# FRE7241 Algorithmic Portfolio Management

## Lecture#4, Fall 2023

Jerzy Pawlowski *jp3900@nyu.edu*

*NYU Tandon School of Engineering*

September 26, 2023

# Centered Price Z-scores

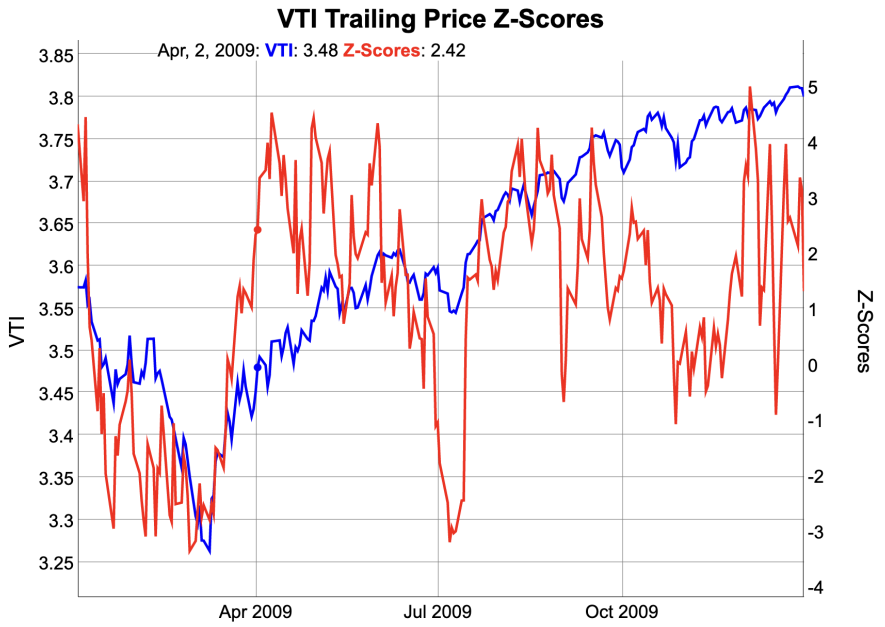
An extreme local price is a price which differs significantly from neighboring prices.

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

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

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

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




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

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

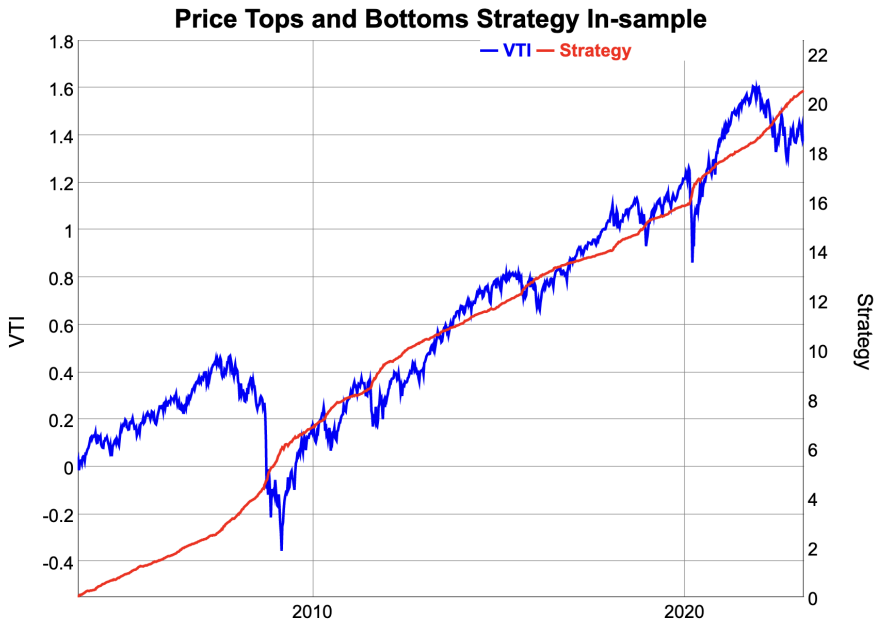
# Labeling the Tops and Bottoms of Prices

