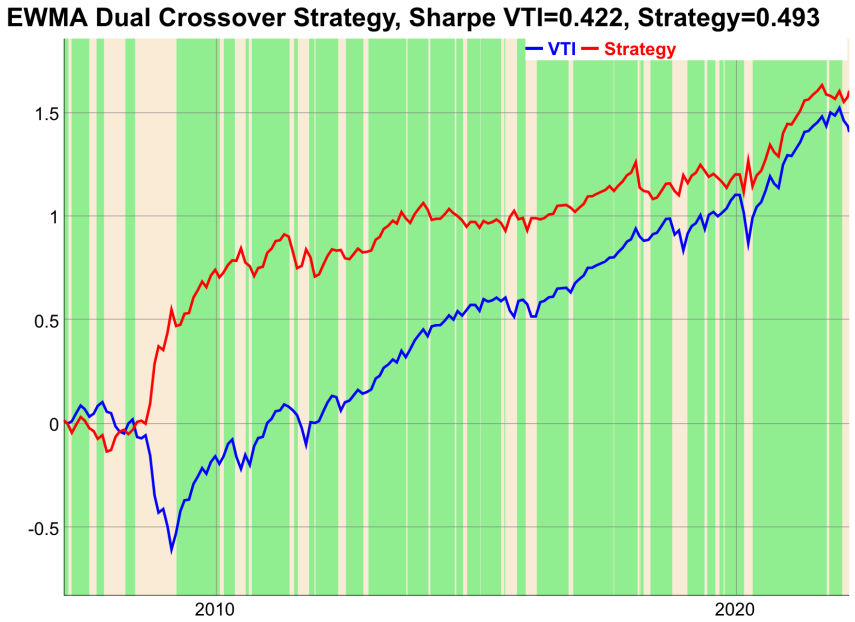

```
> # Create colors for background shading
> crossd <- (rutils::diffit(posit) != 0)
> shadev <- posit[crossd]
> crossd <- c(zoo::index(shadev), end(posit))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen", "
> # Plot dygraph
> colnamev <- colnames(pricev)
> dyplot <- dygraphs::dygraph(pricev[endd], main="VTI Dual EWMA Pric
+   dySeries(name=colnamev[1], label=colnamev[1], strokeWidth=1, col
+   dySeries(name=colnamev[2], label=colnamev[2], strokeWidth=2, col
+   dySeries(name=colnamev[3], label=colnamev[3], strokeWidth=2, col
+   dyLegend(show="always", width=500)
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1],
+ }  # end for
> dyplot
```

Performance of the Dual EWMA Crossover Strategy

The crossover strategy suffers losses when prices are range-bound without a trend, because whenever it switches position the prices soon change direction. (This is called a *"whipsaw"*.)

The crossover strategy performance is worse than the underlying asset (*VTI*), but it has a negative correlation to it, which is very valuable when building a portfolio.

```
> # Calculate daily profits and losses of strategy
> pnls <- retvti*posit
> colnames(pnls) <- "Strategy"
> wealthv <- cbind(retvti, pnls)
> # Annualized Sharpe ratio of Dual EWMA strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```




```
> # Plot Dual EWMA strategy
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main=paste("EWMA Dual
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1],
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Simulation Function for the Dual EWMA Crossover Strategy

The *Dual EWMA* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_ewma2()` performs a simulation of the *Dual EWMA* strategy, given an *OHLC* time series of pricev, and two decay parameters λ_1 and λ_2.

The function `sim_ewma2()` returns the *EWMA* strategy positions and returns, in a two-column *xts* time series.

```
> sim_ewma2 <- function(ohlc, lambda1=0.1, lambda2=0.01, look_back=3
+                 bid_offer=0.001, trend=1, lagg=1) {
+   if (lambda1 >= lambda2) return(NA)
+   closep <- quantmod::Cl(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate EWMA prices
+   ewma1 <- HighFreq::run_mean(closep, lambda=lambda1)
+   ewma2 <- HighFreq::run_mean(closep, lambda=lambda2)
+   # Calculate positions, either: -1, 0, or 1
+   indic <- sign(ewma1 - ewma2)
+   if (lagg > 1) {
+     indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
+     indic[1:lagg] <- 0
+   }  # end if
+   posit <- rep(NA_integer_, nrows)
+   posit[1] <- 0
+   posit <- ifelse(indic == lagg, 1, posit)
+   posit <- ifelse(indic == (-lagg), -1, posit)
+   posit <- zoo::na.locf(posit, na.rm=FALSE)
+   posit <- xts::xts(posit, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posit <- rutils::lagit(posit, lagg=1)
+   # Calculate PnLs of strategy
+   pnls <- retp*posit
+   costs <- 0.5*bid_offer*abs(rutils::diffit(posit))
+   pnls <- (pnls - costs)
+   # Calculate strategy returns
+   pnls <- cbind(posit, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_ewma2
```

Dual EWMA Strategy Performance Matrix

Multiple *Dual EWMA* strategies can be simulated by calling the function `sim_ewma2()` in two loops over the vectors of λ parameters.

The function `outer()` calculates the values of a function over a grid spanned by two variables, and returns a matrix of function values.

The function `Vectorize()` performs an `apply()` loop over the arguments of a function, and returns a vectorized version of the function.

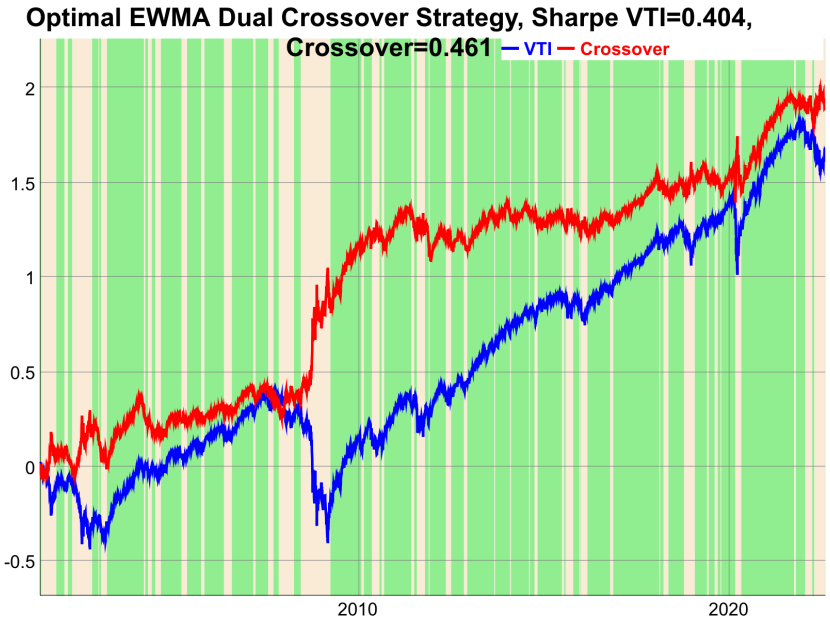
```
> source("/Users/jerzy/Develop/lecture_slides/scripts/ewma_model.R")
> lambdav1 <- seq(from=0.85, to=0.99, by=0.01)
> lambdav2 <- seq(from=0.85, to=0.99, by=0.01)
> # Calculate Sharpe ratio of dual EWMA strategy
> calc_sharpe <- function(ohlc, lambda1, lambda2, look_back, bid_of
+   if (lambda1 >= lambda2) return(NA)
+   pnls <- sim_ewma2(ohlc=ohlc, lambda1=lambda1, lambda2=lambda2,
+     bid_offer=bid_offer, trend=trend, lagg=lagg)[, "pnls"]
+   sqrt(252)*mean(pnls)/sd(pnls)
+ }  # end calc_sharpe
> # Vectorize calc_sharpe with respect to lambda1 and lambda2
> calc_sharpe <- Vectorize(FUN=calc_sharpe,
+   vectorize.args=c("lambda1", "lambda2"))
> # Calculate matrix of PnLs
> sharpem <- outer(lambdav1, lambdav2, FUN=calc_sharpe, ohlc=ohlc,
+            look_back=look_back, bid_offer=0.0, trend=1, lagg=2)
> # Or perform two sapply() loops over lambda vectors
> sharpem <- sapply(lambdav2, function(lambda2) {
+   sapply(lambdav1, function(lambda1) {
+     if (lambda1 >= lambda2) return(NA)
+     calc_sharpe(ohlc=ohlc, lambda1=lambda1, lambda2=lambda2,
+           look_back=look_back, bid_offer=0.0, trend=1, lagg=2)
+   })  # end sapply
+ })  # end sapply
> colnames(sharpem) <- lambdav2
> rownames(sharpem) <- lambdav1
```

Optimal Dual EWMA Strategy

The best *Dual EWMA* strategy performs better than the best *single EWMA* strategy, because it has an extra parameter that can be adjusted to improve in-sample performance.

But this doesn't guarantee better out-of-sample performance.

```
> # Calculate the PnLs for the optimal strategy
> whichv <- which(sharpem == max(sharpem, na.rm=TRUE), arr.ind=TRUE)
> lambda1 <- lambdav1[whichv[1]]
> lambda2 <- lambdav2[whichv[2]]
> ewma_opt <- sim_ewma2(ohlc=ohlc, lambda1=lambda1, lambda2=lambda2,
+   look_back=look_back, bid_offer=0.0, trend=1, lagg=2)
> pnls <- ewma_opt[, "pnls"]
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv)[2] <- "Crossover"
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Annualized Sharpe ratio of Dual EWMA strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```

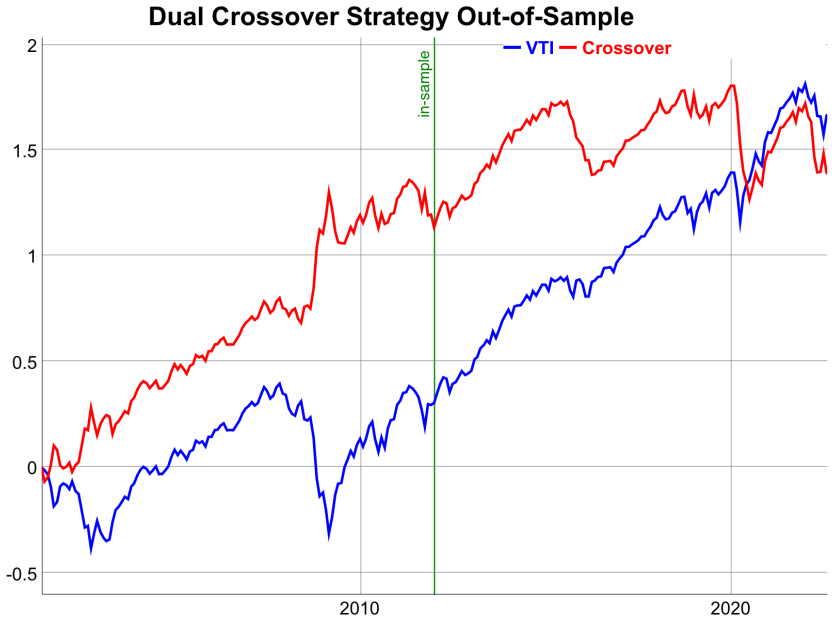



```
> # Create colors for background shading
> posit <- ewma_opt[, "positions"]
> crossd <- (rutils::diffit(posit) != 0)
> shadev <- posit[crossd]
> crossd <- c(zoo::index(shadev), end(posit))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen",
> # Plot Optimal Dual EWMA strategy
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main=paste("Optimal
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1],
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Performance of Dual Crossover Strategy *Out-of-Sample*

In-sample, the best *Dual EWMA* strategy performs better than *VTI*, because it has two parameters that can be adjusted to improve performance.

But out-of-sample, the best *Dual EWMA* strategy performs worse than *VTI*, because it's been *overfitted* in-sample.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate matrix of PnLs
> sharpem <- outer(lambdav1, lambdav2,
+            FUN=calc_sharpe, ohlc=ohlc[insample, ],
+            look_back=look_back, bid_offer=0.0, trend=1, lagg=2)
> colnames(sharpem) <- lambdav2
> rownames(sharpem) <- lambdav1
> # Calculate the PnLs for the optimal strategy
> whichv <- which(sharpem == max(sharpem, na.rm=TRUE), arr.ind=TRU
> lambda1 <- lambdav1[whichv[1]]
> lambda2 <- lambdav2[whichv[2]]
> pnls <- sim_ewma2(ohlc=ohlc, lambda1=lambda1, lambda2=lambda2,
+             look_back=look_back, bid_offer=0.0, trend=1, lagg=2)
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv)[2] <- "Crossover"
> # Calculate Sharpe and Sortino ratios in-sample and out-of-sample
> sqrt(252)*sapply(wealthv[insample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```

```
> # Dygraphs plot with custom line colors
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual Crossover Stra
+   dyEvent(zoo::index(wealthv[last(insample)]), label="in-sample",
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

Volume-Weighted Average Price Indicator

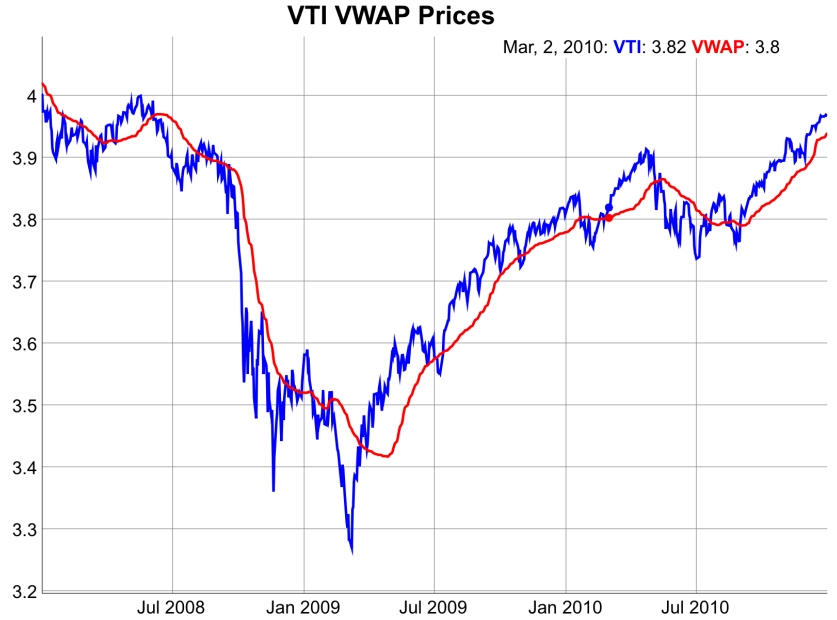
The Volume-Weighted Average Price (*VWAP*) is defined as the sum of prices multiplied by trading volumes, divided by the sum of volumes:

$$p_t^{VWAP} = \frac{\sum_{j=0}^{n} v_{t-j} p_{t-j}}{\sum_{j=0}^{n} v_{t-j}}$$

The *VWAP* applies more weight to prices with higher trading volumes, which allows it to react more quickly to recent market volatility.

The drawback of the *VWAP* indicator is that it applies large weights to prices far in the past.

The *VWAP* is often used as a technical indicator in trend following strategies.




