FRE7241 Algorithmic Portfolio Management

Lecture#5, Spring 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 16, 2024

Idiosyncratic Stock Returns

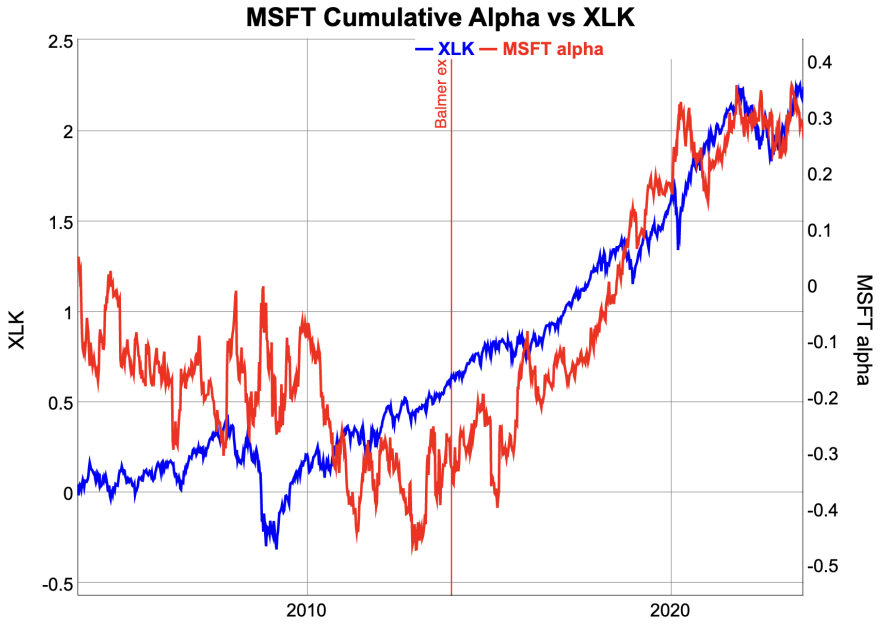
The daily stock returns $r_i - r_f$ in excess of the risk-free rate r_f, can be decomposed into *systematic* returns $\beta(r_m - r_f)$ (r_m are the market returns) plus *idiosyncratic* returns $\alpha + \varepsilon_i$ (which are uncorrelated to the market returns):

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

The *alpha* α are the abnormal returns in excess of the risk premium $\beta(r_m - r_f)$, and ε_i are the regression residuals with zero mean.

We can simplify the formula by setting the risk-free rate to zero $r_f = 0$ and redefining the alpha α.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData")
> # Select ETF returns
> retetf <- rutils::etfenv$returns[, c("VTI", "XLK", "XLF", "XLE")]
> # Calculate the MSFT betas with respect to different ETFs
> betas <- sapply(retetf, function(retetf) {
+   retp <- na.omit(cbind(retstock$MSFT, retetf))
+   # Calculate the MSFT beta
+   drop(cov(retp$MSFT, retp[, 2])/var(retp[, 2]))
+ }) # end sapply
> # Combine MSFT and XLK returns
> retp <- cbind(retstock$MSFT, rutils::etfenv$returns$XLK)
> retp <- na.omit(retp)
> colnames(retp) <- c("MSFT", "XLK")
> # Calculate the beta and alpha of returns MSFT ~ XLK
> betac <- drop(cov(retp$MSFT, retp$XLK)/var(retp$XLK))
> alphac <- retp$MSFT - betac*retp$XLK
> # Scatterplot of returns
> plot(MSFT ~ XLK, data=retp, main="MSFT ~ XLK Returns",
+      xlab="XLK", ylab="MSFT", pch=1, col="blue")
```



The stock of Microsoft *MSFT* began outperforming the *XLK* ETF after Steve Ballmer was replaced as CEO in 2014.

```
> # dygraph plot of MSFT idiosyncratic returns vs XLK
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> datev <- zoo::index(retp)[endd]
> dateb <- datev[findInterval(as.Date("2014-01-01"), datev)] # Steve
> datav <- cbind(retp$XLK, alphac)
> colnames(datav)[2] <- "MSFT alpha"
> colnamev <- colnames(datav)
> dygraphs::dygraph(cumsum(datav)[endd], main="MSFT Cumulative Alpha
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyEvent(dateb, label="Balmer exit", strokePattern="solid", color
+   dyLegend(show="always", width=300)
```

Trailing Idiosyncratic Stock Returns

In practice, the stock beta should be updated over time and applied out-of-sample to calculate the trailing idiosyncratic stock returns.

The trailing beta β of a stock with returns r_t with respect to a stock index with returns R_t can be updated using these recursive formulas with the weight decay factor λ:

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

$$\bar{R}_t = \lambda \bar{R}_{t-1} + (1-\lambda) R_t$$

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

$$\text{cov}_t = \lambda \, \text{cov}_{t-1} + (1-\lambda)(r_t - \bar{r}_t)(R_t - \bar{R}_t)$$

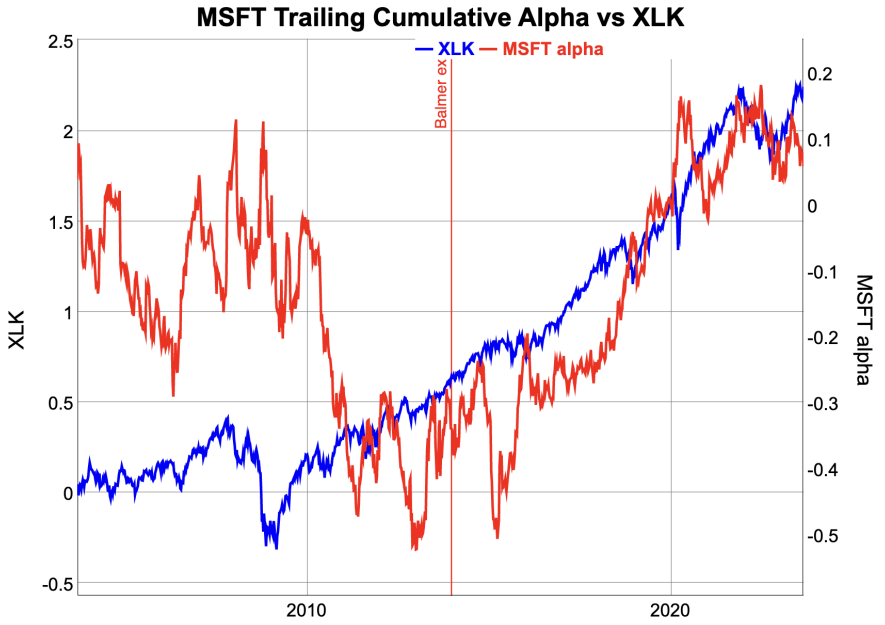
$$\beta_t = \frac{\text{cov}_t}{\sigma_t^2}$$

$$\alpha_t = r_t - \beta_t R_t$$

The parameter λ determines the rate of decay of the weight of past returns. If λ is close to 1 then the decay is weak and past returns have a greater weight. And vice versa if λ is close to 0.

The function `HighFreq::run_covar()` calculates the trailing variances, covariances, and means of two *time series*.

Using a dynamic beta produces a similar picture of *MSFT* stock performance versus the *XLK* ETF.




```
> # Calculate the trailing alphas and betas
> lambdaf <- 0.9
> covarv <- HighFreq::run_covar(retp, lambdaf)
> covarv[1, ] <- 1.0
> betac <- covarv[, 1]/covarv[, 3]
> alphac <- retp$MSFT - betac*retp$XLK
> # dygraph plot of trailing MSFT idiosyncratic returns vs XLK
> datav <- cbind(retp$XLK, alphac)
> colnames(datav)[2] <- "MSFT alpha"
> colnamev <- colnames(datav)
> dygraphs::dygraph(cumsum(datav)[endd], main="MSFT Trailing Cumulat
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyEvent(dateb, label="Balmer exit", strokePattern="solid", colo
+   dyLegend(show="always", width=300)
```

The Dickey-Fuller Process

The *Dickey-Fuller* process is a combination of an *Ornstein-Uhlenbeck* process and an *autoregressive* process.

The returns r_t are equal to the sum of a mean reverting term plus *autoregressive* terms:

$$r_t = \theta(\mu - p_{t-1}) + \varphi_1 r_{t-1} + \ldots + \varphi_n r_{t-n} + \sigma \xi_t$$

$$p_t = p_{t-1} + r_t$$

Where μ is the equilibrium price, σ is the volatility of returns, θ is the strength of mean reversion, and ξ_t are standard normal *innovations*.

Then the prices follow an *autoregressive* process:

$$p_t = \theta\mu + (1 + \varphi_1 - \theta)p_{t-1} + (\varphi_2 - \varphi_1)p_{t-2} + \ldots + (\varphi_n - \varphi_{n-1})p_{t-n} - \varphi_n p_{t-n-1} + \sigma \xi_t$$

The sum of the *autoregressive* coefficients is equal to $1 - \theta$, so if the mean reversion parameter θ is positive: $\theta > 0$, then the time series p_t exhibits mean reversion and has no *unit root*.

```
> # Define Dickey-Fuller parameters
> prici <- 0.0;  priceq <- 1.0
> thetav <- 0.01;  nrows <- 1000
> coeff <- c(0.1, 0.39, 0.5)
> # Initialize the data
> innov <- rnorm(nrows, sd=0.01)
> retp <- numeric(nrows)
> pricev <- numeric(nrows)
> # Simulate Dickey-Fuller process using recursive loop in R
> retp[1] <- innov[1]
> pricev[1] <- prici
> retp[2] <- thetav*(priceq - pricev[1]) + coeff[1]*retp[1] +
+   innov[2]
> pricev[2] <- pricev[1] + retp[2]
> retp[3] <- thetav*(priceq - pricev[2]) + coeff[1]*retp[2] +
+   coeff[2]*retp[1] + innov[3]
> pricev[3] <- pricev[2] + retp[3]
> for (it in 4:nrows) {
+   retp[it] <- thetav*(priceq - pricev[it-1]) +
+     retp[(it-1):(it-3)] %*% coeff + innov[it]
+   pricev[it] <- pricev[it-1] + retp[it]
+ }  # end for
> # Simulate Dickey-Fuller process in Rcpp
> pricecpp <- HighFreq::sim_df(init_price=prici, eq_price=priceq,
+    theta=thetav, coeff=matrix(coeff), innov=matrix(innov))
> # Compare prices
> all.equal(pricev, drop(pricev_cpp))
> # Compare the speed of R code with Rcpp
> library(microbenchmark)
> summary(microbenchmark(
+   Rcode={for (it in 4:nrows) {
+   retp[it] <- thetav*(priceq - pricev[it-1]) + retp[(it-1):(it-3)]
+   pricev[it] <- pricev[it-1] + retp[it]
+   }},
+   Rcpp=HighFreq::sim_df(init_price=prici, eq_price=priceq, theta=t
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```

Augmented Dickey-Fuller ADF Test for Unit Roots

The *Augmented Dickey-Fuller ADF* test is designed to test the *null hypothesis* that a time series has a *unit root*.

The *ADF* test fits an autoregressive model for the prices p_t:

$$r_t = \theta(\mu - p_{t-1}) + \varphi_1 r_{t-1} + \ldots + \varphi_n r_{t-n} + \sigma \xi_t$$

$$p_t = p_{t-1} + r_t$$

Where μ is the equilibrium price, σ is the volatility of returns, and θ is the strength of mean reversion.

ε_i are the *residuals*, which are assumed to be standard normally distributed $\phi(0, \sigma_\varepsilon)$, independent, and stationary.

If the mean reversion parameter θ is positive: $\theta > 0$, then the time series p_t exhibits mean reversion and has no *unit root*.

The *null hypothesis* is that prices have a unit root ($\theta = 0$, no mean reversion), while the alternative hypothesis is that it's *stationary* ($\theta > 0$, mean reversion).

The *ADF* test statistic is equal to the *t*-value of the θ parameter: $t_\theta = \hat{\theta}/SE_\theta$ (which follows a different distribution from the t-distribution).

The function `tseries::adf.test()` performs the *ADF* test.

```
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection"); innov
> # Simulate AR(1) process with coefficient=1, with unit root
> arimav <- HighFreq::sim_ar(coeff=matrix(1), innov=innov)
> x11(); plot(arimav, t="l", main="AR(1) coefficient = 1.0")
> # Perform ADF test with lag = 1
> tseries::adf.test(arimav, k=1)
> # Perform standard Dickey-Fuller test
> tseries::adf.test(arimav, k=0)
> # Simulate AR(1) with coefficient close to 1, without unit root
> arimav <- HighFreq::sim_ar(coeff=matrix(0.99), innov=innov)
> x11(); plot(arimav, t="l", main="AR(1) coefficient = 0.99")
> tseries::adf.test(arimav, k=1)
> # Simulate Ornstein-Uhlenbeck OU process with mean reversion
> prici <- 0.0; priceq <- 0.0; thetav <- 0.1
> pricev <- HighFreq::sim_ou(init_price=prici, eq_price=priceq,
+   theta=thetav, innov=innov)
> x11(); plot(pricev, t="l", main=paste("OU coefficient =", thetav))
> tseries::adf.test(pricev, k=1)
> # Simulate Ornstein-Uhlenbeck OU process with zero reversion
> thetav <- 0.0
> pricev <- HighFreq::sim_ou(init_price=prici, eq_price=priceq,
+   theta=thetav, innov=innov)
> x11(); plot(pricev, t="l", main=paste("OU coefficient =", thetav))
> tseries::adf.test(pricev, k=1)
```

The common practice is to use a small number of lags in the *ADF* test, and if the residuals are autocorrelated, then to increase them until the correlations are no longer significant.

If the number of lags in the regression is zero: $n = 0$ then the *ADF* test becomes the standard *Dickey-Fuller* test: $r_t = \theta(\mu - p_{t-1}) + \varepsilon_i$.

Cointegration of Stocks Prices

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

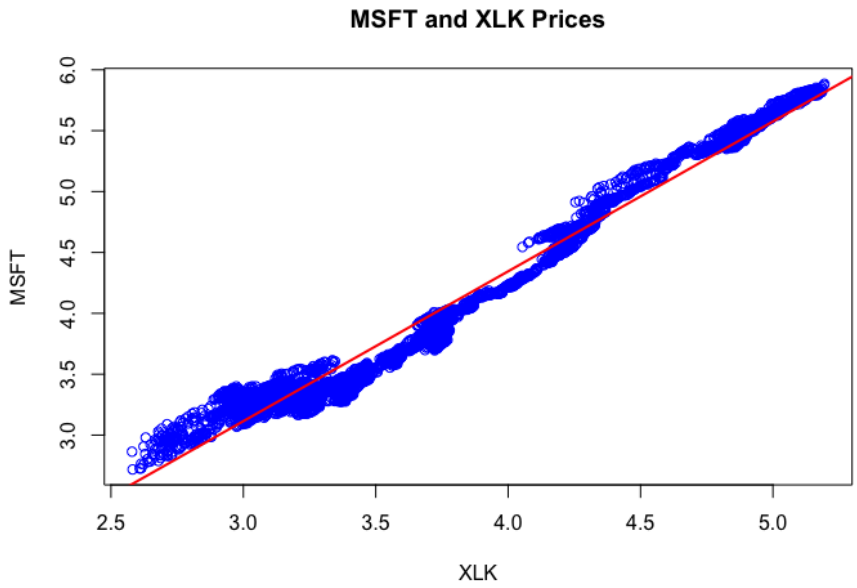
For two stocks, the cointegrating factor β can be obtained from the regression of the stock prices. The cointegrated portfolio price R is the residual of the regression:

$$R = p_1 - \beta p_2$$

Regressing the stock *prices* produces a *cointegrated* portfolio, with a small variance of its *prices*.

