

Univariate Investment Strategies

FRE7241, Spring 2023

Jerzy Pawlowski jp3900@nyu.edu

NYU Tandon School of Engineering

May 14, 2023



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:

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

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 the 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 the simple dollar returns
> retd <- rutils::diffit(pricev)
> # Or
> # retd <- lapply(pricev, rutils::diffit)
> # retd <- rutils::do_call(cbind, retd)
> # Calculate the percentage returns
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate the log returns
> relt <- 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.

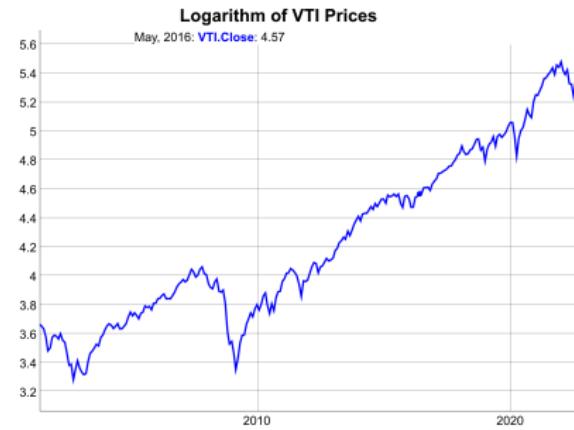
The sum of the percentage returns (without compounding): $\sum_{i=1}^n r_i^p$ represents the wealth path from owning a *fixed dollar amount* of stock.

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.

The logarithm of the wealth of a *fixed number of shares* is approximately equal to the sum of the percentage returns.

```
> # Set the initial dollar returns
> retd[1, ] <- pricev[1, ]
> # Calculate the 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)
> # pricen <- t(t(pricen)*pricesi)
> all.equal(pricen, pricev, check.attributes=FALSE)
```



```
> # Plot log VTI prices
> endd <- rutils::calc_endpoints(rutils::etfenv$VTI, interval="weeks")
> dygraphs::dygraph(log(quantmod::Cl(rutils::etfenv$VTI)[endd]),
+   main="Logarithm of VTI Prices") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Funding Costs of Single Asset Rebalancing

The rebalancing of stock requires borrowing from a *margin account*, and it also incurs trading costs.

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 the percentage VTI returns
> pricev <- rutils::etfenv$prices$VTI
> pricev <- na.omit(pricev)
> retp <- rutils:::diffit(pricev)/rutils::lagit(pricev, lagg=1, pad=
```



```
> # Funding rate per day
> frate <- 0.01/252
> # Margin account
> marginv <- cumsum(retp)
> # Cumulative funding costs
> fcosts <- cumsum(frate*marginv)
> # Add funding costs to margin account
> marginv <- (marginv + fcosts)
> # dygraph plot of margin and funding costs
> datav <- cbind(marginv, fcosts)
> colnamev <- c("Margin", "Cumulative Funding")
> colnames(datav) <- colnamev
> endd <- rutils:::calc_endpoints(datav, interval="weeks")
> 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*.

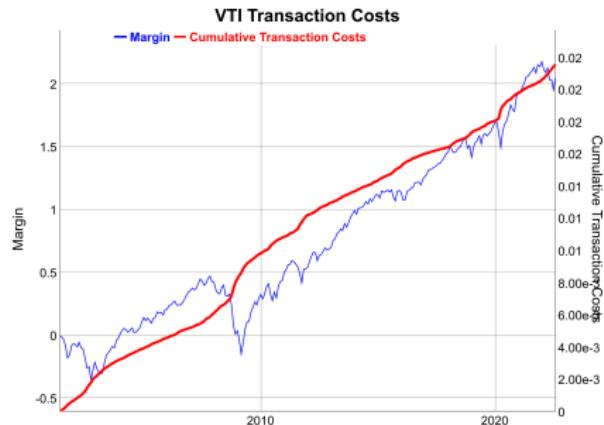
Transaction Costs of Single Asset Rebalancing

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(rtp))/2
> # Subtract transaction costs from margin account
> marginv <- cumsum(rtp)
> marginv <- (marginv - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, costs)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[ennd], 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=300)
```

Combining the Returns of Multiple Assets

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

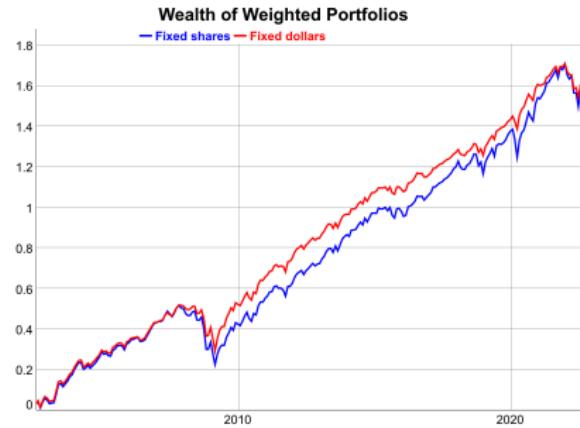
Multiplying the weights times the percentage returns is equivalent to investing in *fixed dollar amounts of stock* proportional to the weights (aka *Fixed Dollar Allocation* or FDA).

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

```
> # Calculate the VTI and IEF dollar returns
> pricev <- rutils::etfenv$prices[, c("VTI", "IEF")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> datev <- zoo::index(pricev)
> # Calculate the VTI and IEF percentage returns
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Wealth of fixed shares equal to $0.5 each (without rebalancing)
> weightv <- c(0.5, 0.5) # dollar weights
> wealthfs <- drop(cumprod(1 + retp) %*% weightv)
> # Or using the dollar returns
> pricessi <- as.numeric(pricev[1, ])
> retd[1, ] <- pricev[1, ]
> wealthfs2 <- cumsum(retd %*% (weightv/pricessi))
```



```
> # Wealth of fixed dollars (with rebalancing)
> wealthfd <- cumsum(retp %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(log(wealthfs), wealthfd)
> 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="weeks")
> dygraphs::dygraph(wealthv[endd], main="Wealth of Weighted Portfolios"
+   + dySeries(name=colnamev[1], col="blue", strokeWidth=2) %>%
+   + dySeries(name=colnamev[2], col="red", strokeWidth=2) %>%
+   + dyLegend(show="always", width=300)
```

Transaction Costs of Weighted Portfolio Rebalancing

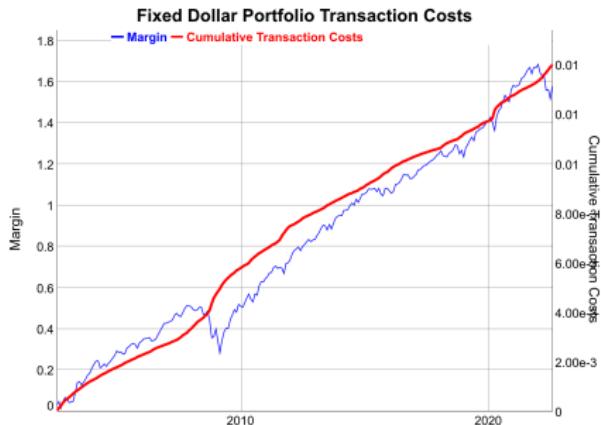
Maintaining a *fixed dollar allocation* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

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: $w_1 |r_t^1| + w_2 |r_t^2|$.

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: $c_t^r = \frac{\delta}{2} (w_1 |r_t^1| + w_2 |r_t^2|)$.

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



```
> # Margin account for fixed dollars (with rebalancing)
> marginv <- cumsum(retp %*% weightv)
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(abs(retp) %*% weightv)/2
> # Subtract transaction costs from margin account
> marginv <- (marginv - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, 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=300)
```

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_t be the percentage returns, ω_i be the portfolio weights, and $\bar{r}_t = \sum_{i=1}^n \omega_i r_t$ 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 $\omega_i r_t$ so it's equal to $\omega_i w_{t-1}(1 + r_t)$, 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:

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

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

Wealth of Proportional Dollar Allocations



```
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- cumsum(retd *%*% weightv/pricesi)
> # Or compound the percentage returns
> wealthfs <- cumprod(1 + retp) *%*% weightv
> # Wealth of proportional allocations (with rebalancing)
> wealthpd <- cumprod(1 + retp *%*% weightv)
> wealthv <- cbind(wealthfs, wealthpd)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Prop dollars")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(utils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> dygraphs::dygraph(wealthv[enda]),
+   main="Wealth of Proportional Dollar Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Transaction Costs With Proportional Dollar Allocations

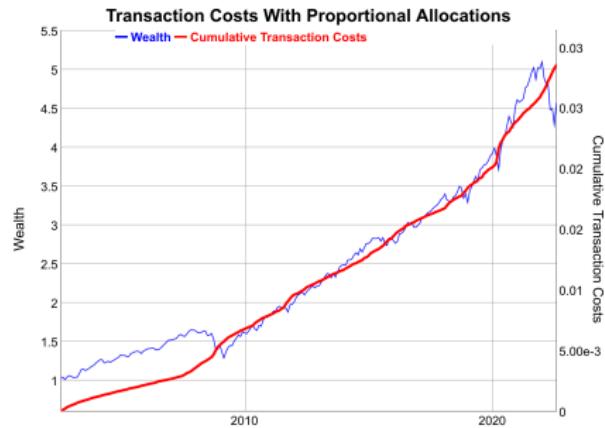
In each period the stocks must be rebalanced to maintain the proportional dollar allocations.

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

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



```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealthpd, 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=300)
```

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: $\omega_i: w_i = \omega_i w_t$.

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)
> wealthfs <- drop(apply(retp, 2, function(x) cumprod(1 + x)) %*% weightv)
> # Wealth of proportional dollar allocations (with rebalancing)
> wealthpd <- cumprod(1 + retp %*% weightv) - 1
> # Wealth of proportional target allocation (with rebalancing)
> retp <- zoo::coredata(retp)
> threshv <- 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(wealthv[it, ])*100 > threshv) {
+     # cat("Rebalance at:", it, "\n")
+     wealthv[it, 1] <- sum(wealthv[it, ])*weightv
+   } # end if
+ } # end for
> wealthv <- rowSums(wealthv) - 1
> wealthv <- cbind(wealthpd, wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Proportional Allocations", "Proportional Target Allocation")
> dygraphs::dygraph(wealthv, main="Wealth of Proportional Target Allocation")
+ dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+ dyLegend(show="always", width=300)
```

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.

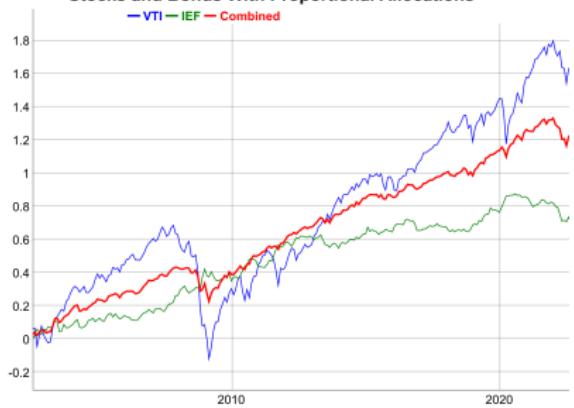
The fixed portfolio weights represent the percentage dollar allocations to stocks and bonds, while the portfolio wealth grows over time.

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 the 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 the correlations
> cor(retp)
> # Calculate the Sharpe ratios
> sqrt(252)*sapply(retp, function(x) mean(x)/sd(x))
> # Calculate the standard deviation, skewness, and kurtosis
> sapply(retp, function(x) {
+   # Calculate the 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
```

Stocks and Bonds With Proportional Allocations



```
> # Wealth of proportional allocations
> wealthv <- cumprod(1 + retp)
> # Calculate the a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> # 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=300)
```

Optimal Stock and Bond Portfolio Allocations

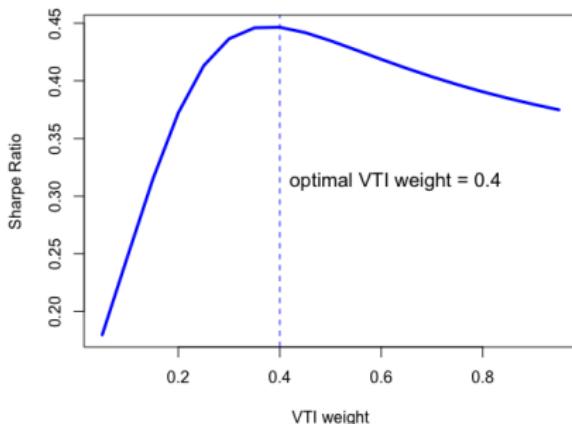
The optimal stock and bond weights can be calculated using optimization.

Using the past 20 years of data, the optimal *VTI* weight is about 0.4.

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
```

Sharpe Ratio as Function of VTI Weight



```
> # 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 the 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 generator
> clusterExport(cluster, c("retp", "nrows"))
> boottd <- 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)
> boottd <- mclapply(1:nboot, function(x) {
+   retp[sample.int(nrows, replace=TRUE), ]
+ }, mc.cores=ncores) # end mclapply
> is.list(boottd); NROW(boottd); dim(boottd[[1]])
```