```
> # Calculate log OHLC prices and volumes
> ohlc <- rutils::etfenv$VTI
> closep <- log(quantmod::Cl(ohlc))
> colnames(closep) <- "VTI"
> volum <- quantmod::Vo(ohlc)
> colnames(volum) <- "Volume"
> nrows <- NROW(closep)
> # Calculate the VWAP prices
> look_back <- 21
> vwap <- roll::roll_sum(closep*volum, width=look_back, min_obs=1)
> volumroll <- roll::roll_sum(volum, width=look_back, min_obs=1)
> vwap <- vwap/volumroll
> colnames(vwap) <- "VWAP"
> pricev <- cbind(closep, vwap)
```

```
> # Dygraphs plot with custom line colors
> colorv <- c("blue", "red")
> dygraphs::dygraph(pricev["2009"], main="VTI VWAP Prices") %>%
+   dyOptions(colors=colorv, strokeWidth=2)
> # Plot VWAP prices with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colors
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI VWAP Prices")
> legend("bottomright", legend=colnames(pricev),
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

Recursive VWAP Price Indicator

The *VWAP* prices p^{VWAP} can also be calculated as the ratio of the volume weighted prices μ^{pv} divided by the mean trading volumes μ^{v}:

$$p^{VWAP} = \frac{\mu^{pv}}{\mu^{v}}$$

The volume weighted prices μ^{pv} and the mean trading volumes μ^{v} are both calculated recursively:

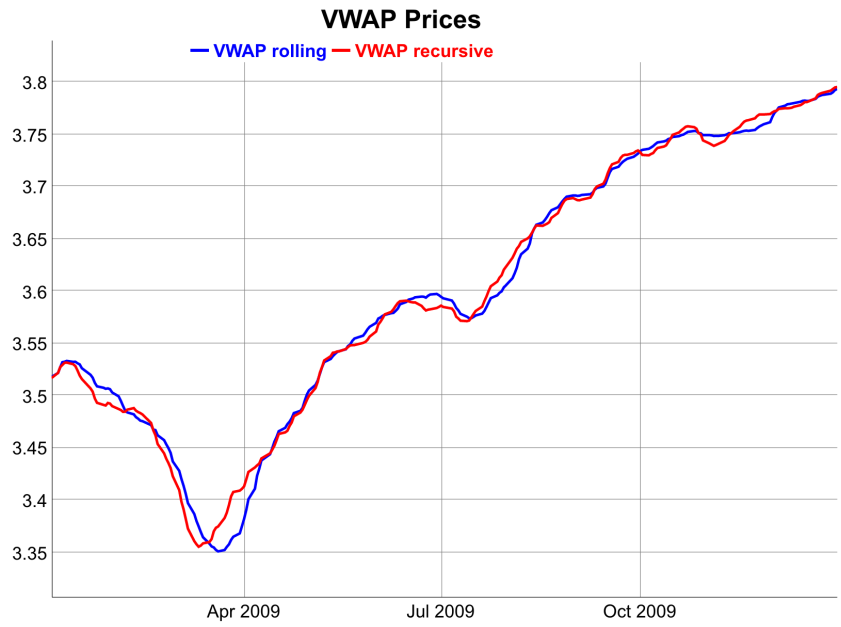
$$\mu^{v}_{t} = \lambda\mu^{v}_{t-1} + (1-\lambda)v_t$$
$$\mu^{pv}_{t} = \lambda\mu^{pv}_{t-1} + (1-\lambda)v_t p_t$$

The recursive *VWAP* prices are slightly different from those calculated as a convolution, because the convolution uses a fixed look-back interval.

The advantage of the recursive *VWAP* indicator is that it gradually "forgets" about large trading volumes far in the past.

The compiled C++ function `stats:::C_rfilter()` calculates the trailing weighted values recursively.

The function `HighFreq::run_mean()` also calculates the trailing weighted values recursively.



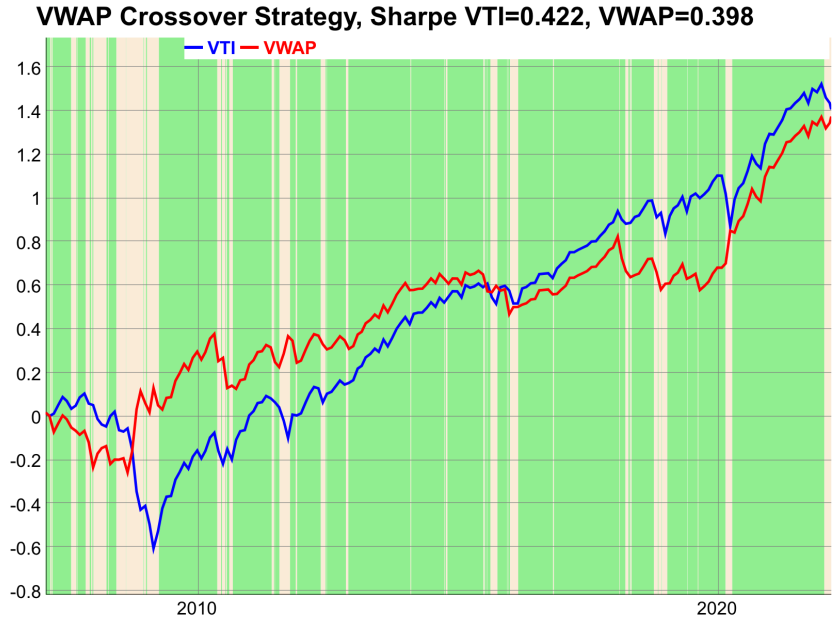

```
> # Calculate VWAP prices recursively using C++ code
> lambda <- 0.9
> volumer <- .Call(stats:::C_rfilter, volum, lambda, c(as.numeric(vo
> pricer <- .Call(stats:::C_rfilter, volum*closep, lambda, c(as.nume
> vwapr <- pricer/volumer
> # Calculate VWAP prices recursively using RcppArmadillo
> vwapcpp <- HighFreq::run_mean(closep, lambda=lambda, weightv=volum
> all.equal(vwapr, drop(vwapcpp))
> # Dygraphs plot the VWAP prices
> pricev <- xts(cbind(vwap, vwapr), zoo::index(ohlc))
> colnames(pricev) <- c("VWAP rolling", "VWAP recursive")
> dygraphs::dygraph(pricev["2009"], main="VWAP Prices") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Simulating the VWAP Crossover Strategy

In the trend following *VWAP Crossover* strategy, the risk position switches depending if the current price is above or below the *VWAP*.

If the current price crosses above the *VWAP*, then the strategy switches its risk position to a fixed unit of long risk, and if it crosses below, to a fixed unit of short risk.

To prevent whipsaws and over-trading, the crossover strategy delays switching positions until the indicator repeats the same value for several periods.




```
> # Calculate positions from lagged indicator
> indic <- sign(closep - vwapcpp)
> lagg <- 2
> indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
> # Calculate positions, either: -1, 0, or 1
> posit <- rep(NA_integer_, nrows)
> posit[1] <- 0
> posit <- ifelse(indic == lagg, 1, posit)
> posit <- ifelse(indic == (-lagg), -1, posit)
> posit <- zoo::na.locf(posit, na.rm=FALSE)
> posit <- xts::xts(posit, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posit <- rutils::lagit(posit, lagg=1)
> # Calculate PnLs of VWAP strategy
> retvti <- rutils::diffit(closep)  # VTI returns
> pnls <- retvti*posit
> colnames(pnls) <- "VWAP"
> wealthv <- cbind(retvti, pnls)
> colnamev <- colnames(wealthv)
> # Annualized Sharpe ratios of VTI and VWAP strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
```

```
> # Create colors for background shading
> crossd <- (rutils::diffit(posit) != 0)
> shadev <- posit[crossd]
> crossd <- c(zoo::index(shadev), end(posit))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen", "
> # Plot dygraph of VWAP strategy
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main=paste("VWAP Cros
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+     dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1],
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Combining VWAP Crossover Strategy with Stocks

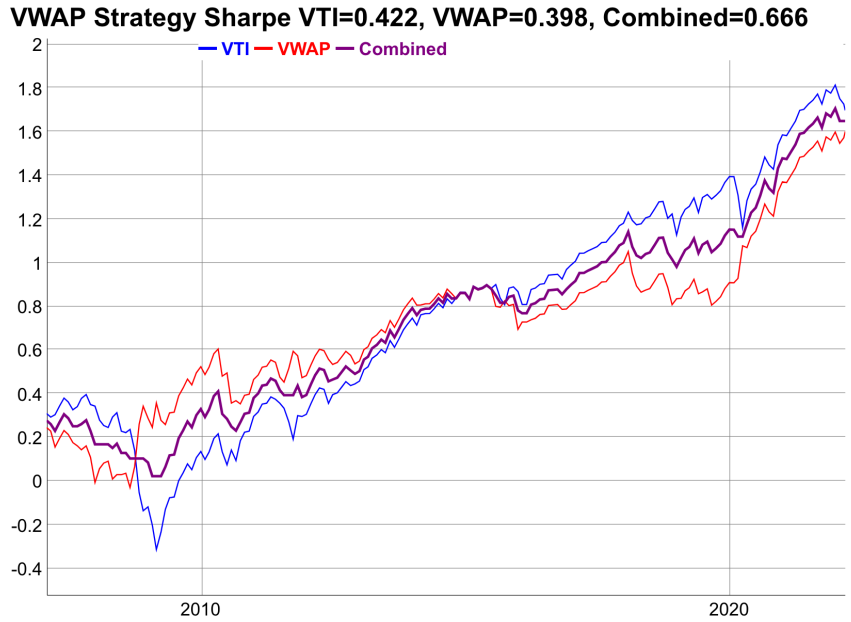
Even though the *VWAP* strategy doesn't perform as well as a static buy-and-hold strategy, it can provide risk reduction when combined with it.

This is because the *VWAP* strategy has a negative correlation with respect to the underlying asset.

In addition, the *VWAP* strategy performs well in periods of extreme market selloffs, so it can provide a hedge for a static buy-and-hold strategy.

The *VWAP* strategy serves as a dynamic put option in periods of extreme market selloffs.

```
> # Calculate correlation of VWAP strategy with VTI
> cor(retvti, pnls)
> # Combine VWAP strategy with VTI
> wealthv <- cbind(retvti, pnls, 0.5*(retvti+pnls))
> colnames(wealthv) <- c("VTI", "VWAP", "Combined")
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
```



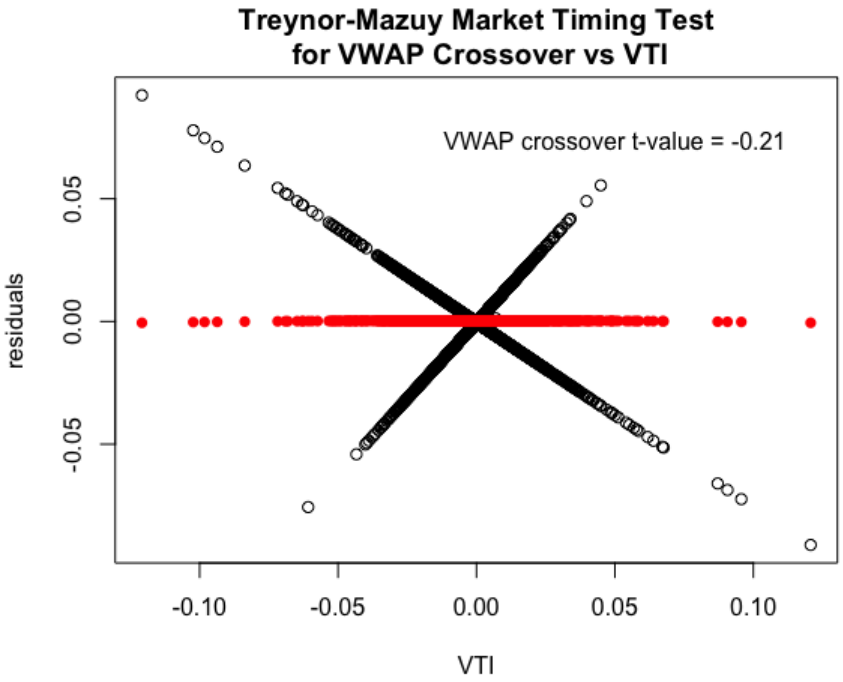

```
> # Plot dygraph of VWAP strategy combined with VTI
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd], paste("VWAP Strategy Shar
+   dySeries(name=colnamev[1], label=colnamev[1], col="blue", stroke
+   dySeries(name=colnamev[2], label=colnamev[2], col="red", strokeW
+   dySeries(name=colnamev[3], label=colnamev[3], col="purple", stro
+   dyLegend(show="always", width=500)
> # Or
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main=paste("VWAP Strategy Sharpe", paste(paste(names(sharper),
+   dyOptions(colors=c("blue", "red", "purple"), strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
```

VWAP Crossover Strategy Market Timing Skill

The VWAP crossover strategy shorts the market during significant selloffs, but otherwise doesn't display market timing skill.

The t-value of the *Treynor-Mazuy* test is negative, but not statistically significant.

```
> # Test VWAP crossover market timing of VTI using Treynor-Mazuy tes
> desv <- cbind(pnls, retvti, retvti^2)
> desv <- na.omit(desv)
> colnames(desv) <- c("VWAP", "VTI", "treynor")
> regmod <- lm(VWAP ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot residual scatterplot
> residv <- (desv$VWAP - regmod$coeff["VTI"]*retvti)
> residv <- regmod$residuals
> # x11(width=6, height=6)
> plot.default(x=retvti, y=residv, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for VWAP Crossover
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fittedv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fittedv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residv), paste("Treynor test t-value =", tvalue))
```



Simulation Function for VWAP Crossover Strategy

The *VWAP* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_vwap()` performs a simulation of the *VWAP* strategy, given an *OHLC* time series of pricev, and the length of the look-back interval (`look_back`).

The function `sim_vwap()` returns the *VWAP* strategy positions and returns, in a two-column *xts* time series.

```
> sim_vwap <- function(ohlc, lambda=0.9, bid_offer=0.001, trend=1, l
+   closep <- log(quantmod::Cl(ohlc))
+   volum <- quantmod::Vo(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate VWAP prices
+   vwap <- HighFreq::run_mean(closep, lambda=lambda, weightv=volum)
+   # Calculate the indicator
+   indic <- trend*sign(closep - vwap)
+   if (lagg > 1) {
+     indic <- roll::roll_sum(indic, width=lagg, min_obs=1)
+     indic[1:lagg] <- 0
+   }  # end if
+   # Calculate positions, either: -1, 0, or 1
+   posit <- rep(NA_integer_, nrows)
+   posit[1] <- 0
+   posit <- ifelse(indic == lagg, 1, posit)
+   posit <- ifelse(indic == (-lagg), -1, posit)
+   posit <- zoo::na.locf(posit, na.rm=FALSE)
+   posit <- xts::xts(posit, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posit <- rutils::lagit(posit, lagg=1)
+   # Calculate PnLs of strategy
+   pnls <- retp*posit
+   costs <- 0.5*bid_offer*abs(rutils::diffit(posit))
+   pnls <- (pnls - costs)
+   # Calculate strategy returns
+   pnls <- cbind(posit, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_vwap
```

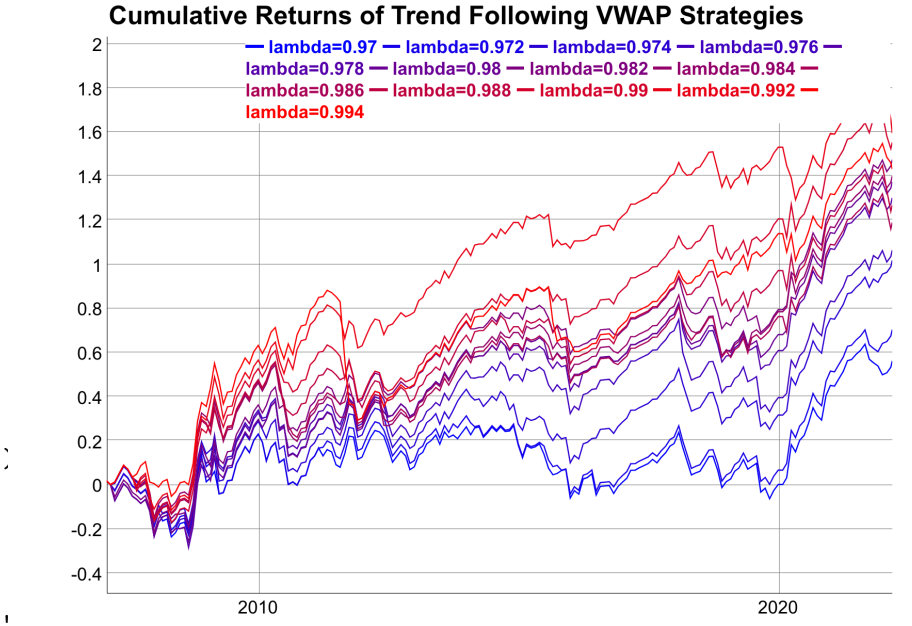
Simulating Multiple Trend Following VWAP Strategies

Multiple *VWAP* strategies can be simulated by calling the function `sim_vwap()` in a loop over a vector of λ parameters.

But `sim_vwap()` returns an *xts* time series, and `sapply()` cannot merge *xts* time series together.