Regressing the stock *returns* produces a *correlated* portfolio, with a small variance of its *returns*.

The standard confidence intervals are not valid for the regression of prices, because prices are not stationary and are not normally distributed.




```
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices
> # Combine MSFT and XLK prices
> pricev <- cbind(rutils::etfenv$prices$XLK, pricestock$MSFT)
> pricev <- log(na.omit(pricev))
> colnames(pricev) <- c("XLK", "MSFT")
> datev <- zoo::index(pricev)
> # Calculate the beta regression coefficient of prices MSFT ~ XLK
> betac <- drop(cov(pricev$MSFT, pricev$XLK)/var(pricev$XLK))
> # Calculate the cointegrated portfolio prices
> pricec <- pricev$MSFT - betac*pricev$XLK
> colnames(pricec) <- "MSFT Coint XLK"
```

```
> # Scatterplot of MSFT and XLK prices
> plot(MSFT ~ XLK, data=pricev, main="MSFT and XLK Prices",
+      xlab="XLK", ylab="MSFT", pch=1, col="blue")
> abline(a=mean(pricec), b=betac, col="red", lwd=2)
> # Plot time series of prices
> endd <- rutils::calc_endpoints(pricev, interval="weeks")
> dygraphs::dygraph(pricev[endd], main="MSFT and XLK Log Prices") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

Cointegrated Portfolio Prices

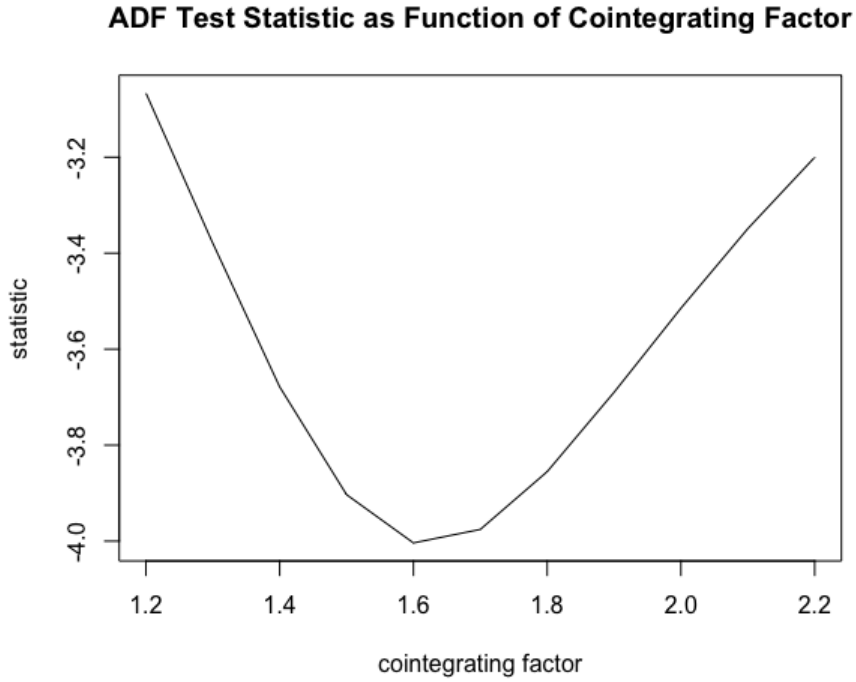
The Engle-Granger two-step procedure can be used to test the cointegrated portfolio of two stocks:

- Perform a regression of the stock prices to calculate the cointegrating factor β,
- Apply the *ADF* test to the cointegrated portfolio price, to determine if it has a unit root (the portfolio price diverges), or if it's mean reverting.

The *p*-value of the *ADF* test for the cointegrated portfolio of *MSFT* and *XLK* is not small, so the *null hypothesis* that it has a *unit root* (it diverges) cannot be rejected.

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

The *ADF* test statistic for the cointegrated portfolio is smaller than for either *MSFT* or *XLK* alone, which indicates that it's more mean-reverting.




```
> # Plot histogram of the cointegrated portfolio prices
> hist(pricec, breaks=100, xlab="Prices",
+   main="Histogram of Cointegrated Portfolio Prices")
> # Plot of cointegrated portfolio prices
> datav <- cbind(pricev$XLK, pricec)[endd]
> colnames(datav)[2] <- "Cointegrated Portfolio"
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav, main="MSFT and XLK Cointegrated Portfol
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
```

```
> # Perform ADF test on the individual stocks
> sapply(pricev, tseries::adf.test, k=1)
> # Perform ADF test on the cointegrated portfolio
> tseries::adf.test(pricec, k=1)
> # Perform ADF test for vector of cointegrating factors
> betas <- seq(1.2, 2.2, 0.1)
> adfstat <- sapply(betas, function(betac) {
+   pricec <- (pricev$MSFT - betac*pricev$XLK)
+   tseries::adf.test(pricec, k=1)$statistic
+ })  # end sapply
> # Plot ADF statistics for vector of cointegrating factors
> plot(x=betas, y=adfstat, type="l", xlab="cointegrating factor", yl
+  main="ADF Test Statistic as Function of Cointegrating Factor")
```

Bollinger Band Strategy for Cointegrated Pairs

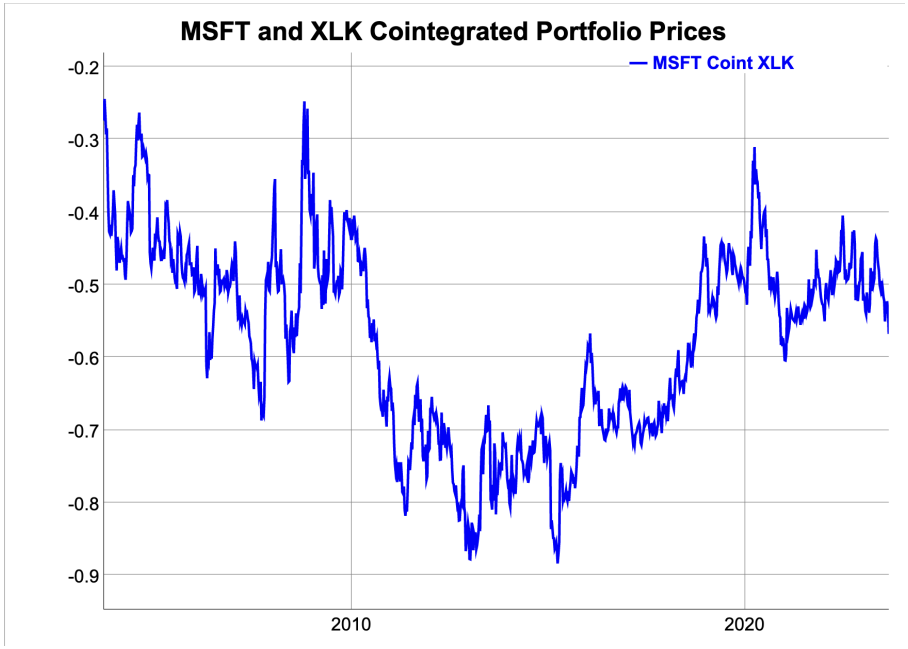
The returns of the cointegrated portfolio have negative autocorrelations, so it can be traded using a mean-reverting strategy.

The *Bollinger Band* strategy switches to long \$1 dollar of the stock when the portfolio price is cheap (below the lower band), and sells short -\$1 dollar of stock when the portfolio is 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 portfolio is cheap if its price is below its mean price minus the portfolio standard deviation, and it's rich (expensive) if its price is above the mean plus the standard deviation. The portfolio price is fair if it's equal to the mean price.

The pairs strategy is path dependent because it depends on the risk position. So the simulation requires performing a loop.



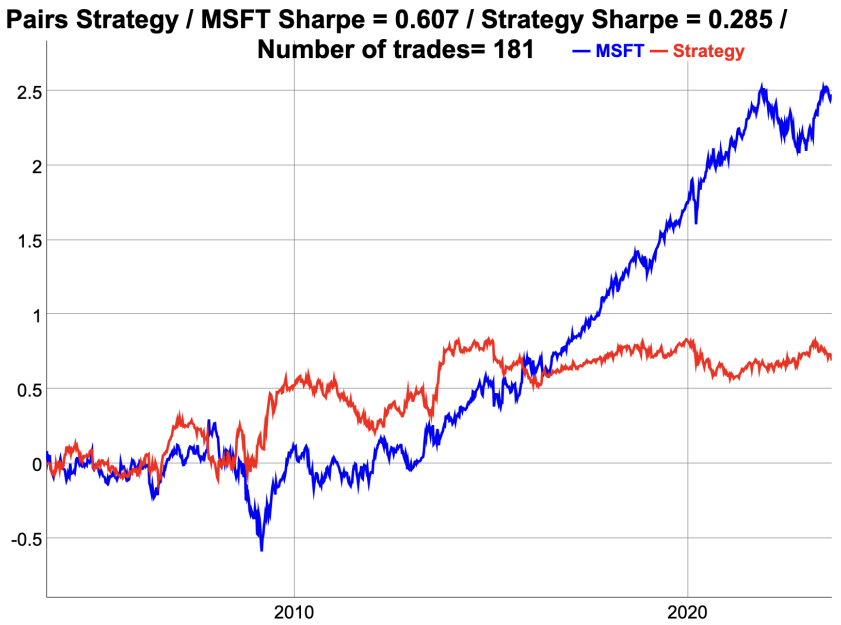

```
> # Plot of PACF of the cointegrated portfolio returns
> pricen <- zoo::coredata(pricec) # Numeric price
> retd <- rutils::diffit(pricen)
> pacf(retd, lag=10, xlab=NA, ylab=NA,
+      main="PACF of Cointegrated Portfolio Returns")
> # Dygraphs plot of cointegrated portfolio prices
> endd <- rutils::calc_endpoints(pricec, interval="weeks")
> dygraphs::dygraph(pricec[endd], main=
+   "MSFT and XLK Cointegrated Portfolio Prices") %>%
+   dyOptions(colors=c("blue"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Pairs Strategy With Fixed Beta

The pairs strategy performs well because it's in-sample - it uses the in-sample beta.

A more realistic strategy requires calculating the trailing betas on past prices, updating the pairs price, and updating the trailing mean and volatility.

```
> # Calculate the trailing mean prices and volatilities
> lambdaf <- 0.9
> meanv <- HighFreq::run_mean(pricen, lambda=lambdaf)
> volv <- HighFreq::run_var(pricen, lambda=lambdaf)
> volv <- sqrt(volv)
> # Simulate the pairs Bollinger strategy
> pricem <- pricen - meanv # De-meaned price
> nrows <- NROW(pricec)
> threshd <- rutils::lagit(volv)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> # Lag the positions to trade in the next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
```



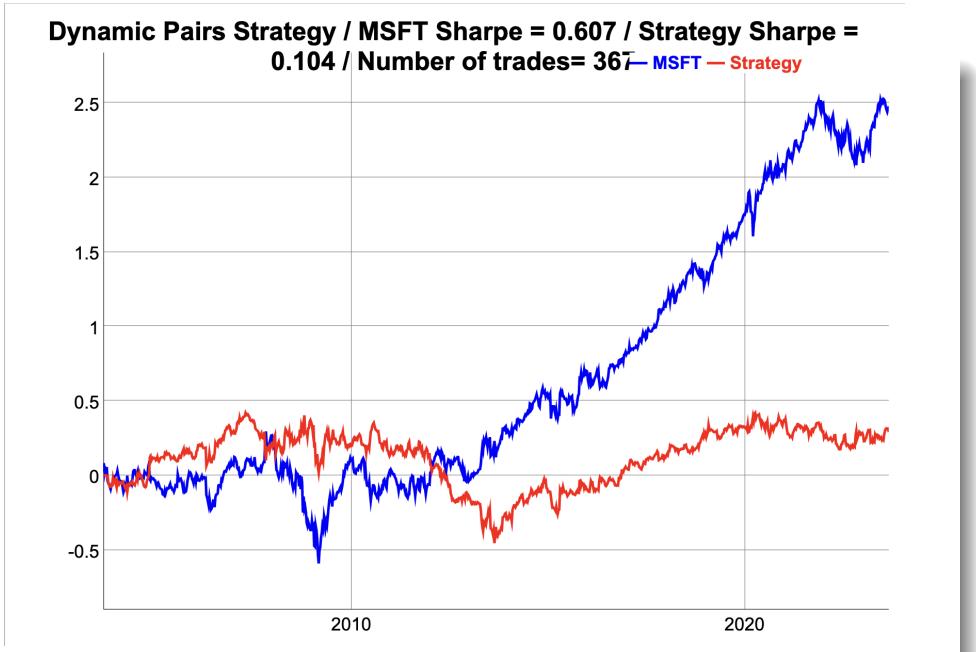

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Pairs 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=200)
```

Pairs Strategy With Dynamic Beta

In practice, the stock beta should be updated over time and applied out-of-sample to calculate the trailing cointegrated pair prices.

Using a dynamic beta produces poor performance for many stock and ETF pairs.

```
> # Calculate the trailing cointegrated pair prices
> covarv <- HighFreq::run_covar(pricev, lambdaf)
> betac <- covarv[, 1]/covarv[, 3]
> pricec <- (pricev$MSFT - betac*pricev$XLK)
> # Recalculate the mean of cointegrated portfolio prices
> meanv <- HighFreq::run_mean(pricec, lambda=lambdaf)
> vars <- sqrt(HighFreq::run_var(pricec, lambda=lambdaf))
> # Simulate the pairs Bollinger strategy
> pricen <- zoo::coredata(pricec) # Numeric price
> pricem <- pricen - meanv # De-meaned price
> threshd <- rutils::lagit(volv)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
```




```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Dynamic Pairs 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=200)
```

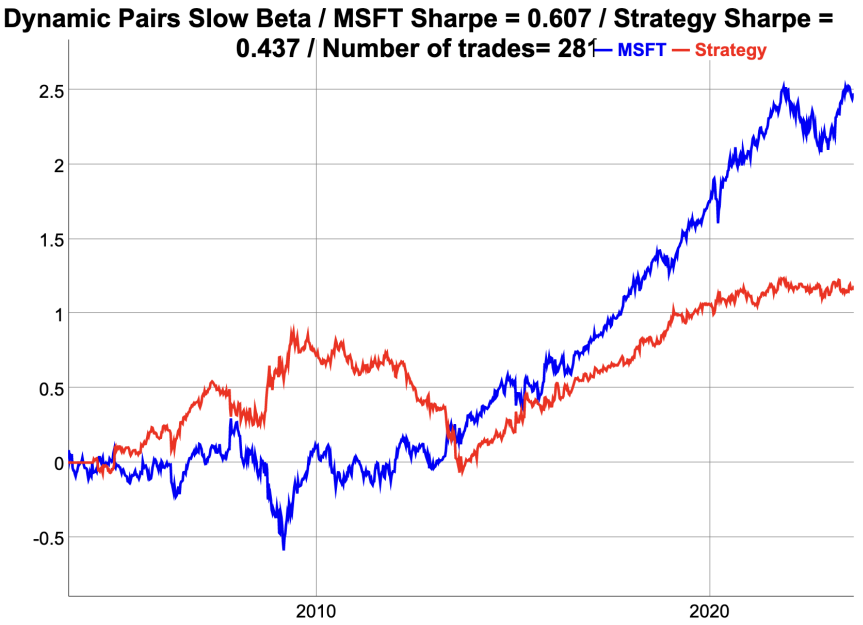
Pairs Strategy With Slow Beta

The pairs strategy can be improved by introducing an independent decay factor λ_β for calculating the trailing stock β.

The pairs strategy performs better with fast decay for calculating the trailing pairs volatility and slow decay for calculating the trailing stock β.

Independent decay factors for the trailing stock β and for the trailing pairs volatility improve the pairs strategy performance, but they also increase the risk of overfitting the model, because the more model parameters, the greater the risk of overfitting.

