FRE7241 Algorithmic Portfolio Management

Lecture#2, Fall 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

September 19, 2023

NYU | TANDON SCHOOL OF ENGINEERING

Tests for Market Timing Skill

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

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

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

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

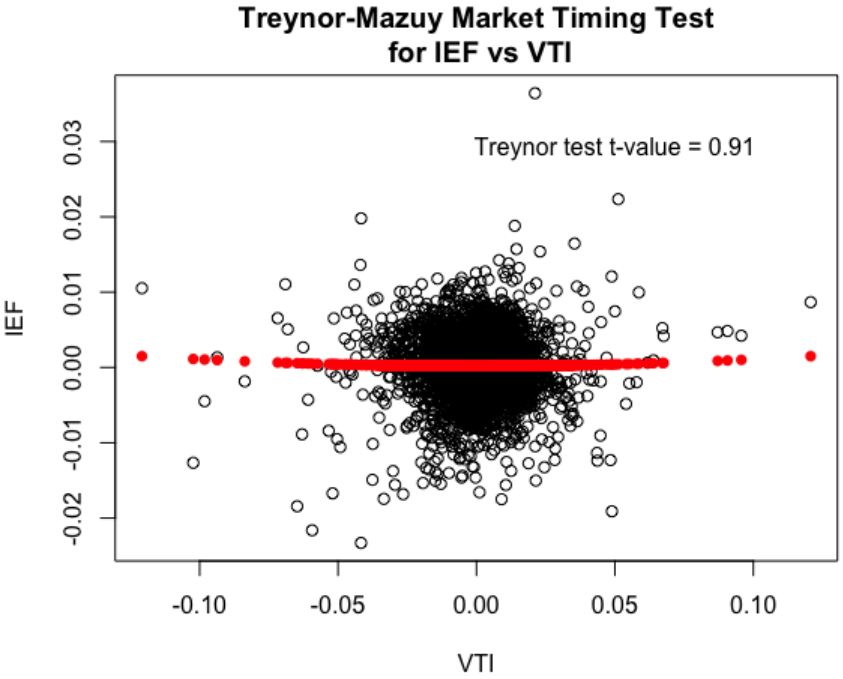
Where R are the strategy returns, R_m are the market returns, and R_f are the risk-free rates.

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

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

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

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




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

```
> # Treynor-Mazuy test
> regmod <- lm(IEF ~ VTI + treynor, data=desv); summary(regmod)
> # Plot residual scatterplot
> x11(width=6, height=5)
> resids <- (desv$IEF - regmod$coeff["VTI"]*retvti)
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="IEF")
> title(main="Treynor-Mazuy Market Timing Test\n for IEF vs VTI", li
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$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(resids), paste("Treynor test t-value =", tva
```

Calculating Asset Returns

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

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

The initial returns are all equal to zero.

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

```
> library(rutils)
> # Extract 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
> retl <- rutils::diffit(log(pricev))
```

Compounding Asset Returns

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

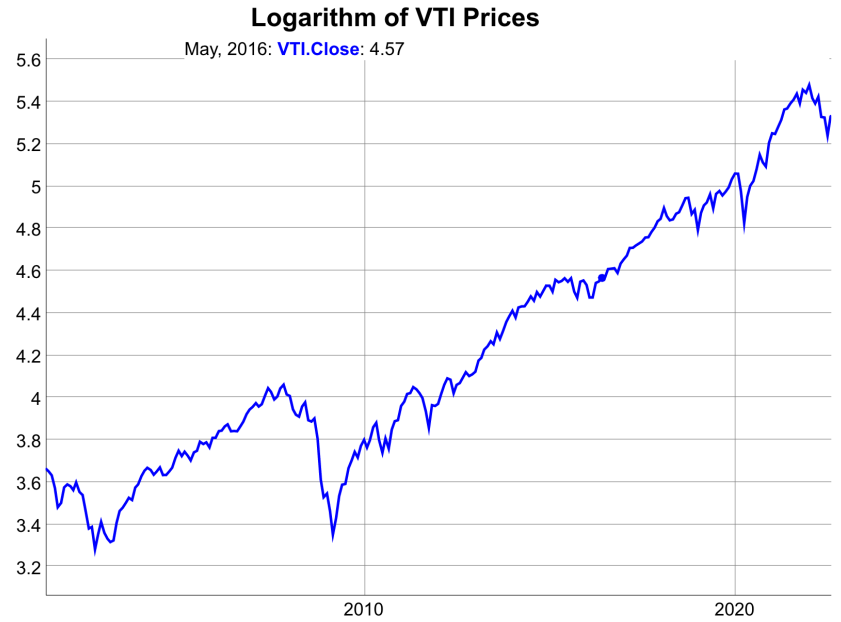
The compounded percentage returns: $\prod_{i=1}^{n}(1 + r_i^p)$ also represent the wealth path from owning a *fixed number of shares*, initially equal to \$1 dollar.

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

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

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

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




```
> # 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)
```

```
> # 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)
```

Funding Costs of Single Asset Rebalancing

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

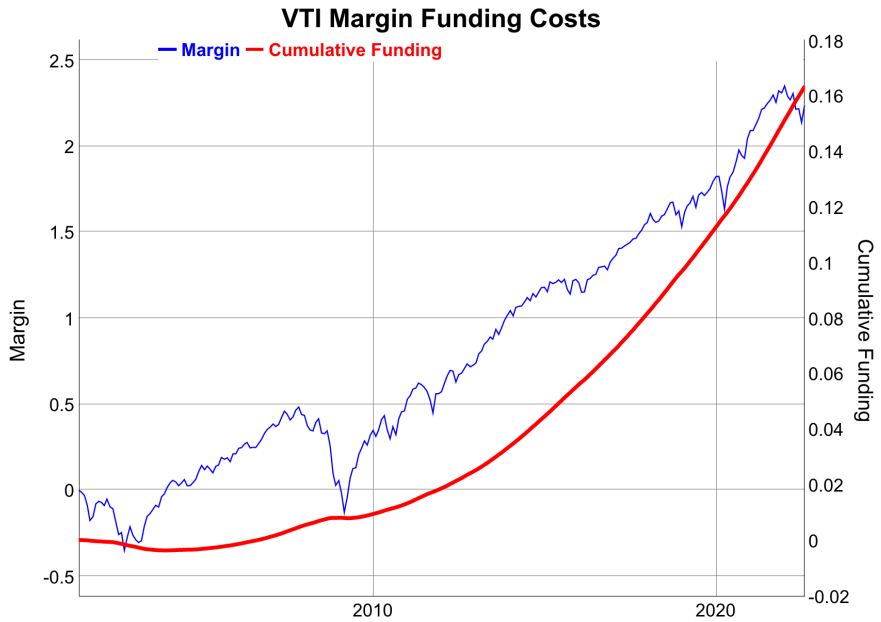
The wealth accumulated from owning a *fixed dollar amount* of stock is equal to the cash earned from rebalancing, which is proportional to the sum of the percentage returns, and it's kept in a *margin account*: $m_t = \sum_{i=1}^{t} r_i^p$.