So instead the loop is performed using `lapply()` which returns a list of *xts*, and the list is merged into a single *xts* using the functions `do.call()` and `cbind()`.

```
> source("/Users/jerzy/Develop/lecture_slides/scripts/ewma_model.R")
> lambdav <- seq(from=0.97, to=0.995, by=0.002)
> # Perform lapply() loop over lambdav
> pnls <- lapply(lambdav, function(lambda) {
+   # Simulate VWAP strategy and calculate returns
+   sim_vwap(ohlc=ohlc, lambda=lambda, bid_offer=0, lagg=2)[, "pnls"]
+ })  # end lapply
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
```




```
> # Plot dygraph of multiple VWAP strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd], main="Cumulative Returns of
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot VWAP strategies with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(pnls), theme=plot_theme,
+   name="Cumulative Returns of VWAP Strategies")
> legend("topleft", legend=colnames(pnls), inset=0.1,
+   bg="white", cex=0.8, lwd=rep(6, NCOL(pnls)),
+   col=plot_theme$col$line.col, bty="n")
```

Centered Price Z-scores

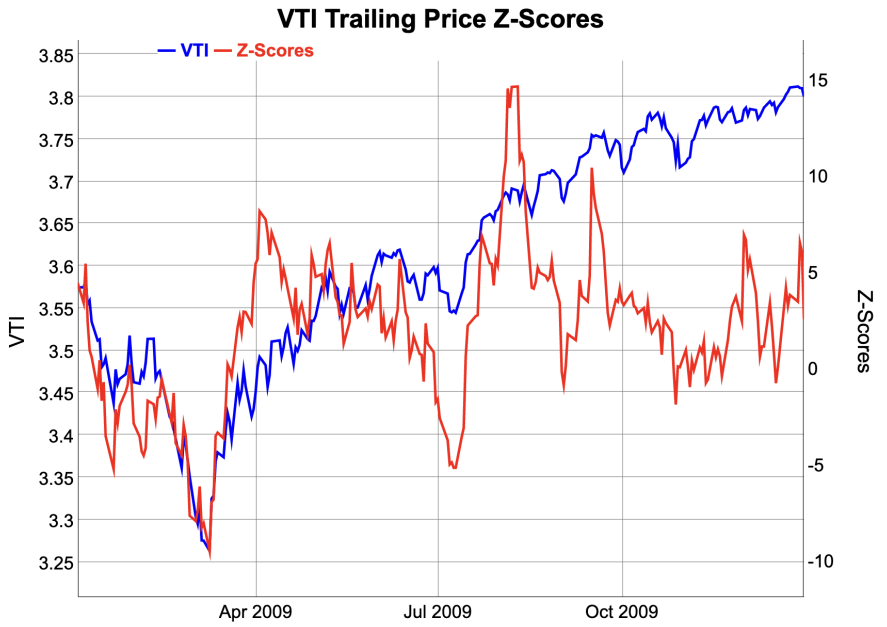
An extreme local price is a price which differs significantly from neighboring prices.

Extreme prices can be identified in-sample using the centered *price z-score* equal to the price difference with neighboring prices divided by the volatility of returns σ_i:

$$z_i = \frac{2p_i - p_{i-k} - p_{i+k}}{\sigma_i}$$

Where p_{i-k} and p_{i+k} are the lagged and advanced prices.

The lag parameter k determines the scale of the extreme local pricev, with smaller k producing larger z-scores for more local price extremes.




```
> # Extract VTI log OHLC prices
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retvti <- rutils::diffit(closep)
> # Calculate the centered volatility
> look_back <- 7
> half_back <- look_back %/% 2
> stdev <- roll::roll_sd(retvti, width=look_back, min_obs=1)
> stdev <- rutils::lagit(stdev, lagg=(-half_back))
> # Calculate the z-scores of prices
> pricez <- (2*closep -
+   rutils::lagit(closep, half_back, pad_zeros=FALSE) -
+   rutils::lagit(closep, -half_back, pad_zeros=FALSE))
> pricez <- ifelse(stdev > 0, pricez/stdev, 0)
```

```
> # Plot dygraph of z-scores of VTI prices
> pricev <- cbind(closep, pricez)
> colnames(pricev) <- c("VTI", "Z-scores")
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2009"], main="VTI Price Z-Scores") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
```

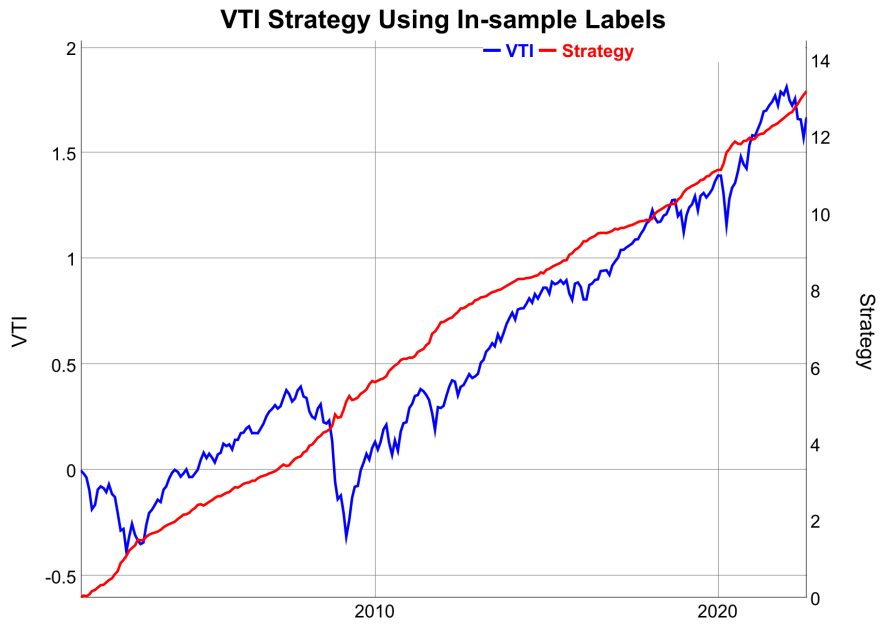
Labeling the Tops and Bottoms of Prices

The local tops and bottoms of prices can be labeled approximately in-sample using the z-scores of prices and threshold values.

The local tops of prices represent *overbought* conditions, while the bottoms represent *oversold* conditions.

The labeled data can be used as a response or target variable in machine learning classifier models.

But it's not feasible to classify the prices out-of-sample exactly according to their in-sample labels.




```
> # Calculate thresholds for labeling tops and bottoms
> confl <- c(0.2, 0.8)
> threshv <- quantile(pricez, confl)
> # Calculate the vectors of tops and bottoms
> tops <- zoo::coredata(pricez > threshv[2])
> colnames(tops) <- "tops"
> bottoms <- zoo::coredata(pricez < threshv[1])
> colnames(bottoms) <- "bottoms"
> # Simulate in-sample VTI strategy
> posit <- rep(NA_integer_, nrows)
> posit[1] <- 0
> posit[tops] <- (-1)
> posit[bottoms] <- 1
> posit <- zoo::na.locf(posit)
> posit <- rutils::lagit(posit)
> pnls <- retvti*posit
```

```
> # Plot dygraph of in-sample VTI strategy
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Strategy Using In-sample Labels") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="Strategy", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", label="VTI", strokeWidth=2, col="
+   dySeries(name="Strategy", axis="y2", label="Strategy", strokeWi
```

Predictors of Price Extremes

The return volatility and trading volumes may be used as predictors in a classification model, in order to identify *overbought* and *oversold* conditions.

The trailing *volume z-score* is equal to the volume v_i minus the trailing average volumes $\bar{v}_i$ divided by the volatility of the volumes σ_i:

$$z_i = \frac{v_i - \bar{v}_i}{\sigma_i}$$

Trading volumes are typically higher when prices drop and they are also positively correlated with the return volatility.

The *volatility z-score* is equal to the spot volatility v_i minus the trailing average volatility $\bar{v}_i$ divided by the standard deviation of the volatility σ_i:

$$z_i = \frac{v_i - \bar{v}_i}{\sigma_i}$$

Volatility is typically higher when prices drop and it's also positively correlated with the trading volumes.

```
> # Calculate volatility z-scores
> volat <- HighFreq::roll_var_ohlc(ohlc=ohlc, look_back=look_back, s
> volatm <- roll::roll_mean(volat, width=look_back, min_obs=1)
> volatsd <- roll::roll_sd(rutils::diffit(volat), width=look_back, m
> volatsd[1] <- 0
> volatz <- ifelse(volatsd > 0, (volat - volatm)/volatsd, 0)
> colnames(volatz) <- "volat"
> # Calculate volume z-scores
> volum <- quantmod::Vo(ohlc)
> volumean <- roll::roll_mean(volum, width=look_back, min_obs=1)
> volumsd <- roll::roll_sd(rutils::diffit(volum), width=look_back, m
> volumsd[1] <- 0
> volumz <- ifelse(volumsd > 0, (volum - volumean)/volumsd, 0)
> colnames(volumz) <- "volume"
```

Regression Z-Scores

The trailing *z-score* z_i of a price p_i can be defined as the *standardized residual* of the linear regression with respect to time t_i or some other variable:

$$z_i = \frac{p_i - (\alpha + \beta t_i)}{\sigma_i}$$

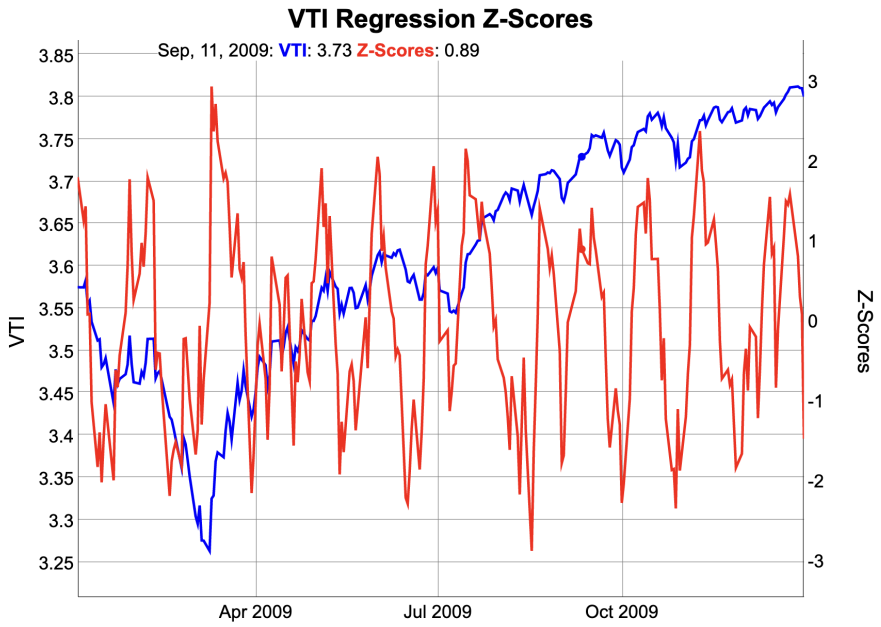
Where α and β are the *regression coefficients*, and σ_i is the standard deviation of the residuals.

The regression *z-scores* can be used as rich or cheap indicators, either relative to past pricev, or relative to prices in a stock pair.

The regression residuals must be calculated in a loop, so it's much faster to calculate them using functions written in C++ code.

The function `HighFreq::roll_zscores()` calculates the residuals of a rolling regression.

```
> # Calculate trailing price regression z-scores
> datev <- matrix(zoo::index(closep))
> look_back <- 21
> regz <- drop(HighFreq::roll_zscores(respv=closep, predv=datev, l
> regz[1:look_back] <- 0
```




```
> # Plot dygraph of z-scores of VTI prices
> pricev <- cbind(closep, regz)
> colnames(pricev) <- c("VTI", "Z-scores")
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2009"], main="VTI Price Z-Scores") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
```

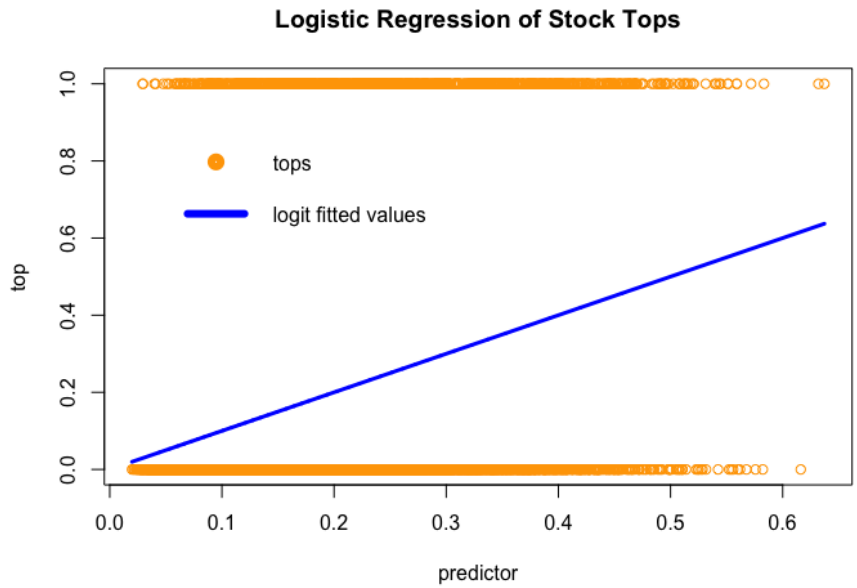
Forecasting Stock Price Tops and Bottoms Using Logistic Regression

The weighted average of the volatility, trading volume, and regression z-scores can be used to forecast a stock top (overbought condition) or a bottom (oversold condition).

The residuals are the differences between the actual response values (0 and 1), and the calculated probabilities of default.

The residuals are not normally distributed, so the data is fitted using the *maximum likelihood* method, instead of least squares.

```
> # Define design matrix for tops including intercept column
> predv <- cbind(volatz, volumz, regz)
> predv[1, ] <- 0
> predv <- rutils::lagit(predv)
> # Fit in-sample logistic regression for tops
> logmod <- glm(tops ~ predv, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcast <- drop(cbind(rep(1, nrows), predv) %*% coeff)
> ordern <- order(fcast)
> # Calculate in-sample forecasts from logistic regression model
> fcast <- 1/(1+exp(-fcast))
> all.equal(logmod$fitted.values, fcast, check.attributes=FALSE)
> hist(fcast)
```




```
> plot(x=fcast[ordern], y=tops[ordern],
+      main="Logistic Regression of Stock Tops",
+      col="orange", xlab="predictor", ylab="top")
> lines(x=fcast[ordern], y=logmod$fitted.values[ordern], col="blue"
> legend(x="topleft", inset=0.1, bty="n", lwd=6,
+  legend=c("tops", "logit fitted values"), y.intersp=0.5,
+  col=c("orange", "blue"), lty=c(NA, 1), pch=c(1, NA))
```

Forecasting Errors of Stock Tops and Bottoms

A *binary classification model* categorizes cases based on its forecasts whether the *null hypothesis* is `TRUE` or `FALSE`.

Let the *null hypothesis* be that the data point is not a top: `tops = FALSE`.

A *positive* result corresponds to rejecting the null hypothesis (`tops = TRUE`), while a *negative* result corresponds to accepting the null hypothesis (`tops = FALSE`).

The forecasts are subject to two different types of errors: *type I* and *type II* errors.

A *type I* error is the incorrect rejection of a `TRUE` *null hypothesis* (i.e. a "false positive"), when `tops = FALSE` but it's classified as `tops = TRUE`.

A *type II* error is the incorrect acceptance of a `FALSE` *null hypothesis* (i.e. a "false negative"), when `tops = TRUE` but it's classified as `tops = FALSE`.

```
> # Define discrimination threshold value
> threshold <- quantile(fcast, confl[2])
> # Calculate confusion matrix in-sample
> confmat <- table(actual=!tops, forecast=(fcast < threshold))
> confmat
> # Calculate FALSE positive (type I error)
> sum(tops & (fcast < threshold))
> # Calculate FALSE negative (type II error)
> sum(!tops & (fcast > threshold))
```

The Confusion Matrix of a Binary Classification Model

The confusion matrix summarizes the performance of a classification model on a set of test data for which the actual values of the *null hypothesis* are known.