```
> # Calculate the trailing cointegrated pair prices
> covarv <- HighFreq::run_covar(pricev, lambda=0.95)
> betac <- covarv[, 1]/covarv[, 3]
> pricec <- (pricev$MSFT - betac*pricev$XLK)
> # Recalculate the mean of cointegrated portfolio prices
> meanv <- HighFreq::run_mean(pricec, lambda=0.3)
> vars <- sqrt(HighFreq::run_var(pricec, lambda=0.3))
> # Simulate the pairs Bollinger strategy
> pricen <- zoo::coredata(pricec) # Numeric price
> pricem <- pricen - meanv # De-meaned price
> threshd <- rutils::lagit(volv)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
```




```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Dynamic Pairs Slow Beta", "/ \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=200)
```

Stock Index Weighting Methods

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

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

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

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

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

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

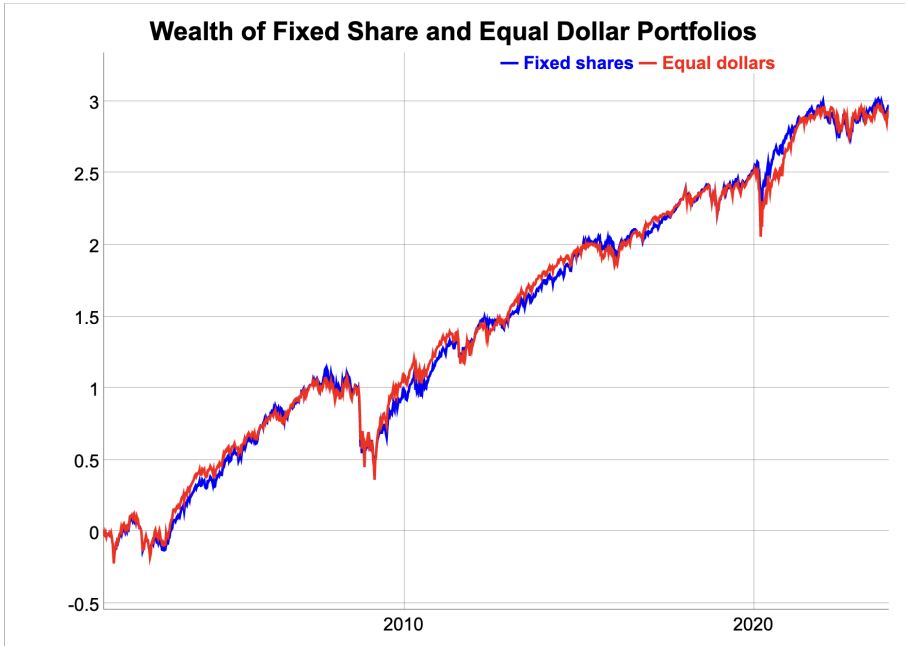
```
> # Calculate the percentage VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> nrows <- NROW(retvti)
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.R
> # Select the stock prices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

The Equal-Weight Portfolio

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

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

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




```
> # Wealth of fixed shares portfolio
> wealthfs <- rowMeans(cumprod(1 + retp))
> # Wealth of proportional dollar allocations (with rebalancing)
> wealthpd <- cumprod(1 + rowMeans(retp))
```

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

Random Stock Selection

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

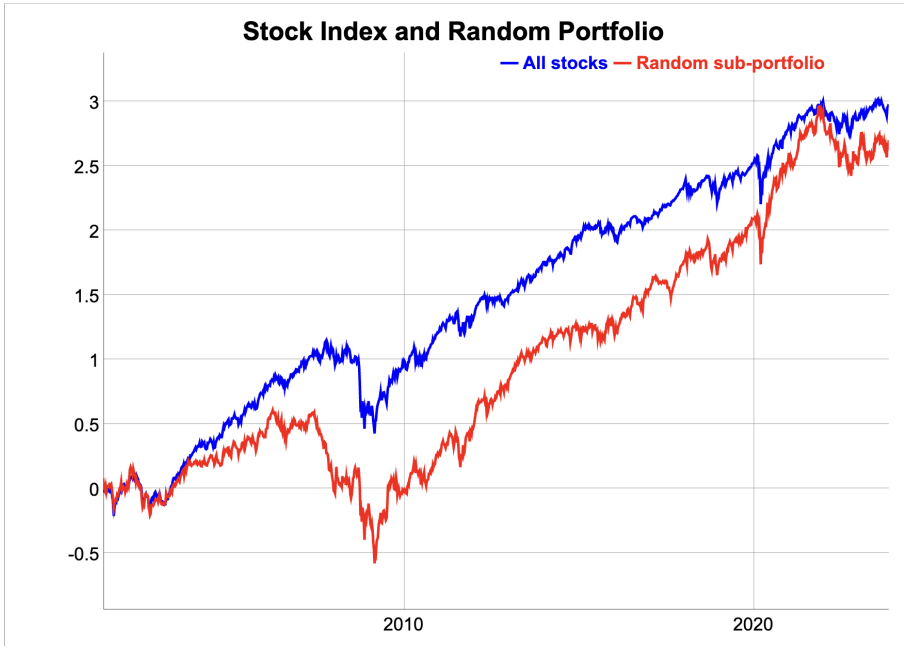
Random portfolios are used as a benchmark for stock pickers (portfolio managers).

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

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

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

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




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

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

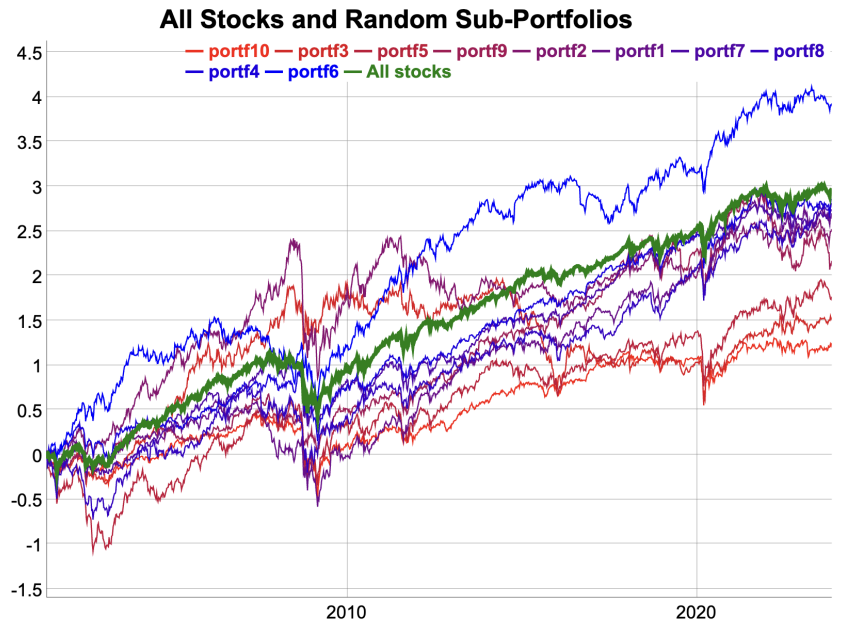
Random Stock Portfolios

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

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

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

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




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

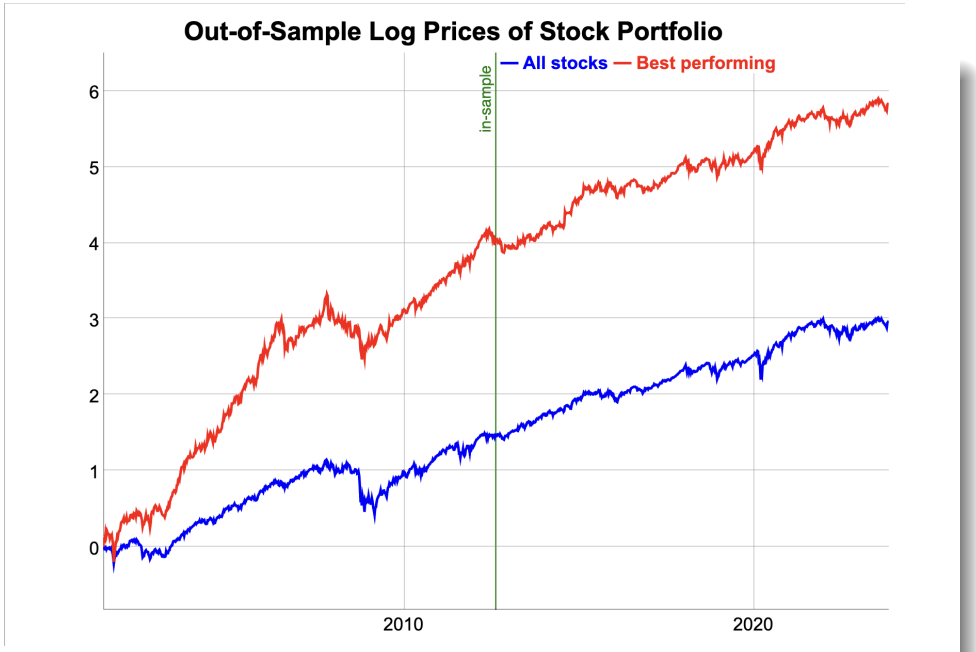
```
> # Plot dygraph of all stocks and random sub-portfolios
> colorv <- colorRampPalette(c("red", "blue"))(nportf)
> colorv <- c("green", colorv)
> wealthv <- cbind(wealthfs, wealthr)
> wealthv <- log(wealthv)
> colnames(wealthv)[1] <- "All stocks"
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(wealthv[endd], main="All Stocks and Random Sub-P
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=colnamev[1], label=colnamev[1], strokeWidth=3) %>%
+   dyLegend(show="always", width=500)
```

Stock Portfolio Selection Out-of-Sample

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

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

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




```
> # Define in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the 10 best performing stocks in-sample
> pricev <- cumprod(1 + retp)
> pricet <- pricev[cutoff, ]
> pricet <- drop(coredata(pricet))
> pricet <- sort(pricet, decreasing=TRUE)
> symbolv <- names(head(pricet, 10))
> # Calculate the wealth of the 10 best performing stocks
> wealthb <- rowMeans(pricev[, symbolv])
```

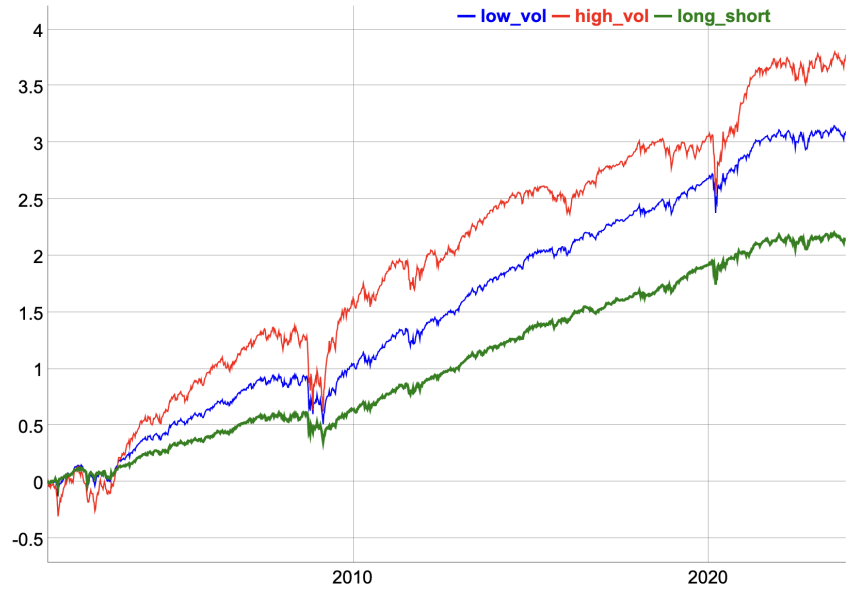
```
> # Combine the fixed share wealth with the 10 best performing stock
> wealthv <- cbind(wealthfs, wealthb)
> wealthv <- xts::xts(log(wealthv), order.by=datev)
> colnames(wealthv) <- c("All stocks", "Best performing")
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv[insample,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

Low and High Volatility Stock Portfolios

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

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

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




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

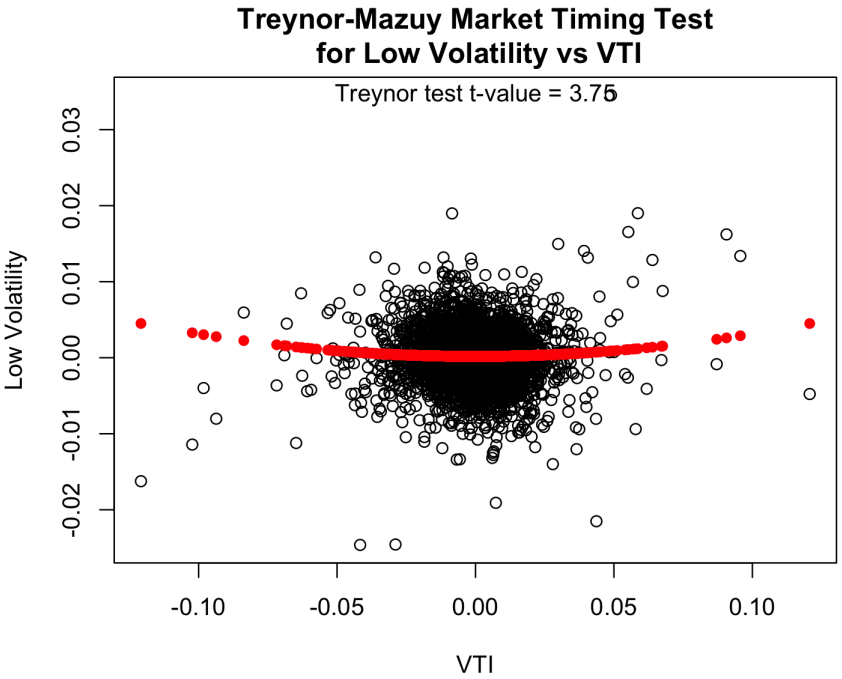
```
> # Calculate the returns of low and high volatility stocks
> retlow <- rowMeans(retp[, names(riskv[riskv <= medianv])], na.rm=T
> rethigh <- rowMeans(retp[, names(riskv[riskv > medianv])], na.rm=T
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Volati
+   dySeries(name=colnamev[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colnamev[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colnamev[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low Volatility Stock Portfolio Market Timing Skill