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* $\sum_{i=1}^{t} c_t^f$ must be added to the *margin account*: $m_t + \sum_{i=1}^{t} c_t^f$.




```
> # 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-ask 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-ask spread* is the percentage difference between the *ask* (offer) minus the *bid* prices, 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-ask 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-ask spread for many liquid ETFs is about 1 basis point. For example the *XLK ETF*

In reality the *bid-ask 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-ask spread* are equal to the number of traded shares times their price, times half the *bid-ask spread*.

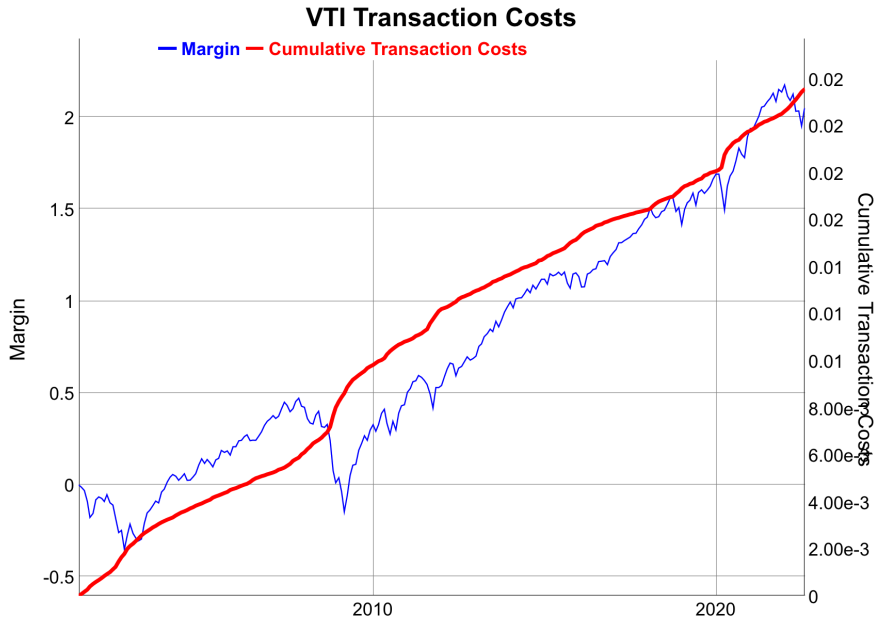
Transaction Costs of Single Asset Rebalancing

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-ask spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2} |r_t|$.

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




```
> # bidask equal to 1 bp for liquid ETFs
> bidask <- 0.001
> # Cumulative transaction costs
> costs <- bidask*cumsum(abs(retp))/2
> # Subtract transaction costs from margin account
> marginv <- cumsum(retp)
> 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[endd], main="VTI Transaction Costs") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=300)
```

Combining the Returns of Multiple Assets

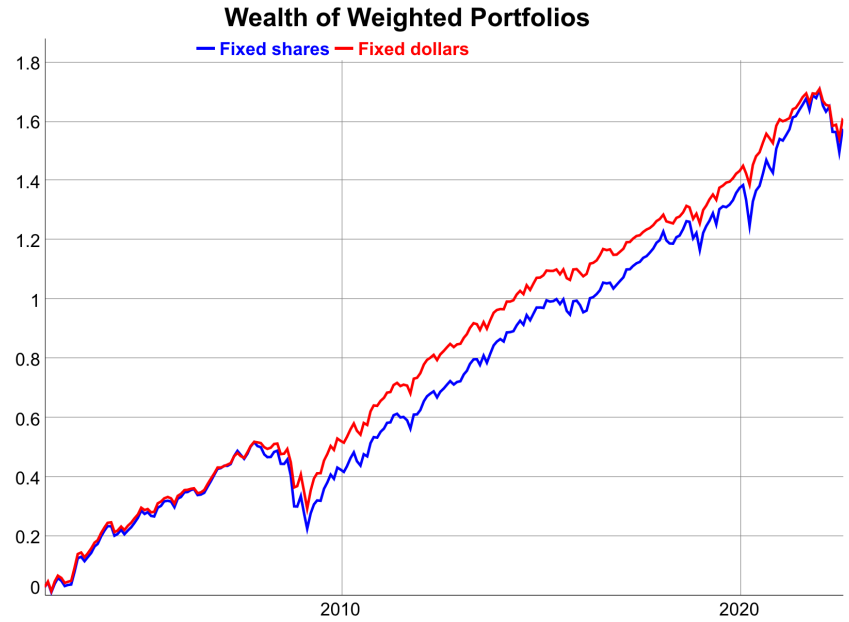
Multiplying the weights times the dollar returns is equivalent to buying a *fixed number of shares* proportional to the weights (aka *Fixed Share Allocation* or FSA).

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
> pricesi <- as.numeric(pricev[1, ])
> retd[1, ] <- pricev[1, ]
> wealthfs2 <- cumsum(retd %*% (weightv/pricesi))
```

```
> # 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 Portfoli
+   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

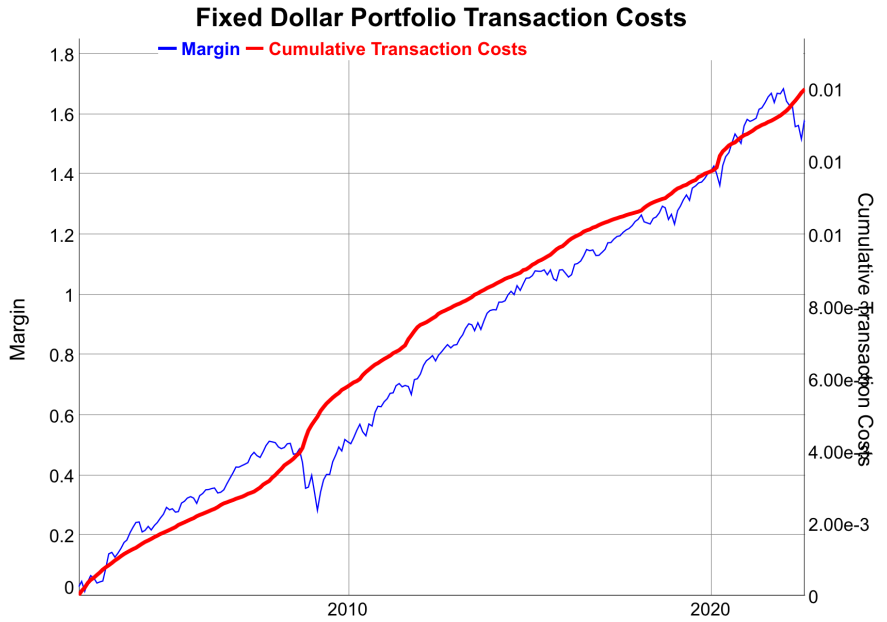
Maintaining a *fixed dollar allocation* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

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: $w1 \left|r_t^1\right| + w2 \left|r_t^2\right|$.

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

The *cumulative transaction costs* $\sum_{i=1}^{t} c_t^r$ 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 <- bidask*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.

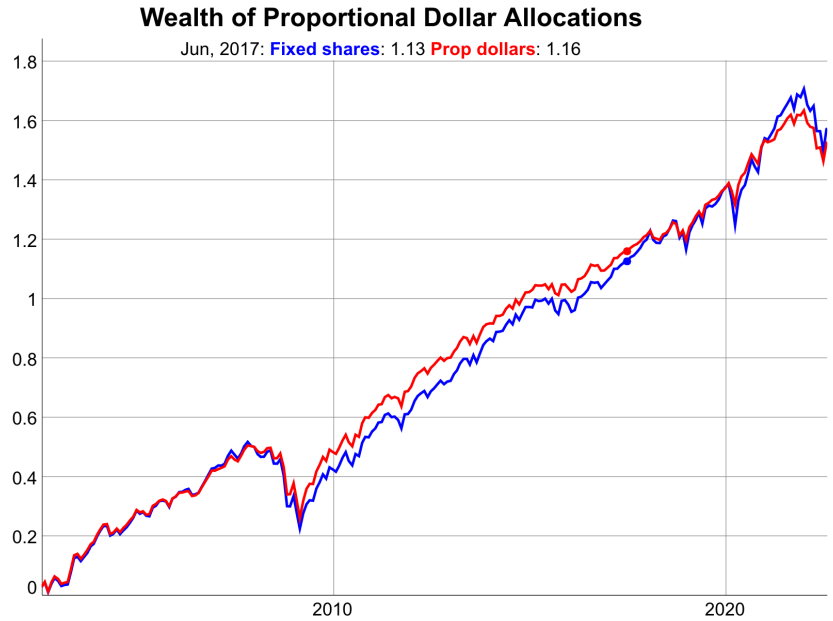
The total portfolio wealth at time t is equal to the wealth at time $t-1$ multiplied by the weighted returns: $w_t = w_{t-1}(1 + \bar{r}_t)$.

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 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(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Proportional Dollar Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

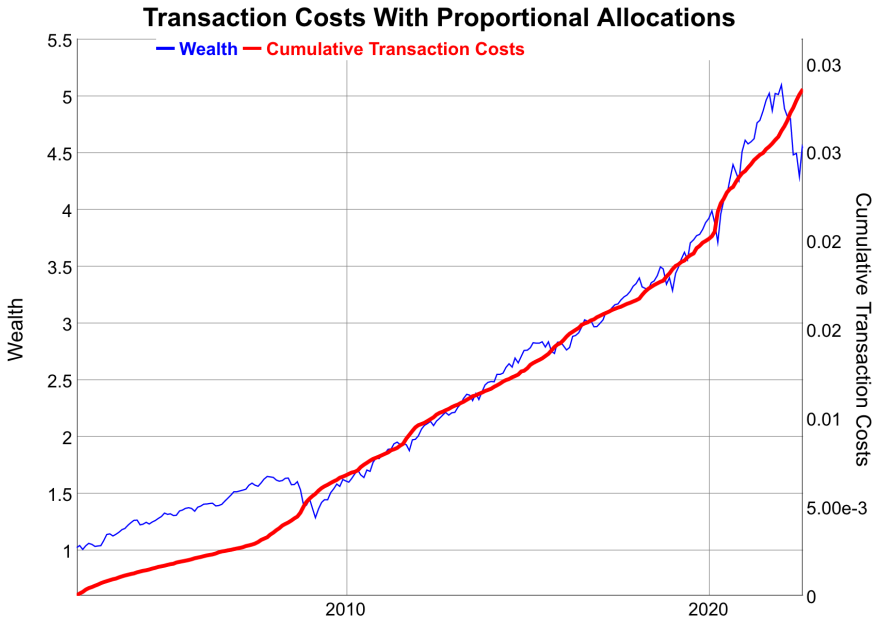
Transaction Costs With Proportional Dollar Allocations

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: $\sum_{i=1}^{n} \varepsilon_i = w_{t-1} \sum_{i=1}^{n} \omega_i \left| \bar{r}_t - r_t \right|$

The *transaction costs* c_t^r are equal to half the *bid-ask 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* $\sum_{i=1}^{t} c_t^r$ must be subtracted from the *wealth* w_t: $w_t - \sum_{i=1}^{t} c_t^r$.