		Forecast	
		Null is FALSE	**Null is TRUE**
Actual			
Null is FALSE		True Positive (sensitivity)	False Negative (type II error)
Null is TRUE		False Positive (type I error)	True Negative (specificity)

```
> # Calculate FALSE positive and FALSE negative rates
> confmat <- confmat / rowSums(confmat)
> c(typeI=confmat[2, 1], typeII=confmat[1, 2])
```

Let the *null hypothesis* be that the data point is not a top: `tops = FALSE`.

The *true positive* rate (known as the *sensitivity*) is the fraction of `FALSE` *null hypothesis* cases that are correctly classified as `FALSE`.

The *false negative* rate is the fraction of `FALSE` *null hypothesis* cases that are incorrectly classified as `TRUE` (*type II* error).

The sum of the *true positive* plus the *false negative* rate is equal to 1.

The *true negative* rate (known as the *specificity*) is the fraction of `TRUE` *null hypothesis* cases that are correctly classified as `TRUE`.

The *false positive* rate is the fraction of `TRUE` *null hypothesis* cases that are incorrectly classified as `FALSE` (*type I* error).

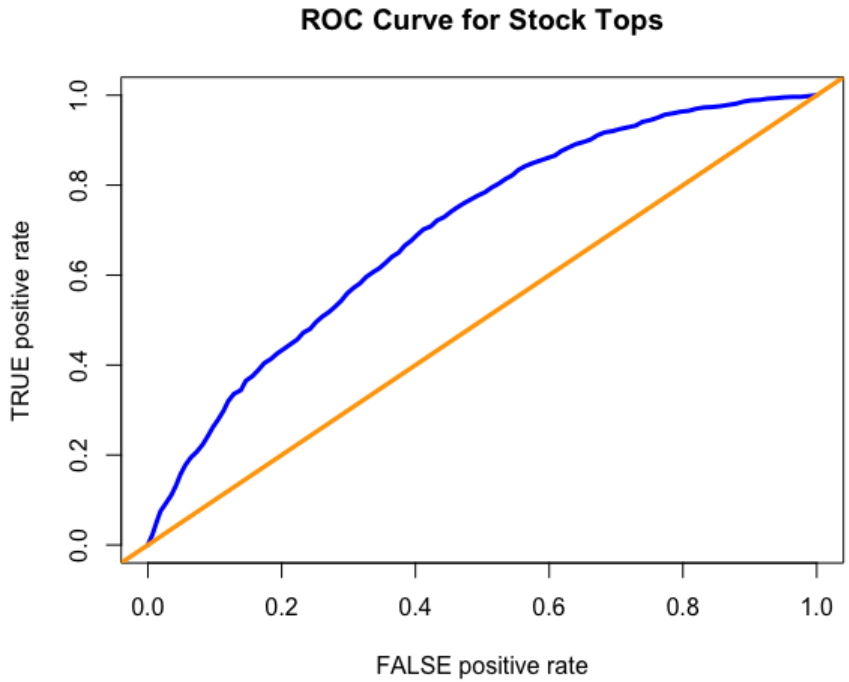
The sum of the *true negative* plus the *false positive* rate is equal to 1.

Receiver Operating Characteristic (ROC) Curve for Stock Tops

The *ROC curve* is the plot of the *true positive* rate, as a function of the *false positive* rate, and illustrates the performance of a binary classifier.

The area under the *ROC curve* (AUC) measures the classification ability of a binary classifier.

The *informedness* is equal to the sum of the sensitivity plus the specificity, and measures the performance of a binary classification model.




```
> # Confusion matrix as function of threshold
> confun <- function(actual, fcast, threshold) {
+   forb <- (fcast < threshold)
+   conf <- matrix(c(sum(!actual & !forb), sum(actual & !forb),
+              sum(!actual & forb), sum(actual & forb)), ncol=2)
+   conf <- conf / rowSums(conf)
+   c(typeI=conf[2, 1], typeII=conf[1, 2])
+ }  # end confun
> confun(!tops, fcast, threshold=threshold)
> # Define vector of discrimination thresholds
> threshv <- quantile(fcast, seq(0.01, 0.99, by=0.01))
> # Calculate error rates
> error_rates <- sapply(threshv, confun,
+   actual=!tops, fcast=fcast)  # end sapply
> error_rates <- t(error_rates)
> rownames(error_rates) <- threshv
> # Calculate the informedness
> informv <- 2 - rowSums(error_rates)
> plot(threshv, informv, t="l", main="Informedness")
> # Find the threshold corresponding to highest informedness
> threshm <- threshv[which.max(informv)]
> forecastops <- (fcast > threshm)
```

```
> # Calculate area under ROC curve (AUC)
> error_rates <- rbind(c(1, 0), error_rates)
> error_rates <- rbind(error_rates, c(0, 1))
> truepos <- (1 - error_rates[, "typeII"])
> truepos <- (truepos + rutils::lagit(truepos))/2
> falsepos <- rutils::diffit(error_rates[, "typeI"])
> abs(sum(truepos*falsepos))
> # Plot ROC Curve for stock tops
> plot(x=error_rates[, "typeI"], y=1-error_rates[, "typeII"],
+      xlab="FALSE positive rate", ylab="TRUE positive rate",
+      main="ROC Curve for Stock Tops", type="l", lwd=3, col="blue")
> abline(a=0.0, b=1.0, lwd=3, col="orange")
```

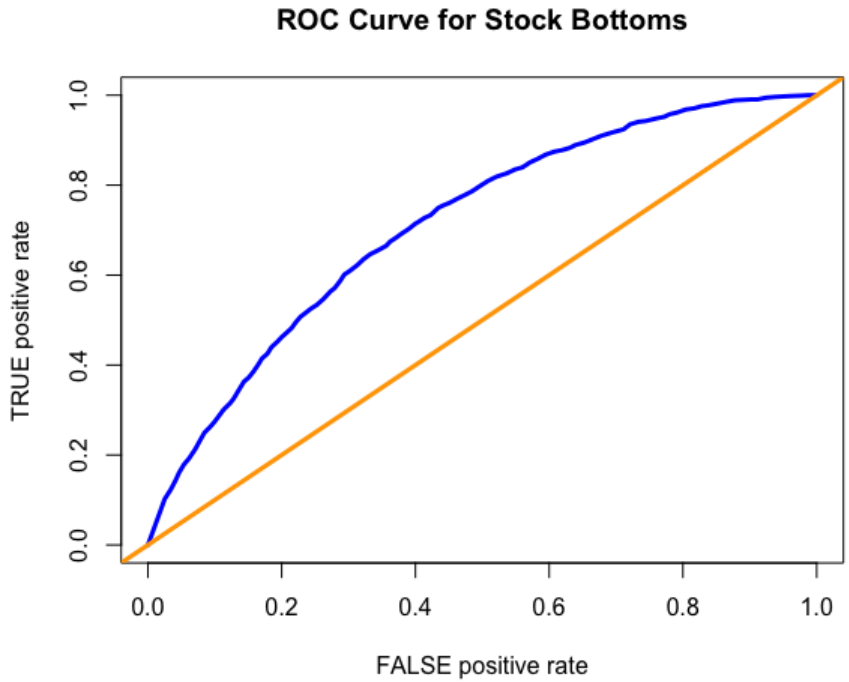
Receiver Operating Characteristic (ROC) Curve for Stock Bottoms

The *ROC curve* is the plot of the *true positive* rate, as a function of the *false positive* rate, and illustrates the performance of a binary classifier.

The area under the *ROC curve* (AUC) measures the classification ability of a binary classifier.

The *informedness* is equal to the sum of the sensitivity plus the specificity, and measures the performance of a binary classification model.

```
> # Fit in-sample logistic regression for bottoms
> logmod <- glm(bottoms ~ predv, family=binomial(logit))
> summary(logmod)
> # Calculate in-sample forecast from logistic regression model
> coeff <- logmod$coefficients
> fcast <- drop(cbind(rep(1, nrows), predv) %*% coeff)
> fcast <- 1/(1+exp(-fcast))
> # Calculate error rates
> error_rates <- sapply(threshv, confun,
+   actual=!bottoms, fcast=fcast)  # end sapply
> error_rates <- t(error_rates)
> rownames(error_rates) <- threshv
> # Calculate the informedness
> informv <- 2 - rowSums(error_rates)
> plot(threshv, informv, t="l", main="Informedness")
> # Find the threshold corresponding to highest informedness
> threshm <- threshv[which.max(informv)]
> forecastbot <- (fcast > threshm)
```




```
> # Calculate area under ROC curve (AUC)
> error_rates <- rbind(c(1, 0), error_rates)
> error_rates <- rbind(error_rates, c(0, 1))
> truepos <- (1 - error_rates[, "typeII"])
> truepos <- (truepos + rutils::lagit(truepos))/2
> falsepos <- rutils::diffit(error_rates[, "typeI"])
> abs(sum(truepos*falsepos))
> # Plot ROC Curve for stock tops
> plot(x=error_rates[, "typeI"], y=1-error_rates[, "typeII"],
+      xlab="FALSE positive rate", ylab="TRUE positive rate",
+      main="ROC Curve for Stock Bottoms", type="l", lwd=3, col="blu
> abline(a=0.0, b=1.0, lwd=3, col="orange")
```

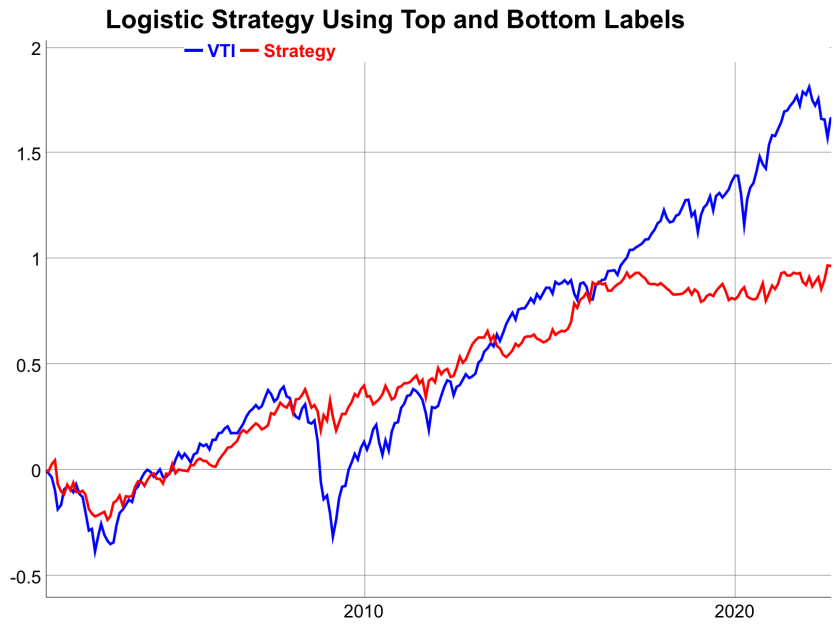
Logistic Tops and Bottoms Strategy In-sample

The logistic strategy forecasts the tops and bottoms of pricev, using a logistic regression model with the volatility and trading volumes as predictors.

Averaging the forecasts over time improves strategy performance because of the bias-variance tradeoff.

It makes sense to average the forecasts over time because they are forecasts for future time intervals, not just a single point in time.

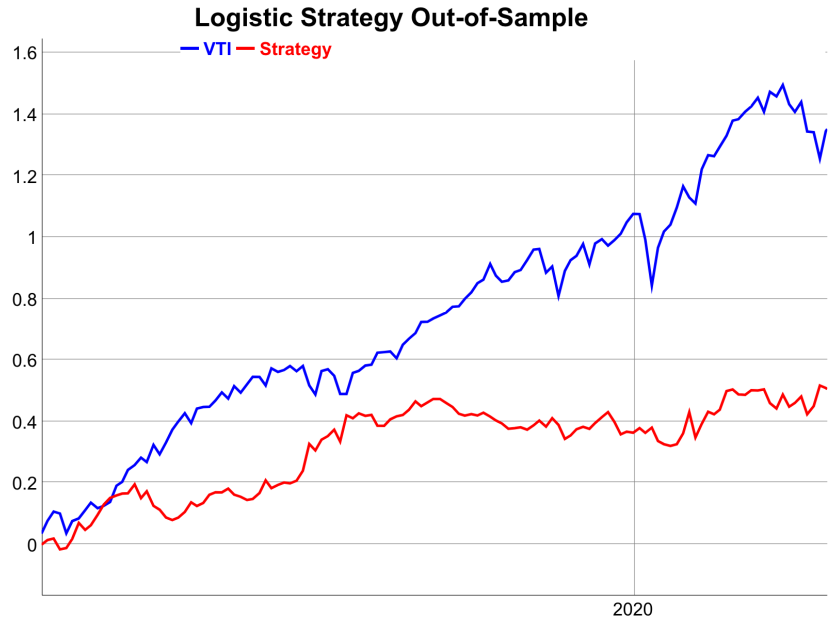
```
> # Average the signals over time
> topsav <- HighFreq::roll_sum(matrix(forecastops), 5)/5
> botsav <- HighFreq::roll_sum(matrix(forecastbot), 5)/5
> # Simulate in-sample VTI strategy
> posit <- (botsav-topsav)
> # Standard strategy
> # posit <- rep(NA_integer_, NROW(retvti))
> # posit[1] <- 0
> # posit[forecastops] <- (-1)
> # posit[forecastbot] <- 1
> # posit <- zoo::na.locf(posit)
> posit <- rutils::lagit(posit)
> pnls <- retvti*posit
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Using Top and Bottom Labels") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Logistic Tops and Bottoms Strategy Out-of-Sample

The logistic strategy forecasts the tops and bottoms of pricev, using a logistic regression model with the volatility and trading volumes as predictors.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Fit in-sample logistic regression for tops
> logmod <- glm(tops[insample] ~ predv[insample, ], family=binomial(
> fittedv <- logmod$fitted.values
> coefftop <- logmod$coefficients
> # Calculate error rates and best threshold value
> error_rates <- sapply(threshv, confun,
+   actual=!tops[insample], fcast=fittedv)  # end sapply
> error_rates <- t(error_rates)
> informv <- 2 - rowSums(error_rates)
> threshtop <- threshv[which.max(informv)]
> # Fit in-sample logistic regression for bottoms
> logmod <- glm(bottoms[insample] ~ predv[insample, ], family=binor
> fittedv <- logmod$fitted.values
> coeffbot <- logmod$coefficients
> # Calculate error rates and best threshold value
> error_rates <- sapply(threshv, confun,
+   actual=!bottoms[insample], fcast=fittedv)  # end sapply
> error_rates <- t(error_rates)
> informv <- 2 - rowSums(error_rates)
> threshbot <- threshv[which.max(informv)]
> # Calculate out-of-sample forecasts from logistic regression mode
> predictout <- cbind(rep(1, NROW(outsample)), predv[outsample, ])
> fcast <- drop(predictout %*% coefftop)
> fcast <- 1/(1+exp(-fcast))
> forecastops <- (fcast > threshtop)
> fcast <- drop(predictout %*% coeffbot)
> fcast <- 1/(1+exp(-fcast))
> forecastbot <- (fcast > threshbot)
```

```
> # Simulate in-sample VTI strategy
> topsav <- HighFreq::roll_sum(matrix(forecastops), 5)/5
> botsav <- HighFreq::roll_sum(matrix(forecastbot), 5)/5
> posit <- (botsav-topsav)
> posit <- rutils::lagit(posit)
> pnls <- retvti[outsample, ]*posit
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retvti[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

draft: Forecasting Stock Tops and Bottoms Out-of-Sample

The function `predict()` is a *generic function* for forecasting based on a given model.

The method `predict.glm()` produces forecasts for a generalized linear (*glm*) model, in the form of `numeric` probabilities, not the `Boolean` response variable.

The `Boolean` forecasts are obtained by comparing the *forecast probabilities* with a *discrimination threshold*.

Let the *null hypothesis* be that the data point is not a top: `tops = FALSE`.

If the *forecast probability* is greater than the *discrimination threshold*, then the forecast is that the data point is not a top and that the *null hypothesis* is `TRUE`.

The *in-sample forecasts* are just the *fitted values* of the *glm* model.