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

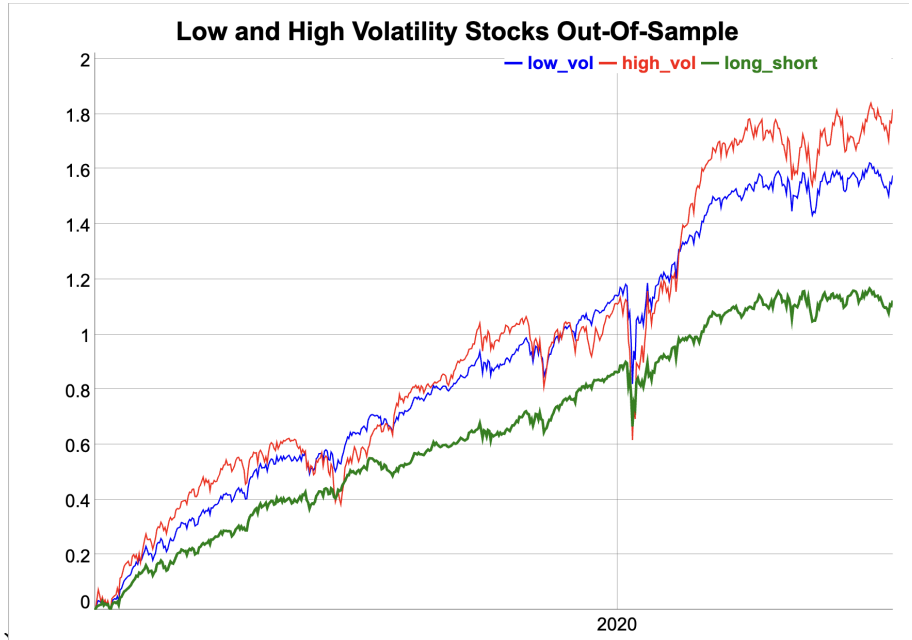
The *Treynor-Mazuy* test shows that the *betting against volatility* strategy has very small *market timing* skill.

```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm); sum
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="Low Volatility'
> title(main="Treynor-Mazuy Market Timing Test\n for Low Volatility
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), paste("Treynor test t-value =", tvalue))
```



Low and High Volatility Stock Portfolios Out-Of-Sample

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




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

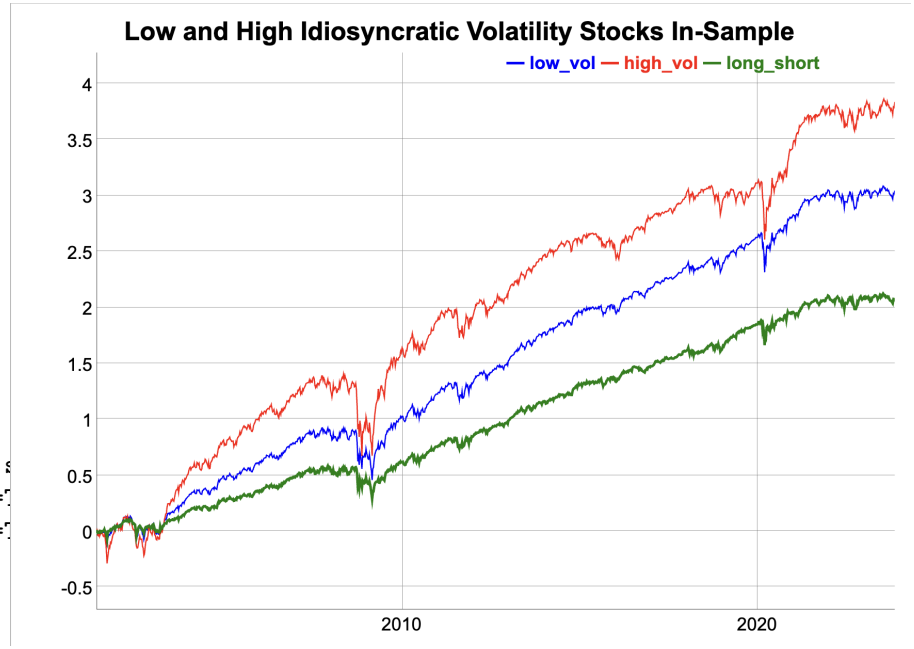
```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low and High Idiosyncratic Volatility Stock Portfolios

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

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

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



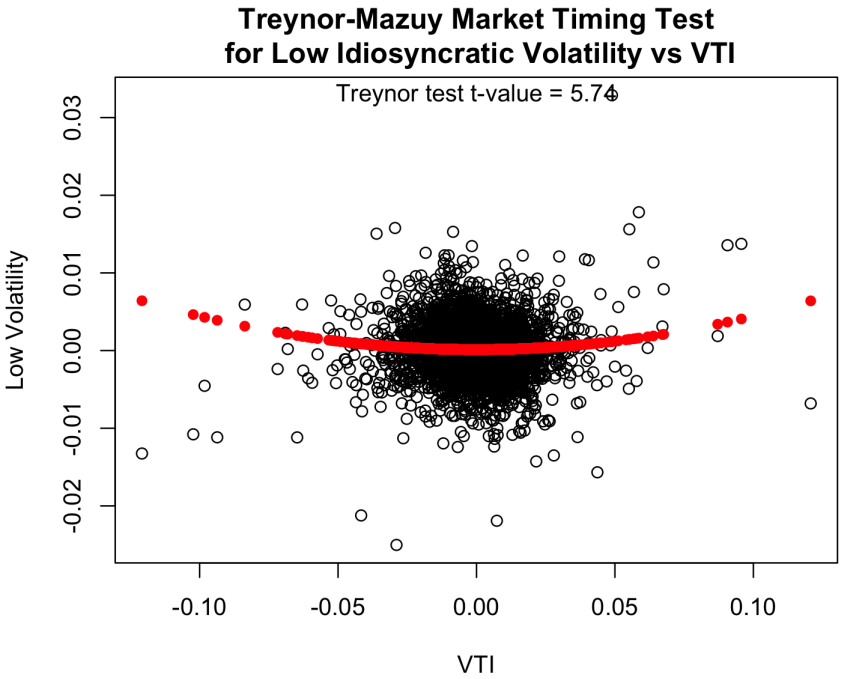

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of returns of low and high idiosyncratic volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low Idiosyncratic Volatility Stock Portfolio Market Timing Skill

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

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

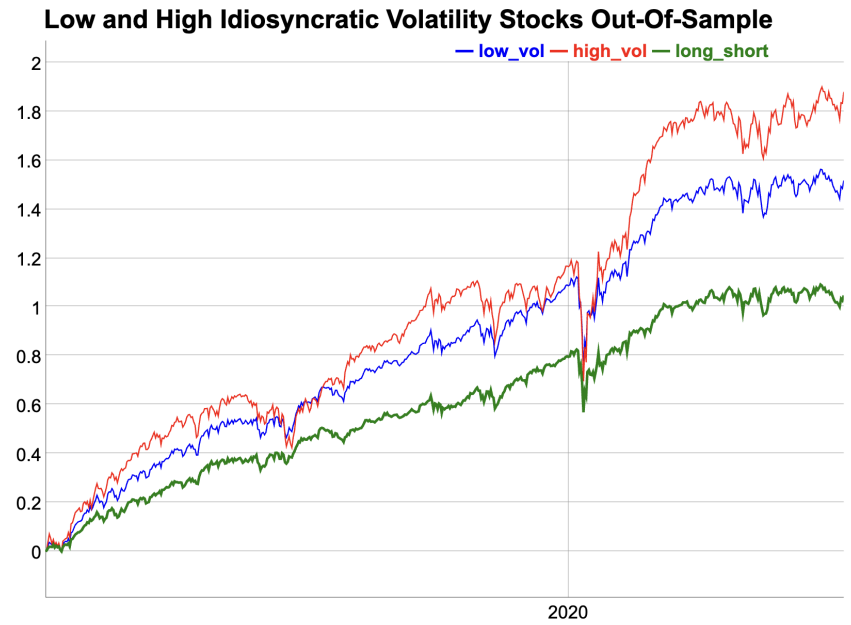
```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm);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



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

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

```
> # Calculate the median in-sample idiosyncratic volatility
> riskv <- riskretis[, "ivol"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high idiosyncrati
> retlow <- rowMeans(retp[outsample, names(riskv[riskv <= medianv])]
> rethigh <- rowMeans(retp[outsample, names(riskv[riskv > medianv])]
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
```



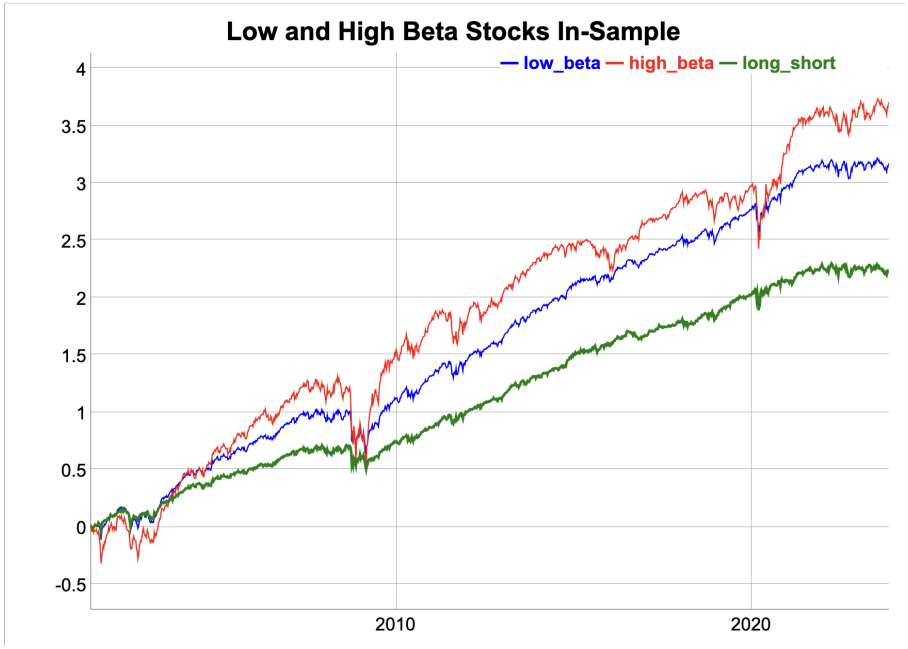

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of out-of-sample returns of low and high volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low and High Beta Stock Portfolios

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

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

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




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

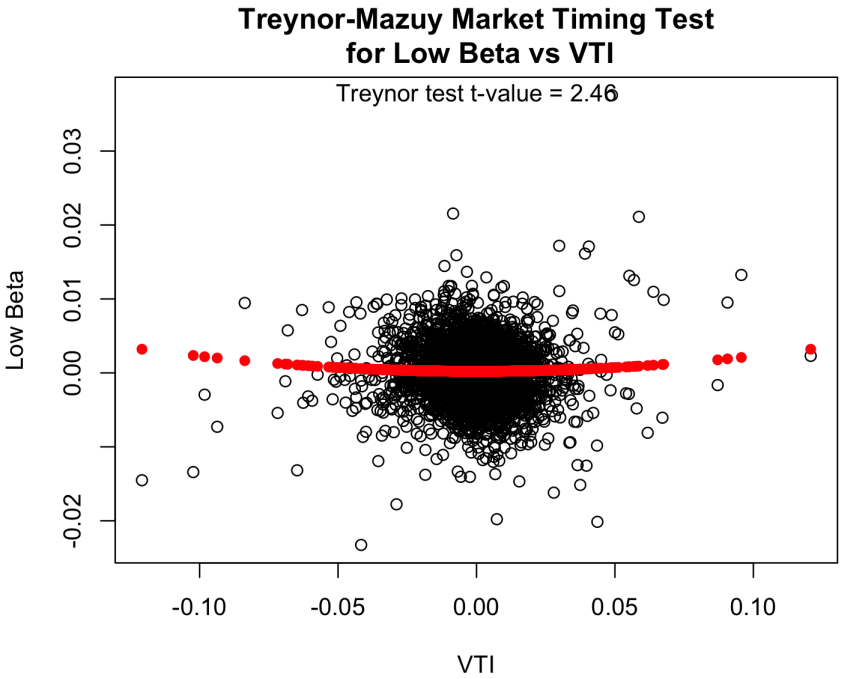
```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high beta stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low Beta Stock Portfolio Market Timing Skill

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

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

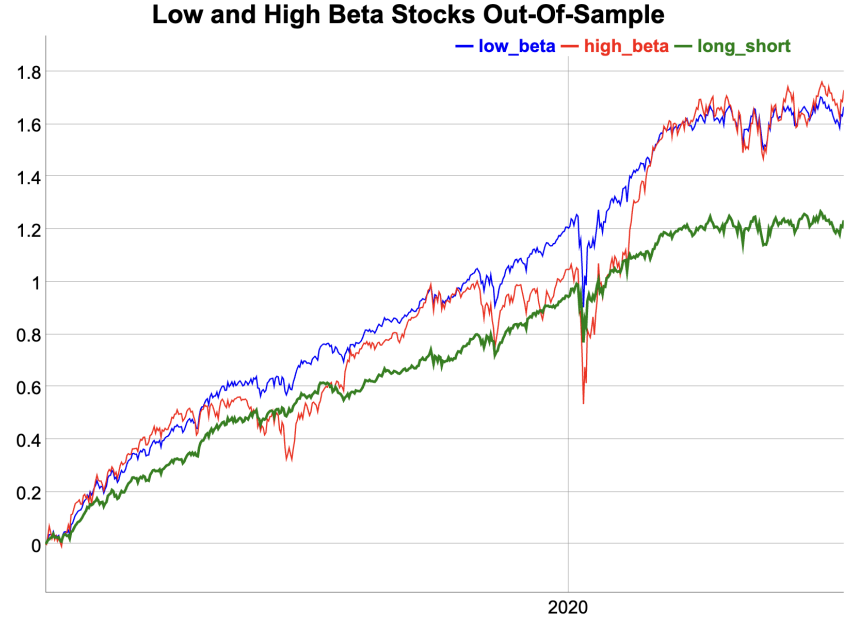
```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm);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



Low and High Beta Stock Portfolios Out-Of-Sample

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

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




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

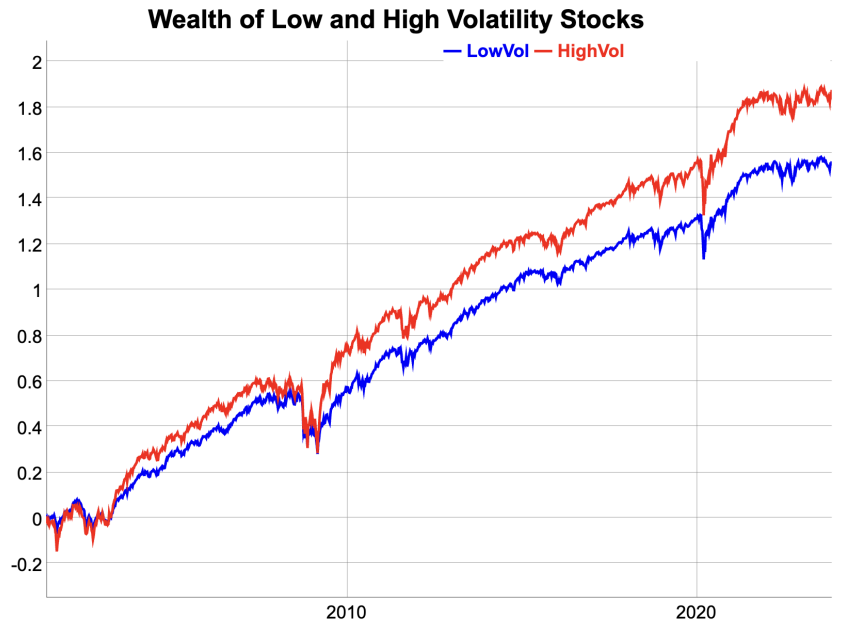
Stocks With Low and High Trailing Volatilities

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

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

The portfolios are rebalanced daily, as the volatility changes.