```
> # 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*rutils::lagit(wealthpd)
> # Cumulative transaction costs
> costs <- bidask*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: ω_i: $w_i = \omega_i w_t$.

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

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

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

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

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

```
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- drop(apply(retp, 2, function(x) cumprod(1 + x)) %*% we
> # 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(wea
+     # cat("Rebalance at:", it, "\n")
+     wealthv[it, ] <- 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 T
> dygraphs::dygraph(wealthv, main="Wealth of Proportional Target All
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

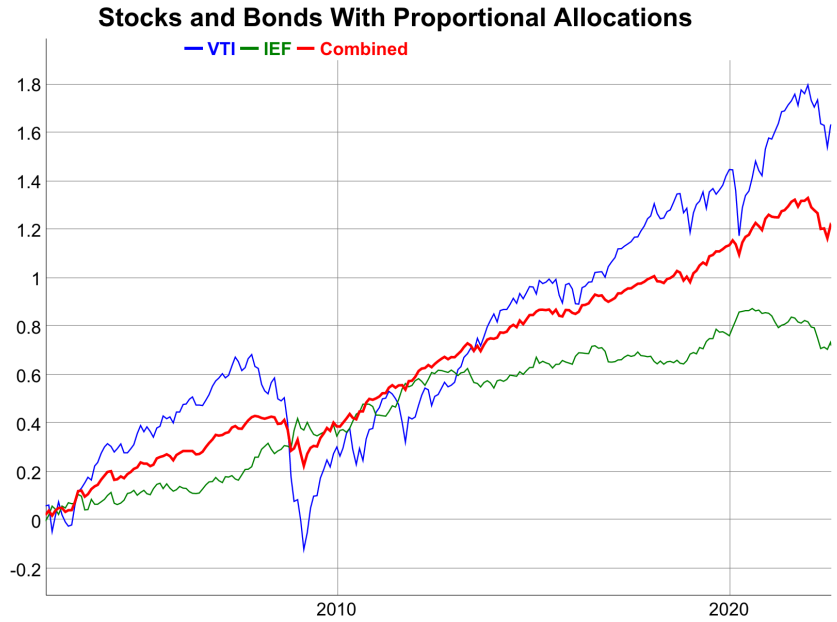
Stock and Bond Portfolio With Proportional Dollar Allocations

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
```

```
> # 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)
```

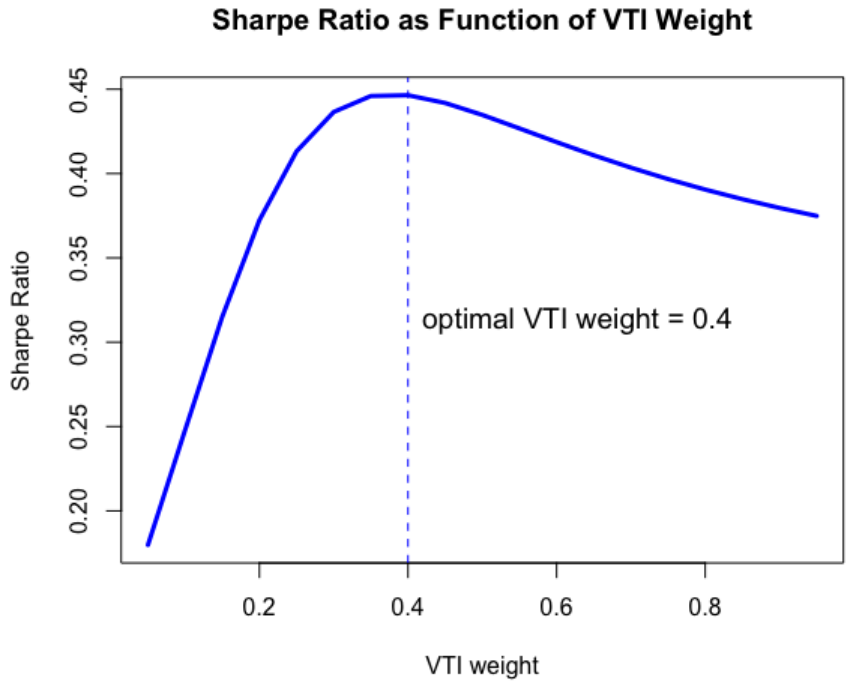
Optimal Stock and Bond Portfolio Allocations

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
```




```
> # 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.