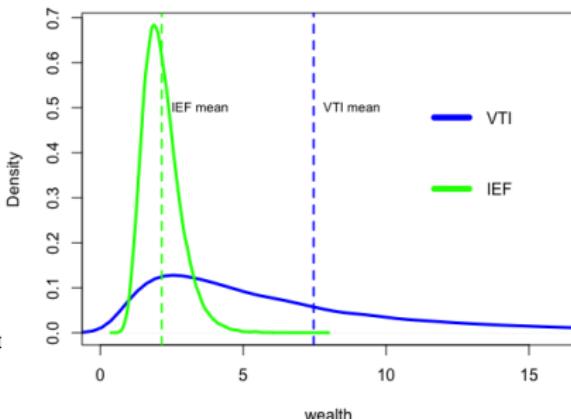
The Distributions of Terminal Wealth From Bootstrap

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 wealths
> apply(wealthv, 2, mean)
> apply(wealthv, 2, sd)
> # Extract the terminal wealths of VTI and IEF
> wealthvti <- wealthv[, "VTI"]
> wealthief <- wealthv[, "IEF"]
```

Terminal Wealth Distributions of VTI and 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).

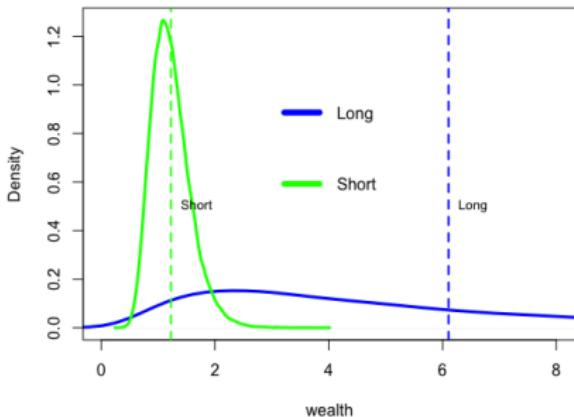
The distribution for long holding periods is highly positively skewed with a much larger mean.

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 <- nrow(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")
```

Wealth Distributions for Long and Short Holding Periods



```
> # 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", v.intersp=0.5,
```

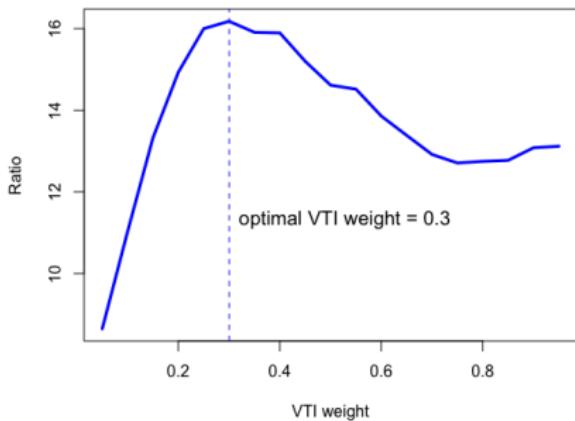
Optimal Stock and Bond Portfolio Allocations From Bootstrap

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 0.3.

```
> # 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)]
```

Portfolio Risk-Return Ratio as Function of VTI Weight



```
> # 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

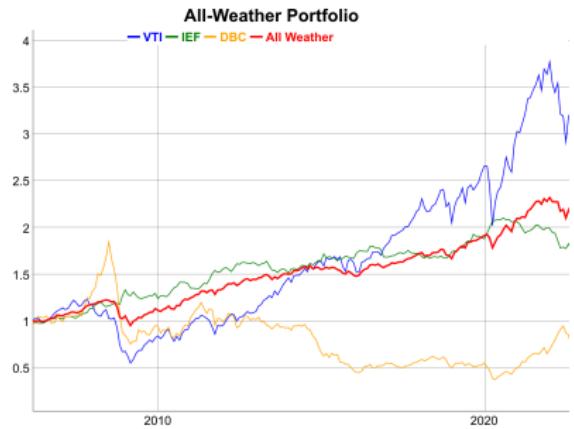
The *All-Weather* portfolio is a portfolio with proportional allocations of stocks (30%), bonds (55%), and commodities and precious metals (15%) (approximately).

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



```
> # Calculate the cumulative wealth from returns
> wealthv <- cumprod(1 + retp)
> # Calculate the a vector of monthly end points
> endd <- rutilis::calc_endpoints(wealthv, interval="weeks")
> # 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=300)
> # 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,
+   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")
```

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 the VTI returns
> retp <- na.omit(rutils::etfenv$returns$VTI["2008/2009"])
> datev <- zoo::index(retp)
> nrows <- NROW(retp)
> retp <- drop(zoo::coredata(retp))
> # 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]*retp[t]
+   stockv[t] <- min(coeff*(portfv[t] - bfloor), portfv[t])
+   bondv[t] <- (portfv[t] - stockv[t])
+ } # end for
> # dygraph plot of CPPI strategy
> pricev <- 100*cumprod(1 + retp)
> datav <- xts::xts(cbind(stockv, bondv, portfv, pricev), 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"), strokeWidth=3)
+   dyLegend(show="always", width=300)
```

Combining the Standardized Returns of Multiple Assets

Standardized returns are returns divided by their volatilities.

Multiplying the weights times the *standardized dollar returns* is equivalent to buying *share amounts* such that their dollar volatilities are proportional to the weights.

Multiplying the weights times the *standardized percentage returns* is equivalent to buying *dollar amounts* such that their percentage volatilities are proportional to the weights.

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

```
> # Calculate the dollar and percentage returns for VTI and IEF
> pricev <- rutils::etfenv$prices[, c("VTI", "IEF")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate the standardized simple dollar returns
> retstd <- lapply(retd, function(x) x/sd(x))
> retstd <- do.call(cbind, retstd)
> retstd <- sapply(retstd, sd)
> # Wealth of fixed number of shares (without rebalancing)
> weightv <- c(0.5, 0.5)
> pricesi <- as.numeric(pricev[1, ])
> wealthfs <- cumsum(retd %*% (weightv/pricesi))
> # Calculate the standardized percentage returns
> retstp <- lapply(retp, function(x) x/sd(x))
> retstp <- do.call(cbind, retstp)
> retstp <- sapply(retstp, sd)
> # Wealth of target dollar allocation of shares (with rebalancing)
> wealthfd <- cumsum(retstp %*% weightv)
> # Plot log wealth
> wealthv <- cbind(wealthfd, log(wealthfs))
> # 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=300)
```

Risk Parity Strategy For Stocks and Bonds

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

This means that the weights w_i are proportional to the *standardized prices* ($\frac{p_i}{\sigma_i^d}$ - the dollar amounts of stocks

with unit dollar volatilities): $w_i \propto \frac{p_i}{\sigma_i^d}$, 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 weights

w_i are proportional to the inverse of the percentage volatilities $w_i \propto \frac{1}{\sigma_i}$.

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

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

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

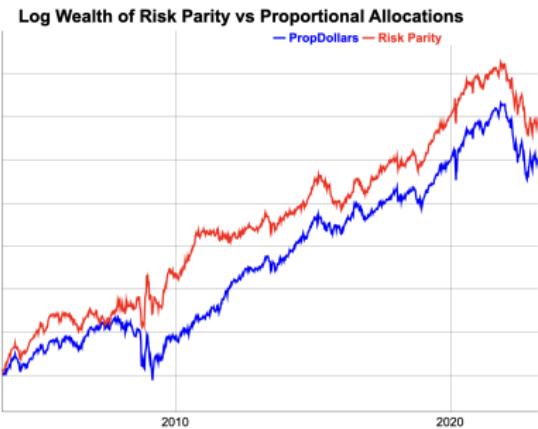
```
> # Calculate the dollar and percentage returns for stocks and bonds
> pricev <- rutils::etfenv$prices[, c("VTI", "TLT")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate the wealth of proportional dollar allocations
> weightv <- c(0.5, 0.5)
> retw <- retp %*% weightv
> wealthpd <- cumprod(1 + retw)
> # Calculate the trailing percentage volatility
> lambda <- 0.15
> volat <- HighFreq::run_var(retp, lambda=lambda)
> volat <- sqrt(volat)
> # Calculate the risk parity portfolio weights
> weightv <- ifelse(volat > 1e-4, 1/volat, 0)
> # Scale weights to 1 dollar total
> weightv <- weightv/rowSums(weightv)
> weightv[1, ] <- 0
> # Lag the weights
> weightv <- rutils::lagit(weightv)
> # Calculate the wealth of risk parity
> retw <- rowSums(retp*weightv)
> wealthrp <- cumprod(1 + retw)
```

Risk Parity Strategy Performance

The risk parity strategy for stocks and bonds has a slightly higher Sharpe ratio, and also higher absolute returns than the proportional dollar strategy, despite the fact that it's more overweight bonds.

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(wealthpd, wealthrp))
> wealthv <- xts::xts(wealthv, zoo::index(pricev))
> colnames(wealthv) <- c("PropDollars", "Risk Parity")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils:::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot a dygraph of the log wealths
> endd <- rutils:::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Log Wealth of Risk Parity vs Proportional Allocations") %
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy Market Timing Skill

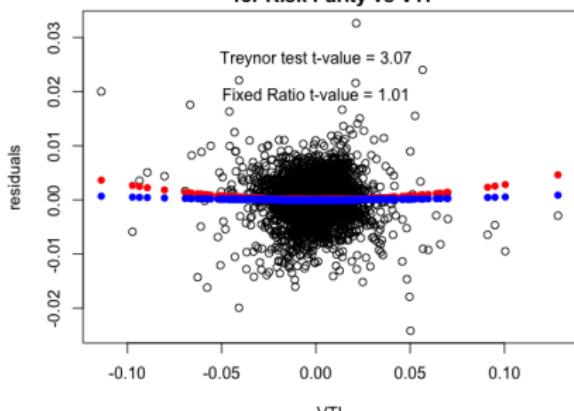
The risk parity strategy reduces allocations to assets with rising volatilities, which is often accompanied by negative returns.

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

**Treynor-Mazuy Market Timing Test
for Risk Parity vs VTI**



```
> # Test for fixed ratio market timing of VTI using Treynor-Mazuy t
> regmod <- lm(fixed ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> points.default(x=retvti, y=fitv, pch=16, col="blue")
> text(x=0.0, y=0.6*max(resid), paste("Fixed Ratio t-value =", round
```

draft: Transaction Costs of Risk Parity Strategy

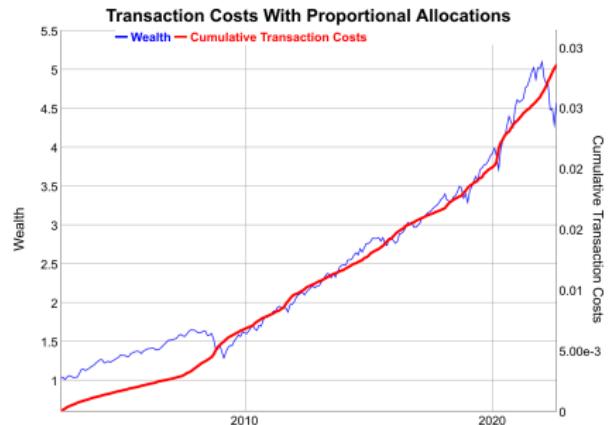
In each period the stocks must be rebalanced to maintain the proportional allocations.

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

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



```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealthpd, 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=300)
```

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 the positions
> retp <- na.omit(rutils::etfenv$returns$VTI)
> posv <- rep(NA_integer_, NROW(retp))
> datev <- zoo::index(retp)
> datev <- format(datev, "%m-%d")
> posv[datev == "05-01"] <- 0
> posv[datev == "05-03"] <- 0
> posv[datev == "11-01"] <- 1
> posv[datev == "11-03"] <- 1
> # Carry forward and backward non-NA posv
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- zoo::na.locf(posv, fromLast=TRUE)
> # Calculate the strategy returns
> pnlinmay <- posv*retp
> wealthv <- cbind(retp, pnlinmay)
> colnames(wealthv) <- c("VTI", "sell_in_may")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*apply(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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Sell in May Strategy"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # 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 May Strategy"
> 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")
```

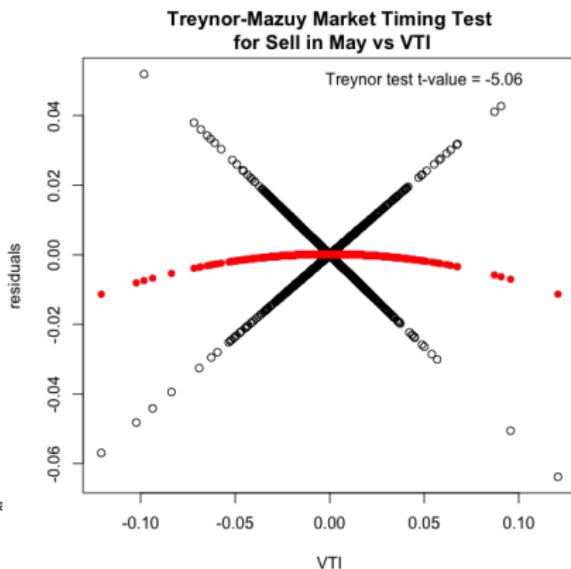
Sell in May Strategy Market Timing

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$sell_in_may, 0.5*(retlp+abs(retlp)), retp^2)
> colnames(desv) <- c("VTI", "merton", "treynor")
> # Perform Merton-Henriksson test
> regmod <- lm(pnlinmay ~ VTI + merton, data=desv)
> summary(regmod)
> # Perform Treynor-Mazuy test
> regmod <- lm(pnlinmay ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot Treynor-Mazuy residual scatterplot
> resids <- (pnlinmay - regmod$coeff["VTI"]*retlp)
> plot.default(x=retlp, y=resids, 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)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retlp
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retlp, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(resids), paste("Treynor test t-value = ", tvalue))

```



Seasonal Overnight Market Anomaly

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

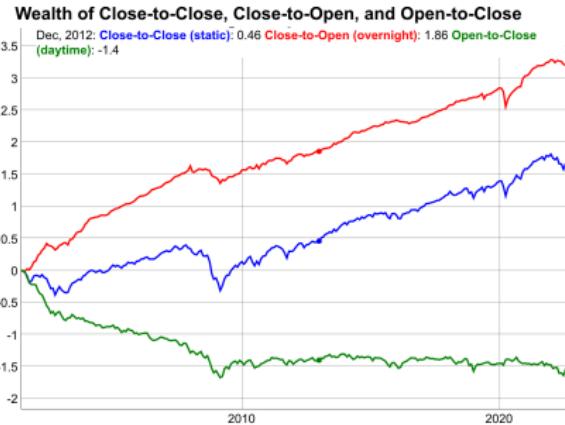
The Overnight Strategy consists of holding a long position only overnight (buying at the close and selling at the open the next day).