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




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

```
> # Combined wealth
> wealthv <- cbind(retlow, rethigh)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("LowVol", "HighVol")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Low and High Volatility Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

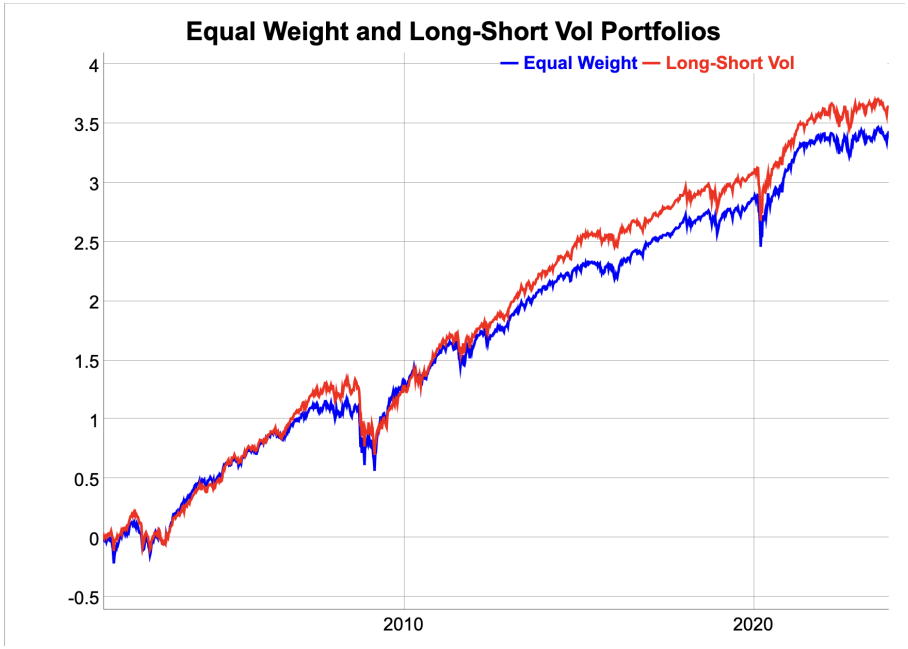
Long-Short Stock Volatility Strategy

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

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

The *Long-Short Volatility* strategy has a higher *Sharpe ratio* and also higher absolute returns than the equal weight (proportional dollar) strategy.

```
> # Calculate the returns of equal weight portfolio
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
> # Calculate the long-short volatility returns
> retls <- (retlow - 0.25*rethigh)
> # Scale the PnL volatility to that of wealthpd
> retls <- retls*sd(retew)/sd(retls)
> # Combined wealth
> wealthv <- cbind(retew, retls)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Equal Weight", "Long-Short Vol")
> 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])))
```




```
> # Plot of log wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Equal Weight and Long-Short Vol Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy for Stocks

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

The dollar amount of stock that has unit dollar volatility is equal to the *standardized prices* $\frac{p_i}{\sigma_i^d}$. Where σ_i^d is the dollar volatility.

So the weights w_i should be proportional to the *standardized prices*: $w_i \propto \frac{p_i}{\sigma_i^d}$,

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

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

Where σ_t^2 is the trailing variance at time t, and r_t is the *time series* of returns.

```
> # Calculate the trailing dollar volatilities
> volv <- HighFreq::run_var(retd, lambda=0.15)
> volv <- sqrt(volv)
> volv <- rutils::lagit(volv)
> volv[volv == 0] <- 1
> # Calculate the standardized prices with unit dollar volatility
> pricerp <- pricestock/volv
> # Scale the sum of stock prices to $1
> pricerp <- pricerp/rowMeans(pricerp)
> # Calculate the risk parity returns
> retrp <- retp*pricerp
> # Calculate the wealth of risk parity
> wealthrp <- cumprod(1 + rowMeans(retrp))
```

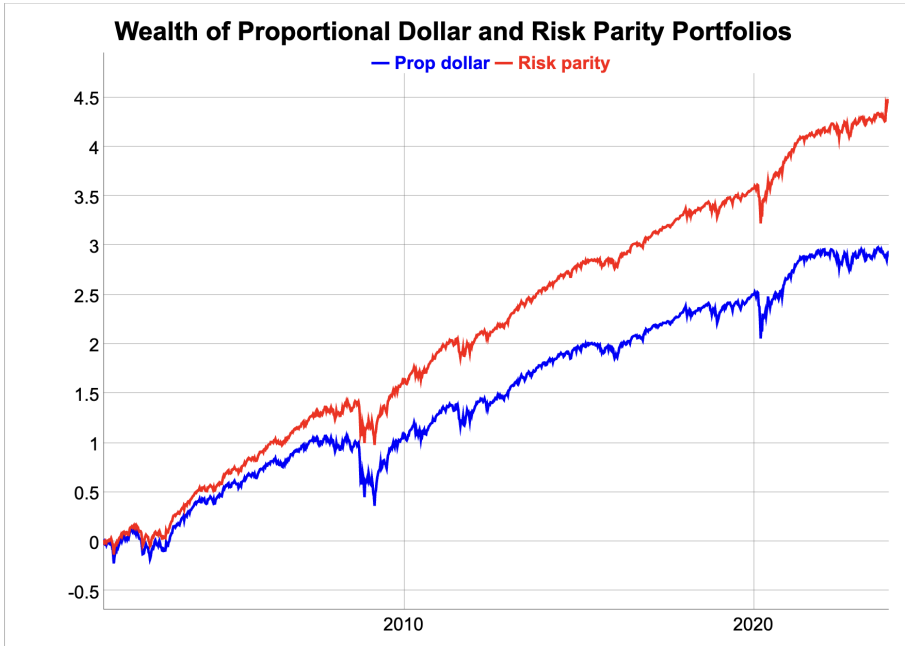
Risk Parity Strategy for Stocks Performance

The risk parity strategy for stocks has a significantly higher *Sharpe ratio* than the proportional dollar strategy because it's overweight low volatility stocks.

The risk parity strategy also has higher absolute returns.

But the risk parity strategy also has higher transaction costs.

```
> # Combined wealth
> wealthv <- cbind(wealthpd, wealthrp)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Prop dollar", "Risk parity")
> 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])))
```



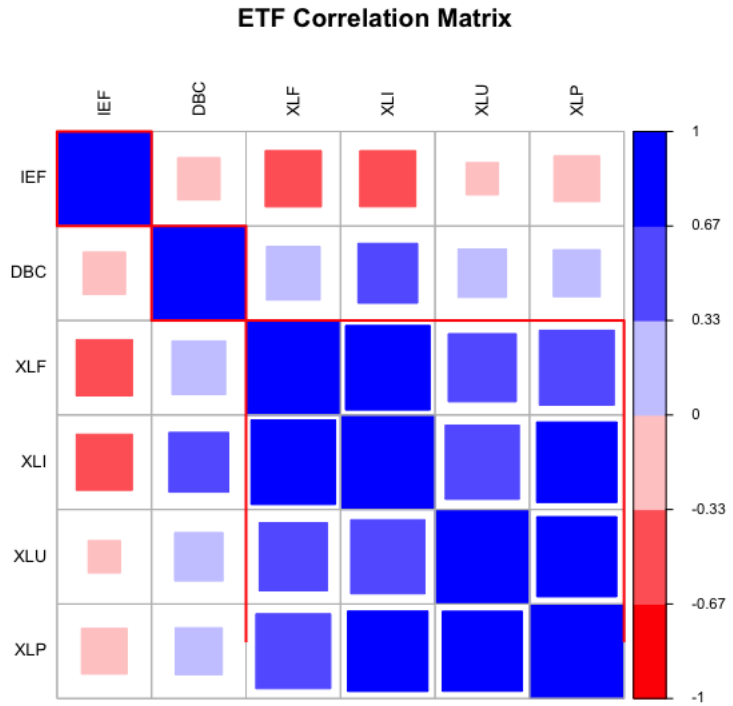

```
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Proportional Dollar and Risk Parity Portfolios")
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

Covariance Matrix of ETF Returns

The covariance matrix $\mathbb{C}$, of the return matrix $\mathbf{r}$ is given by:

$$\mathbb{C} = \frac{(\mathbf{r} - \bar{\mathbf{r}})^T (\mathbf{r} - \bar{\mathbf{r}})}{n - 1}$$

If the returns are *standardized* (centered and scaled) then the covariance matrix is equal to the correlation matrix.




```
> # Select ETF symbols
> symbolv <- c("IEF", "DBC", "XLU", "XLF", "XLP", "XLI")
> # Calculate the ETF prices and log returns
> pricev <- rutils::etfenv$prices[, symbolv]
> # Applying zoo::na.locf() can produce bias of the correlations
> # pricev <- zoo::na.locf(pricev, na.rm=FALSE)
> # pricev <- zoo::na.locf(pricev, fromLast=TRUE)
> pricev <- na.omit(pricev)
> retp <- rutils::diffit(log(pricev))
> # Calculate the covariance matrix
> covmat <- cov(retp)
> # Standardize (de-mean and scale) the returns
> retp <- lapply(retp, function(x) {(x - mean(x))/sd(x)})
> retp <- rutils::do_call(cbind, retp)
> round(sapply(retp, mean), 6)
> sapply(retp, sd)
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- apply(retp, 2, scale)
> # retp <- xts::xts(retp, zoo::index(pricev))
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- scale(retp, center=TRUE, scale=TRUE)
> # retp <- xts::xts(retp, zoo::index(pricev))
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- t(retp) - colMeans(retp)
> # retp <- retp/sqrt(rowSums(retp^2)/(NCOL(retp)-1))
> # retp <- t(retp)
```

```
> # Calculate the correlation matrix
> cormat <- cor(retp)
> # Reorder correlation matrix based on clusters
> library(corrplot)
> ordern <- corrMatOrder(cormat, order="hclust",
+   hclust.method="complete")
> cormat <- cormat[ordern, ordern]
> # Plot the correlation matrix
> colorv <- colorRampPalette(c("red", "white", "blue"))
> # x11(width=6, height=6)
> corrplot(cormat, title=NA, tl.col="black", mar=c(0,0,0,0),
+     method="square", col=colorv(NCOL(cormat)), tl.cex=0.8,
+     cl.offset=0.75, cl.cex=0.7, cl.align.text="l", cl.ratio=0.25)
```

Principal Component Vectors

Principal components are linear combinations of the k return vectors $\mathbf{r}_i$:

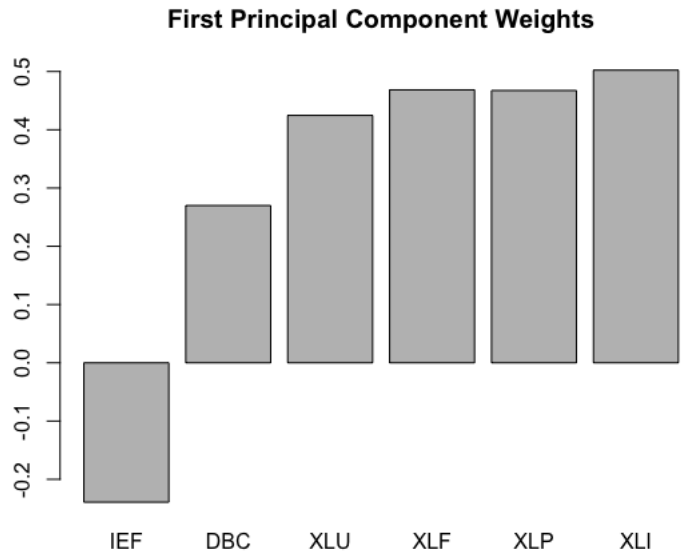
$$\mathbf{pc}_j = \sum_{i=1}^{k} w_{ij}\, \mathbf{r}_i$$

Where $\mathbf{w}_j$ is a vector of weights (loadings) of the *principal component* j, with $\mathbf{w}_j^T \mathbf{w}_j = 1$.

The weights $\mathbf{w}_j$ are chosen to maximize the variance of the *principal components*, under the condition that they are orthogonal:

$$\mathbf{w}_j = \arg\max \left\{ \mathbf{pc}_j^T\, \mathbf{pc}_j \right\}$$

$$\mathbf{pc}_i^T\, \mathbf{pc}_j = 0 \; (i \neq j)$$




```
> # Create initial vector of portfolio weights
> nweights <- NROW(symbolv)
> weightv <- rep(1/sqrt(nweights), nweights)
> names(weightv) <- symbolv
> # Objective function equal to minus portfolio variance
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   -sum(retp^2) + 1e4*(1 - sum(weightv^2))^2
+ }  # end objfun
> # Objective for equal weight portfolio
> objfun(weightv, retp)
> # Compare speed of vector multiplication methods
> summary(microbenchmark(
+   transp=(t(retp[, 1]) %*% retp[, 1]),
+   sumv=sum(retp[, 1]^2),
+   times=10))[, c(1, 4, 5)]
```

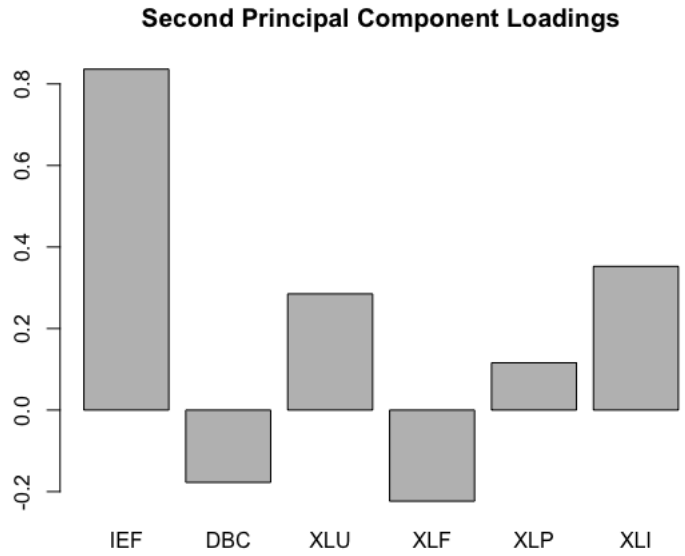
```
> # Find weights with maximum variance
> optim1 <- optim(par=weightv,
+   fn=objfun,
+   retp=retp,
+   method="L-BFGS-B",
+   upper=rep(10.0, nweights),
+   lower=rep(-10.0, nweights))
> # Optimal weights and maximum variance
> weights1 <- optim1$par
> -objfun(weights1, retp)
> # Plot first principal component weights
> barplot(weights1, names.arg=names(weights1), xlab="", ylab="",
+   main="First Principal Component Weights")
```

Higher Order Principal Components

The second *principal component* can be calculated by maximizing its variance, under the constraint that it must be orthogonal to the first *principal component*.

Similarly, higher order *principal components* can be calculated by maximizing their variances, under the constraint that they must be orthogonal to all the previous *principal components*.