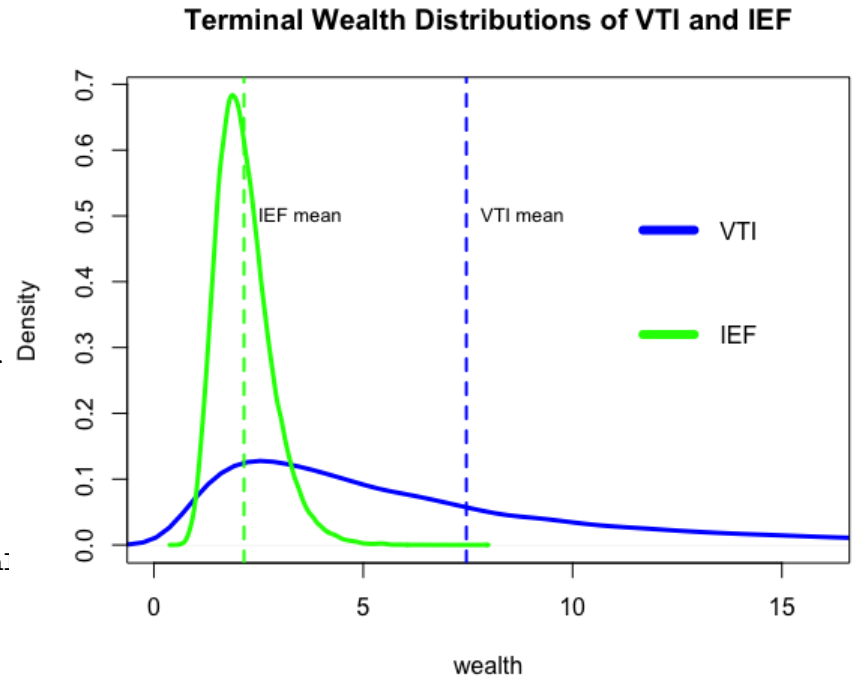
```
> # Coerce the returns from xts time series to matrix
> retp <- zoo::coredata(retp[, 1:2])
> nrows <- NROW(retp)
> # Bootstrap the returns and Calculate the a list of random returns
> nboot <- 1e4
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> # Perform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

The Distributions of Terminal Wealth From Bootstrap

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

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

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




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

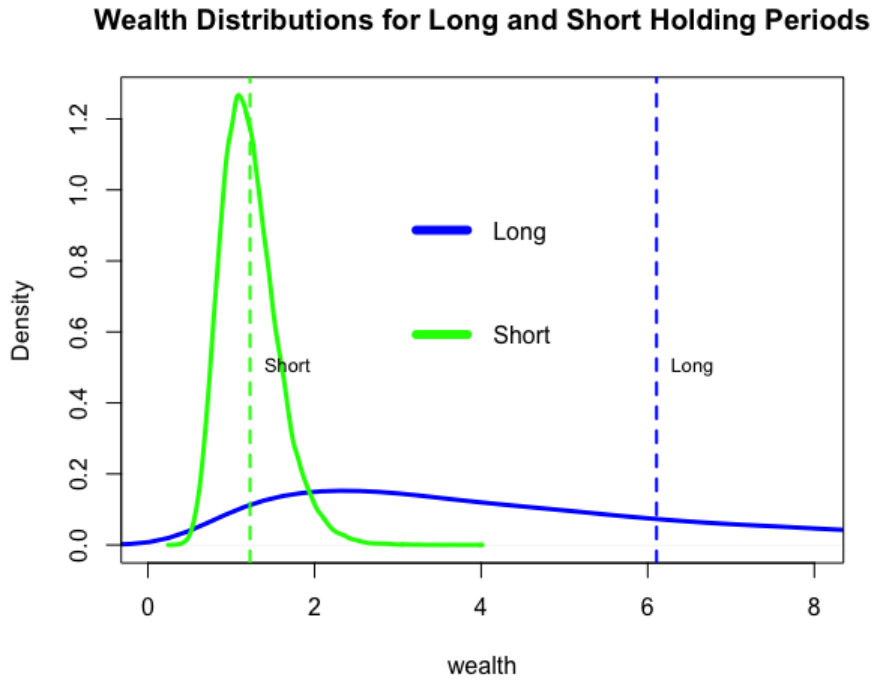
The Distribution of Stock Wealth and Holding Period

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

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

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

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




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

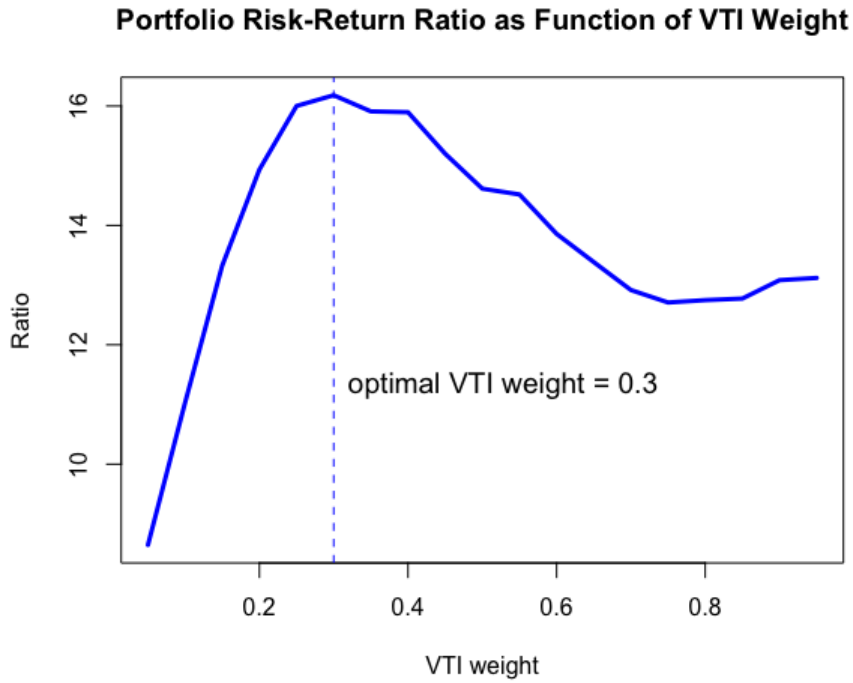
```
> # Plot the stock wealth for long and short holding periods
> wealth1 <- wealthm[, 9]
> wealth2 <- wealthm[, 1]
> mean1 <- mean(wealth1); mean2 <- mean(wealth2)
> dens1 <- density(wealth1); dens2 <- density(wealth2)
> plot(dens1, col="blue", lwd=3, xlab="wealth",
+      xlim=c(0, 2*max(dens2$x)),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

Optimal Stock and Bond Portfolio Allocations From Bootstrap

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

Bootstrapping the past 20 years of data, the optimal *VTI* weight is about 0.3.

```
> # 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)]
```




```
> # 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)))
```

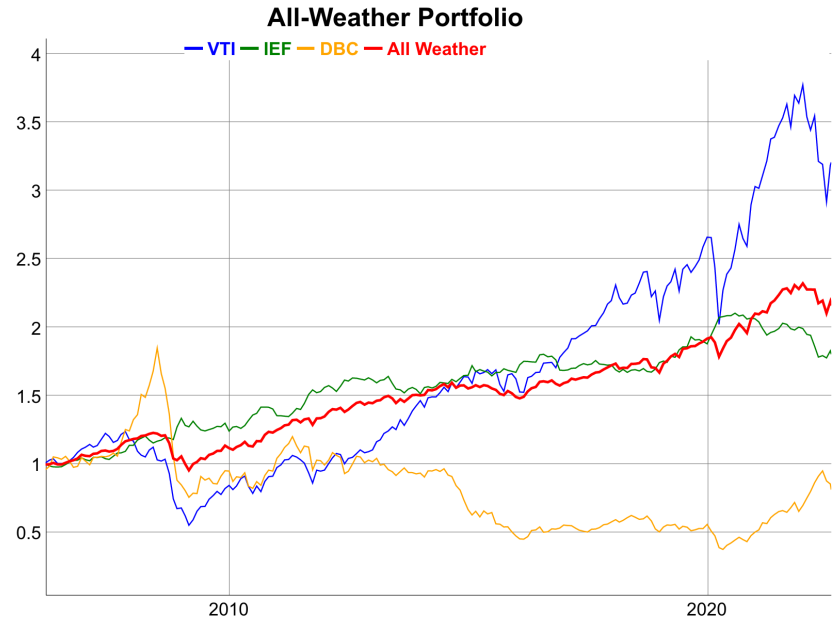
The *All-Weather* Portfolio

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(rutils::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 <- rutils::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, 4
+        name="All-Weather Portfolio")
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

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"), strokeWidt
+   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)
> 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)
> 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.