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

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

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

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




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

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

# Predictors of Price Extremes

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

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

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

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

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

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

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

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

# Regression Z-Scores

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

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

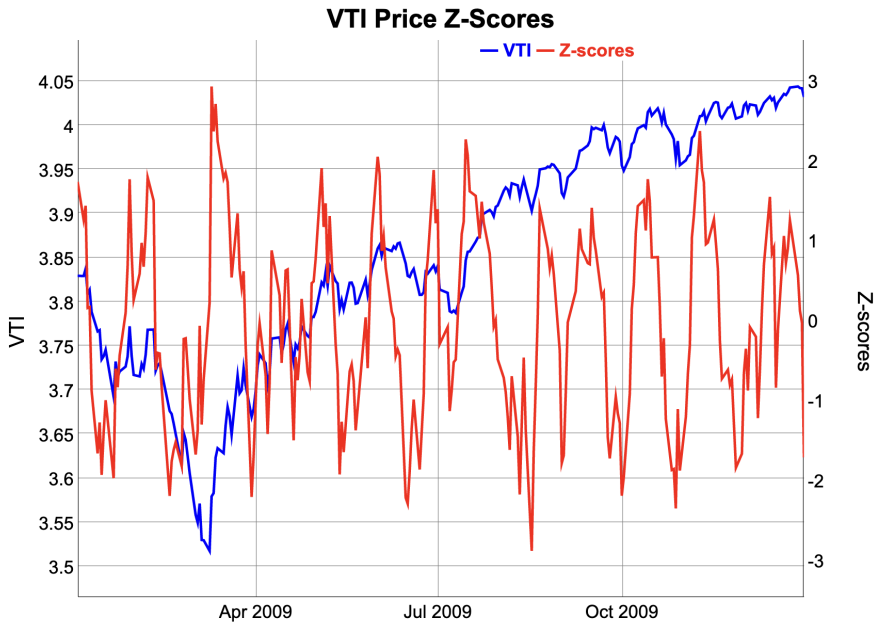
Where $\alpha$ and $\beta$ are the *regression coefficients*, and $\sigma_i$ is the standard deviation of the residuals.

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

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

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

```
> # Calculate the trailing price regression z-scores
> datev <- matrix(zoo::index(closep))
> look_back <- 21
> controlv <- HighFreq::param_reg()
> regs <- HighFreq::roll_reg(respv=closep, predm=datev, look_back=
> regs <- drop(regs[, NCOL(regs)])
> regs[1:look_back] <- 0
```




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

# The *Logistic* Function

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

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

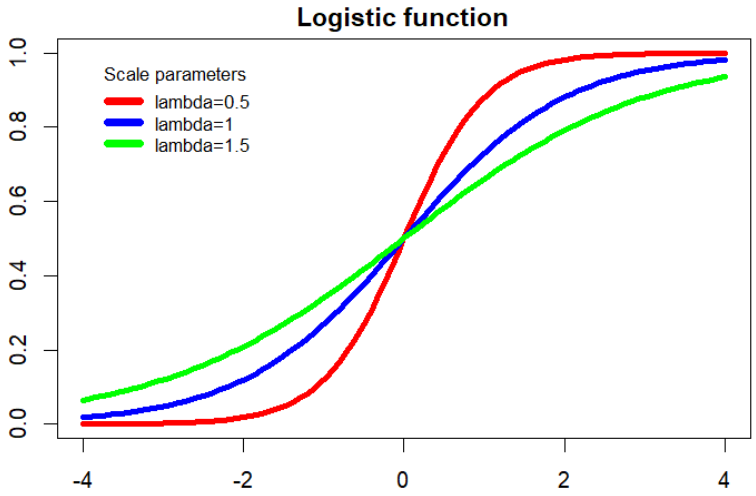
Where $\lambda$ is the scale (dispersion) parameter.

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

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

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

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




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

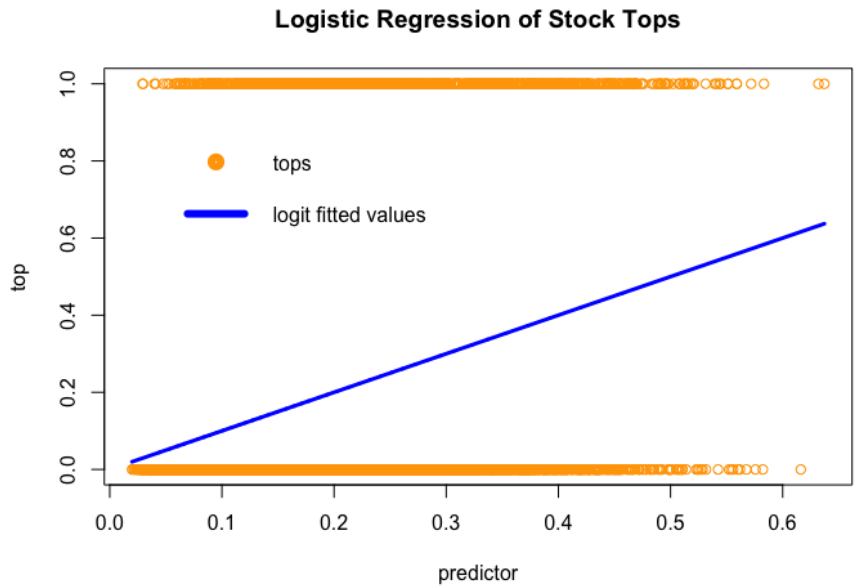
# Forecasting Stock Price Tops and Bottoms Using Logistic Regression

Consider a model which uses the weighted average of the volatility, trading volume, and regression z-scores, to forecast a stock top (overbought condition) or a bottom (oversold condition).