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 daytime open-to-close returns
> # and the overnight close-to-open returns.
> clcl <- rutils::diffit(closep)
> colnames(clcl) <- "clcl"
> opcl <- (closep - openp)
> colnames(opcl) <- "daytime"
> close_open <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(close_open) <- "overnight"
```



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(clcl, close_open, opcl)
> 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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Close-to-Close, Overnight, and Daytime Strategies",
+   dySeries(name="clcl", label="Close-to-Close (static)", strokeWeight=2, color="red"),
+   dySeries(name="close_open", label="overnight", strokeWeight=2, color="blue"),
+   dySeries(name="opcl", label="daytime", strokeWeight=2, color="green"),
+   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
> retp <- na.omit(rutls::etfenv$returns$VTI)
> datev <- zoo::index(retp)
> # Calculate the first business day of every month
> dayv <- as.numeric(format(datev, "%d"))
> indeks <- which(rutls::diffit(dayv) < 0)
> datev[head(indeks)]
> # Calculate the 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 the Turn of the Month pnls
> pnls <- numeric(NROW(retp))
> pnls[indeks] <- retp[indeks, ]
```



```
> # Combine data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "TOM Strategy")
> colnames(wealthv) <- colnamev
> # 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])))
> # dygraph plot VTI Turn of the Month strategy
> endd <- rutls::calc_endpoints(wealthv, interval="weeks")
> 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")
```

The Stop-loss Strategy

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

For example, a simple stop-loss rule is to sell the stock if its price drops below the stop-loss level, equal to the stop-loss percentage times the previous maximum price.

The stock is bought back after the price recovers, for example after the price reaches its previous maximum price.

The stop-loss strategy trades a single \$1 of stock, and is either long \$1 of stock or it's flat (\$0 of stock).

The stop-loss strategy is trend-following because it expects prices to continue dropping.

```
> # Calculate the VTI prices and returns
> pricev <- na.omit(rutils::etfenv$prices$VTI)
> nrows <- NROW(pricev)
> datev <- zoo::index(pricev)
> retp <- rutils::diffit(log(pricev))
> # Simulate stop-loss strategy
> stopl <- 0.05 # Stop-loss percentage
> pricem <- cummax(pricev) # Trailing maximum prices
> # Calculate the drawdown
> dd <- (pricev - pricem)
> pnls <- retp # Initialize PnLs
> for (i in 1:(nrows-1)) {
+ # Check for stop-loss
+   if (dd[i] < -stopl*pricem[i])
+     pnls[i+1] <- 0 # Set PnLs = 0 if in stop-loss
+ } # end for
> # Same but without using loops in R
> pnls2 <- retp
> insl <- rutils::lagit(dd < -stopl*pricem)
> pnls2 <- ifelse(insl, 0, pnls2)
> all.equal(pnls, pnls2, check.attributes=FALSE)
```

Stop-loss Strategy Performance

The stop-loss strategy underperforms because it waits too long for prices to recover.

And the stop-loss strategy also underperforms because it gets "*whipsawed*" when prices are range-bound without a trend.

When prices are range-bound without a trend, the stop-loss strategy often stops because of a drawdown (goes flat risk), but if the prices soon rebound, then it's forced to buy back the stock. (This is called a "*whipsaw*".)

```
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # 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])))
> # dygraph plot the stop-loss strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Stop-loss Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```



```
> # Plot dygraph with shading
> # Create colors for background shading
> indic <- (rutils::diffit(insl) != 0) # Indices of stop-loss
> crossd <- c(datev[indic], datev[nrows]) # Dates of stop-loss
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="VTI Stop-loss Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # 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
```

Optimal Stop-loss Strategy

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

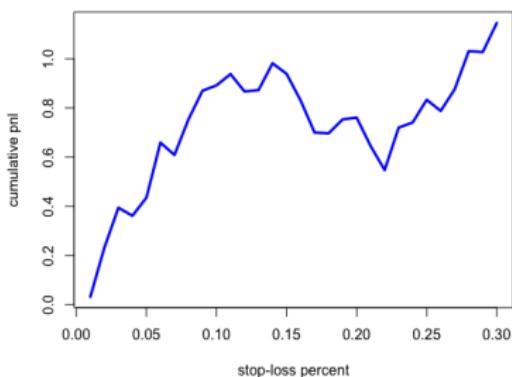
The best performing stop-loss strategy for *VTI* has the largest stop-loss percentage - i.e. it almost never enters into a stop-loss.

That's because *VTI* has had positive returns in the last 20 years, so a stop-loss rule had little benefit.

That's why many quantitative investment funds do not use stop-loss rules if they have strong convictions that the stocks will have positive returns.

```
> # Simulate multiple stop-loss strategies
> dd <- (pricev - pricem)
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   pnls <- rtrp
+   insl <- rutils::lagit(dd < -stopl*pricem)
+   pnls <- ifelse(insl, 0, pnls)
+   sum(pnls)
+ }) # end sapply
```

Cumulative PnLs for Stop-loss Strategies



```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=stopv, y=pnlc,
+       main="Cumulative PnLs for Stop-loss Strategies",
+       xlab="stop-loss percent", ylab="cumulative pnl",
+       t="l", lwd=3, col="blue")
```

Stop-Start Strategy

The stop-loss strategy can be improved by introducing a start-gain rule: buy back the stock if its price rebounds from the previous minimum and exceeds the start-gain level.

The stop-start strategy implements both stop-loss events due to price drawdowns and start-gain events due to price draw-ups.

In order to determine the stop-loss and start-gain events, the stop-start strategy follows the trailing maximum and trailing minimum prices.

A start-gain event is when the stock price rebounds from the previous minimum and exceeds the start-gain level, equal to the start-gain percentage times the previous minimum price.

After the start-gain level is crossed, the strategy buys back the stock which was sold under the stop-loss.

The stop-start strategy is trend-following because it profits if price trends are persistent. But it loses if prices are range-bound.

```
> # Define function for simulating a stop-start strategy
> sim_stopstart <- function(stopl) {
+   maxp <- pricev[1] # Trailing maximum price
+   minp <- pricev[1] # Trailing minimum price
+   insl <- FALSE # Is in stop-loss?
+   insg <- FALSE # Is in start-gain?
+   pnls <- rep(0, length(pricev)) # Initialize PnLs
+   for (i in 1:length(pricev)) {
+     if (insl) { # In stop-loss
+       pnls[i] <- 0 # Set PnLs = 0 if in stop-loss
+       minp <- min(minp, pricev[i]) # Update minimum price to current price
+       if (pricev[i] > ((1 + stopl)*minp)) { # Check for start-gain
+         insg <- TRUE # Is in start-gain?
+         insl <- FALSE # Is in stop-loss?
+         maxp <- pricev[i] # Reset trailing maximum price
+       } # end if
+     } else if (insg) { # In start-gain
+       maxp <- max(maxp, pricev[i]) # Update maximum price to current price
+       if (pricev[i] < ((1 - stopl)*maxp)) { # Check for stop-loss
+         insl <- TRUE # Is in stop-loss?
+         insg <- FALSE # Is in start-gain?
+         minp <- pricev[i] # Reset trailing minimum price
+       } # end if
+     } # end else { # Warmup period
+   } # Update the maximum and minimum prices
+   maxp <- max(maxp, pricev[i])
+   minp <- min(minp, pricev[i])
+   # Update the stop-loss and start-gain indicators
+   insl <- (pricev[i] < ((1 - stopl)*maxp)) # Is in stop-loss?
+   insg <- (pricev[i] > ((1 + stopl)*minp)) # Is in start-gain?
+ } # end if
+ } # end for
+ return(pnls)
+ } # end sim_stopstart
```

Stop-Start Strategy Performance

The stop-start strategy spends less time in a stop-loss because prices tend to rebound sharply after a steep loss.

The start-gain rule tends to quickly override the stop-loss rule, so that the strategy is long the stock after it recovers after a stop-loss.

```
> # Simulate stop-start strategy
> pnls <- sim_stopstart(0.1)
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # 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])))
> # dygraph plot the stop-loss strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```



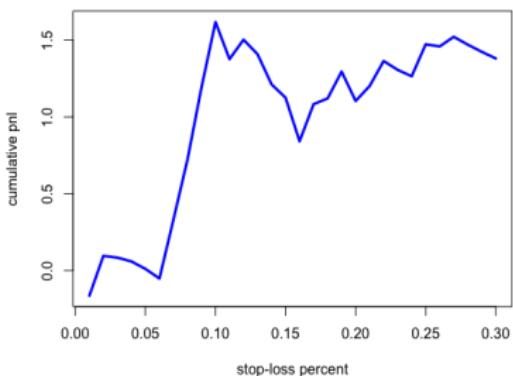
```
> # Plot dygraph with shading
> # Create colors for background shading
> insl <- (pnls == 0) # Is in stop-loss?
> indic <- (rutils::diffit(insl) != 0) # Indices of crosses
> crossd <- c(datev[indic], datev[nrows]) # Dates of crosses
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="VTI Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # 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
```

Optimal Stop-Start Strategy

The stop-start strategy performs much better than the stop-loss strategy because it captures stock gains.

```
> # Simulate multiple stop-loss strategies
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   sum(sim_stopstart(stopl))
+ }) # end sapply
> stop1 <- stopv[which.max(pnlc)]
```

Cumulative PnLs for Stop-Start Strategies



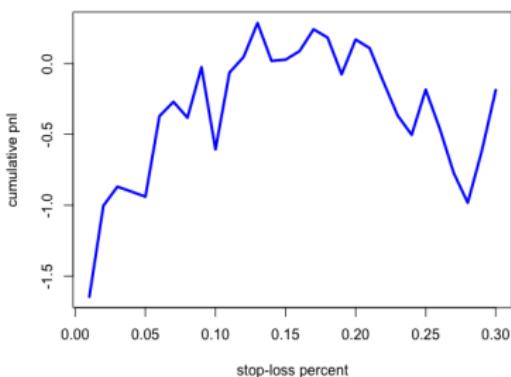
```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=stopv, y=pnlc,
+       main="Cumulative PnLs for Stop-Start Strategies",
+       xlab="stop-loss percent", ylab="cumulative pnl",
+       t="l", lwd=3, col="blue")
```

Stop-Start Strategy for Other ETFs

The stop-start strategy can prevent losses for stocks with significant negative returns, like the *USO* ETF (oil fund).

```
> # Calculate the USO prices and returns
> pricev <- na.omit(rutils::etfenv$prices$USO)
> nrowv <- NROW(pricev)
> datev <- zoo::index(pricev)
> retp <- rutils::diffit(log(pricev))
> # Simulate multiple stop-start strategies
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   sum(sim_stopstart(stopl))
+ }) # end sapply
```

Cumulative PnLs for USO Stop-Start Strategies



```
> # Plot cumulative pns for stop-start strategies
> plot(x=stopv, y=pnlc,
+       main="Cumulative PnLs for USO Stop-Start Strategies",
+       xlab="stop-loss percent", ylab="cumulative pnl",
+       t="l", lwd=3, col="blue")
```

Optimal Stop-Start Strategy For USO

The stop-start strategy for the *USO* ETF has performed well because *USO* has had very negative returns in the last 20 years.