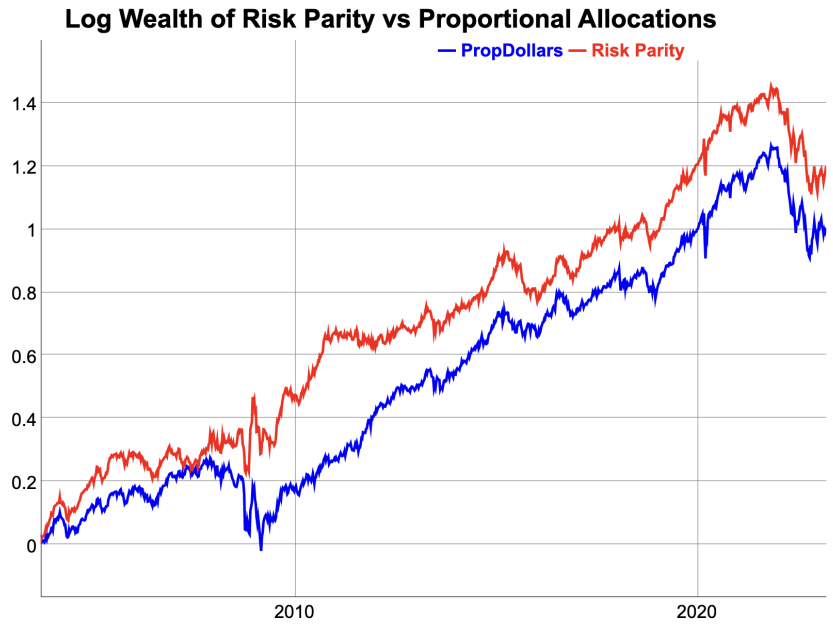
```
> # Calculate the dollar and percentage returns for stocks and bonds
> pricev <- rutils::etfenv$prices[, c("VTI", "TLT")]
> pricev <- na.omit(pricev)
> retd <- rutils::diffit(pricev)
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> # Calculate the wealth of proportional dollar allocations
> 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.



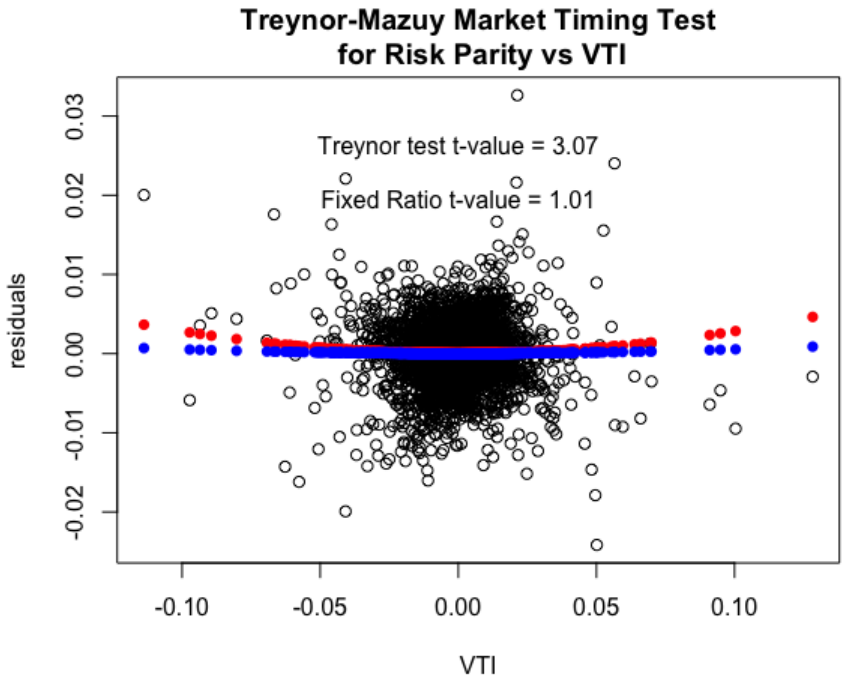

```
> # Calculate the log wealths
> wealthv <- log(cbind(wealthpd, wealthrp))
> wealthv <- xts::xts(wealthv, zoo::index(pricev))
> colnames(wealthv) <- c("PropDollars", "Risk Parity")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), 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*.



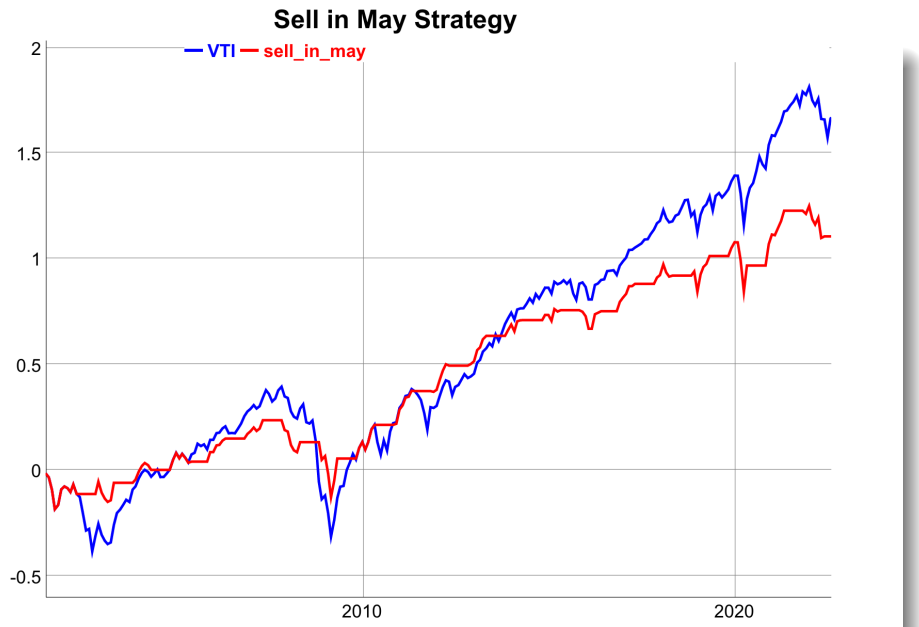

```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> retrp <- rutils::diffit(wealthv)
> retvti <- retp$VTI
> desv <- cbind(retrp, retvti, retvti^2)
> desv <- na.omit(desv)
> colnames(desv)[1:2] <- c("fixed", "risk_parity")
> colnames(desv)[4] <- "treynor"
> regmod <- lm(risk_parity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

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

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)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```

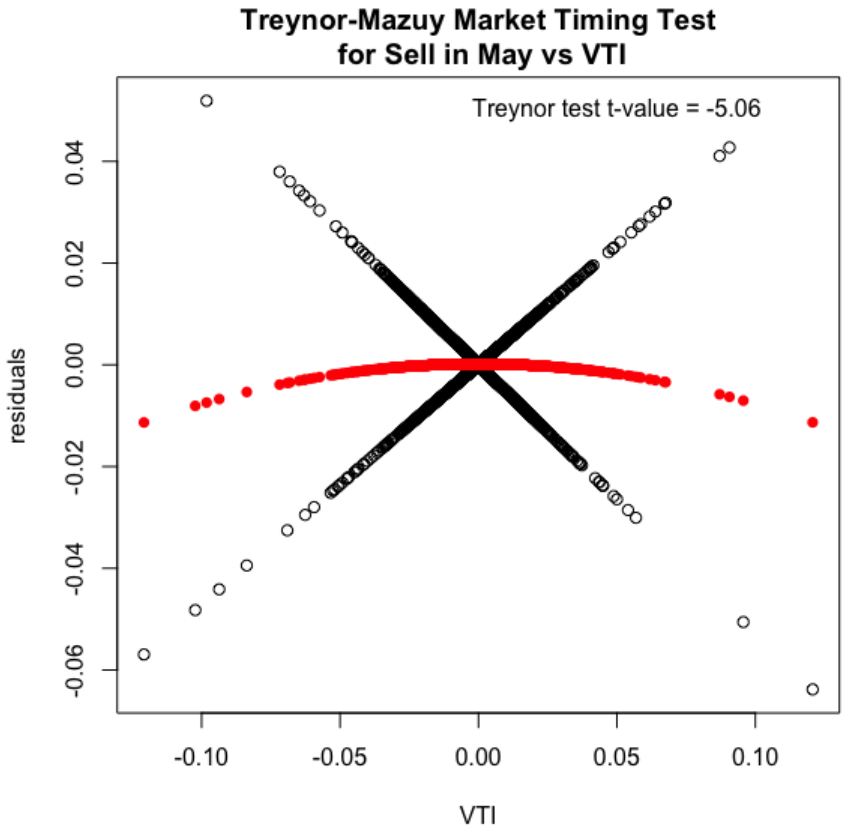