```
> # PC1 returns
> pc1 <- drop(retp %*% weights1)
> # Redefine objective function
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   -sum(retp^2) + 1e4*(1 - sum(weightv^2))^2 +
+     1e4*(sum(weights1*weightv))^2
+ }  # end objfun
> # Find second PC weights using parallel DEoptim
> optim1 <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retp)),
+   lower=rep(-10, NCOL(retp)),
+   retp=retp, control=list(parVar="weights1",
+     trace=FALSE, itermax=1000, parallelType=1))
```




```
> # PC2 weights
> weights2 <- optim1$optim$bestmem
> names(weights2) <- colnames(retp)
> sum(weights2^2)
> sum(weights1*weights2)
> # PC2 returns
> pc2 <- drop(retp %*% weights2)
> # Plot second principal component loadings
> barplot(weights2, names.arg=names(weights2), xlab="", ylab="",
+   main="Second Principal Component Loadings")
```

Eigenvalues of the Correlation Matrix

The portfolio variance: $\mathbf{w}^T \mathbb{C} \mathbf{w}$ can be maximized under the *quadratic* weights constraint $\mathbf{w}^T \mathbf{w} = 1$, by maximizing the *Lagrangian* $\mathcal{L}$:

$$\mathcal{L} = \mathbf{w}^T \mathbb{C} \mathbf{w} - \lambda (\mathbf{w}^T \mathbf{w} - 1)$$

Where λ is a *Lagrange multiplier*.

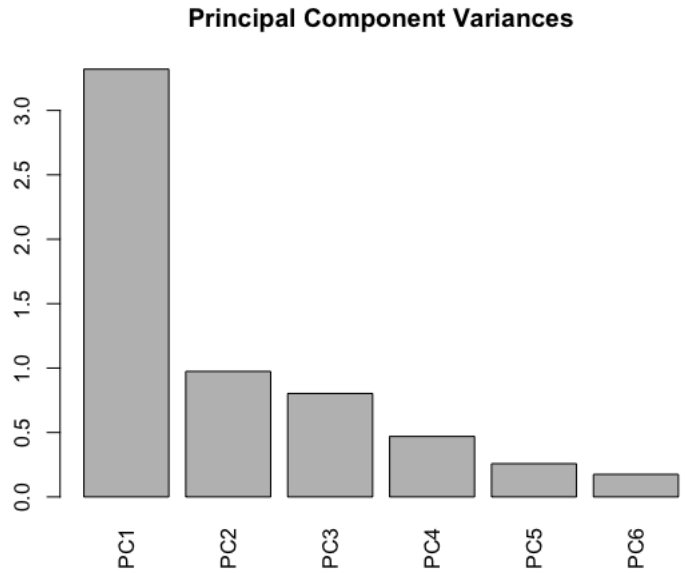
The maximum variance portfolio weights can be found by differentiating $\mathcal{L}$ with respect to $\mathbf{w}$ and setting it to zero:

$$\mathbb{C} \mathbf{w} = \lambda \mathbf{w}$$

This is the *eigenvalue* equation of the covariance matrix $\mathbb{C}$, with the optimal weights $\mathbf{w}$ forming an *eigenvector*, and λ is the *eigenvalue* corresponding to the *eigenvector* $\mathbf{w}$.

The *eigenvalues* are the variances of the *eigenvectors*, and their sum is equal to the sum of the return variances:

$$\sum_{i=1}^{k} \lambda_i = \frac{1}{1-k} \sum_{i=1}^{k} \mathbf{r}_i^T \mathbf{r}_i$$




```
> # Calculate the eigenvalues and eigenvectors
> eigend <- eigen(cormat)
> eigend$vectors
> # Compare with optimization
> all.equal(sum(diag(cormat)), sum(eigend$values))
> all.equal(abs(eigend$vectors[, 1]), abs(weights1), check.attribute
> all.equal(abs(eigend$vectors[, 2]), abs(weights2), check.attribute
> all.equal(eigend$values[1], var(pc1), check.attributes=FALSE)
> all.equal(eigend$values[2], var(pc2), check.attributes=FALSE)
> # Eigenvalue equations
> (cormat %*% weights1) / weights1 / var(pc1)
> (cormat %*% weights2) / weights2 / var(pc2)
> # Plot eigenvalues
> barplot(eigend$values, names.arg=paste0("PC", 1:nweights),
+   las=3, xlab="", ylab="", main="Principal Component Variances")
```

Principal Component Analysis Versus *Eigen Decomposition*

Principal Component Analysis (*PCA*) is equivalent to the *eigen decomposition* of either the correlation or the covariance matrix.

If the input time series *are* scaled, then *PCA* is equivalent to the eigen decomposition of the *correlation matrix*.

If the input time series *are not* scaled, then *PCA* is equivalent to the eigen decomposition of the *covariance matrix*.

Scaling the input time series improves the accuracy of the *PCA dimension reduction*, allowing a smaller number of *principal components* to more accurately capture the data contained in the input time series.

The number of *eigenvalues* is equal to the dimension of the covariance matrix.

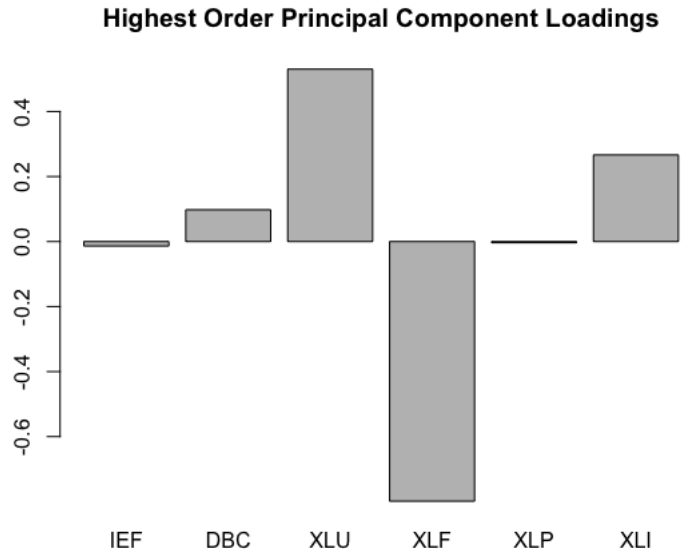
```
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> # Perform PCA with scaling
> pcad <- prcomp(retp, scale=TRUE)
> # Compare outputs
> all.equal(eigend$values, pcad$sdev^2)
> all.equal(abs(eigend$vectors), abs(pcad$rotation),
+     check.attributes=FALSE)
> # Eigen decomposition of covariance matrix
> eigend <- eigen(covmat)
> # Perform PCA without scaling
> pcad <- prcomp(retp, scale=FALSE)
> # Compare outputs
> all.equal(eigend$values, pcad$sdev^2)
> all.equal(abs(eigend$vectors), abs(pcad$rotation),
+     check.attributes=FALSE)
```

Minimum Variance Portfolio

The highest order *principal component*, with the smallest eigenvalue, has the lowest possible variance, under the *quadratic* weights constraint: $\mathbf{w}^T\mathbf{w} = 1$.

So the highest order *principal component* is equal to the *Minimum Variance Portfolio*.

```
> # Redefine objective function to minimize variance
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   sum(retp^2) + 1e4*(1 - sum(weightv^2))^2
+ }  # end objfun
> # Find highest order PC weights using parallel DEoptim
> optiml <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retp)),
+   lower=rep(-10, NCOL(retp)),
+   retp=retp, control=list(trace=FALSE,
+     itermax=1000, parallelType=1))
> # PC6 weights and returns
> weights6 <- optiml$optim$bestmem
> names(weights6) <- colnames(retp)
> sum(weights6^2)
> sum(weights1*weights6)
> # Compare with eigend vector
> weights6
> eigend$vectors[, 6]
> # Calculate the objective function
> objfun(weights6, retp)
> objfun(eigend$vectors[, 6], retp)
```




```
> # Plot highest order principal component loadings
> weights6 <- eigend$vectors[, 6]
> names(weights6) <- colnames(retp)
> barplot(weights6, names.arg=names(weights6), xlab="", ylab="",
+   main="Highest Order Principal Component Loadings")
```

Principal Component Analysis of ETF Returns

Principal Component Analysis (*PCA*) is a *dimension reduction* technique, that explains the returns of a large number of correlated time series as linear combinations of a smaller number of principal component time series.

The input time series are often scaled by their standard deviations, to improve the accuracy of *PCA dimension reduction*, so that more information is retained by the first few *principal component* time series.

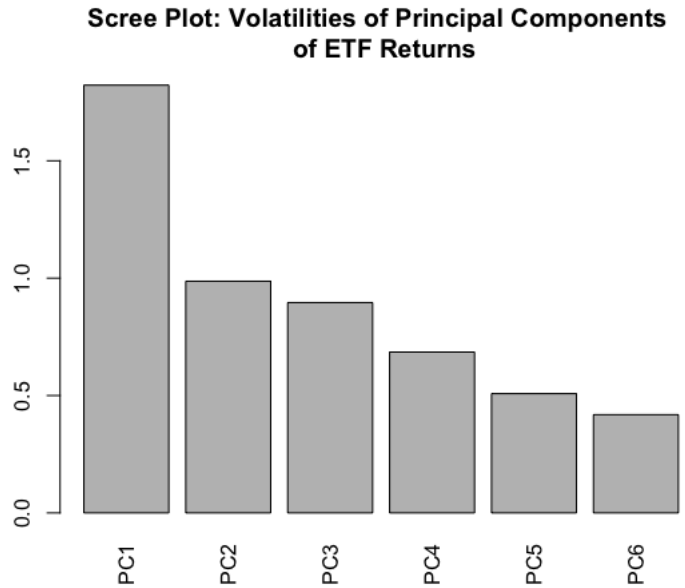
If the input time series are not scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the covariance matrix, and if they are scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the correlation matrix.

The function `prcomp()` performs *Principal Component Analysis* on a matrix of data (with the time series as columns), and returns the results as a list of class `prcomp`.

The `prcomp()` argument `scale=TRUE` specifies that the input time series should be scaled by their standard deviations.

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

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



A *scree plot* is a bar plot of the volatilities of the *principal components*.

```
> # Perform principal component analysis PCA
> pcad <- prcomp(retp, scale=TRUE)
> # Plot standard deviations of principal components
> barplot(pcad$sdev, names.arg=colnames(pcad$rotation),
+   las=3, xlab="", ylab="",
+   main="Scree Plot: Volatilities of Principal Components \n of ETF
> # Calculate the number of principal components which sum up to at
> pcavar <- pcad$sdev^2
> which(cumsum(pcavar)/sum(pcavar) > 0.8)[1]
```

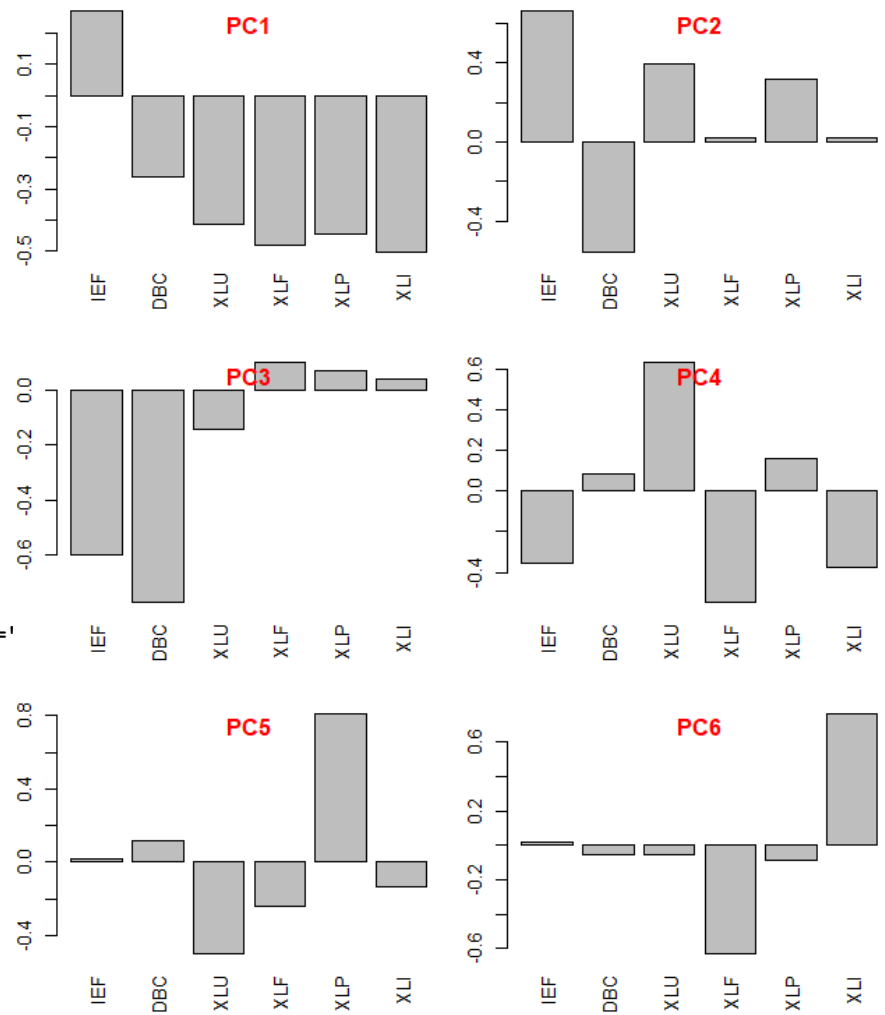
Principal Component Loadings (Weights)

Principal component loadings are the weights of portfolios which have mutually orthogonal returns.

The *principal component* (*PC*) portfolios represent the different orthogonal modes of the return variance.

The *PC* portfolios typically consist of long or short positions of highly correlated groups of assets (compclusts), so that they represent relative value portfolios.

```
> # Plot barplots with PCA loadings (weights) in multiple panels
> pcad$rotation
> # x11(width=6, height=7)
> par(mfrow=c(nweights/2, 2))
> par(mar=c(3, 2, 2, 1), oma=c(0, 0, 0, 0))
> for (ordern in 1:nweights) {
+   barplot(pcad$rotation[, ordern], las=3, xlab="", ylab="", main='
+   title(paste0("PC", ordern), line=-1, col.main="red")
+ }  # end for
```



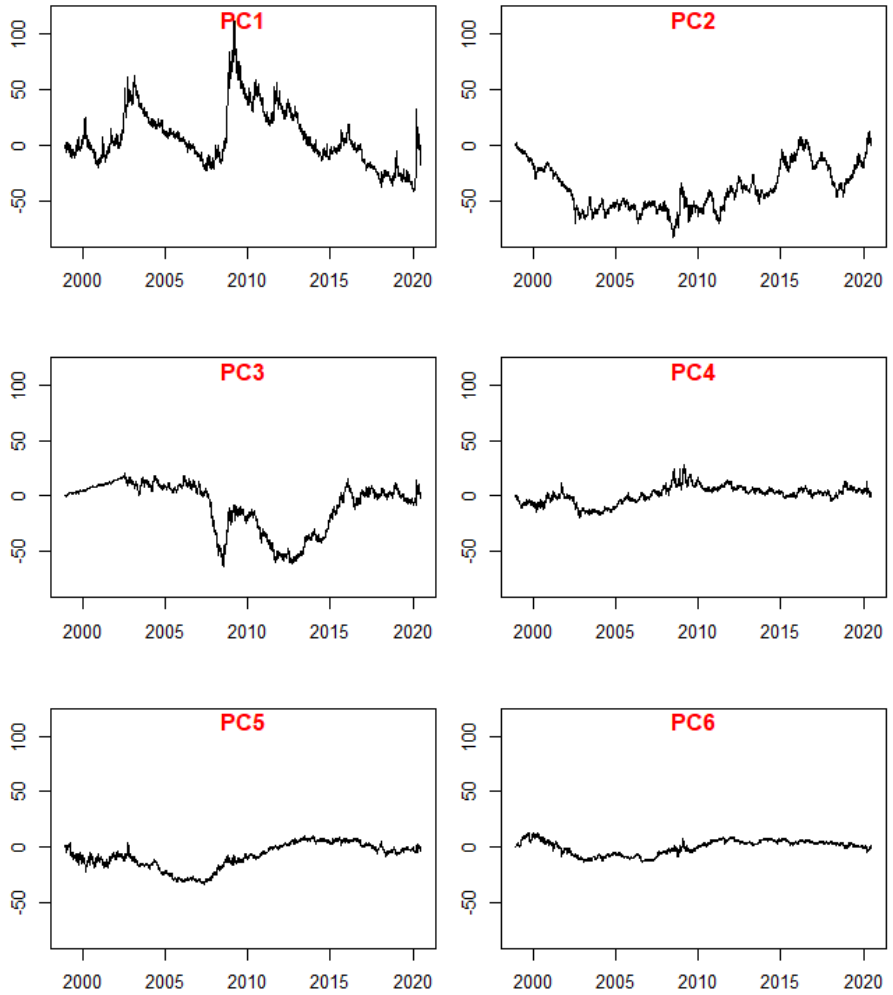
Principal Component Time Series

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

The *principal component* time series have mutually orthogonal returns.

Higher order *principal components* are gradually less volatile.