```
> # Fit logistic regression over training data
> set.seed(1121)  # Reset random number generator
> nrows <- NROW(Default)
> samplev <- sample.int(n=nrows, size=nrows/2)
> trainset <- Default[samplev, ]
> logmod <- glm(formulav, data=trainset, family=binomial(logit))
> # Forecast over test data out-of-sample
> testset <- Default[-samplev, ]
> fcast <- predict(logmod, newdata=testset, type="response")
> # Calculate confusion matrix out-of-sample
> table(actual=!testset$default,
+ forecast=(fcast < threshold))
```

Forecasting Returns Using Logistic Regression

The weighted average of the volatility, trading volume, and regression z-scores can be used to forecast the sign of future returns.

The residuals are the differences between the actual response values (0 and 1), and the calculated probabilities of default.

The residuals are not normally distributed, so the data is fitted using the *maximum likelihood* method, instead of least squares.

```
> # Define response as the multi-day returns
> lagg <- 5
> retsf <- rutils::diffit(closep, lagg=5)
> retsf <- drop(coredata(retsf))
> # Fit in-sample logistic regression for positive returns
> retvtios <- (retsf > 0)
> logmod <- glm(retspos ~ predv - 1, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcast <- drop(predv %*% coeff)
> fcast <- 1/(1+exp(-fcast))
> # Calculate error rates
> threshv <- quantile(fcast, seq(0.01, 0.99, by=0.01))
> error_rates <- sapply(threshv, confun,
+   actual=!retspos, fcast=fcast)  # end sapply
> error_rates <- t(error_rates)
> # Calculate the threshold corresponding to highest informedness
> informv <- 2 - rowSums(error_rates)
> plot(threshv, informv, t="l", main="Informedness")
> threshm <- threshv[which.max(informv)]
> forecastpos <- (fcast > threshm)
> # Fit in-sample logistic regression for negative returns
> retsneg <- (retsf < 0)
> logmod <- glm(retsneg ~ predv - 1, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcast <- drop(predv %*% coeff)
> fcast <- 1/(1+exp(-fcast))
> # Calculate error rates
> error_rates <- sapply(threshv, confun,
+   actual=!retsneg, fcast=fcast)  # end sapply
> error_rates <- t(error_rates)
> # Calculate the threshold corresponding to highest informedness
> informv <- 2 - rowSums(error_rates)
> plot(threshv, informv, t="l", main="Informedness")
> threshm <- threshv[which.max(informv)]
> forecastneg <- (fcast > threshm)
```

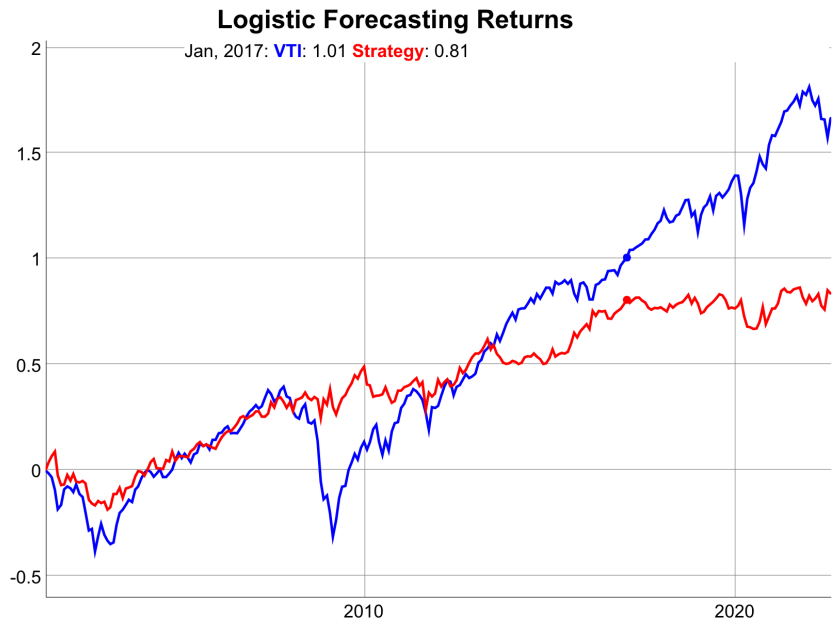
draft: Logistic Forecasting Returns Strategy In-sample

Explain why the strategy uses the minus of the signals.

The logistic strategy forecasts the sign of returns, using a logistic regression model with the volatility and trading volumes as predictors.

Averaging the forecasts over time improves strategy performance because of the bias-variance tradeoff.

It makes sense to average the forecasts over time because they are forecasts for future time intervals, not just a single point in time.

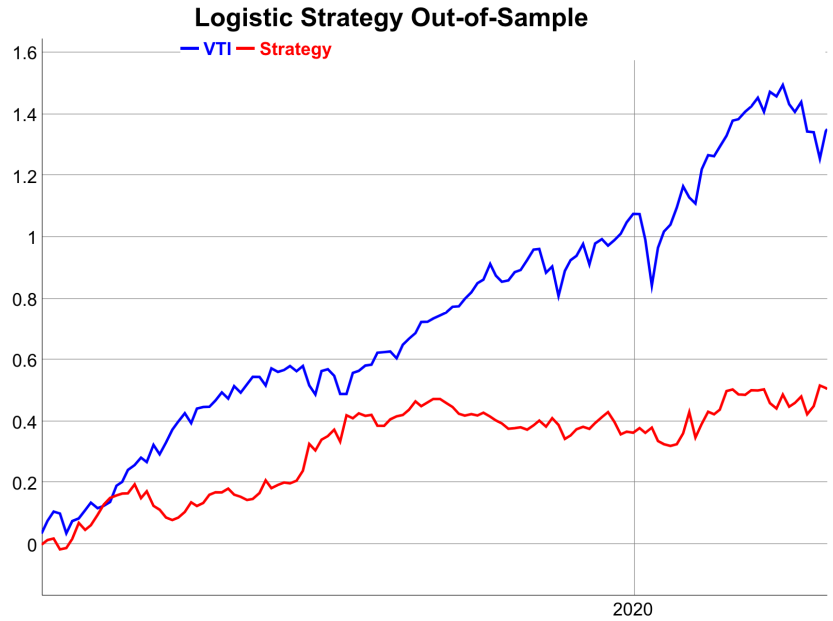



```
> # Simulate in-sample VTI strategy
> negav <- HighFreq::roll_sum(matrix(forecastneg), lagg)/lagg
> posav <- HighFreq::roll_sum(matrix(forecastpos), lagg)/lagg
> posit <- (negav - posav)
> # posit <- ifelse(forecastpos, 1, 0)
> # posit <- ifelse(forecastneg, -1, posit)
> posit <- rutils::lagit(posit)
> pnls <- retvti*posit
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```

```
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Forecasting Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

draft: Logistic Strategy Out-of-Sample

The logistic strategy forecasts the tops and bottoms of pricev, using a logistic regression model with the volatility and trading volumes as predictors.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Fit in-sample logistic regression for tops
> logmod <- glm(tops[insample] ~ predv[insample, ], family=binomial(
> fittedv <- logmod$fitted.values
> coefftop <- logmod$coefficients
> # Calculate error rates and best threshold value
> error_rates <- sapply(threshv, confun,
+   actual=!tops[insample], fcast=fittedv)  # end sapply
> error_rates <- t(error_rates)
> informv <- 2 - rowSums(error_rates)
> threshtop <- threshv[which.max(informv)]
> # Fit in-sample logistic regression for bottoms
> logmod <- glm(bottoms[insample] ~ predv[insample, ], family=binor
> fittedv <- logmod$fitted.values
> coeffbot <- logmod$coefficients
> # Calculate error rates and best threshold value
> error_rates <- sapply(threshv, confun,
+   actual=!bottoms[insample], fcast=fittedv)  # end sapply
> error_rates <- t(error_rates)
> informv <- 2 - rowSums(error_rates)
> threshbot <- threshv[which.max(informv)]
> # Calculate out-of-sample forecasts from logistic regression mode
> predictout <- cbind(rep(1, NROW(outsample)), predv[outsample, ])
> fcast <- drop(predictout %*% coefftop)
> fcast <- 1/(1+exp(-fcast))
> forecastops <- (fcast > threshtop)
> fcast <- drop(predictout %*% coeffbot)
> fcast <- 1/(1+exp(-fcast))
> forecastbot <- (fcast > threshbot)
```

```
> # Simulate out-of-sample VTI strategy
> posit <- rep(NA_integer_, NROW(outsample))
> posit[1] <- 0
> posit[forecastops] <- (-1)
> posit[forecastbot] <- 1
> posit <- zoo::na.locf(posit)
> posit <- rutils::lagit(posit)
> pnls <- retvti[outsample, ]*posit
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retvti[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
```

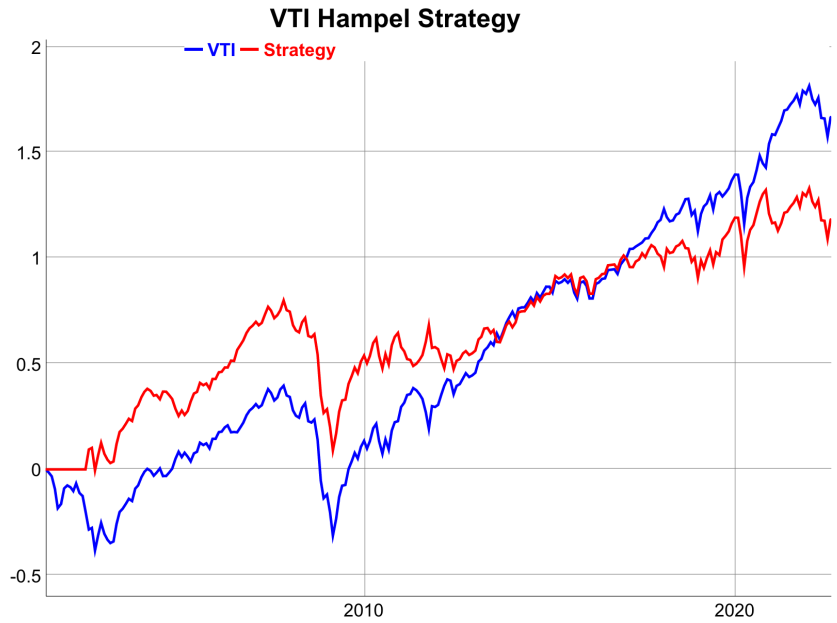
Hampel Filter Strategy

The Hampel filter strategy is a contrarian strategy that uses Hampel z-scores to establish long and short positions.

The Hampel strategy has two meta-parameters: the look-back interval and the threshold level.

The best choice of the meta-parameters can be determined through simulation.

```
> # Calculate VTI percentage returns
> closep <- log(na.omit(rutils::etfenv$prices$VTI))
> retvti <- rutils::diffit(closep)
> # Define look-back window
> look_back <- 11
> # Calculate time series of medians
> medianv <- roll::roll_median(closep, width=look_back)
> # medianv <- TTR::runMedian(closep, n=look_back)
> # Calculate time series of MAD
> madv <- HighFreq::roll_var(closep, look_back=look_back, method="
> # madv <- TTR::runMAD(closep, n=look_back)
> # Calculate time series of z-scores
> zscores <- (closep - medianv)/madv
> zscores[1:look_back, ] <- 0
> tail(zscores, look_back)
> range(zscores)
> # Define threshold value
> threshold <- sum(abs(range(zscores)))/8
> # Simulate VTI strategy
> posit <- rep(NA_integer_, NROW(closep))
> posit[1] <- 0
> posit[zscores < -threshold] <- 1
> posit[zscores > threshold] <- (-1)
> posit <- zoo::na.locf(posit)
> posit <- rutils::lagit(posit)
> pnls <- retvti*posit
```




```
> # Plot dygraph of Hampel strategy pnls
> wealthv <- cbind(retvti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Hampel Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Engle-Granger Two-step Procedure for Cointegration

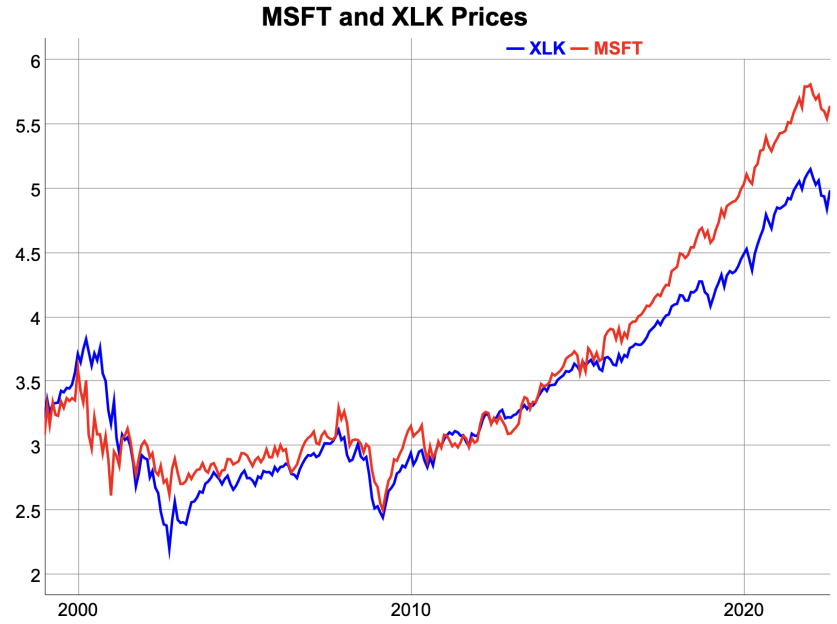
A group of stocks is *cointegrated* if there is a portfolio of the stocks whose price is range-bound.

The Engle-Granger two-step procedure can be used to find the cointegrated portfolio of two stocks:

- Perform a regression of the stock prices to calculate the cointegrating factor β,
- Apply the *ADF* test to the regression residuals (the portfolio price) to determine if they have a unit root (the portfolio price diverges), or if they are mean reverting.

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

The *null hypothesis* of the *ADF* test is that the time series has a *unit root* (it diverges). So a small p-value suggests that the *null hypothesis* is FALSE and that the time series is range-bound.




```
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices
> # Subset (select) the stock prices after the start date of XLK
> pricev <- cbind(pricev$MSFT, rutils::etfenv$prices$XLK)
> pricev <- log(na.omit(pricev))
> colnames(pricev) <- c("MSFT", "XLK")
> datev <- zoo::index(pricev)
> # Calculate the regression coefficients of MSFT ~ XLK
> betav <- drop(cov(pricev$MSFT, pricev$XLK)/var(pricev$XLK))
> # Calculate the regression residuals
> residv <- (pricev$MSFT - betav*pricev$XLK)
> # Perform ADF test on residuals
```

```
> # Perform ADF test for vector of cointegrating factors
> betas <- seq(0.5, 1.5, 0.1)
> adfstat <- sapply(betas, function(betav) {
+   residv <- (pricev$MSFT - betav*pricev$XLK)
+   tseries::adf.test(residv, k=1)$statistic
+ })  # end sapply
> plot(x=betas, y=adfstat, type="l", xlab="cointegrating factor", yl
+  main="ADF Test Statistic as Function of Cointegrating Factor")
> # Plot the prices
> endd <- rutils::calc_endpoints(pricev, interval="months")
> dygraphs::dygraph(pricev[endd], main="MSFT and XLK Prices") %>%
+   dyOptions(colors=c("red", "blue"), strokeWidth=2)
> # Plot the cointegration residuals
> dygraphs::dygraph(residv[endd], main="MSFT and XLK Cointegration R
+   dyOptions(colors=c("blue"), strokeWidth=2)
```

draft: Trading Cointegrated Pairs

Explain that the cointegration residual is not stationary (it has a time-dependent volatility), so it's hard to trade. We need to make the volatility less time-dependent.

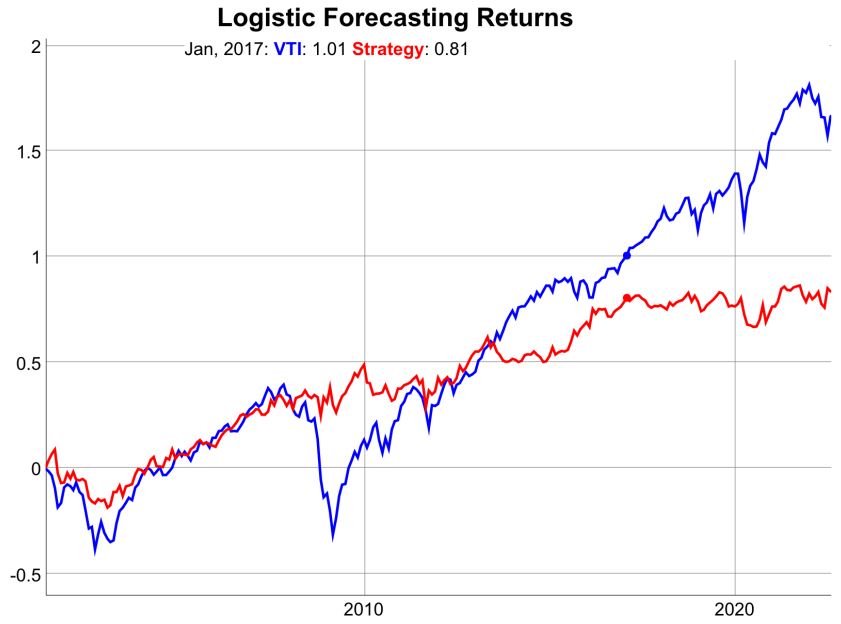
Explain why the strategy uses the minus of the signals.

Explain why the strategy uses the minus of the signals.

The logistic strategy forecasts the sign of returns, using a logistic regression model with the volatility and trading volumes as predictors.

Averaging the forecasts over time improves strategy performance because of the bias-variance tradeoff.

It makes sense to average the forecasts over time because they are forecasts for future time intervals, not just a single point in time.