```
> # Plot wealth of Sell in May strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Sell in May Strateg
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # OR: Open x11 for plotting
> 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 Ma
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

Sell in May Strategy Market Timing

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

```
> # Test if Sell in May strategy can time VTI
> desv <- cbind(wealth$sell_in_may, 0.5*(retp+abs(retp)), retp^2)
> colnames(desv) <- c("VTI", "merton", "treynor")
> # Perform Merton-Henriksson test
> regmod <- lm(pnlinmay ~ VTI + merton, data=desv)
> summary(regmod)
> # Perform Treynor-Mazuy test
> regmod <- lm(pnlinmay ~ VTI + treynor, data=desv)
> summary(regmod)
> # Plot Treynor-Mazuy residual scatterplot
> resids <- (pnlinmay - regmod$coeff["VTI"]*retp)
> plot.default(x=retp, 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"]*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 =", tva
```



Overnight Market Anomaly

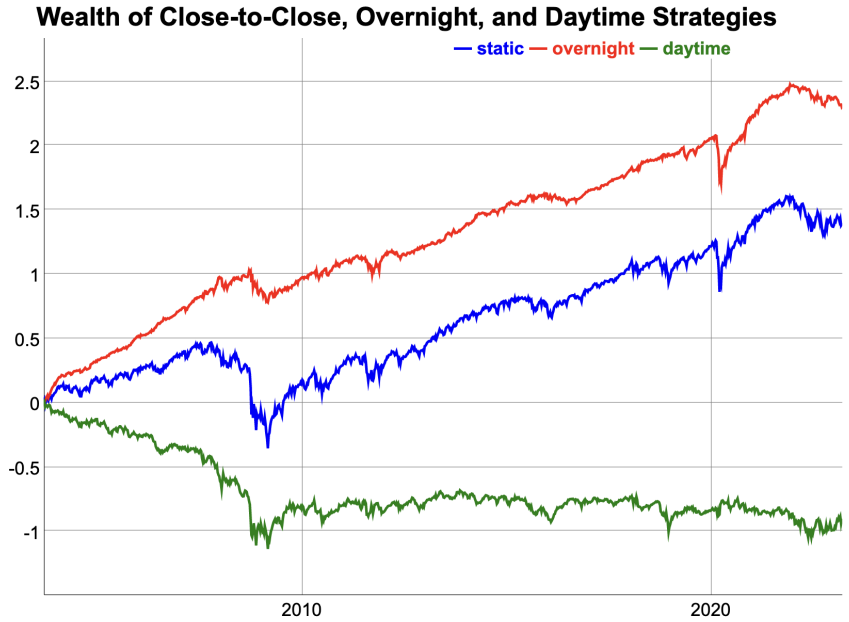
The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the intraday returns.

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

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

The Intraday 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* is not as pronounced 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.
> retp <- rutils::diffit(closep)
> colnames(retp) <- "daily"
> retd <- (closep - openp)
> colnames(retd) <- "intraday"
> reton <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(reton) <- "overnight"
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, reton, retd)
> 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 Intraday Strategi
+   dySeries(name="daily", strokeWidth=2, col="blue") %>%
+   dySeries(name="overnight", strokeWidth=2, col="red") %>%
+   dySeries(name="intraday", strokeWidth=2, col="green") %>%
+   dyLegend(width=500)
```

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(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retp)
> # Calculate the first business day of every month
> dayv <- as.numeric(format(datev, "%d"))
> indeks <- which(rutils::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 <- rutils::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.

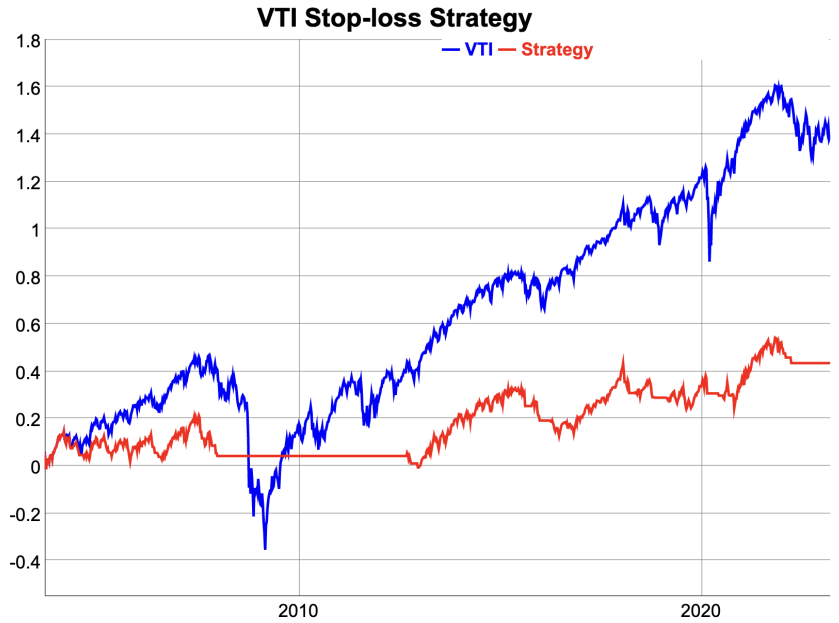
```
> # Calculate the VTI prices and returns
> pricev <- na.omit(rutils::etfenv$prices$VTI)
> nrows <- NROW(pricev)
> datev <- zoo::index(pricev)
> retp <- rutils::diffit(log(pricev))
> # Simulate stop-loss strategy
> stopl <- 0.05 # Stop-loss percentage
> pricem <- cummax(pricev) # Trailing maximum 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], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

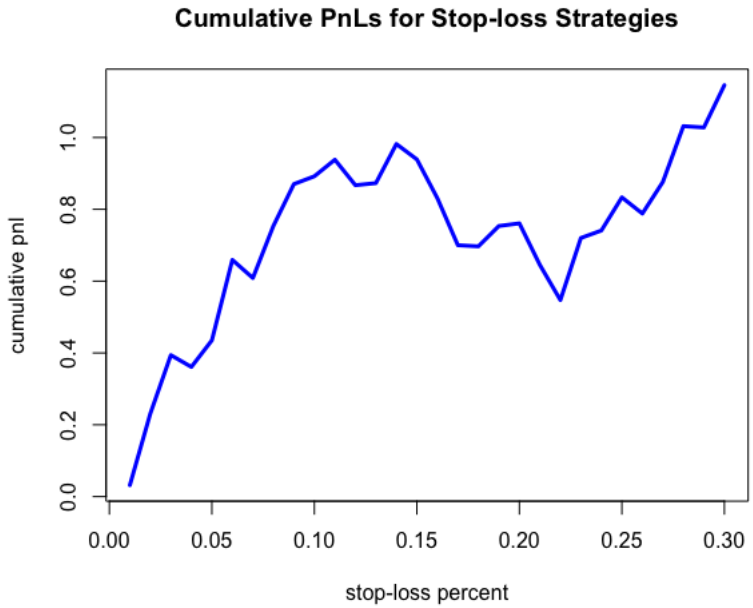
Optimal Stop-loss Strategy

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 <- retp
+   insl <- rutils::lagit(dd < -stopl*pricem)
+   pnls <- ifelse(insl, 0, pnls)
+   sum(pnls)
+ })  # end sapply
```

```
> # 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 <- retp # Initialize PnLs
+   for (i in 1:nrows) {
+     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 pri
+ 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 pri
+ 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
+     } 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.