```
> # Calculate the products of principal component time series
> round(t(pcad$x) %*% pcad$x, 2)
> # Calculate the principal component time series from returns
> datev <- zoo::index(pricev)
> retpca <- xts::xts(retp %*% pcad$rotation, order.by=datev)
> round(cov(retpca), 3)
> all.equal(coredata(retpca), pcad$x, check.attributes=FALSE)
> retpcac <- cumsum(retpca)
> # Plot principal component time series in multiple panels
> rangev <- range(retpcac)
> for (ordern in 1:nweights) {
+   plot.zoo(retpcac[, ordern], ylim=rangev, xlab="", ylab="")
+   title(paste0("PC", ordern), line=-1, col.main="red")
+ }  # end for
```



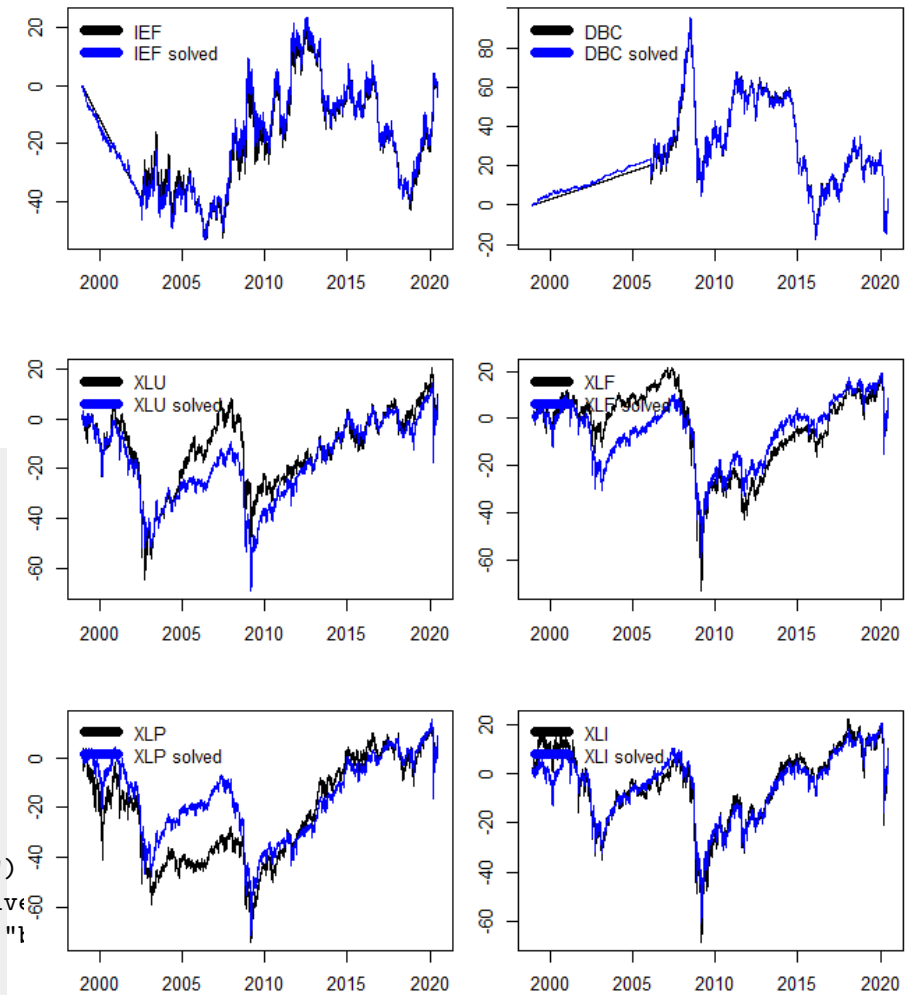
Dimension Reduction Using Principal Component Analysis

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

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

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

```
> # Invert all the principal component time series
> retpca <- retp %*% pcad$rotation
> solved <- retpca %*% solve(pcad$rotation)
> all.equal(coredata(retp), solved)
> # Invert first 3 principal component time series
> solved <- retpca[, 1:3] %*% solve(pcad$rotation)[1:3, ]
> solved <- xts::xts(solved, datev)
> solved <- cumsum(solved)
> retc <- cumsum(retp)
> # Plot the solved returns
> for (symbol in symbolv) {
+   plot.zoo(cbind(retc[, symbol], solved[, symbol]),
+     plot.type="single", col=c("black", "blue"), xlab="", ylab="")
+   legend(x="topleft", bty="n", legend=paste0(symbol, c("", " solve
+    title=NULL, inset=0.0, cex=1.0, lwd=6, lty=1, col=c("black", "b
+ }  # end for
```



Condition Number of Correlation Matrices

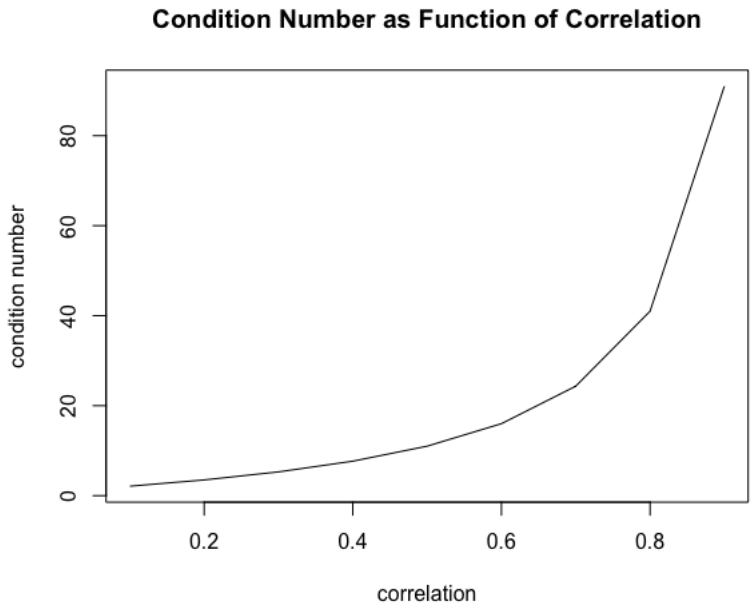
The condition number κ of a correlation matrix is equal to the ratio of its largest eigenvalue divided by the smallest:

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$$

The condition number depends on the level of correlations. If correlations are small then the eigenvalues are close to 1 and the condition number is also close to 1. If the correlations are close to 1 then the condition number is large.

A large condition number indicates the presence of small eigenvalues, and a correlation matrix close to *singular*, with a poorly defined inverse matrix.

A very large condition number indicates that the correlation matrix is close to being *singular*.




```
> # Create a matrix with low correlation
> ndata <- 10
> cormat <- matrix(rep(0.1, ndata^2), nc=ndata)
> diag(cormat) <- rep(1, ndata)
> # Calculate the condition number
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> max(eigenval)/min(eigenval)
> # Create a matrix with high correlation
> cormat <- matrix(rep(0.9, ndata^2), nc=ndata)
> diag(cormat) <- rep(1, ndata)
> # Calculate the condition number
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> max(eigenval)/min(eigenval)
```

```
> # Calculate the condition numbers as function correlation
> corv <- seq(0.1, 0.9, 0.1)
> condv <- sapply(corv, function(corv) {
+   cormat <- matrix(rep(corv, ndata^2), nc=ndata)
+   diag(cormat) <- rep(1, ndata)
+   eigend <- eigen(cormat)
+   eigenval <- eigend$values
+   max(eigenval)/min(eigenval)
+ })  # end sapply
> # Plot the condition numbers
> plot(x=corv, y=condv, t="l",
+   main="Condition Number as Function of Correlation",
+   xlab="correlation", ylab="condition number")
```

Condition Number for Small Number of Observations

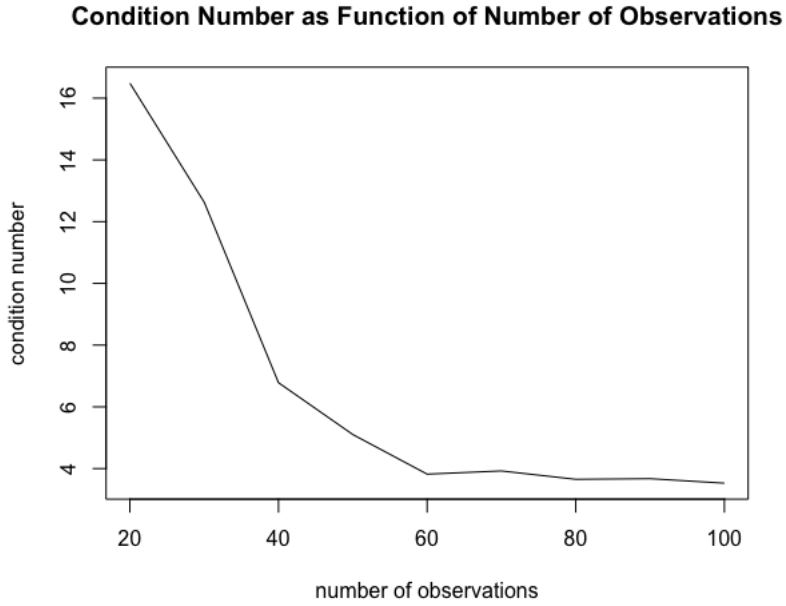
The condition number also depends on the number of observations.

If the number of observations (rows of data) is small compared to the number of stocks (columns), then the condition number can be large, even if the returns are not correlated.

That's because as the number of rows of data decreases, the returns become more *collinear*, and the sample correlation matrix becomes more *singular*, with some very small eigenvalues.

In practice, calculating the inverse correlation matrix of returns faces two challenges: not enough rows of data and correlated returns.

In both cases, the problem is that the columns of returns are close to *collinear*.




```
> # Simulate uncorrelated stock returns
> nstocks <- 10
> nrows <- 100
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> retp <- matrix(rnorm(nstocks*nrows), nc=nstocks)
> # Calculate the condition numbers as function of number of observations
> obsvec <- seq(20, nrows, 10)
> condv <- sapply(obsvec, function(nobs) {
+   cormat <- cor(retp[1:nobs, ])
+   eigend <- eigen(cormat)
+   eigenval <- eigend$values
+   max(eigenval)/min(eigenval)
+ })  # end sapply
```

```
> # Plot the condition numbers
> plot(x=obsvec, y=condv, t="l",
+   main="Condition Number as Function of Number of Observations",
+   xlab="number of observations", ylab="condition number")
```

The Correlations of Stock Returns

Estimating the correlations of stock returns is complicated because their date ranges may not overlap in time. Stocks may trade over different date ranges because of IPOs and corporate events (takeovers, mergers).

The function `cor()` calculates the correlation matrix of time series. The argument `use="pairwise.complete.obs"` removes `NA` values from pairs of stock returns.

But removing `NA` values in pairs of stock returns can produce correlation matrices which are not positive semi-definite.

The reason is because the correlations are calculated over different time intervals for different pairs of stock returns.

```
> # Load daily S&P500 log percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> # Calculate the number of NA values in retstock
> retp <- retstock
> colSums(is.na(retp))
> # Calculate the correlations ignoring NA values
> cor(retp$DAL, retp$FOXA, use="pairwise.complete.obs")
> cor(na.omit(retp[, c("DAL", "FOXA")]))[2]
> cormat <- cor(retp, use="pairwise.complete.obs")
> sum(is.na(cormat))
> cormat[is.na(cormat)] <- 0
```

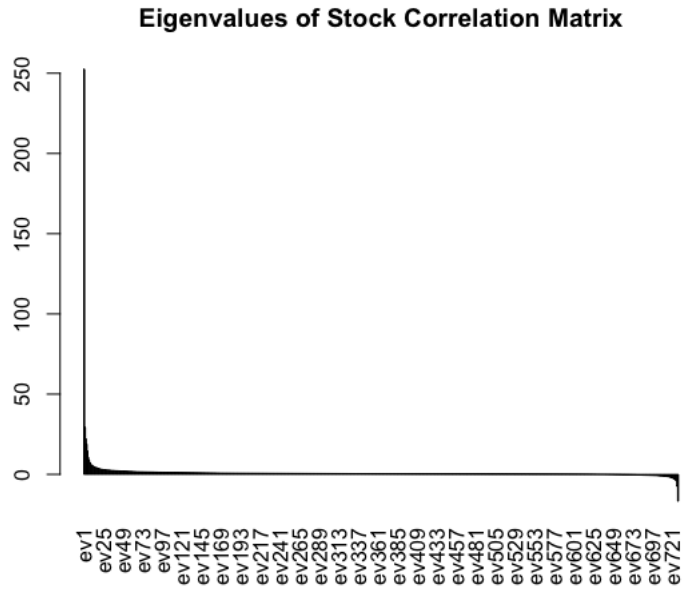
Principal Component Analysis of Stock Returns

Removing `NA` values in pairs of stock returns can produce correlation matrices which are not positive semi-definite.

The function `prcomp()` produces an error when the correlation matrix is not positive semi-definite, so instead, *eigen decomposition* can be applied to perform *Principal Component Analysis*.

If some of the eigenvalues are negative, then the condition number is calculated using the eigenvalue with the smallest absolute value.