```
> retsres <- rutils::diffit(residv)
> lambda <- 0.6
> volat <- HighFreq::run_var(retsres, lambda=lambda)
> volat <- sqrt(volat)
> volat <- xts::xts(volat, datev)
> dygraphs::dygraph(volat[endd], main="Residual Volatility") %>%
+   dyOptions(colors=c("blue"), strokeWidth=2)
>
> # Calculate the regression residuals
> residv <- (pricev$MSFT - betav*pricev$XLK)
> # Perform ADF test on residuals
> retp <- rutils::diffit(pricev$XLK)
> retsres <- rutils::diffit(residv)
> resids <- retsres/volat
> resids[1] <- 0
> resids <- HighFreq::run_mean(resids, lambda=lambda)
```

```
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Forecasting Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

The *Leverage* Function

The *leverage function* maps the *predictor* value p into a dollar amount for investment.

A possible choice for the leverage function is the hyperbolic tangent function:

$$L(x) = \frac{\exp(\lambda p) - \exp(-\lambda p)}{\exp(\lambda p) + \exp(-\lambda p)}$$

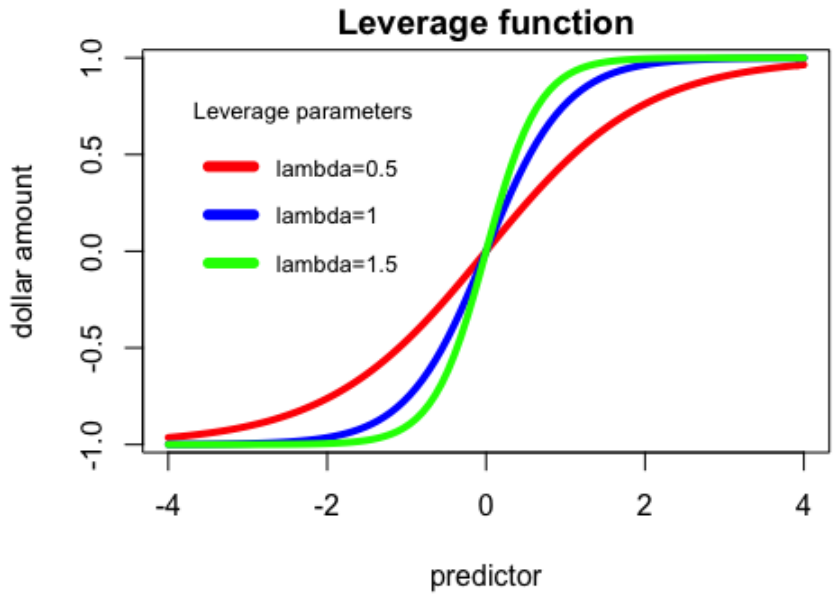
Where λ is the leverage parameter.

The hyperbolic tangent is close to linear for small values of the *predictor* p, and saturates to $+1\$/-1\$$ for very large positive and negative values of the *predictor*.

The saturation effect limits (caps) the leverage in the strategy to $+1\$/-1\$$.

For very small values of the leverage parameter λ, the invested dollar amount is linear for a wide range of *predictor* values p. So the strategy is mostly invested in dollar amounts proportional to the *predictor* values.

For very large values of the leverage parameter λ, the invested dollar amount jumps from $-1\$$ for negative *predictor* values to $+1\$$ for positive *predictor* values. So the strategy is invested in either $-1\$$ or $+1\$$ dollar amounts.




```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the leverage function
> leverage <- function(p, lambda) tanh(lambda*p)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=leverage(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="predictor", ylab="dollar amount",
+ col=colorv[indeks], add=(indeks>1))
+ }  # end for
> # Add title
> title(main="Leverage function", line=0.5)
> # Add legend
> legend("topleft", title="Leverage parameters",
+        paste("lambda", lambdav, sep="="))
```

Forecasting Stock Returns Using Autoregressive Models

Assume that the stock returns r_i follow an *autoregressive* process *AR(n)* with a constant term (intercept):

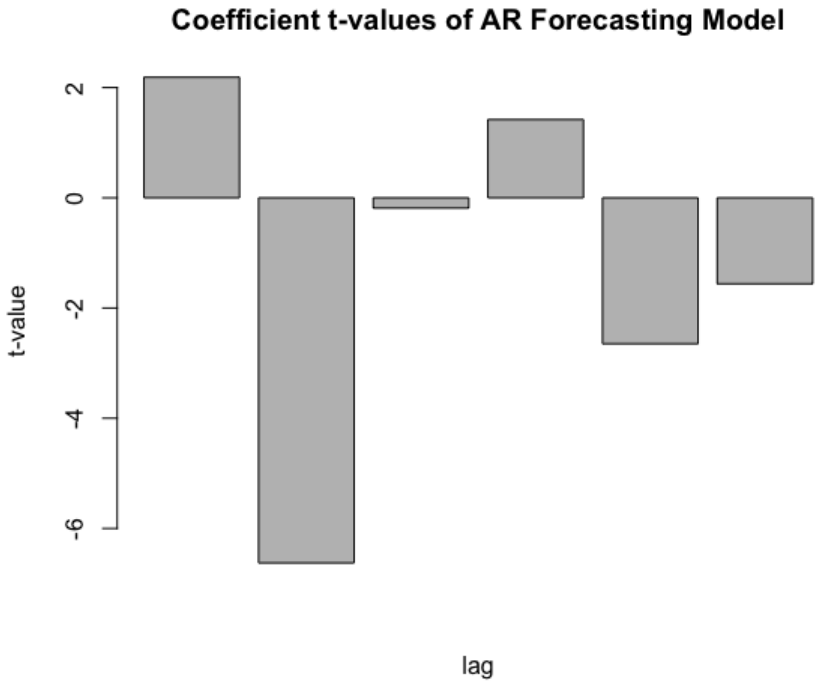
$$r_i = \varphi_0 + \varphi_1 r_{i-1} + \varphi_2 r_{i-2} + \ldots + \varphi_n r_{i-n} + \xi_i$$

The coefficients φ can be calculated using linear regression, with the *response* equal to $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ equal to the lags of $\mathbf{r}$:

$$\varphi = \mathbb{P}^{-1}\mathbf{r}$$

The *in-sample AR(n)* autoregressive forecasts are calculated by multiplying the predictor matrix by the fitted coefficients:

$$f_i = \varphi_0 + \varphi_1 r_{i-1} + \varphi_2 r_{i-2} + \ldots + \varphi_n r_{i-n}$$




```
> # Calculate a vector of daily VTI percentage returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retvti)
> retvti <- as.numeric(retvti)
> respv <- retvti
> nrows <- NROW(retvti)
> # Define predictor matrix for forecasting
> orderp <- 5
> predv <- sapply(1:orderp, rutils::lagit, input=respv)
> predv <- cbind(rep(1, nrows), predv)
> colnames(predv) <- paste0("pred", 1:NCOL(predv))
> predinv <- MASS::ginv(predv)
> coeff <- drop(predinv %*% respv)
> # Calculate in-sample forecasts of VTI
> fcast <- drop(predv %*% coeff)
> # Calculate the residuals (forecast errors)
> residv <- drop(retvti - fcast)
```

```
> # Calculate the variance of the residuals
> residvar <- sum(residv^2)/(nrows-NROW(coeff))
> # Calculate the predictor matrix squared
> predictor2 <- crossprod(predv)
> # Calculate covariance matrix of the AR coefficients
> covar <- residvar*MASS::ginv(predv2)
> coeffsd <- sqrt(diag(covar))
> # Calculate the t-values of the AR coefficients
> coefftv <- coeff/coeffsd
> # Plot the t-values of the AR coefficients
> barplot(coefftv, xlab="lag", ylab="t-value",
+   main="Coefficient t-values of AR Forecasting Model")
```

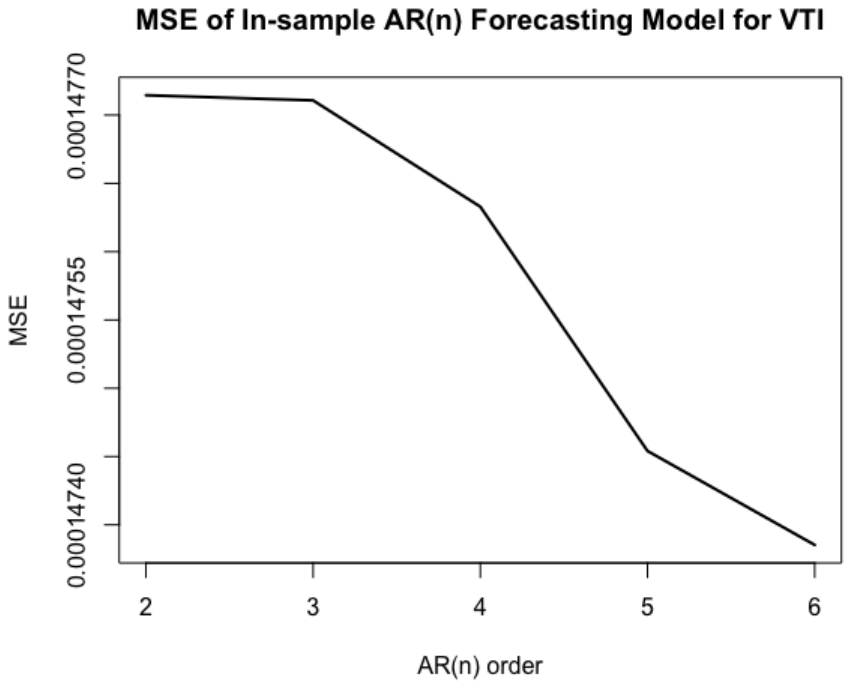
In-sample Order Selection of Autoregressive Forecasting

The mean squared errors (*MSE*) of the *in-sample* forecasts decrease steadily with the increasing order parameter *n* of the *AR(n)* forecasting model.

In-sample forecasting consists of first fitting an *AR(n)* model to the data, and calculating its coefficients.

The *in-sample* forecasts are calculated by multiplying the predictor matrix by the fitted coefficients.

```
> # Define predictor matrix for forecasting
> ordmax <- 5
> predv <- sapply(1:ordmax, rutils::lagit, input=respv)
> predv <- cbind(rep(1, nrows), predv)
> colnames(predv) <- paste0("pred", 1:NCOL(predv))
> # Calculate forecasts as function of the AR order
> fcast <- lapply(2:NCOL(predv), function(ordern) {
+   # Calculate fitted coefficients
+   predinv <- MASS::ginv(predv[, 1:ordern])
+   coeff <- drop(predinv %*% respv)
+   # Calculate in-sample forecasts of VTI
+   drop(predv[, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcast) <- paste0("n=", 2:NCOL(predv))
```




```
> # Calculate mean squared errors
> mse <- sapply(fcast, function(x) {
+   c(mse=mean((respv - x)^2), cor=cor(respv, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcast)
> # Plot forecasting MSE
> plot(x=2:NCOL(predv), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of In-sample AR(n) Forecasting Model for VTI")
```

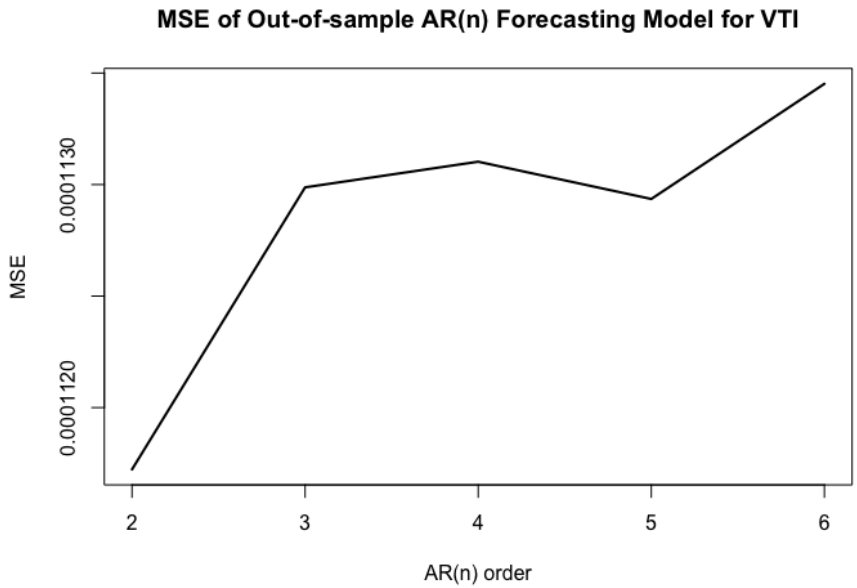
Out-of-sample Forecasting Using Autoregressive Models

The mean squared errors (*MSE*) of the *out-of-sample* forecasts increase with the increasing order parameter *n* of the *AR(n)* model.

The reason for the increasing out-of-sample MSE is the *overfitting* of the coefficients to the training data for larger order parameters.

Out-of-sample forecasting consists of first fitting an *AR(n)* model to the training data, and calculating its coefficients.

The *out-of-sample* forecasts are calculated by multiplying the *out-of-sample* predictor matrix by the fitted coefficients.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate forecasts as function of the AR order
> fcast <- lapply(2:NCOL(predv), function(ordern) {
+   # Calculate fitted coefficients
+   predinv <- MASS::ginv(predv[insample, 1:ordern])
+   coeff <- drop(predinv %*% respv[insample])
+   # Calculate out-of-sample forecasts of VTI
+   drop(predv[outsample, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcast) <- paste0("n=", 2:NCOL(predv))
```

```
> # Calculate mean squared errors
> mse <- sapply(fcast, function(x) {
+   c(mse=mean((respv[outsample] - x)^2), cor=cor(respv[outsample],
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcast)
> # Plot forecasting MSE
> plot(x=2:NCOL(predv), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Out-of-sample AR(n) Forecasting Model for VTI")
```

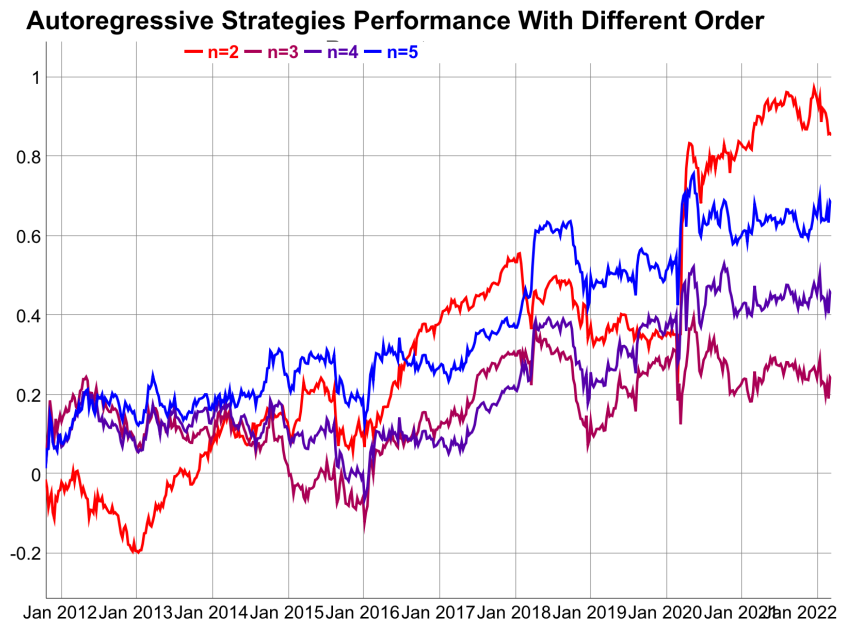
Autoregressive Strategy Out-of-sample Performance

The autoregressive strategy invests a single dollar amount of *VTI* equal to the sign of the forecasts.

The performance of the autoregressive strategy is better with a smaller order parameter *n* of the *AR(n)* model.

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.