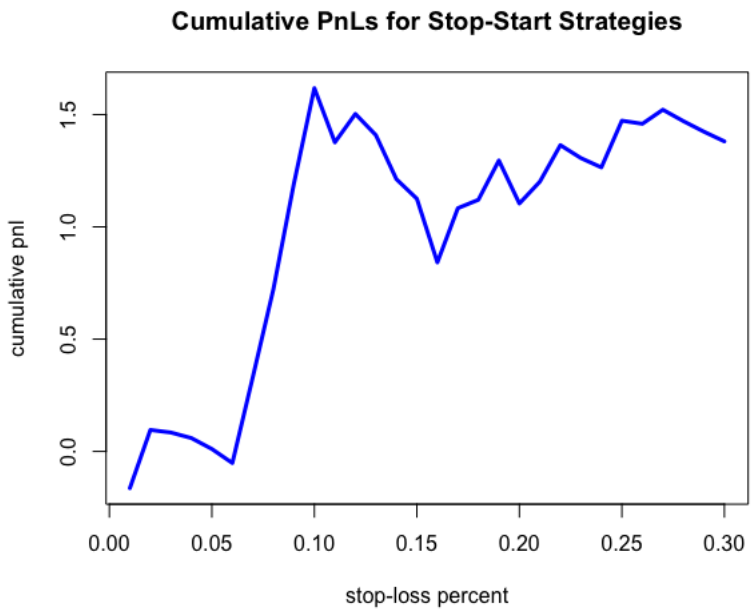
```
> # Simulate stop-start strategy
> pnls <- sim_stopstart(0.1)
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<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], c
+ }  # 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
> stopl <- stopv[which.max(pnlc)]
```

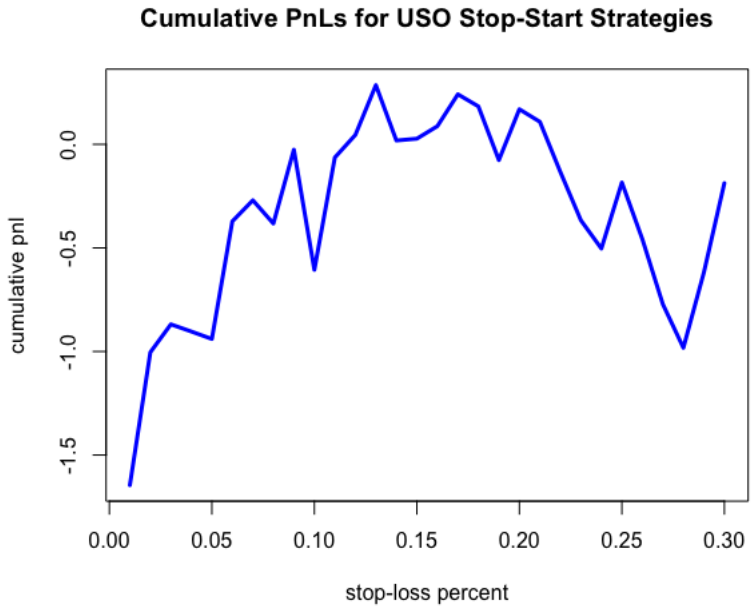



```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=stopv, y=pnlc,
+    main="Cumulative PnLs for Stop-Start Strategies",
+    xlab="stop-loss percent", ylab="cumulative pnl",
+    t="l", lwd=3, col="blue")
```

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)
> nrows <- 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
```




```
> # Plot cumulative pnls 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 <- stopv[which.max(pnlc)]
> 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], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Dynamic Documents Using *R markdown*

markdown is a simple markup language designed for creating documents in different formats, including *pdf* and *html*.

R Markdown is a modified version of *markdown*, which allows creating documents containing *math formulas* and `R` code embedded in them.

An *R Markdown* document (with extension `.Rmd`) contains:

- A *YAML* header,
- Text in *R Markdown* code format,
- Math formulas (equations), delimited using either single "$" symbols (for inline formulas), or double "$$" symbols (for display formulas),
- `R` code chunks, delimited using either single "`" backtick symbols (for inline code), or triple "```" backtick symbols (for display code).

The packages *rmarkdown* and *knitr* compile `R` documents into either *pdf*, *html*, or *MS Word* documents.

````
---
title: "My First R Markdown Document"
author: Jerzy Pawlowski
date: ''`r format(Sys.time(), "%m/%d/%Y")`''
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

# install package quantmod if it can't be loaded success
if (!require("quantmod"))
  install.packages("quantmod")
```

### R Markdown
This is an *R Markdown* document. Markdown is a simple f

One of the advantages of writing documents *R Markdown*

You can read more about publishing documents using *R* h
https://algoquant.github.io/r,/markdown/2016/07/02/Publi

You can read more about using *R* to create *HTML* docum
https://algoquant.github.io/2016/07/05/Interactive-Plots

Clicking the **Knit** button in *RStudio*, compiles the

Example of an *R* code chunk:
```{r cars}
summary(cars)
```

### Plots in *R Markdown* documents

Plots can also be embedded, for example:
```{r pressure, echo=FALSE}
plot(pressure)
````
````

Package *shiny* for Creating Interactive Applications

The package *shiny* creates interactive applications running in R, with their outputs presented as live visualizations.

Shiny allows changing the model parameters, recalculating the model, and displaying the resulting outputs as plots and charts.

A *shiny app* is a file with *shiny* commands and R code.

The *shiny* code consists of a *shiny interface* and a *shiny server*.

The *shiny interface* contains widgets for data input and an area for plotting.

The *shiny server* contains the R model code and the plotting code.

The function `shiny::fluidPage()` creates a GUI layout for the user inputs of model parameters and an area for plots and charts.

The function `shiny::renderPlot()` renders a plot from the outputs of a live model.

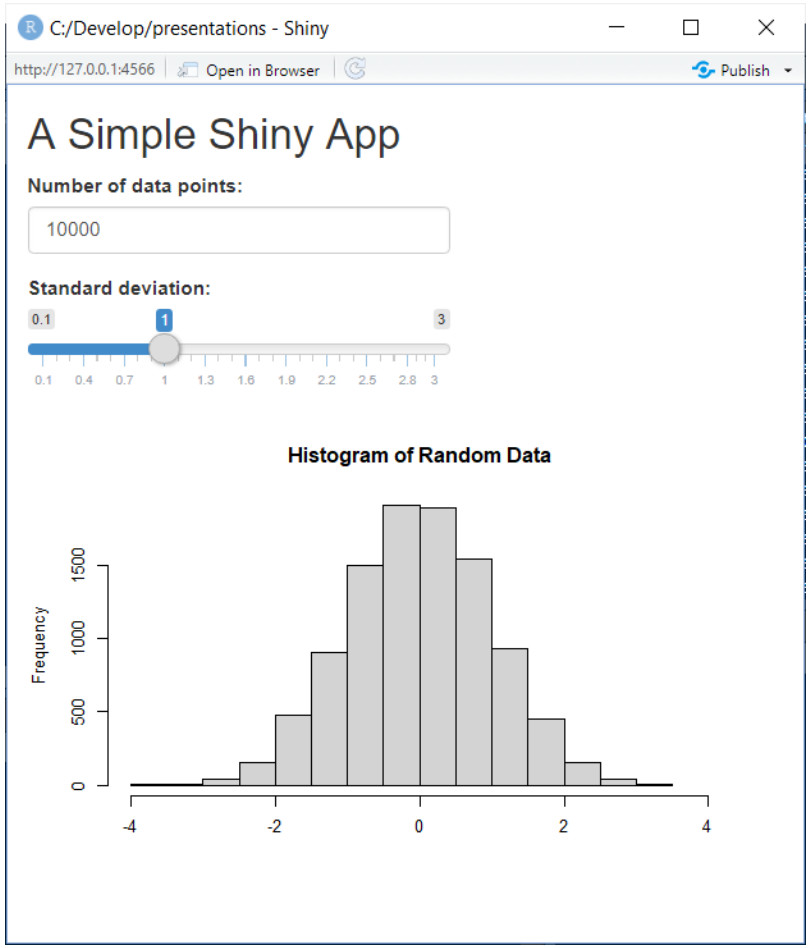
The function `shiny::shinyApp()` creates a `shiny` app from a *shiny interface* and a *shiny server*.