```
> # Perform principal component analysis PCA - produces error
> pcad <- prcomp(retp, scale=TRUE)
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> # Calculate the eigenvalues and eigenvectors
> eigenval <- eigend$values
> eigenvec <- eigend$vectors
> # Calculate the number of negative eigenvalues
> sum(eigenval<0)
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the number of eigenvalues which sum up to at least 80% of the total variance
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
```




```
> # Plot the eigenvalues
> barplot(eigenval, xlab="", ylab="", las=3,
+   names.arg=paste0("ev", 1:NROW(eigenval)),
+   main="Eigenvalues of Stock Correlation Matrix")
```

Principal Component Analysis of Low and High Volatility Stocks

Low and high volatility stocks have different correlations and principal components.

Low volatility stocks have higher correlations than high volatility stocks, so their correlation matrix has a larger condition number than high volatility stocks.

But low volatility stocks can be explained by a smaller number of principal components, compared to high volatility stocks.

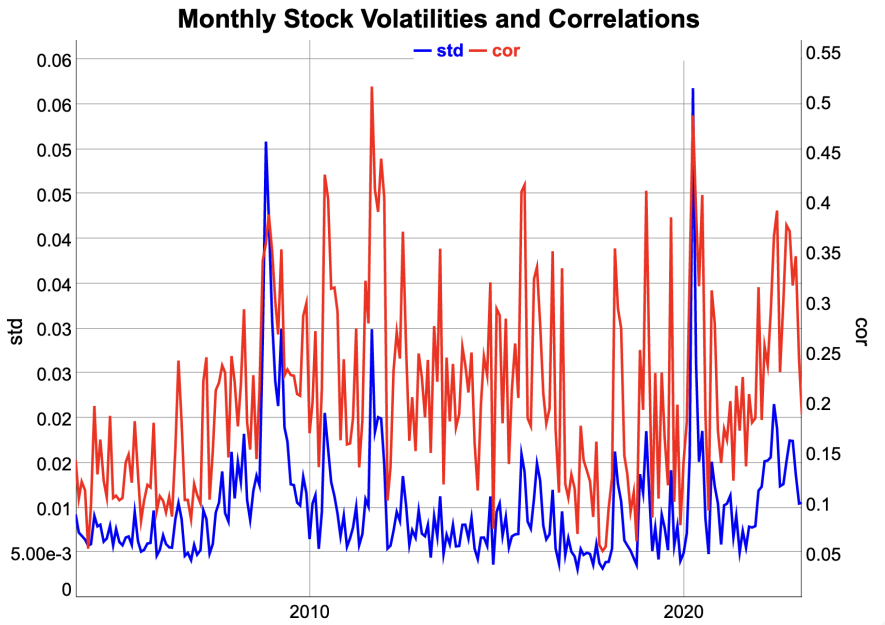
```
> # Calculate the stock variance
> varv <- sapply(retp, var, na.rm=TRUE)
> # Calculate the returns of low and high volatility stocks
> nstocks <- NCOL(retp)
> medianv <- median(varv)
> retlow <- retp[, varv <= medianv]
> rethigh <- retp[, varv > medianv]
> # Calculate the correlations of low volatility stocks
> cormat <- cor(retlow, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> # Calculate the mean correlations
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> # Calculate the number of negative eigenvalues
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the correlations of high volatility stocks
> cormat <- cor(rethigh, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> # Calculate the mean correlations
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> # Calculate the number of negative eigenvalues
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
```

Stock Correlations in Periods of Low and High Volatility

Correlations of stock returns are higher in time intervals with high volatility.

Stock returns have *high correlations* in time intervals with *high volatility*, and vice versa.

```
> # Subset (select) the stock returns after the start date of VTI
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> retp <- retstock[zoo::index(retvti)]
> datev <- zoo::index(retp)
> retvti <- retvti[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
> # Calculate the monthly end points
> endd <- rutils::calc_endpoints(retvti, interval="months")
> retvti[head(endd)]
> retvti[tail(endd)]
> # Remove stub interval at the end
> endd <- endd[-NROW(endd)]
> npts <- NROW(endd)
> # Calculate the monthly stock volatilities and correlations
> stdcor <- sapply(2:npts, function(endp) {
+   # cat("endp = ", endp, "\n")
+   retp <- retp[endd[endp-1]:endd[endp]]
+   cormat <- cor(retp, use="pairwise.complete.obs")
+   cormat[is.na(cormat)] <- 0
+   c(stdev=sd(retvti[endd[endp-1]:endd[endp]]),
+     cor=mean(cormat[upper.tri(cormat)]))
+ })  # end sapply
> stdcor <- t(stdcor)
```




```
> # Scatterplot of stock volatilities and correlations
> plot(x=stdcor[, "stdev"], y=stdcor[, "cor"],
+  xlab="volatility", ylab="correlation",
+  main="Monthly Stock Volatilities and Correlations")
> # Plot stock volatilities and correlations
> colnamev <- colnames(stdcor)
> stdcor <- xts(stdcor, zoo::index(retvti[endd]))
> dygraphs::dygraph(stdcor,
+   main="Monthly Stock Volatilities and Correlations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Principal Component Analysis in Periods of Low and High Volatility

Stock returns in time intervals with *high volatility* have *high correlations* and therefore require fewer eigenvalues to explain 80% of their total variance.

Stock returns in time intervals with *low volatility* have *low correlations* and therefore require more eigenvalues to explain 80% of their total variance.

```
> # Calculate the median VTI volatility
> medianv <- median(stdcor[, "stdev"])
> # Calculate the stock returns of low volatility intervals
> retlow <- lapply(2:npts, function(endp) {
+   if (stdcor[endp-1, "stdev"] <= medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> retlow <- rutils::do_call(rbind, retlow)
> # Calculate the stock returns of high volatility intervals
> rethigh <- lapply(2:npts, function(endp) {
+   if (stdcor[endp-1, "stdev"] > medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> rethigh <- rutils::do_call(rbind, rethigh)
```

```
> # Calculate the correlations of low volatility intervals
> cormat <- cor(retlow, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the correlations of high volatility intervals
> cormat <- cor(rethigh, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
```

Trailing Correlations of Stock Returns

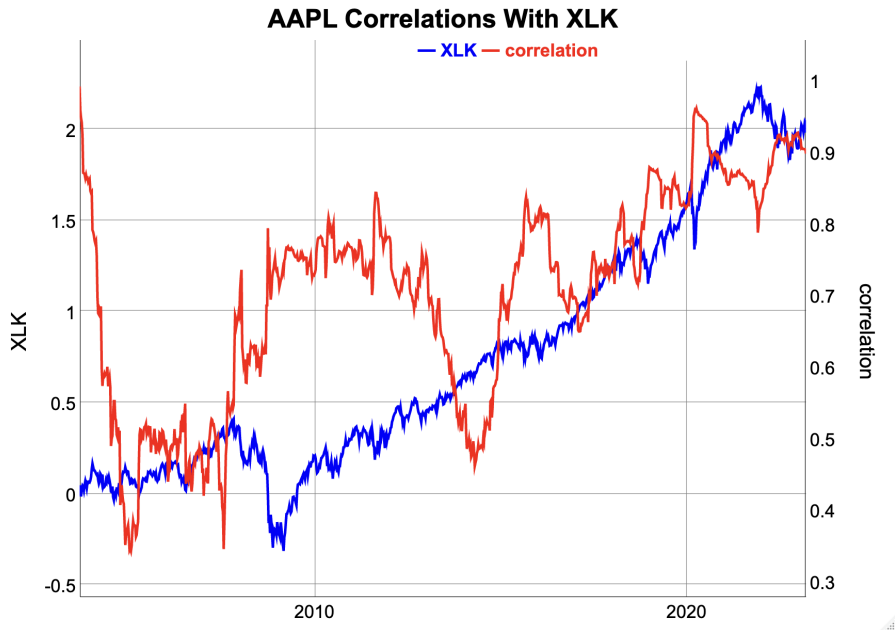
The trailing covariance can be updated using *online* recursive formulas with the weight decay factor λ:

$$\bar{x}_t = \lambda \bar{x}_{t-1} + (1-\lambda)x_t$$
$$\bar{y}_t = \lambda \bar{y}_{t-1} + (1-\lambda)y_t$$
$$\sigma^2_{xt} = \lambda \sigma^2_{x(t-1)} + (1-\lambda)(x_t - \bar{x}_t)^2$$
$$\sigma^2_{yt} = \lambda \sigma^2_{y(t-1)} + (1-\lambda)(y_t - \bar{y}_t)^2$$
$$\mathrm{cov}_t = \lambda\, \mathrm{cov}_{t-1} + (1-\lambda)(x_t - \bar{x}_t)(y_t - \bar{y}_t)$$

The parameter λ determines the rate of decay of the weight of past returns. If λ is close to 1 then the decay is weak and past returns have a greater weight, and the trailing mean values have a stronger dependence on past returns. This is equivalent to a long look-back interval. And vice versa if λ is close to 0.

The function `HighFreq::run_covar()` calculates the trailing variances, covariances, and means of two *time series*.

```
> # Calculate the AAPL and XLK returns
> retp <- na.omit(cbind(returns$AAPL, rutils::etfenv$returns$XLK))
> # Calculate the trailing correlations
> lambdaf <- 0.99
> covarv <- HighFreq::run_covar(retp, lambdaf)
> correlv <- covarv[, 1, drop=FALSE]/sqrt(covarv[, 2]*covarv[, 3])
```



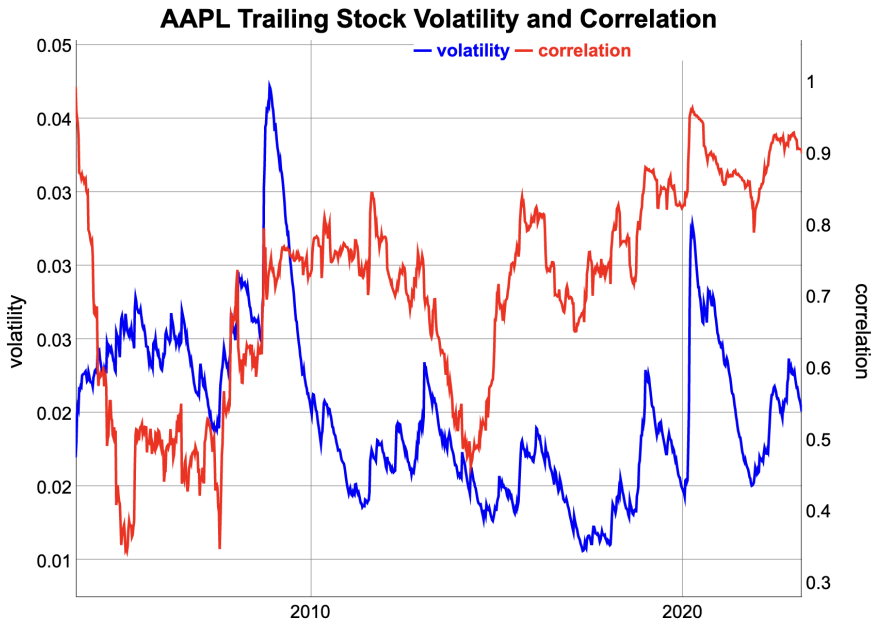

```
> # Plot dygraph of XLK returns and AAPL correlations
> datav <- cbind(cumsum(retp$XLK), correlv)
> colnames(datav)[2] <- "correlation"
> colnamev <- colnames(datav)
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(datav[endd], main="AAPL Correlations With XLK")
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Trailing Stock Correlations and Volatilities

The correlations of stock returns are typically higher in periods of higher volatility, and vice versa.

But stock correlations have increased after the 2008-09 financial crisis, while volatilities have decreased.

The correlation of *AAPL* and *XLK* has increased over time because *AAPL* has become a much larger component of *XLK*, as its stock has rallied.



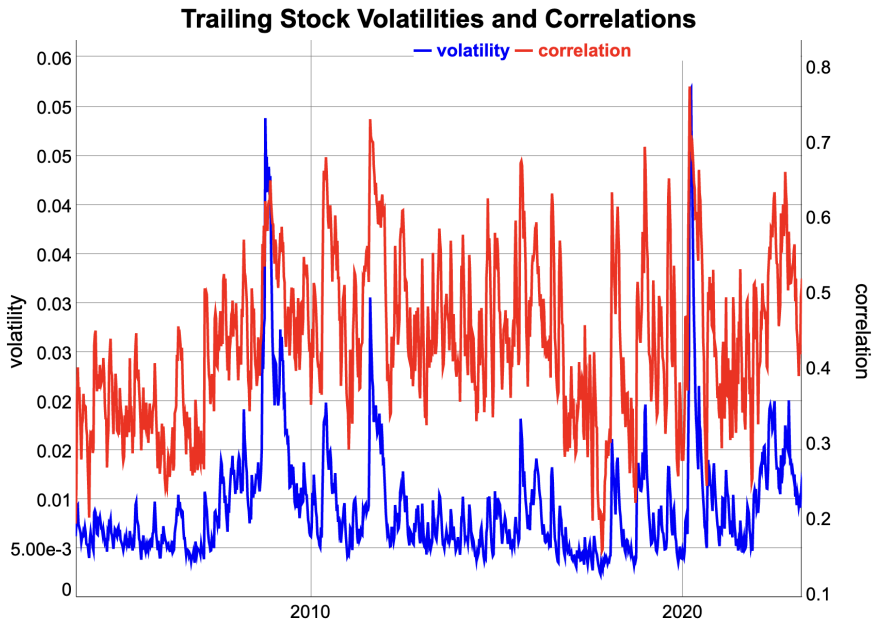

```
> # Scatterplot of trailing stock volatilities and correlations
> volv <- sqrt(covarv[, 2])
> plot(x=volv[endd], y=correlv[endd, ], pch=1, col="blue",
+  xlab="AAPL volatility", ylab="Correlation",
+  main="Trailing Volatilities and Correlations of AAPL vs XLK")
> # Interactive scatterplot of trailing stock volatilities and corre
> datev <- zoo::index(retp[endd])
> datav <- data.frame(datev, volv[endd], correlv[endd, ])
> colnames(datav) <- c("date", "volatility", "correlation")
> library(plotly)
> plotly::plot_ly(data=datav, x=~volatility, y=~correlation,
+   type="scatter", mode="markers", text=datev) %>%
+   layout(title="Trailing Volatilities and Correlations of AAPL vs
```

```
> # Plot trailing stock volatilities and correlations
> datav <- xts(cbind(volv, correlv), zoo::index(retp))
> colnames(datav) <- c("volatility", "correlation")
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav[endd], main="AAPL Trailing Stock Volatilit
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Stock Portfolio Correlations and Volatilities

The average correlations of a stock portfolio are typically higher in periods of higher volatility, and vice versa.

But stock correlations have increased after the 2008-09 financial crisis, while volatilities have decreased.




```
> # Calculate the portfolio returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> retp <- retstock100
> retp[is.na(retp)] <- 0
> retp <- retp[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
> # Calculate the average trailing portfolio correlations
> lambdaf <- 0.9
> correlv <- sapply(retp, function(retp) {
+   covarv <- HighFreq::run_covar(cbind(retvti, retp), lambdaf)
+   covarv[, 1, drop=FALSE]/sqrt(covarv[, 2]*covarv[, 3])
+ })  # end sapply
> correlv[is.na(correlv)] <- 0
> correlp <- rowMeans(correlv)
> # Scatterplot of trailing stock volatilities and correlations
> volvti <- sqrt(HighFreq::run_var(retvti, lambdaf))
> endd <- rutils::calc_endpoints(retvti, interval="weeks")
> plot(x=volvti[endd], y=correlp[endd],
+  xlab="volatility", ylab="correlation",
+  main="Trailing Stock Volatilities and Correlations")
```

```
> # Plot trailing stock volatilities and correlations
> datav <- xts(cbind(volvti, correlp), datev)
> colnames(datav) <- c("volatility", "correlation")
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav[endd],
+   main="Trailing Stock Volatilities and Correlations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Homework Assignment

Required

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

Recommended

- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
- Read about *PCA* in:
  pca-handout.pdf
  pcaTutorial.pdf