```
> # Calculate out-of-sample PnLs
> pnls <- sapply(fcast, function(x) {
+   cumsum(sign(x)*retvti[outsample])
+ })  # end sapply
> colnames(pnls) <- names(fcast)
> pnls <- xts::xts(pnls, datev[outsample])
```




```
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls[, 1:4]))
> colnamev <- colnames(pnls[, 1:4])
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(pnls[endd, 1:4],
+   main="Autoregressive Strategies Performance With Different Orde
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=500)
```

Autoregressive Strategy Using Average Past Returns

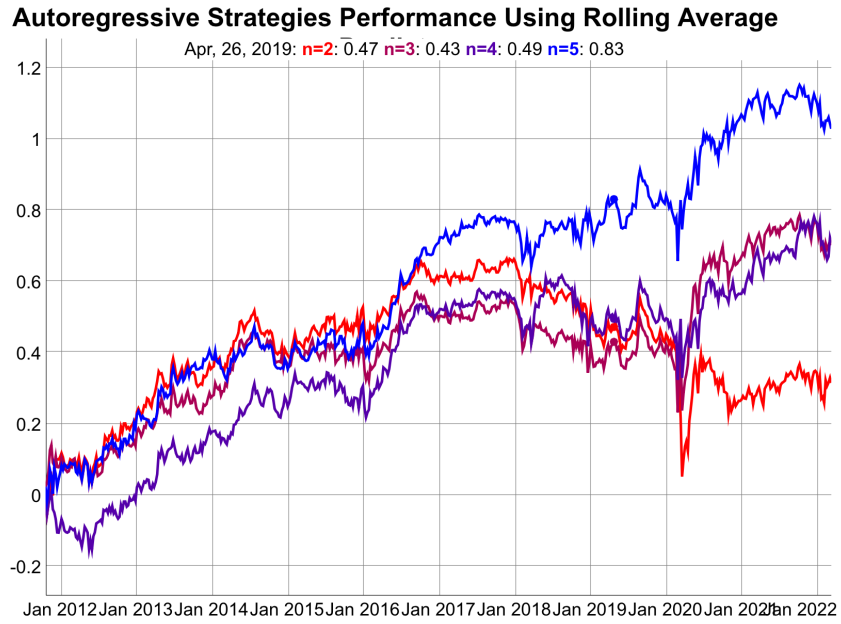
The *out-of-sample* forecasts can be improved by using the rolling average of the returns as a predictor.

This is because the average of returns has a lower *variance*.

But the average also has a higher *bias* because it includes past returns that may be unrelated to the present.

Using the rolling average of returns as a predictor reduces the forecast variance at the expense of increasing its bias (known as the *bias-variance tradeoff*).

```
> # Define predictor as a rolling mean
> nagg <- 5
> predv <- roll::roll_mean(retvti, width=nagg, min_obs=1)
> respv <- retvti
> # Define predictor matrix for forecasting
> predv <- sapply(1+nagg*(0:ordmax), rutils::lagit,
+                 input=predv)
> predv <- cbind(rep(1, nrows), predv)
> # Calculate forecasts as function of the AR order
> fcast <- lapply(2:NCOL(predv), function(ordern) {
+   predinv <- MASS::ginv(predv[insample, 1:ordern])
+   coeff <- drop(predinv %*% respv[insample])
+   drop(predv[outsample, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcast) <- paste0("n=", 2:NCOL(predv))
```




```
> # Calculate out-of-sample PnLs
> pnls <- sapply(fcast, function(x) {
+   cumsum(sign(x)*retvti[outsample])
+ })  # end sapply
> colnames(pnls) <- names(fcast)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot dygraph of out-of-sample PnLs
> dygraphs::dygraph(pnls[endd, 1:4],
+   main="Autoregressive Strategies Performance Using Rolling Averag
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=500)
```

Autoregressive Strategy Using Average of Past Forecasts

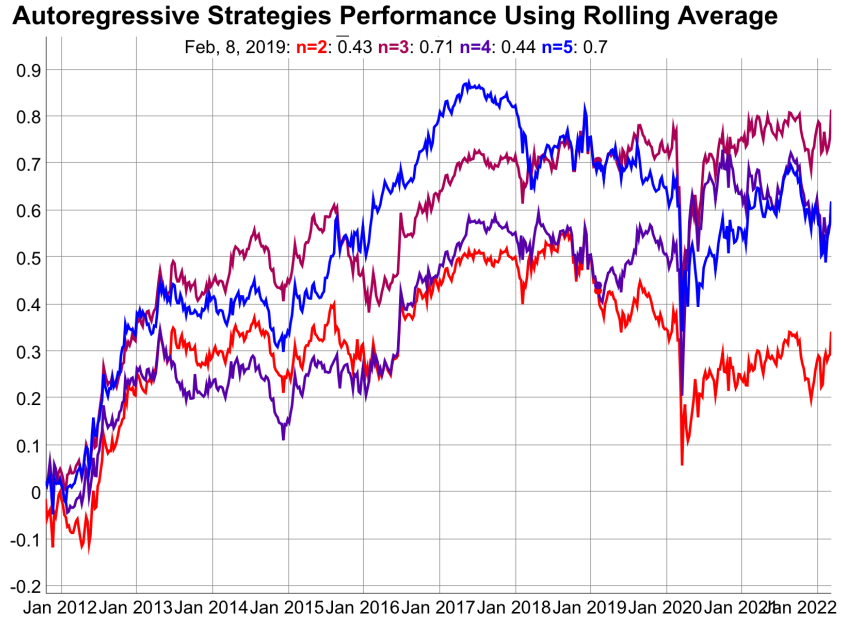
The *out-of-sample* forecasts can be further improved by using the average of past forecasts.

This is because the average of forecasts has a lower *variance*.

But the average also has a higher *bias* because it includes past forecasts that may be unrelated to the present.

Using the rolling average of past forecasts reduces the forecast variance at the expense of increasing its bias (known as the *bias-variance tradeoff*).

```
> # Calculate PnLs using the average of past forecasts
> nagg <- 5
> pnls <- sapply(fcast, function(x) {
+   x <- roll::roll_mean(x, width=nagg, min_obs=1)
+   cumsum(sign(x)*retvti[outsample])
+ })  # end sapply
> colnames(pnls) <- names(fcast)
> pnls <- xts::xts(pnls, datev[outsample])
```




```
> # Plot dygraph of out-of-sample PnLs
> dygraphs::dygraph(pnls[endd, 1:4],
+   main="Autoregressive Strategies Performance Using Rolling Averag
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=500)
```

depr: Forecasting Returns Using Autoregressive Models

Forecasting using an autoregressive model is performed by first fitting an *AR(n)* model to past data, and calculating its coefficients.

The fitted coefficients are then applied to calculating the *out-of-sample* forecasts.

The forecasting model depends on two unknown *meta-parameters*: the order n of the *AR(n)* model and the length of the look-back interval (`look_back`).

```
> library(rutils)
> # Calculate a vector of daily VTI log returns
> pricev <- log(quantmod::Cl(rutils::etfenv$VTI))
> retvti <- rutils::diffit(pricev)
> retvti <- as.numeric(retvti)
> nrows <- NROW(retvti)
> # Define predictor matrix for forecasting
> ordmax <- 5
> desv <- sapply(1:ordmax, rutils::lagit, input=respv)
> colnames(desv) <- paste0("pred", 1:NCOL(desv))
> # Add response equal to VTI
> desv <- cbind(retvti, desv)
> colnames(desv)[1] <- "response"
> # Specify length of look-back interval
> look_back <- 100
> # Invert the predictor matrix
> rangev <- (nrows-look_back):(nrows-1)
> designinv <- MASS::ginv(desv[rangev, -1])
> # Calculate fitted coefficients
> coeff <- drop(desvinv %*% desv[rangev, 1])
> # Calculate forecast of VTI for nrows
> drop(desv[nrows, -1] %*% coeff)
> # Compare with actual value
> desv[nrows, 1]
```

Rolling Autoregressive Forecasting Model

The autoregressive coefficients can be calibrated dynamically over a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`look_back`).

```
> # Define predictor as a rolling sum
> nagg <- 5
> predv <- rutils::roll_sum(retvti, look_back=nagg)
> # Define predictor matrix for forecasting
> ordmax <- 5
> predv <- sapply(1+nagg*(0:ordmax), rutils::lagit,
+                 input=predv)
> predv <- cbind(rep(1, nrows), predv)
> # Perform rolling forecasting
> look_back <- 100
> fcast <- sapply((look_back+1):nrows, function(endd) {
+   # Define rolling look-back range
+   startp <- max(1, endd-look_back)
+   # Or expanding look-back range
+   # startp <- 1
+   rangev <- startp:(endd-1)
+   # Invert the predictor matrix
+   designinv <- MASS::ginv(predv[rangev, ])
+   # Calculate fitted coefficients
+   coeff <- drop(desvinv %*% retvti[rangev])
+   # Calculate forecast
+   drop(predv[endd, ] %*% coeff)
+ })  # end sapply
> # Add warmup period
> fcast <- c(rep(0, look_back), fcast)
```

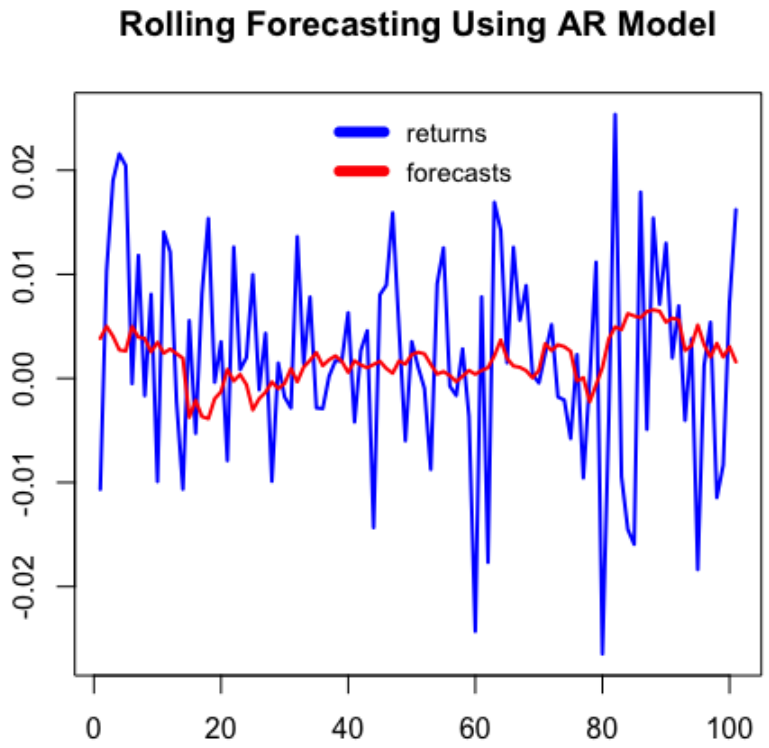
Mean Squared Error of the Autoregressive Forecasting Model

The accuracy of a forecasting model can be measured using the *mean squared error* and the *correlation*.

The mean squared error (*MSE*) of a forecasting model is the average of the squared forecasting residuals ε_i, equal to the differences between the actual values r_i minus the *forecasts* f_i: $\varepsilon_i = r_i - f_i$:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(r_i - f_i)^2$$

```
> # Mean squared error
> mean((retvti - fcast)^2)
> # Correlation
> cor(fcast, retvti)
```




```
> # Plot forecasting series with legend
> x11(width=6, height=5)
> par(mar=c(3, 3, 2, 1), oma=c(0, 0, 0, 0))
> plot(retvti[(nrows-look_back):nrows], col="blue",
+      xlab="", ylab="", type="l", lwd=2,
+      main="Rolling Forecasting Using AR Model")
> lines(fcast[(nrows-look_back):nrows], col="red", lwd=2)
> legend(x="topleft", legend=c("returns", "forecasts"),
+  col=c("blue", "red"), lty=1, lwd=6, y.intersp=0.5,
+  cex=0.8, bg="white", bty="n")
```

Backtesting Function for the Forecasting Model

The *meta-parameters* of the *backtesting* function are the order *n* of the *AR(n)* model and the length of the look-back interval (`look_back`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`look_back`).

```
> # Define backtesting function
> sim_fcasts <- function(respv, nagg=5, ordern=5,
+                        look_back=100, rollp=TRUE) {
+   nrows <- NROW(respv)
+   # Define predictor as a rolling sum
+   predv <- rutils::roll_sum(respv, look_back=nagg)
+   # Define predictor matrix for forecasting
+   predv <- sapply(1+nagg*(0:ordern), rutils::lagit,
+                   input=predv)
+   predv <- cbind(rep(1, nrows), predv)
+   # Perform rolling forecasting
+   fcast <- sapply((look_back+1):nrows, function(endd) {
+     # Define rolling look-back range
+     if (rollp)
+ startp <- max(1, endd-look_back)
+     else
+     # Or expanding look-back range
+     startp <- 1
+     rangev <- startp:(endd-1)
+     # Invert the predictor matrix
+     designinv <- MASS::ginv(predv[rangev, ])
+     # Calculate fitted coefficients
+     coeff <- drop(desvinv %*% respv[rangev])
+     # Calculate forecast
+     drop(predv[endd, ] %*% coeff)
+   })  # end sapply
+   # Add warmup period
+   fcast <- c(rep(0, look_back), fcast)
+   # Aggregate the forecasts
+   rutils::roll_sum(fcast, look_back=nagg)
+ }  # end sim_fcasts
> # Simulate the rolling autoregressive forecasts
> fcast <- sim_fcasts(respv=retvti, ordern=5, look_back=100)
> c(mse=mean((retvti - fcast)^2), cor=cor(retvti, fcast))
```

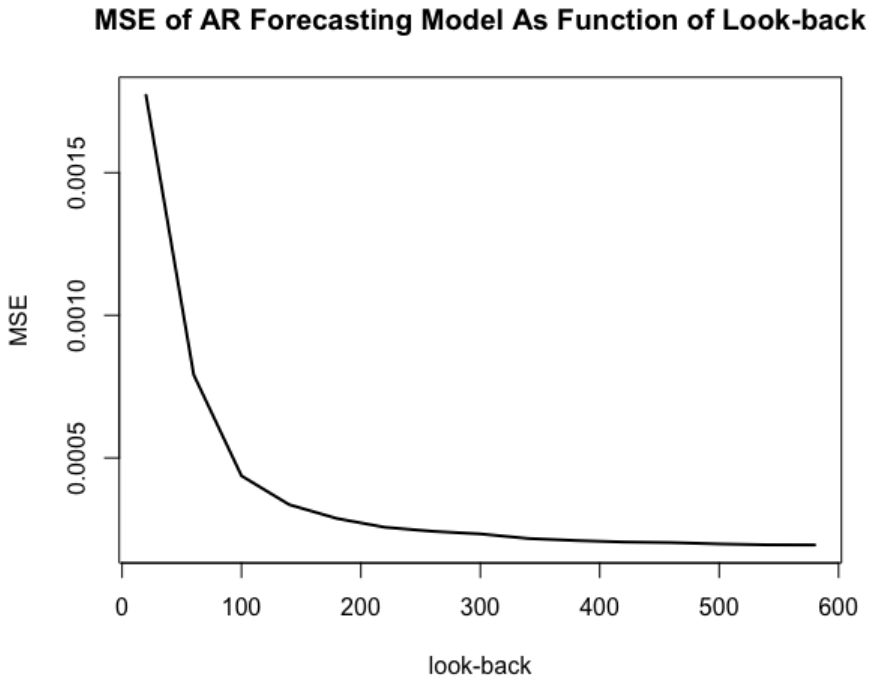
Forecasting Dependence On the Look-back Interval

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`look_back`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model increases with longer look-back intervals (`look_back`), because more data improves the estimates of the autoregressive coefficients.