Stop-start strategies are able to preserve profits for the best performing stocks, and avoid losses for the worst performing stocks.

```
> # Simulate optimal stop-start strategy for USO
> stopl <- stopy[which.max(pnls)]
> pnls <- sim_stopstart(stopl)
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("USO", "Strategy")
> colnames(wealthv) <- colnamev
> # 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])))
> # dygraph plot the stop-start strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="USO Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```



```
> # Plot dygraph with shading
> # Create colors for background shading
> insl <- (pnls == 0) # Is in stop-loss?
> indic <- (rutils::diffit(insl) != 0) # Indices of crosses
> crossd <- c(datev[indic], datev[nrows]) # Dates of crosses
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="USO Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # 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
```

EWMA Price Technical Indicator

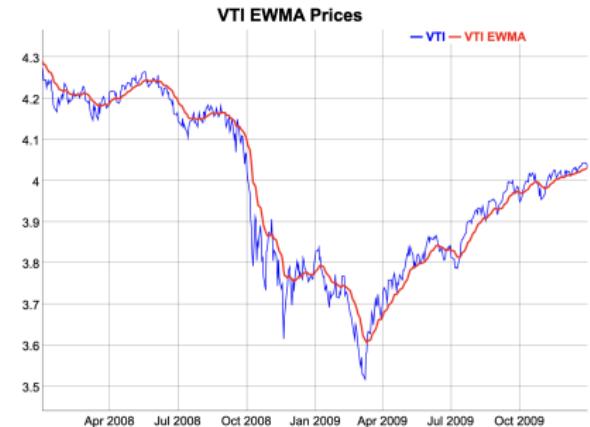
The *Exponentially Weighted Moving Average Price (EWMA)* is defined as the weighted average of prices over a rolling interval:

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

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 prices, and vice versa.

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

```
> # Extract the log VTI prices
> ohlc <- log(quantmod::etfenv$VTI)
> closep <- quantmod::Cl(ohlc)
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> # Calculate the EWMA weights
> look_back <- 111
> lambda <- 0.9
> weightv <- lambda^(0:look_back)
> weightv <- weightv/sum(weights)
> # Calculate the EWMA prices as a convolution
> ewmacpp <- HighFreq::roll_sumw(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], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+   dyLegend(show="always", width=300)
> # 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 prices, 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 the EWMA prices recursively using C++ code
> ewmar <- .Call(stats:::C_rfilter, closep, lambda, c(as.numeric(c(
> # Or R code
> # ewmar <- filter(closep, filter=lambda, init=as.numeric(closep[1])
> ewmar <- (1-lambda)*ewmar
> # Calculate the 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(
+     time=10)))[1:4], 5))
Jerzy Pawlowski (NYU Tandon)
```



```
> # 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 Prices")
+   dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+   dyLegend(show="always", width=300)
> # 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")
```

The EWMA Crossover Strategy

The trend following *EWMA Crossover* strategy switches its stock 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 of stock, and if it is below, to short \$1 dollar of stock.

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 of stock or short \$1 dollar of stock.

```
> # Calculate the EWMA prices recursively using C++ code
> lambda <- 0.984
> ewmacpp <- HighFreq::run_mean(clossep, lambda=lambda)
> # Calculate the positions, either: -1, 0, or 1
> indic <- sign(clossep - ewmacpp)
> posv <- rutils::lagit(indic, lagg=1)
> # Create colors for background shading
> crosssd <- (rutils:::diffit(posv) != 0)
> shadev <- posv[crosssd]
> crosssd <- c(zoo::index(shadev), end(posv))
> 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], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=3, col="red") %>%
+   dyLegend(show="always", width=300)
```



```
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crosssd[i], to=crosssd[i+1],
+ } # 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(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 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")
```

EWMA Crossover Strategy Performance

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 the daily profits and losses of EWMA strategy
> retp <- rutils::diffrt(closep) # VTI returns
> pnls <- retp*posv
> colnames(pnls) <- "EWMA"
> wealthv <- cbind(retp, 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=300)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+ dyplot <- dyplot %>%
+ dyShading(from=crosssd[i], to=crosssd[i+1], color=shadev[i])
+ } # end for
> # Plot the dygraph object
> dyplot
```



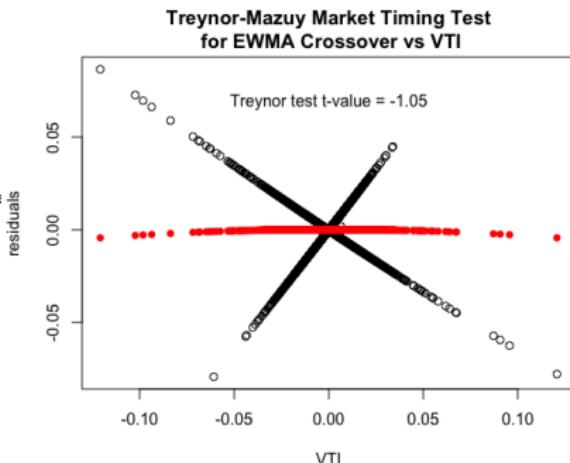
```
> # Standard plot of EWMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+ name="Performance of EWMA Strategy")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 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 test
> desv <- cbind(pnls, retp, retp^2)
> desv <- na.omit(desv)
> colnames(desv) <- c("EWMA", "VTI", "treynor")
> regmod <- lm(EWMA ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot residual scatterplot
> residss <- (desv$EWMA - regmod$coeff["VTI"]*retp)
> residuals <- regmod$residuals
> plot.default(x=retp, y=residss, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\nfor EWMA Crossover")
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retp
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retp, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(residss), paste("Treynor test t-value =", tvalue))
```



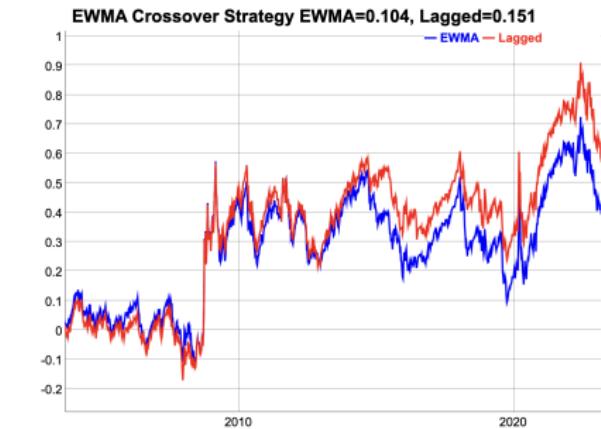
EWMA Crossover Strategy With Lag

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 the positions from lagged indicator
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> # Calculate the positions, either: -1, 0, or 1
> posv <- rep(NA_integer_, nrow)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the PnLs of lagged strategy
> pnlslag <- rep*posv
> 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
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=paste("EWMA Crossover"))
+ dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+ dyLegend(show="always", width=300)
```

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 the positions, either: -1, 0, or 1
> indic <- sign(clossep - ewmacpp)
> posv <- rutils::lagit(indic, lagg=1)
> # Calculate the daily pnl for days without trades
> pnls <- retp*posv
> # Determine trade dates right after EWMA has crossed prices
> crosssd <- which(rutils::diffit(posv) != 0)
> # Calculate the realized pnl for days with trades
> openp <- quantmod::Op(ohlc)
> closelag <- rutils::lagit(clossep)
> poslag <- rutils::lagit(posv)
> pnls[crosssd] <- poslag[crosssd]*(openp[crosssd] - closelag[crosssd])
> # Calculate the unrealized pnl for days with trades
> pnls[crosssd] <- pnls[crosssd] +
+ posv[crosssd]*(closelag[crosssd] - openp[crosssd])
> # Calculate the wealth
> wealthv <- cbind(retp, 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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="EWMA Strategy Trading at the Open Price")
+ dyOptions(colors=colrv, strokeWidth=2) %>%
+ dyLegend(show="always", width=300)
> # 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 the transaction costs
> costs <- 0.5*bid_offer*abs(poslag - posv)
> # 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=300)
```

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

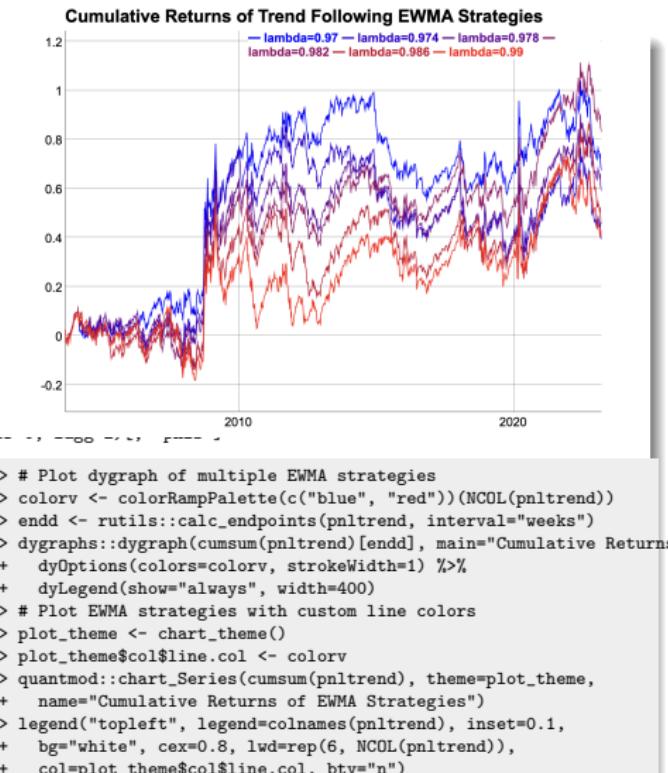
Simulating Multiple Trend Following EWMA Strategies

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=0.97, to=0.99, by=0.004)
> # Perform lapply() loop over lambdav
> pnltrend <- lapply(lambdav, function(lambda) {
+   # Simulate EWMA strategy and Calculate the returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, look_back=look_back, bid_offset=0, ask_offset=0, r=0)
+ }) # end lapply
> pnltrend <- do.call(cbind, pnltrend)
> colnames(pnltrend) <- paste0("lambda=", lambdav)
```



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 the returns
+   sim_ewma(ohlc=ohlc, lambda=lambda, look_back=look_back)[, "pnls"]
+ }) # end parLapply
> stopCluster(cluster) # Stop R processes over cluster under Windows
> # Perform parallel loop over lambdav under Mac-OSX or Linux
> pnltrend <- mclapply(lambdav, function(lambda) {
+   library(quantmod)
+   # Simulate EWMA strategy and Calculate the 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.

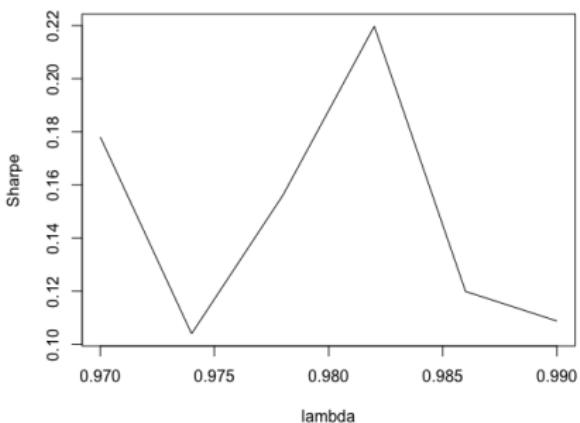
The *Sharpe ratios* of *EWMA* strategies with different λ parameters can be calculated by performing an `sapply()` loop over the *columns* of returns.

`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 the 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=lambda, y=sharpetrend, t="l",
+       xlab="lambda", ylab="Sharpe",
+       main="Performance of EWMA Trend Following Strategies
+             as Function of the Decay Parameter Lambda")
```

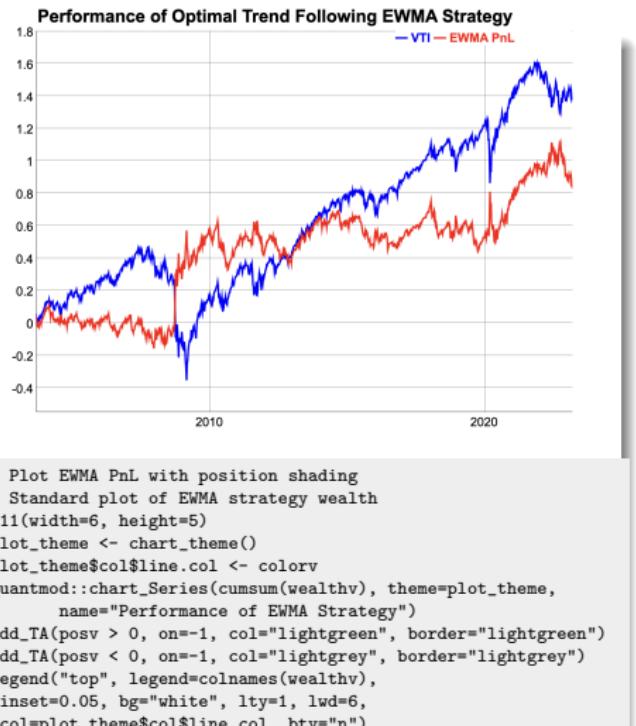
Performance of EWMA Trend Following Strategies as Function of the Decay Parameter Lambda



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 price), and producing less frequent trading.

```
> # Calculate the optimal lambda
> lambda <- lambda[which.max(sharpetrend)]
> # Simulate best performing strategy
> ewmatrend <- sim_ewma(ohlc=ohlc, lambda=lambda, bid_offer=0, lagg=0)
> posv <- ewmatrend[, "positions"]
> trendopt <- ewmatrend[, "pnls"]
> wealthv <- cbind(retp, 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)[end], main="Performance of Optimal Trend Following EWMA Strategy")
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```



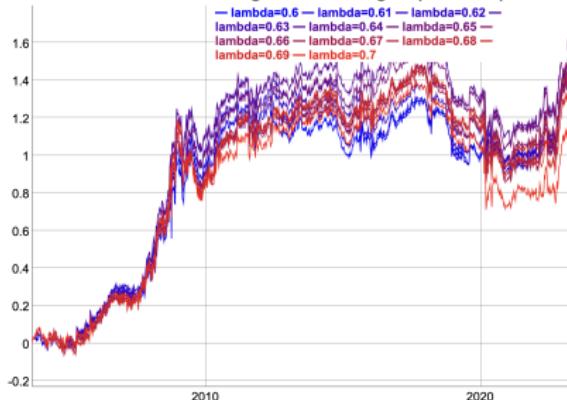
Mean Reverting EWMA Crossover Strategies

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

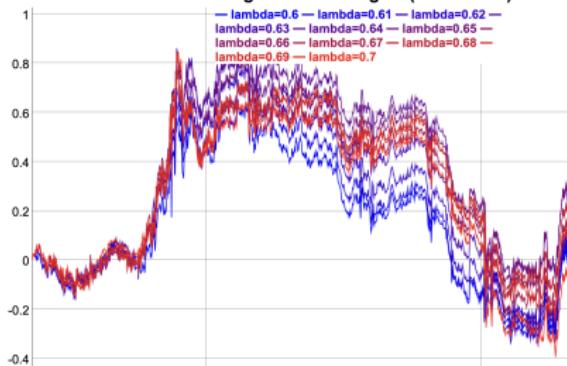
The profitability of mean reverting strategies can be significantly improved by using limit orders, to reduce transaction costs.