```
> ## App setup code that runs only once at startup.
> ndata <- 1e4
> stdev <- 1.0
>
> ## Define the user interface
> uiface <- shiny::fluidPage(
+   # Create numeric input for the number of data points.
+   numericInput("ndata", "Number of data points:", value=ndata),
+   # Create slider input for the standard deviation parameter.
+   sliderInput("stdev", label="Standard deviation:",
+         min=0.1, max=3.0, value=stdev, step=0.1),
+   # Render plot in a panel.
+   plotOutput("plotobj", height=300, width=500)
+ )  # end user interface
>
> ## Define the server function
> servfun <- function(input, output) {
+   output$plotobj <- shiny::renderPlot({
+     # Simulate the data
+     datav <- rnorm(input$ndata, sd=input$stdev)
+     # Plot the data
+     par(mar=c(2, 4, 4, 0), oma=c(0, 0, 0, 0))
+     hist(datav, xlim=c(-4, 4), main="Histogram of Random Data")
+   })  # end renderPlot
+ }  # end servfun
>
> # Return a Shiny app object
> shiny::shinyApp(ui=uiface, server=servfun)
```

Running Shiny Apps in *RStudio*

A *shiny app* can be run by pressing the "Run App" button in *RStudio*.

When the *shiny app* is run, the *shiny* commands are translated into *JavaScript* code, which creates a graphical user interface (GUI) with buttons, sliders, and boxes for data input, and also with the output plots and charts.



Positioning and Sizing Widgets Within the Shiny GUI

The functions `shiny::fluidRow()` and `shiny::column()` allow positioning and sizing widgets within the *shiny* GUI.

```
> ## Create elements of the user interface
> uiface <- shiny::fluidPage(
+   titlePanel("VWAP Moving Average"),
+   # Create single row of widgets with two slider inputs
+   fluidRow(
+     # Input stock symbol
+     column(width=3, selectInput("symbol", label="Symbol",
+                           choices=symbolv, selected=symbol)),
+     # Input look-back interval
+     column(width=3, sliderInput("look_back", label="Lookback inter
+                           min=1, max=150, value=11, step=1))
+   ),  # end fluidRow
+   # Create output plot panel
+   mainPanel(dygraphs::dygraphOutput("dyplot"), width=12)
+ )  # end fluidPage interface
```

Shiny Apps With Reactive Expressions

The package *shiny* allows specifying reactive expressions which are evaluated only when their input data is updated.

Reactive expressions avoid performing unnecessary calculations.

If the reactive expression is invalidated (recalculated), then other expressions that depend on its output are also recalculated.

This way calculations cascade through the expressions that depend on each other.

The function `shiny::reactive()` transforms an expression into a reactive expression.

```
> ## Define the server function
> servfun <- shiny::shinyServer(function(input, output) {
+   # Get the close and volume data in a reactive environment
+   closep <- shiny::reactive({
+     # Get the data
+     ohlc <- get(input$symbol, data_env)
+     closep <- log(quantmod::Cl(ohlc))
+     volum <- quantmod::Vo(ohlc)
+     # Return the data
+     cbind(closep, volum)
+   })  # end reactive code
+
+   # Calculate the VWAP indicator in a reactive environment
+   vwapv <- shiny::reactive({
+     # Get model parameters from input argument
+     look_back <- input$look_back
+     # Calculate the VWAP indicator
+     closep <- closep()[, 1]
+     volum <- closep()[, 2]
+     vwapv <- HighFreq::roll_sum(tseries=closep*volum, look_back=lo
+     volumroll <- HighFreq::roll_sum(tseries=volum, look_back=look_
+     vwapv <- vwapv/volumroll
+     vwapv[is.na(vwapv)] <- 0
+     # Return the plot data
+     datav <- cbind(closep, vwapv)
+     colnames(datav) <- c(input$symbol, "VWAP")
+     datav
+   })  # end reactive code
+
+   # Return the dygraph plot to output argument
+   output$dyplot <- dygraphs::renderDygraph({
+     colnamev <- colnames(vwapv())
+     dygraphs::dygraph(vwapv(), main=paste(colnamev[1], "VWAP")) %>
+ dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+ dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+ dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeWidt
+ dySeries(name=colnamev[2], axis="y2", label=colnamev[2], strokeWid
+   })  # end output plot
+ })  # end server code
```

Reactive Event Handlers

Event handlers are functions which evaluate expressions when an event occurs (like a button press).

The functions `shiny::observeEvent()` and `shiny::eventReactive()` are event handlers.

The function `shiny::eventReactive()` returns a value, while `shiny::observeEvent()` produces a side-effect, without returning a value.

The function `shiny::reactiveValues()` creates a list for storing reactive values, which can be updated by event handlers.

```
> ## Define the server function
> servfun <- shiny::shinyServer(function(input, output) {
+
+   # Create an empty list of reactive values.
+   value_s <- reactiveValues()
+
+   # Get input parameters from the user interface.
+   nrows <- reactive({
+     # Add nrows to list of reactive values.
+     value_s*nrows <- input$nrows
+     input$nrows
+   })  # end reactive code
+
+   # Broadcast a message to the console when the button is pressed.
+   observeEvent(eventExpr=input$button, handlerExpr={
+     cat("Input button pressed\n")
+   })  # end observeEvent
+
+   # Send the data when the button is pressed.
+   datav <- eventReactive(eventExpr=input$button, valueExpr={
+     # eventReactive() executes on input$button, but not on nrows()
+     cat("Sending", nrows(), "rows of data\n")
+     datav <- head(mtcars, input$nrows)
+     value_s$mpg <- mean(datav$mpg)
+     datav
+   })  # end eventReactive
+   #   datav
+
+   # Draw table of the data when the button is pressed.
+   observeEvent(eventExpr=input$button, handlerExpr={
+     datav <- datav()
+     cat("Received", value_s*nrows, "rows of data\n")
+     cat("Average mpg = ", value_s$mpg, "\n")
+     cat("Drawing table\n")
+     output$tablev <- renderTable(datav)
+   })  # end observeEvent
+
+ })  # end server code
>
```

Homework Assignment

Required

- Study all the lecture slides in `FRE7241_Lecture_2.pdf`, and run all the code in `FRE7241_Lecture_2.R`,
- Study *bootstrap simulation* from the files *bootstrap_technique.pdf* and *doBootstrap_primer.pdf*,
- Study the following sections in the file `numerical_analysis.pdf`:
  - Numerical Calculations,
  - Optimizing `R` Code for Speed and Memory Usage,
  - Writing Fast `R` Code Using Vectorized Operations,
  - Simulation,
  - Parallel Computing in `R`,
  - Run the code corresponding to the above sections from `numerical_analysis.R`

Recommended

Read the following sections in the file `R_environment.pdf`:

- *Environments in R*,
- *Data Input and Output*,
- Run the code corresponding to the above sections from `R_environment.R`