```
> library(parallel)  # Load package parallel
> # Calculate number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> look_backs <- seq(20, 600, 40)
> fcast <- parLapply(cluster, look_backs, sim_fcasts,
+   response=retvti, nagg=5, ordern=5)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcast <- mclapply(look_backs, sim_fcasts, response=retvti,
+   nagg=5, ordern=5, mc.cores=ncores)
```

```
> # Calculate mean squared errors
> mse <- sapply(fcast, function(x) {
+   c(mse=mean((retvti - x)^2), cor=cor(retvti, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- look_backs
> # Select optimal look_back interval
> look_back <- look_backs[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=look_backs, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

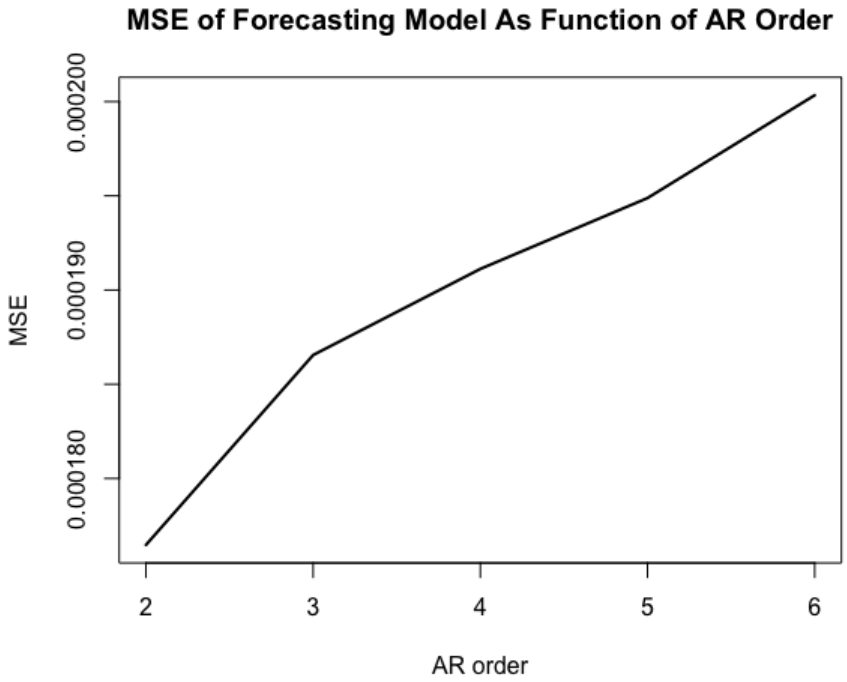
The Dependence On the Order Parameter

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`look_back`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model decreases for larger AR order parameters, because of overfitting in-sample.




```
> library(parallel)  # Load package parallel
> # Calculate number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcast <- parLapply(cluster, orderv, sim_fcasts, response=retvti,
+   nagg=5, look_back=look_back)
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcast <- mclapply(orderv, sim_fcasts, response=retvti,
+   nagg=5, look_back=look_back, mc.cores=ncores)
```

```
> # Calculate mean squared errors
> mse <- sapply(fcast, function(x) {
+   c(mse=mean((retvti - x)^2), cor=cor(retvti, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Forecasting Model As Function of AR Order")
```

Performance of the Rolling Autoregressive Strategy

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy is dominated by a few periods with very large returns, without producing profits for the remaining periods.

Using the return forecasts as portfolio weights produces very large weights in periods of high volatility, and creates excessive risk.

To reduce excessive risk, a binary strategy uses portfolio weights equaly to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcast <- sim_fcasts(retvti, ordern=ordern, look_back=look_back)
> # Calculate strategy PnLs
> pnls <- sign(fcast)*retvti
> pnls <- cbind(retvti, pnls, (retvti+pnls)/2)
> colnames(pnls) <- c("VTI", "AR_Strategy", "Combined")
> cor(pnls)
> # Annualized Sharpe ratios of VTI and AR strategy
> pnls <- xts::xts(pnls, datev)
> sqrt(252)*sapply(pnls, function (x) mean(x)/sd(x))
```


```
> # Plot the cumulative strategy PnLs
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd], main="Rolling Autoregressive
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

draft: Backtesting of Strategies

Backtesting is the simulation of a rolling strategy's performance on historical data.

A *rolling strategy* can be *backtested* by specifying the parameter updating frequency, the formation interval, and the holding period:

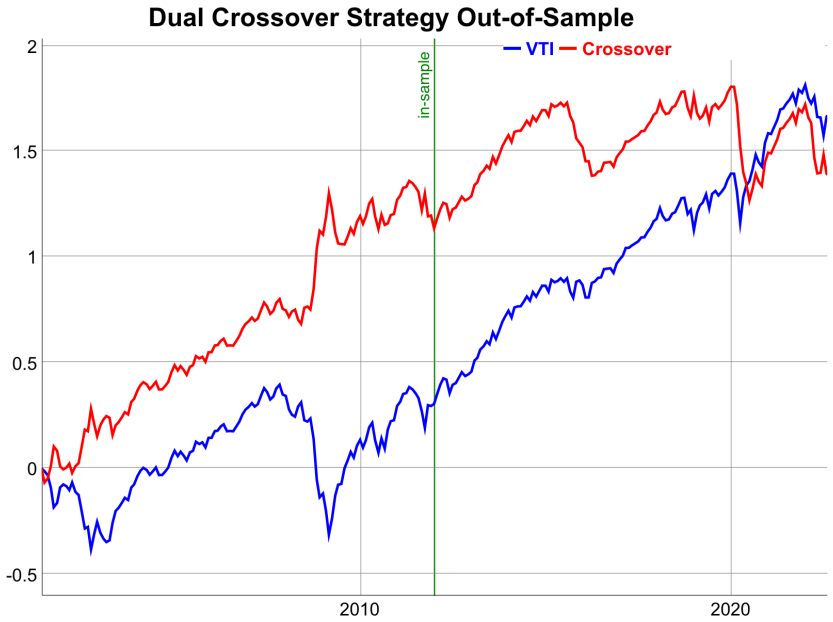
- Calculate the *end points* for parameter updating,
- Define an objective function for parameter optimization,
- Calculate the optimal parameters in the in-sample formation interval,
- Calculate the out-of-sample strategy returns,
- Calculate the transaction costs and subtract them from the strategy returns.

But using a different updating 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.

In-sample, the best *Dual EWMA* strategy performs




```
> # Dygraphs plot with custom line colors
> endd <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual Crossover Stra
+   dyEvent(zoo::index(wealthv[last(insample)]), label="in-sample",
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

depr: The Dependence On the Order Parameter

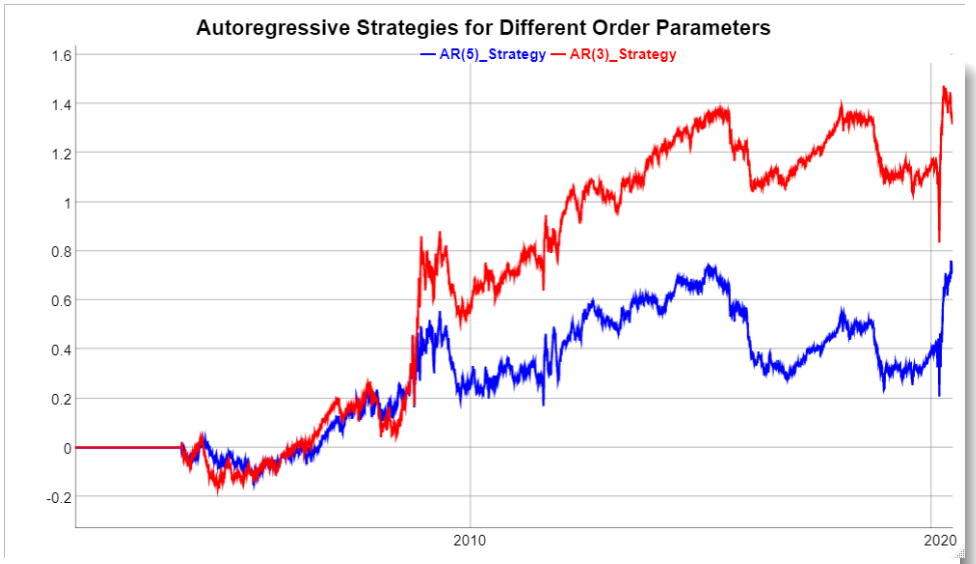
The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Longer look-back intervals (`look_back`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy is dominated by a few periods with very large returns, without producing profits for the remaining periods.




```
> # Calculate PnLs for ordern=5
> fcast <- sim_fcasts(retvti, ordern=ordern, look_back=look_back)
> pnls5 <- cumsum(sign(fcast)*retvti)
> # Calculate PnLs for ordern=3
> fcast <- sim_fcasts(retvti, ordern=ordern, look_back=look_back)
> pnls3 <- cumsum(sign(fcast)*retvti)
```

```
> # Plot the cumulative strategy returns
> wealthv <- cbind(pnls5, pnls3)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("AR(5)_Strategy", "AR(3)_Strategy")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv, main="Autoregressive Strategies for Di
+   dySeries(name=colnamev[1], label=colnamev[1], col="blue", stroke
+   dySeries(name=colnamev[2], label=colnamev[2], col="red", strokeW
+   dyLegend(width=500)
```

draft: Autoregressive Strategy With an Expanding Look-back Interval

Study why the *expanding* look-back interval doesn't improve performance.

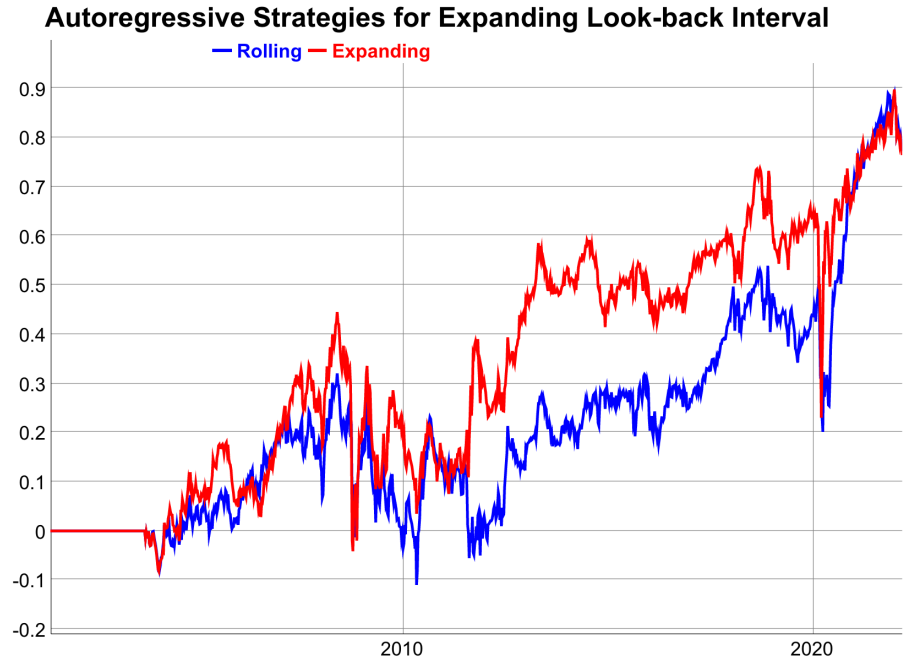
The accuracy of the forecasting model depends on whether a *rolling* or an *expanding* look-back interval is used.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Longer look-back intervals (`look_back`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy is dominated by a few periods with very large returns, without producing profits for the remaining periods.




```
> # Calculate PnLs for rolling look-back
> fcast <- sim_fcasts(retvti, ordern=ordern, look_back=look_back,
+                     rollp=TRUE)
> pnls_roll <- cumsum(sign(fcast)*retvti)
> # Calculate PnLs for expanding look-back
> fcast <- sim_fcasts(retvti, ordern=ordern, look_back=look_back,
+                     rollp=FALSE)
> pnls_expand <- cumsum(sign(fcast)*retvti)
```

```
> # Plot the cumulative strategy returns
> wealthv <- cbind(pnls_roll, pnls_expand)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Rolling", "Expanding")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv[endd], main="Autoregressive Strategies f
+   dySeries(name=colnamev[1], label=colnamev[1], col="blue", stroke
+   dySeries(name=colnamev[2], label=colnamev[2], col="red", strokeW
+   dyLegend(width=500)
```

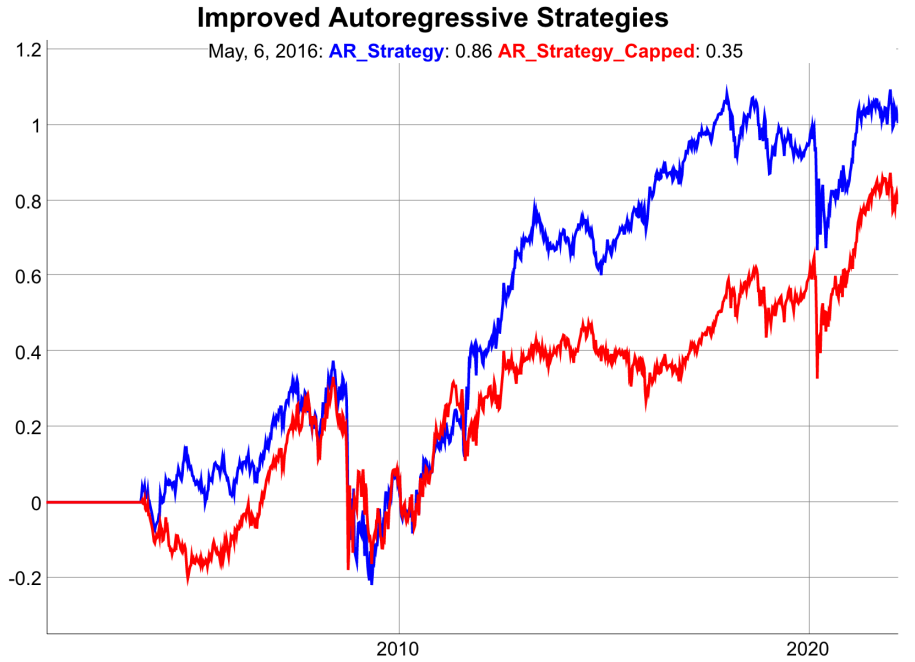
draft: Improved Autoregressive Strategy

Capping returns doesn't really help.

The performance of the autoregressive strategy can be improved by fitting its coefficients using the *capped returns*, to reduce the leverage of very large returns.

The performance can be further improved by fitting the coefficients over an *expanding look-back interval*, instead of a *rolling look-back interval*.

A longer look-back interval (`look_back`) also improves the performance.




```
> # Cap the VTI returns
> cutoff <- 0.03
> capped <- ifelse(retvti > cutoff, cutoff, retvti)
> capped <- ifelse(capped < (-cutoff), -cutoff, capped)
> # Calculate PnLs for VTI
> fcast <- sim_fcasts(retvti, ordern=3, look_back=look_back, rollp=FALSE)
> pnls <- cumsum(sign(fcast)*retvti)
> # Calculate PnLs for capped VTI returns
> fcast <- sim_fcasts(capped, ordern=3, look_back=look_back, rollp=
> pnls_capped <- cumsum(sign(fcast)*retvti)
```

```
> # Plot the cumulative strategy returns
> wealthv <- cbind(pnls, pnls_capped)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("AR(3)_Rolling", "AR(3)_Expanding")
> colnamev <- c("AR_Strategy", "AR_Strategy_Capped")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv[endd], main="Improved Autoregressive Str
+   dySeries(name=colnamev[1], label=colnamev[1], col="blue", stroke
+   dySeries(name=colnamev[2], label=colnamev[2], col="red", strokeW
+   dyLegend(width=500)
```