```
> 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 the 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="Returns of Mean I"
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
> # 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 EW!
> legend("topleft", legend=colnames(pnlrevert),
+   inset=0.1, bg="white", cex=0.8, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```

Returns of Mean Reverting EWMA Strategies (No Costs)



Returns of Mean Reverting EWMA Strategies (With Costs)



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.

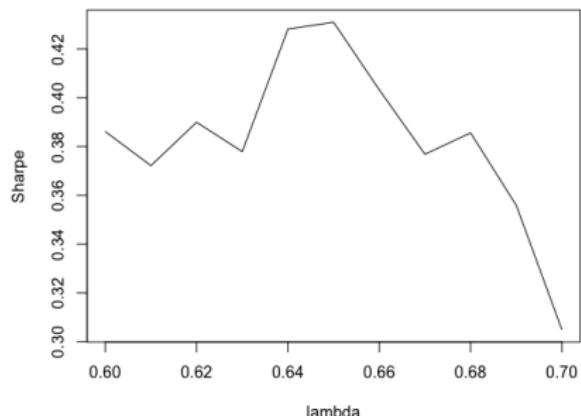
Performing loops in R over the *columns* of returns is acceptable, but R loops over the *rows* of returns should be avoided.

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.

Performance of EWMA Mean Reverting Strategies as Function of the Decay Parameter Lambda



```
> # Calculate the Sharpe ratios of strategy returns
> sharparevert <- sqrt(252)*sapply(pnlrevert, function(xtsv) {
+   mean(xtsv)/sd(xtsv)
+ }) # end sapply
> plot(x=lambda, y=sharparevert, 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 prices), and producing more frequent trading.

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

```
> # Calculate the optimal lambda
> lambda <- lambda[which.max(sharpererevert)]
> # Simulate best performing strategy
> ewmarevert <- sim_ewma(ohlc=ohlc, bid_offer=0.0,
+   lambda=lambda, trend=(-1))
> posv <- ewmarevert[, "positions"]
> revertopt <- ewmarevert[, "pnls"]
> wealthv <- cbind(retp, revertopt)
> colnames(wealthv) <- c("VTI", "EWMA PnL")
> # Plot dygraph of EWMA strategy wealth
> colrv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Optimal Mean Reve:
+   dyOptions(colors=colrv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```



```
> # Standard plot of EWMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colrv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+   name="Optimal Mean Reverting EWMA Strategy")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 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 the correlation between trend following and mean reverting
> trendopt <- ewmatrend[, "pnls"]
> colnames(trendopt) <- "trend"
> revertopt <- ewmarevert[, "pnls"]
> colnames(revertopt) <- "revert"
> cor(cbind(retp, trendopt, revertopt))
> # Calculate the combined strategy
> combstrat <- (retlp + trendopt + revertopt)/3
> colnames(combstrat) <- "combined"
> # Calculate the annualized Sharpe ratio of strategy returns
> retc <- cbind(retlp, trendopt, revertopt, combstrat)
> colnames(retc) <- c("VTI", "Trending", "Reverting", "Combined")
> sqrt(252)*sapply(retc, function(xtsv) mean(xtsv)/sd(xtsv))
```



```
> # Plot dygraph of EWMA strategy wealth
> colorv <- c("blue", "red", "green", "purple")
> dygraphs::dygraph(cumsum(retc)[endd], main="Performance of Combined EWMA Strategies")
+ dyOptions(colors=colorv, strokeWidth=1) %>%
+ dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+ dyLegend(show="always", width=300)
> # 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 the weights proportional to Sharpe ratios
> weightvt <- c(sharpetrend, sharprevert)
> weightvt[weightvt < 0] <- 0
> weightvt <- weightvt/sum(weights)
> retc <- cbind(pnltrend, pnlrevert)
> retc <- retc %*% weightvt
> retc <- cbind(retp, retc)
> colnames(retc) <- c("VTI", "EWMA PnL")
> # Plot dygraph of EWMA strategy wealth
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(retc)[endd], main="Performance of Ensemble of EWMA Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Standard plot of EWMA strategy wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(retc), 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")
```

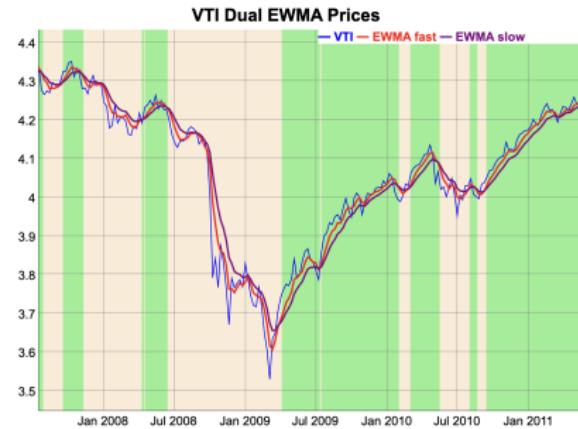


Simulating the Dual EWMA Crossover Strategy

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

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

```
> # Calculate the fast and slow EWMA
> lambdaf <- 0.89
> lambdas <- 0.95
> # Calculate the EWMA prices
> ewmaf <- HighFreq::run_mean(closep, lambda=lambdaf)
> ewmas <- HighFreq::run_mean(closep, lambda=lambdas)
> # Calculate the EWMA prices
> pricev <- cbind(closep, ewmaf, ewmas)
> colnames(pricev) <- c("VTI", "EWMA fast", "EWMA slow")
> # Calculate the positions, either: -1, 0, or 1
> indic <- sign(ewmaf - ewmas)
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> posv <- rutils::lagit(posv, lagg=1)
```



```
> # Create colors for background shading
> crossd <- (rutils:::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen", "lightpink")
> # Plot dygraph
> colnamev <- colnames(pricev)
> dyplot <- dygraphs::dygraph(pricev[,endd], main="VTI Dual EWMA Prices")
+ dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+ dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+ dySeries(name=colnamev[3], strokeWidth=2, col="purple") %>%
+ dyLegend(show="always", width=300)
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], color=shadev[i])
+ } # end for
> dyplot
```

Dual EWMA Crossover Strategy Performance

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 the daily profits and losses of strategy
> pnls <- retp*posv
> colnames(pnls) <- "Strategy"
> wealthv <- cbind(retp, 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]
```

EWMA Dual Crossover Strategy, Sharpe VTI=0.37, Strategy=0.222



```
> # Plot Dual EWMA strategy
> dyplot <- dygraphs::dygraph(cumsum(wealthv)[endd], main=paste("EWMA", "Dual Crossover Strategy"))
+   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], fill=TRUE)
+ } # 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, lambdaf=0.1, lambdas=0.01,
+                         bid_offer=0.001, trend=1, lagg=1) {
+   if (lambdaf >= lambdas) return(NA)
+   closep <- quantmod::Cl(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate the EWMA prices
+   ewmaf <- HighFreq::run_mean(closep, lambda=lambdaf)
+   ewmas <- HighFreq::run_mean(closep, lambda=lambdas)
+   # Calculate the positions, either: -1, 0, or 1
+   indic <- sign(ewmaf - ewmas)
+   if (lagg > 1) {
+     indic <- HighFreq::roll_sum(indic, lagg)
+     indic[1:lagg] <- 0
+   } # end if
+   posv <- rep(NA_integer_, nrows)
+   posv[1] <- 0
+   posv <- ifelse(indic == lagg, 1, posv)
+   posv <- ifelse(indic == (-lagg), -1, posv)
+   posv <- zoo::na.locf(posv, na.rm=FALSE)
+   posv <- xts::xts(posv, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posv <- rutils::lagit(posv, lagg=1)
+   # Calculate the PnLs of strategy
+   pnls <- retp*posv
+   costs <- 0.5*bid_offer*abs(rutils::diffit(posv))
+   pnls <- (pnls - costs)
+   # Calculate the strategy returns
+   pnls <- cbind(posv, 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")
> lambdafv <- seq(from=0.85, to=0.99, by=0.01)
> lambdasv <- seq(from=0.85, to=0.99, by=0.01)
> # Calculate the Sharpe ratio of dual EWMA strategy
> calc_sharpe <- function(ohlc, lambdaf, lambdas, bid_offer, trend,
+   if (lambdaf >= lambdas) return(NA)
+   pnls <- sim_ewma2(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+   bid_offer=bid_offer, trend=trend, lagg=lagg)[, "pnls"]
+   sqrt(252)*mean(pnls)/sd(pnls)
+ } # end calc_sharpe
> # Vectorize calc_sharpe with respect to lambdaf and lambdas
> calc_sharpe <- Vectorize(FUN=calc_sharpe,
+   vectorize.args=c("lambdaf", "lambdas"))
> # Calculate the matrix of PnLs
> sharphem <- outer(lambdafv, lambdasv, FUN=calc_sharpe, ohlc=ohlc,
+   bid_offer=0.0, trend=1, lagg=2)
> # Or perform two sapply() loops over lambda vectors
> sharphem <- sapply(lambdasv, function(lambdas) {
+   sapply(lambdafv, function(lambdaf) {
+     if (lambdaf >= lambdas) return(NA)
+     calc_sharpe(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+     bid_offer=0.0, trend=1, lagg=2)
+   }) # end sapply
+ }) # end sapply
> colnames(sharphem) <- lambdasv
> rownames(sharphem) <- lambdafv
```

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(sharperm == max(sharperm, na.rm=TRUE), arr.ind=TRUE)
> lambdaf <- lambdafv[whichv[1]]
> lambdas <- lambdasv[whichv[2]]
> ewma_opt <- sim_ewma2(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+   bid_offer=0.0, trend=1, lagg=2)
> pnls <- ewma_opt[, "pnls"]
> wealthv <- cbind(retp, pnls)
> colnames(wealthv)[2] <- "EWMA"
> # 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])))
> # 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
> posv <- ewma_opt[, "positions"]
> crossd <- (rutils:::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo:::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo:::coredata(shadev)) == 1, "lightgreen",
+   "lightorange")
> # Plot Optimal Dual EWMA strategy
> dyplot <- dygraphs:::dygraph(cumsum(wealthv), main=paste("Optimal Dual EWMA Strategy, Sharpe VTI=0.37, EWMA=0.352"))
> 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],
+     fill=shadev[i])
+ } # 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 the matrix of PNLs
> sharpmem <- outer(lambdadafv, lambdaDasv,
+                      FUN=calc_sharpe, ohlc=ohlc[insample, ],
+                      bid_offer=0.0, trend=1, lagg=2)
> colnames(sharpmem) <- lambdaDasv
> rownames(sharpmem) <- lambdadafv
> # Calculate the PNLs for the optimal strategy
> whichv <- which(sharpmem == max(sharpmem, na.rm=TRUE), arr.ind=TRUE)
> lambdaDaf <- lambdadafv[whichv[1]]
> lambdaDas <- lambdaDasv[whichv[2]]
> pnls <- sim_ewma2(ohlc=ohlc, lambdaDaf=lambdaDaf, lambdaDas=lambdaDas,
+                     bid_offer=0.0, trend=1, lagg=2)[, "pnls"]
> wealthv <- cbind(retP, pnls)
> colnames(wealthv)[2] <- "EWMA"
> # Calculate the 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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual EWMA Strategy",
+   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 the 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 <- HighFreq::roll_sum(closep*volum, look_back)
> volumr <- HighFreq::roll_sum(volum, look_back)
> vwap <- vwap/volumr
> colnames(vwap) <- "VWAP"
> pricev <- cbind(closep, vwap)
```



```
> # Dygraphs plot with custom line colors
> colrv <- c("blue", "red")
> dygraphs::dygraph(pricev["2009"], main="VTI VWAP Prices") %>%
+   dyOptions(colors=colrv, strokeWidth=2)
> # Plot VWAP prices with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colrv
> 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:

$$\mu_t^V = \lambda \mu_{t-1}^V + (1 - \lambda) v_t$$

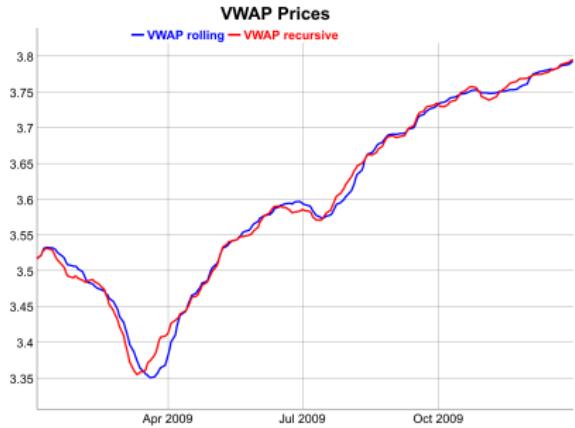
$$\mu_t^{PV} = \lambda \mu_{t-1}^{PV} + (1 - \lambda) v_t p_t$$

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 the VWAP prices recursively using C++ code
> lambda <- 0.9
> volumer <- .Call(stats:::C_rfilter, volum, lambda, c(as.numeric(volum)))
> pricer <- .Call(stats:::C_rfilter, volum*clossep, lambda, c(as.numeric(clossep)))
> vwapr <- pricer/volumer
> # Calculate the VWAP prices recursively using RcppArmadillo
> vwapcpp <- HighFreq::run_mean(clossep, lambda=lambda, weightv=volum)
> all.equal(vwapr, drop(vwapcpp))
> # Dygraphs plot the VWAP prices
> pricev <- xts(cbind(vwapr, vwapcpp), 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=300)
```

Simulating the VWAP Crossover Strategy

In the trend following *VWAP Crossover* strategy, the stock 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 stock 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 the VWAP prices recursively using RcppArmadillo
> lambda <- 0.99
> vwapcpp <- HighFreq::run_mean(closep, lambda=lambda, weightv=volum
> # Calculate the positions from lagged indicator
> indic <- sign(closep - vwapcpp)
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> # Calculate the positions, either: -1, 0, or 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the PnLs of VWAP strategy
> retp <- rutils::dfdiff(closep) # VTI returns
> pnls <- retp*posv
> colnames(pnls) <- "VWAP"
> wealthv <- cbind(retp, pnls)
> 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:::dfdiff(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo:::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo:::coredata(shadev)) == 1, "lightgreen", "red")
> # Plot dygraph of VWAP strategy
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main=paste("VWAP Crossover Strategy, Sharpe VTI=", sharper[1], ", VWAP=", sharper[2]))
> + dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
> + dyLegend(show="always", width=300)
> # 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
```

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

VWAP Strategy Sharpe VTI=0.37, VWAP=0.256, Combined=0.488



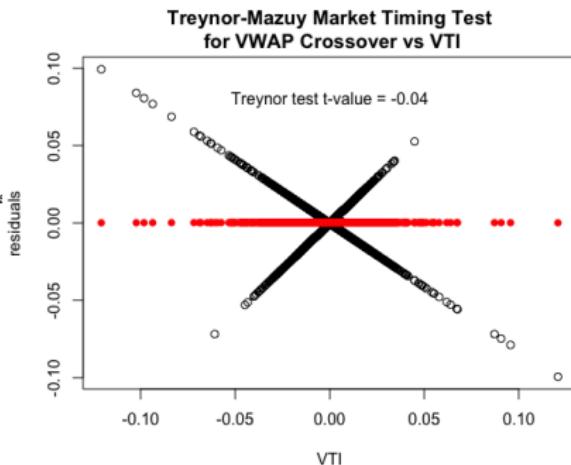
```
> # Plot dygraph of VWAP strategy combined with VTI
> colorv <- c("blue", "red", "purple")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   paste("VWAP Strategy Sharpe", paste(paste(names(sharper)), round(
+     sharper, 2), sep=","), sep=""))
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

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 test
> desv <- cbind(pnls, retp, retp^2)
> desv <- na.omit(desv)
> colnames(desv) <- c("VWAP", "VTI", "treynor")
> regmod <- lm(VWAP ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot residual scatterplot
> resids <- (desv$VWAP - regmod$coeff["VTI"]*retp)
> resids <- regmod$residuals
> # x11(width=6, height=6)
> plot.default(x=retp, y=resids, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\nfor VWAP Crossover")
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retp
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retp, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(resids), 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, lagg=1) {
+   closep <- log(quantmod::Cl(ohlc))
+   volum <- quantmod::Vo(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate the VWAP prices
+   vwap <- HighFreq::run_mean(closep, lambda=lambda, weightv=volum)
+   # Calculate the indicator
+   indic <- trend*sign(closep - vwap)
+   if (lagg > 1) {
+     indic <- HighFreq::roll_sum(indic, lagg)
+     indic[1:lagg] <- 0
+   } # end if
+   # Calculate the positions, either: -1, 0, or 1
+   posv <- rep(NA_integer_, nrows)
+   posv[1] <- 0
+   posv <- ifelse(indic == lagg, 1, posv)
+   posv <- ifelse(indic == (-lagg), -1, posv)
+   posv <- zoo::na.locf(posv, na.rm=FALSE)
+   posv <- xts::xts(posv, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posv <- rutils::lagit(posv, lagg=1)
+   # Calculate the PnLs of strategy
+   pnls <- retp*posv
+   costs <- 0.5*bid_offer*abs(rutils::diffit(posv))
+   pnls <- (pnls - costs)
+   # Calculate the strategy returns
+   pnls <- cbind(posv, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ } # end sim_vwap
```

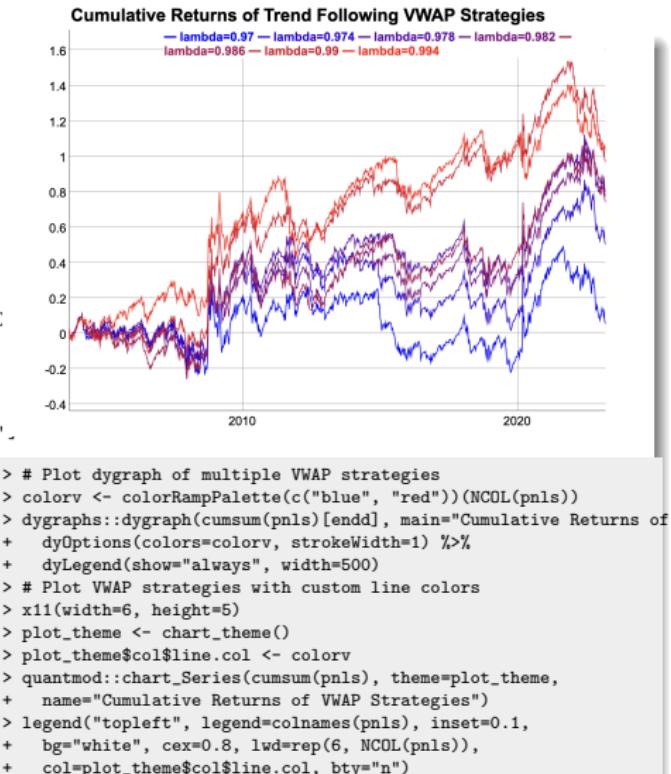
Simulating Multiple Trend Following VWAP Strategies

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=0.004)
> # Perform lapply() loop over lambdav
> pnls <- lapply(lambdav, function(lambda) {
+   # Simulate VWAP strategy and Calculate the 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)
```



Bollinger Bands

The *Bollinger Bands* improve the moving average feature by adding information about the volatility.

The Bollinger Bands are three time series, with the middle band equal to the trailing mean prices, the upper band equal to the mean prices plus the trailing standard deviation, and the lower band equal to the mean prices minus the standard deviation.

The Bollinger Bands are often used to indicate that prices are cheap if they are below the lower band, and rich (expensive) if they are above the upper band.

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

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

```
> # Extract the log VTI prices
> pricev <- log(na.omit(rutils::etfenv$prices$VTI))
> nrow <- NROW(pricev)
> # Calculate the trailing mean prices
> lambda <- 0.9
> meanv <- HighFreq::run_mean(pricev, lambda=lambda)
> # Calculate the trailing volatilities
> volat <- HighFreq::run_var(pricev, lambda=lambda)
> volat <- sqrt(volat)
```



```
> # Dygraphs plot of Bollinger bands
> priceb <- cbind(pricev, meanv, meanv+volat, meanv-volat)
> colnames(priceb)[2:4] <- c("mean", "upper", "lower")
> colnamev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"], main="VTI Prices and"
+ dySeries(name=colnamev[1], strokeWidth=2, col="blue") %>%
+ dySeries(name=colnamev[2], strokeWidth=2, col="green") %>%
+ dySeries(name=colnamev[3], strokeWidth=2, strokePattern="dashed")
+ dySeries(name=colnamev[4], strokeWidth=2, strokePattern="dashed")
+ dyLegend(show="always", width=300)
```

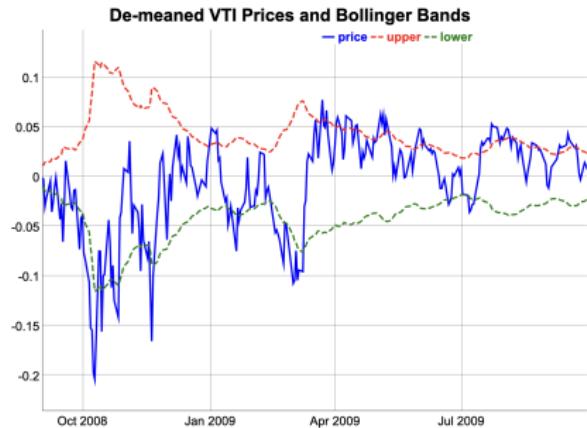
Bollinger Bands With De-meanned Prices

De-meaning the prices provides a better view of the *Bollinger Bands*.

The de-meanned prices tend to be range-bound between the *Bollinger Bands*.

When the de-meanned price is below the lower band it's considered cheap, and if it's above the upper band it's considered rich (expensive).

When the de-meanned price is close to zero it's considered fair (neutral).



```
> # De-mean the prices
> pricem <- pricev - meanv
> # Dygraphs plot of Bollinger bands
> priceb <- cbind(pricem, volat, -volat)
> colnames(priceb) <- c("price", "upper", "lower")
> colnameev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"],
+   main="De-meanned VTI Prices and Bollinger Bands") %>%
+   dySeries(name=colnameev[1], strokeWidth=2, col="blue") %>%
+   dySeries(name=colnameev[2], strokeWidth=2, strokePattern="dashed")
+   dySeries(name=colnameev[3], strokeWidth=2, strokePattern="dashed")
+   dyLegend(show="always", width=300)
```

The Bollinger Band Strategy

The *Bollinger Band* strategy switches to long \$1 dollar of the stock when prices are cheap (below the lower band), and sells short -\$1 dollar of stock when prices are rich (expensive - above the upper band). It goes flat \$0 dollar of stock (unwinds) if the stock reaches a fair (mean) price.

The strategy is therefore always either long \$1 dollar of stock, or short -\$1 dollar of stock, or flat \$0 dollar of stock.

The upper and lower Bollinger bands can be chosen to be a multiple of the standard deviations above and below the mean prices.

The Bollinger Strategy is a *mean reverting* (contrarian) strategy because it bets on prices reverting to their mean value.

The Bollinger Strategy is a type of *statistical arbitrage* strategy.

Statistical arbitrage strategies try to exploit short-term anomalies in prices, when prices diverge from their equilibrium values and then revert back to them.

```
> # Calculate the trailing mean prices and volatilities
> lambda <- 0.1
> meanv <- HighFreq::run_mean(pricev, lambda=lambda)
> volat <- HighFreq::run_var(pricev, lambda=lambda)
> volat <- sqrt(volat)
> # Prepare the simulation parameters
> pricen <- as.numeric(pricev) # Numeric price
> pricem <- pricen - meanv # De-meanned price
> threshv <- volat
> posv <- integer(nrows) # Stock positions
> posv[1] <- 0 # Initial position
> # Calculate the positions from Bollinger bands
> for (it in 2:nrows) {
+   if (pricem[it-1] > threshv[it-1]) {
+     # Enter short
+     posv[it] <- (-1)
+   } else if (pricem[it-1] < (-threshv[it-1])) {
+     # Enter long
+     posv[it] <- 1
+   } else if ((posv[it-1] < 0) && (pricem[it-1] < 0)) {
+     # Unwind short
+     posv[it] <- 0
+   } else if ((posv[it-1] > 0) && (pricem[it-1] > 0)) {
+     # Unwind long
+     posv[it] <- 0
+   } else {
+     # Do nothing
+     posv[it] <- posv[it-1]
+   } # end if
+ } # end for
> # Calculate the number of trades
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> # Calculate the plns
> retp <- rutils::diffit(pricev)
> plns <- retp*posv
```

Bollinger Strategy Performance

The *Bollinger Band* strategy has two parameters: the weight decay factor λ and the standard deviation multiple n .

The best strategy parameters can be found using backtest simulation, but it risks overfitting the parameters to the in-sample data, and poor performance out-of-sample.

The *Bollinger Band* strategy has performed well for *VTI* with $\lambda = 0.1$, but it hasn't performed well for most other stocks.

The *Bollinger Band* strategy had its best performance for *VTI* prior to the financial crisis of 2008–2009.

Bollinger Strategy / VTI Sharpe = 0.492 / Strategy Sharpe = 0.671 / Number of trades= 2769 — VTI — Strategy



```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0])
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=""),
+   "Number of trades=", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

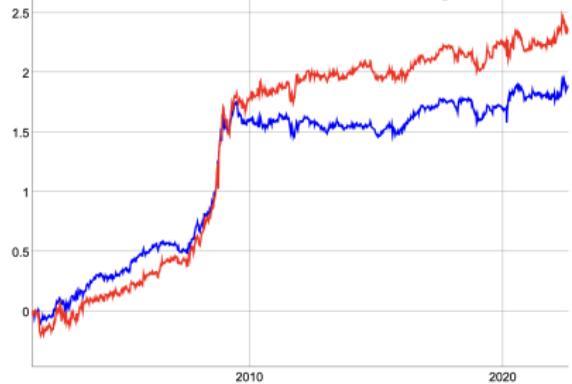
The Modified Bollinger Strategy

Unwinding the position when the stock reaches a fair (mean) price doesn't necessarily produce better performance.

In the modified Bollinger Strategy the positions are held until the price reaches the opposite extreme, so that the strategy is always either long \$1 dollar of stock or short \$1 dollar of stock.

```
> # Simulate the modified Bollinger strategy
> posv <- integer(nrows) # Stock positions
> posv[1] <- 0 # Initial position
> for (it in 2:nrows) {
+   if (pricem[it-1] > threshv[it-1]) {
+     # Enter short
+     posv[it] <- (-1)
+   } else if (pricem[it-1] < (-threshv[it-1])) {
+     # Enter long
+     posv[it] <- 1
+   } else {
+     # Do nothing
+     posv[it] <- posv[it-1]
+   } # end if
+ } # end for
> # Calculate the PnLs
> pnls2 <- retp*posv
```

Bollinger Strategy / Bollinger Sharpe = 0.671 / Modified Sharpe = 0.81 / Number of trades= 2769 — Bollinger — Modified



```
> # Calculate the Sharpe ratios
> wealthv <- cbind(pnls, pnls2)
> colnames(wealthv) <- c("Bollinger", "Modified")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]))
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=""),
+   "Number of trades=", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Fast Bollinger Strategy Simulation

The Bollinger Strategy is path-dependent so simulating it requires performing a loop, which can be slow in R.

The modified Bollinger Strategy can be simulated quickly using the compiled C++ functions `ifelse()` and `zoo::na.locf()`.

```
> # Simulate the modified Bollinger strategy quickly
> posf <- rep(NA_integer_, nrow)
> posf[1] <- 0
> posf <- ifelse(pricem > threshv, -1, posf)
> posf <- ifelse(pricem < -threshv, 1, posf)
> posf <- zoo::na.locf(posf)
> # Lag the positions to trade in the next period
> posf <- rutils::lagit(posf, lag=1)
> # Compare the positions
> all.equal(posv, posf)
```

Bollinger Strategy For Intraday Data

The Bollinger Strategy has performed well for 1-minute prices of the SPY ETF, because intraday prices exhibit significant mean-reversion.

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

```
> # Calculate the trailing mean prices and volatilities of SPY
> pricev <- log(quantmod::Cl(HighFreq::SPY))
> nrows <- NROW(pricev)
> lambda <- 0.1
> meanv <- HighFreq::run_mean(pricev, lambda=lambda)
> volat <- HighFreq::run_var(pricev, lambda=lambda)
> volat <- sqrt(volat)
> # Calculate the positions from Bollinger bands
> threshv <- volat
> pricem <- zoo::coredata(pricev - meanv)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshv, -1, posv)
> posv <- ifelse(pricem < -threshv, 1, posv)
> posv <- zoo::na.locf(posv)
> # Lag the positions to trade in the next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
```

Bollinger Strategy for Minute SPY / SPY Sharpe = 0.189 / Strategy Sharpe = 1.417 / Number of trades = SPY Strategy



```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("SPY", "Strategy")
> nyyears <- as.numeric(end(pricev)-start(pricev))/365
> sharper <- sqrt(nrows/nyyears)*sapply(wealthv, function(x) mean(x))
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy for Minute SPY", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=""),
+   "Number of trades=", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The Hampel Filter Bands

The Bollinger bands can be improved by using nonparametric measures of location (*median*) and dispersion (*MAD*).

The *Median Absolute Deviation (MAD)* is a nonparametric measure of dispersion (variability):

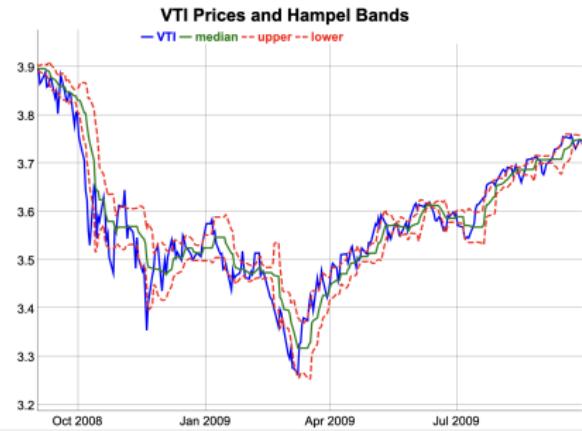
$$\text{MAD} = \text{median}(\text{abs}(p_t - \text{median}(p)))$$

The *Hampel z-score* is equal to the deviation from the median divided by the *MAD*:

$$z_i = \frac{p_t - \text{median}(p)}{\text{MAD}}$$

A time series of *z-scores* over past data can be calculated using a trailing look-back window.

```
> # Extract time series of VTI log prices
> pricev <- log(na.omit(ztutils::etfenv$prices$VTI))
> nrows <- NROW(pricev)
> # Define look-back window
> look_back <- 11
> # Calculate time series of trailing medians
> medianv <- HighFreq::roll_mean(pricev, look_back, method="nonpar")
> # medianv <- TTR::runMedian(pricev, n=look_back)
> # Calculate time series of MAD
> madv <- HighFreq::roll_var(pricev, look_back=look_back, method="")
> # madv <- TTR::runMAD(pricev, n=look_back)
> # Calculate time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:look_back, ] <- 0
> tail(zscores, look_back)
> range(zscores)
```



```
> # Plot histogram of z-scores
> histp <- hist(zscores, col="lightgrey",
+ xlab="z-scores", breaks=50, xlim=c(-4, 4),
+ ylab="frequency", freq=FALSE, main="Hampel Z-Scores histogram")
> lines(density(zscores, adjust=1.5), lwd=3, col="blue")
> # Dygraphs plot of Hampel bands
> priceb <- cbind(pricev, medianv, medianv+madv, medianv-madv)
> colnames(priceb)[2:4] <- c("median", "upper", "lower")
> colnamev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"], main="VTI Prices and
+ dySeries(name=colnamev[1], strokeWidth=2, col="blue") %>%
+ dySeries(name=colnamev[2], strokeWidth=2, col="green") %>%
+ dySeries(name=colnamev[3], strokeWidth=2, strokePattern="dashed")
+ dySeries(name=colnamev[4], strokeWidth=2, strokePattern="dashed")
+ dyLegend(show="always", width=300)
```

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 the time series of trailing medians and MAD
> look_back <- 3
> medianv <- HighFreq::roll_mean(pricev, look_back, method="nonparametric")
> madv <- HighFreq::roll_var(pricev, look_back=look_back, method="nonparametric")
> # Calculate the time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:look_back, ] <- 0
> range(zscores)
> # Calculate the positions
> threshv <- 1
> posv <- rep(NA_integer_, nrow(pricev))
> posv[1] <- 0
> posv[zscores > threshv] <- (-1)
> posv[zscores < -threshv] <- 1
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
```

Hampel Strategy / VTI Sharpe = 0.492 / Strategy Sharpe = 0.686 /



```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]))
> sharper <- round(sharper, 3)
> # Dygraphs plot of Hampel strategy
> captiont <- paste("Hampel Strategy", "/ \n",
+   + paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=""),
+   + "Number of trades=", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   + dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   + dyLegend(show="always", width=300)
```

Hampel Filter Strategy For Intraday Data

The Hampel Filter strategy has performed well for 1-minute prices of the *SPY* ETF, because intraday prices exhibit significant mean-reversion.

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

```
> # Calculate the trailing mean prices and volatilities of SPY
> pricev <- log(quantmod::Cl(HighFreq:::SPY))
> nrows <- NROW(pricev)
> # Calculate the price medians and MAD
> look_back <- 3
> medianv <- HighFreq::roll_mean(pricev, look_back, method="nonpara")
> madv <- HighFreq::roll_var(pricev, look_back=look_back, method="no")
> # Calculate the time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:look_back, ] <- 0
> # Calculate the positions
> threshv <- 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv[zscores < -threshv] <- 1
> posv[zscores > threshv] <- (-1)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils:::diffit(pricev)
> pnls <- retp*posv
```

Hampel Strategy for Minute SPY / SPY Sharpe = 0.189 / Strategy



```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("SPY", "Strategy")
> nyyears <- as.numeric(end(pricev)-start(pricev))/365
> sharper <- sqrt(nrows/nyyears)*sapply(wealthv, function(x) mean(x))
> sharper <- round(sharper, 3)
> # Dygraphs plot of Hampel strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Hampel Strategy for Minute SPY", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=""),
+   "Number of trades=", ntrades)
> endd <- rutils:::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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 price, with smaller k producing larger z-scores for more local price extremes.

```
> # Extract the VTI log OHLC prices
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(closep)
> # Calculate the centered volatility
> look_back <- 7
> half_back <- look_back %/% 2
> stdev <- sqrt(HighFreq::roll_var(retp, look_back))
> 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", strokeWidth=2, col="blue")
+   dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="red")
```

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 the 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])
> bottoms <- zoo::coredata(pricez < threshv[1])
> # Simulate in-sample VTI strategy
> posv <- rep(NA_integer_, nrow(pricez))
> posv[1] <- 0
> posv[tops] <- (-1)
> posv[bottoms] <- 1
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> pnls <- retp*posv
```



```
> # Plot dygraph of in-sample VTI strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Price Tops and Bottoms Strategy In-Sample") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="Strategy", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", label="VTI", strokeWeight=2, col="blue")
+   dySeries(name="Strategy", axis="y2", label="Strategy", strokeWeight=2, col="red")
```

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 the volatility z-scores
> volat <- HighFreq::roll_var_ohlc(ohlc=ohlc, look_back=look_back)
> volatm <- HighFreq::roll_mean(volat, look_back)
> volatsd <- sqrt(HighFreq::roll_var(rutils::diffit(volat), look_back))
> volatsd[1] <- 0
> volatz <- ifelse(volatsd > 0, (volat - volatm)/volatsd, 0)
> colnames(volatz) <- "volat"
> # Calculate the volume z-scores
> volum <- quantmod::Vo(ohlc)
> volumean <- HighFreq::roll_mean(volum, look_back)
> volumsd <- sqrt(HighFreq::roll_var(rutils::diffit(volum), look_back))
> 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:

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

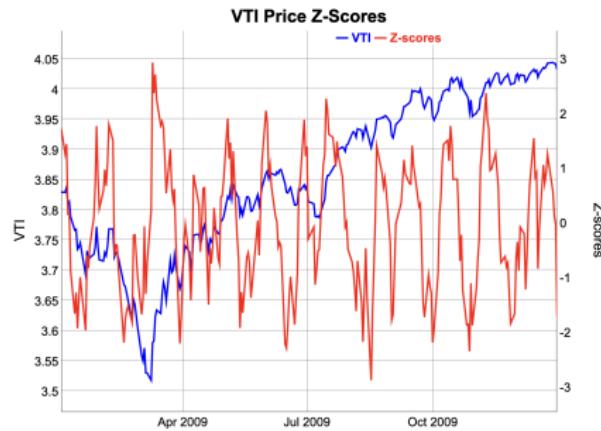
Where α and β are the *regression coefficients*, and σ_i is the standard deviation of the residuals.

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 the them using functions written in C++ code.

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

```
> # Calculate the trailing price regression z-scores
> datev <- matrix(zoo::index(closep))
> look_back <- 21
> controlv <- HighFreq::param_reg()
> regz <- HighFreq::roll_reg(respv=closep, predm=datev, look_back=look_back)
> regz <- drop(regz[, NCOL(regz)])
> 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", strokeWidth=2, col="blue") +
+   dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="red")
```

The Logistic Function

The *logistic* function expresses the probability of a numerical variable ranging over the whole interval of real numbers:

$$p(x) = \frac{1}{1 + \exp(-\lambda x)}$$

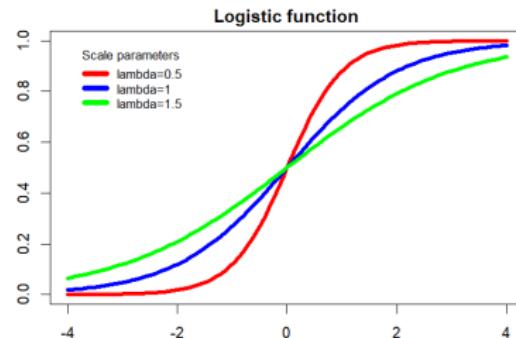
Where λ is the scale (dispersion) parameter.

The *logistic* function is often used as an activation function in neural networks, and logistic regression can be viewed as a perceptron (single neuron network).

The *logistic* function can be inverted to obtain the *Odds Ratio* (the ratio of probabilities for favorable to unfavorable outcomes):

$$\frac{p(x)}{1 - p(x)} = \exp(\lambda x)$$

The function `plogis()` gives the cumulative probability of the *Logistic* distribution,



```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Plot three curves in loop
> for (it in 1:3) {
+   curve(expr=plogis(x, scale=lambdav[it]),
+         xlim=c(-4, 4), type="l", xlab="", ylab="", lwd=4,
+         col=colorv[it], add=(it>1))
+ } # end for
> # Add title
> title(main="Logistic function", line=0.5)
> # Add legend
> legend("topleft", title="Scale parameters",
+        paste("lambda", lambdav, sep="="), y.intersp=0.4,
+        inset=0.05, cex=0.8, lwd=6, bty="n", lty=1, col=colorv)
```

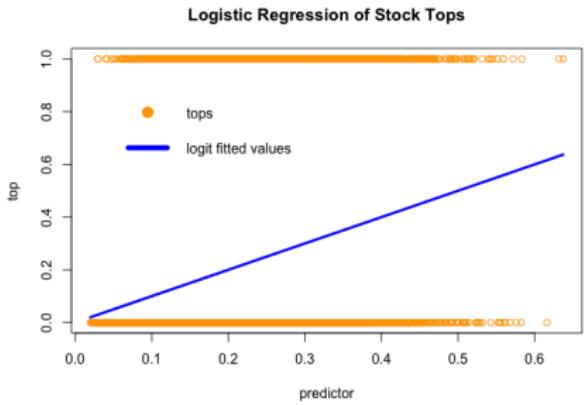
Forecasting Stock Price Tops and Bottoms Using Logistic Regression

Consider a model which uses the weighted average of the volatility, trading volume, and regression z-scores, 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 predictor for tops including intercept column
> predm <- cbind(volatz, volumz, regz)
> predm[1, ] <- 0
> predm <- rutils::lagit(predm)
> # Fit in-sample logistic regression for tops
> logmod <- glm(tops ~ predm, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcasts <- drop(cbind(rep(1, nrows), predm )%*% coeff)
> ordern <- order(fcasts)
> # Calculate the in-sample forecasts from logistic regression model
> fcasts <- 1/(1 + exp(-fcasts))
> all.equal(logmod$fitted.values, fcasts, check.attributes=FALSE)
> hist(fccasts)
```



```
> plot(x=fcasts[ordern], y=tops[ordern],
+       main="Logistic Regression of Stock Tops",
+       col="orange", xlab="predictor", ylab="top")
> lines(x=fcasts[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  
> threshv <- quantile(fcasts, conf1[2])  
> # Calculate the confusion matrix in-sample  
> confmat <- table(actual=!tops, forecast=(fcasts < threshv))  
> confmat  
> # Calculate the FALSE positive (type I error)  
> sum(tops & (fcasts < threshv))  
> # Calculate the FALSE negative (type II error)  
> sum(!tops & (fcasts > threshv))
```

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 the 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.

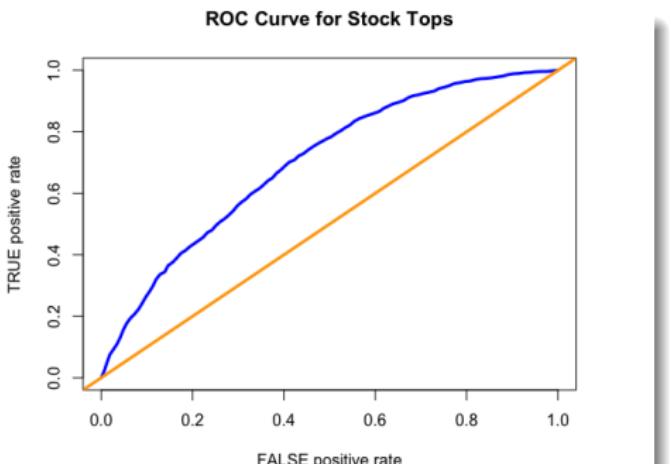
Receiver Operating Characteristic (ROC) Curve for Stock Tops

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, fcasts, threshv) {
+   forb <- (fcasts < threshv)
+   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, fcasts, threshv=threshv)
> # Define vector of discrimination thresholds
> threshv <- quantile(fcasts, seq(0.01, 0.99, by=0.01))
> # Calculate the error rates
> errorr <- sapply(threshv, confun,
+   actual=!tops, fcasts=fcasts) # end sapply
> errorr <- t(errorr)
> rownames(errorr) <- threshv
> # Calculate the informedness
> informv <- 2 - rowSums(errorr)
> plot(threshv, informv, t="l", main="Informedness")
> # Find the threshold corresponding to highest informedness
> threshm <- threshv[which.max(informv)]
> topf <- (fcasts > threshm)
```



```
> # Calculate the area under ROC curve (AUC)
> errorr <- rbind(c(1, 0), errorr)
> errorr <- rbind(errorr, c(0, 1))
> truepos <- (1 - errorr[, "typeII"])
> truepos <- (truepos + rutils::lagit(truepos))/2
> falsepos <- rutils::diffit(errorr[, "typeI"])
> abs(sum(truepos*falsepos))
> # Plot ROC Curve for stock tops
> plot(x=errorr[, "typeI"], y=1-errorr[, "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")
```

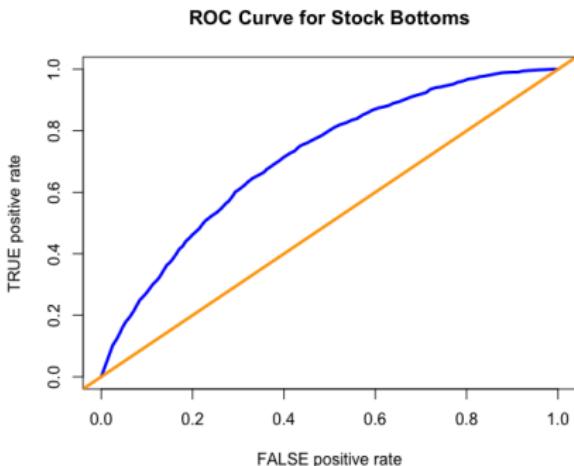
Receiver Operating Characteristic (ROC) Curve for Stock Bottoms

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



```
> # Calculate the area under ROC curve (AUC)
> errorr <- rbind(c(1, 0), errorr)
> errorr <- rbind(errorr, c(0, 1))
> truepos <- (1 - errorr[, "typeII"])
> truepos <- (truepos + rutils::lagit(truepos))/2
> falsepos <- rutils::diffit(errorr[, "typeI"])
> abs(sum(truepos*falsepos))
> # Plot ROC Curve for stock tops
> plot(x=errorr[, "typeI"], y=1-errorr[, "typeII"],
+       xlab="FALSE positive rate", ylab="TRUE positive rate",
+       main="ROC Curve for Stock Bottoms", type="l", lwd=3, col="blue")
> 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(topf), 5)/5
> botsav <- HighFreq::roll_sum(matrix(botf), 5)/5
> # Simulate in-sample VTI strategy
> posv <- (botsav - topsav)
> # Standard strategy
> # posv <- rep(NA_integer_, NROW(retp))
> # posv[1] <- 0
> # posv[topf] <- (-1)
> # posv[botf] <- 1
> # posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> pnls <- retp*posv
```

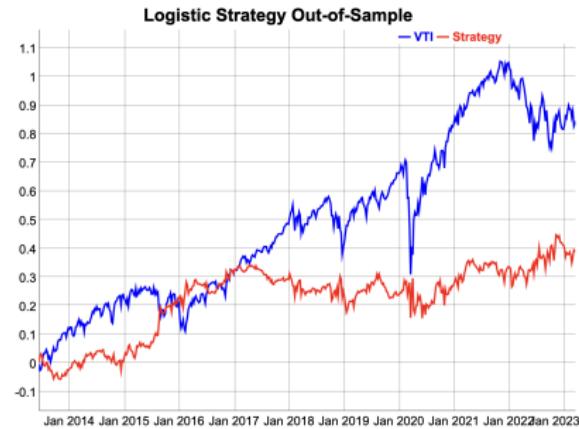


```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Top and Bottom Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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] ~ predm[insample, ], family=binomial)
> fitv <- logmod$fitted.values
> coefftop <- logmod$coefficients
> # Calculate the error rates and best threshold value
> errorr <- sapply(threshv, confun,
+   actual=!tops[insample], fcasts=fitv) # end sapply
> errorr <- t(errorr)
> informv <- 2 - rowSums(errorr)
> threshtop <- threshv[which.max(informv)]
> # Fit in-sample logistic regression for bottoms
> logmod <- glm(bottoms[insample] ~ predm[insample, ], family=binomial)
> fitv <- logmod$fitted.values
> coeffbot <- logmod$coefficients
> # Calculate the error rates and best threshold value
> errorr <- sapply(threshv, confun,
+   actual=!bottoms[insample], fccasts=fitv) # end sapply
> errorr <- t(errorr)
> informv <- 2 - rowSums(errorr)
> threshbot <- threshv[which.max(informv)]
> # Calculate the out-of-sample forecasts from logistic regression
> predictout <- cbind(rep(1, NROW(outsample)), predm[outsample, ])
> fccasts <- drop(predictout %*% coefftop)
> fccasts <- 1/(1 + exp(-fccasts))
> topf <- (fccasts > threshtop)
> fccasts <- drop(predictout %*% coeffbot)
> fccasts <- 1/(1 + exp(-fccasts))
> botf <- (fccasts > threshbot)
```



```
> # Simulate in-sample VTI strategy
> topsav <- HighFreq::roll_sum(matrix(topf), 5)/5
> botsav <- HighFreq::roll_sum(matrix(botf), 5)/5
> posv <- (botsav - topsav)
> posv <- rutils::lagit(posv)
> pnls <- retp[outsample, ]*posv
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp[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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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, ]
> fcasts <- predict(logmod, newdata=testset, type="response")
> # Calculate the confusion matrix out-of-sample
> table(actual=!testset$default,
+ forecast=(fcasts < threshv))
```

draft: 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
> retos <- (retsf > 0)
> logmod <- glm(retspos ~ predm - 1, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcasts <- predm %*% coeff
> fcasts <- 1/(1 + exp(-fcasts))
> # Calculate the error rates
> threshv <- quantile(fcasts, seq(0.01, 0.99, by=0.01))
> errorr <- sapply(threshv, confun,
+   actual=!retspos, fcasts=fcasts) # end sapply
> errorr <- t(errorr)
> # Calculate the threshold corresponding to highest informedness
> informv <- 2 - rowSums(errorr)
> plot(threshv, informv, t="l", main="Informedness")
> threshm <- threshv[which.max(informv)]
> forecastpos <- (fcasts > threshm)
> # Fit in-sample logistic regression for negative returns
> retsneg <- (retsf < 0)
> logmod <- glm(retsneg ~ predm - 1, family=binomial(logit))
> summary(logmod)
> coeff <- logmod$coefficients
> fcasts <- predm %*% coeff
> fcasts <- 1/(1 + exp(-fcasts))
> # Calculate the error rates
> errorr <- sapply(threshv, confun,
+   actual=!retsneg, fcasts=fcasts) # end sapply
> errorr <- t(errorr)
> # Calculate the threshold corresponding to highest informedness
> informv <- 2 - rowSums(errorr)
> plot(threshv, informv, t="l", main="Informedness")
> threshm <- threshv[which.max(informv)]
> forecastneg <- (fcasts > threshm)
```

draft: Logistic Forecasting Returns Strategy In-sample

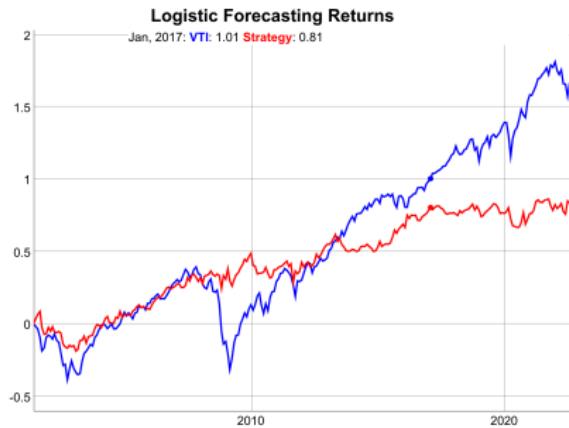
Explain why the strategy uses the minus of the signals.

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
> posv <- (negav - posav)
> # posv <- ifelse(forecastpos, 1, 0)
> # posv <- ifelse(forecastneg, -1, posv)
> posv <- rutils::lagit(posv)
> pnls <- retp*posv
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, 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])))
```

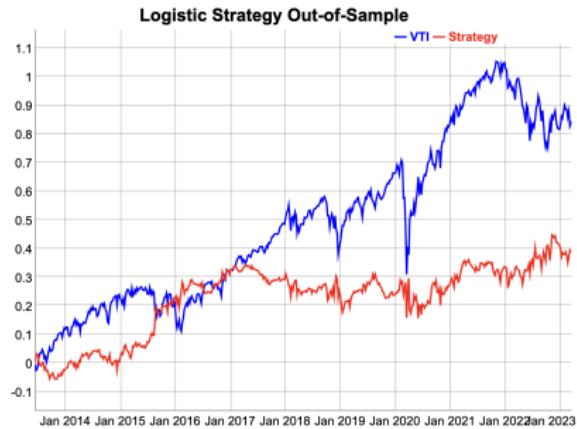


```
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Forecasting Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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] ~ predm[insample, ], family=binomial)
> fitv <- logmod$fitted.values
> coefftop <- logmod$coefficients
> # Calculate the error rates and best threshold value
> errorr <- sapply(threshv, confun,
+   actual=!tops[insample], fcasts=fitv) # end sapply
> errorr <- t(errorr)
> informv <- 2 - rowSums(errorr)
> threshtop <- threshv[which.max(informv)]
> # Fit in-sample logistic regression for bottoms
> logmod <- glm(bottoms[insample] ~ predm[insample, ], family=binomial)
> fitv <- logmod$fitted.values
> coeffbot <- logmod$coefficients
> # Calculate the error rates and best threshold value
> errorr <- sapply(threshv, confun,
+   actual=!bottoms[insample], fcasts=fitv) # end sapply
> errorr <- t(errorr)
> informv <- 2 - rowSums(errorr)
> threshbot <- threshv[which.max(informv)]
> # Calculate the out-of-sample forecasts from logistic regression
> predictout <- cbind(rep(1, NROW(outsample)), predm[outsample, ])
> fcasts <- drop(predictout %*% coefftop)
> fcasts <- 1/(1 + exp(-fcasts))
> topf <- (fcasts > threshtop)
> fcasts <- drop(predictout %*% coeffbot)
> fcasts <- 1/(1 + exp(-fcasts))
> botf <- (fcasts > threshbot)
```



```
> # Simulate out-of-sample VTI strategy
> posv <- rep(NA_integer_, NROW(outsample))
> posv[1] <- 0
> posv[topf] <- (-1)
> posv[botf] <- 1
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> pnls <- rep[outsample, ]*posv
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(rep[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="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
```