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

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

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




```
> plot(x=fcasts[ordern], y=tops[ordern],
+      main="Logistic Regression of Stock Tops",
+      col="orange", xlab="predictor", ylab="top")
> lines(x=fcasts[ordern], y=logmod$fitted.values[ordern], col="blue"
> legend(x=0.1, y=1.2, inset=0.0, bty="n", lwd=6,
+  legend=c("tops", "logit fitted values"), y.intersp=0.1,
+  col=c("orange", "blue"), lty=c(NA, 1), pch=c(1, NA))
```

# Forecasting Errors of Stock Tops and Bottoms

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

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

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

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

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

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

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

# The Confusion Matrix of a Binary Classification Model

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

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

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

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

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

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

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

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

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

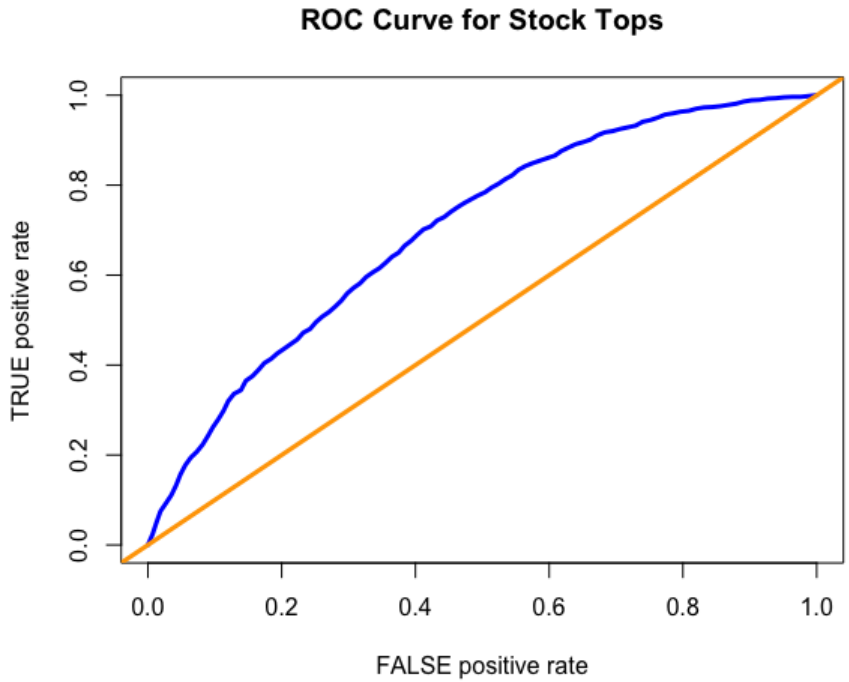
The sum of the *true negative* plus the *false positive* rate is equal to 1.

# Receiver Operating Characteristic (ROC) Curve for Stock Tops

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

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

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




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

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

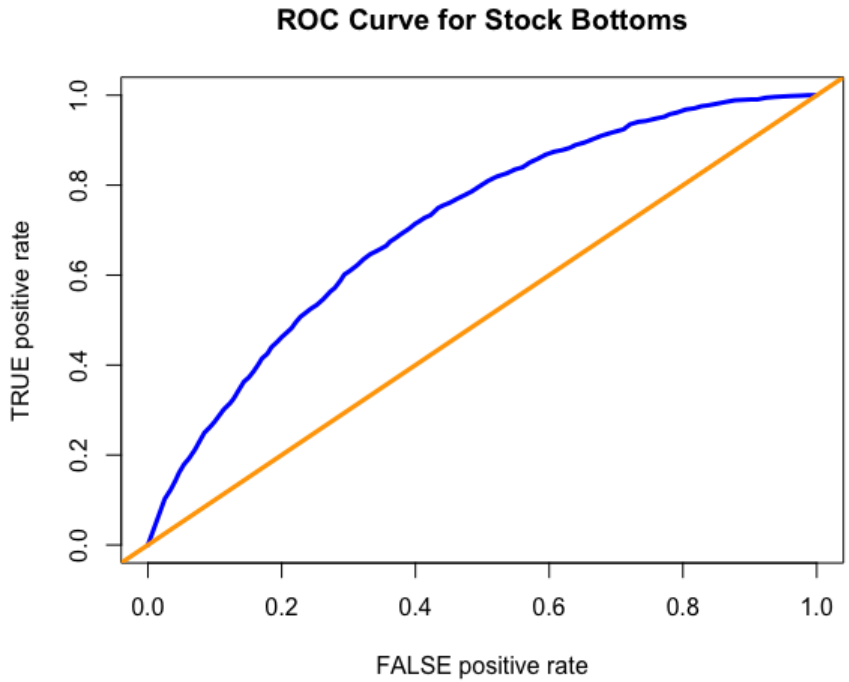
# Receiver Operating Characteristic (ROC) Curve for Stock Bottoms

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

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

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

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




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

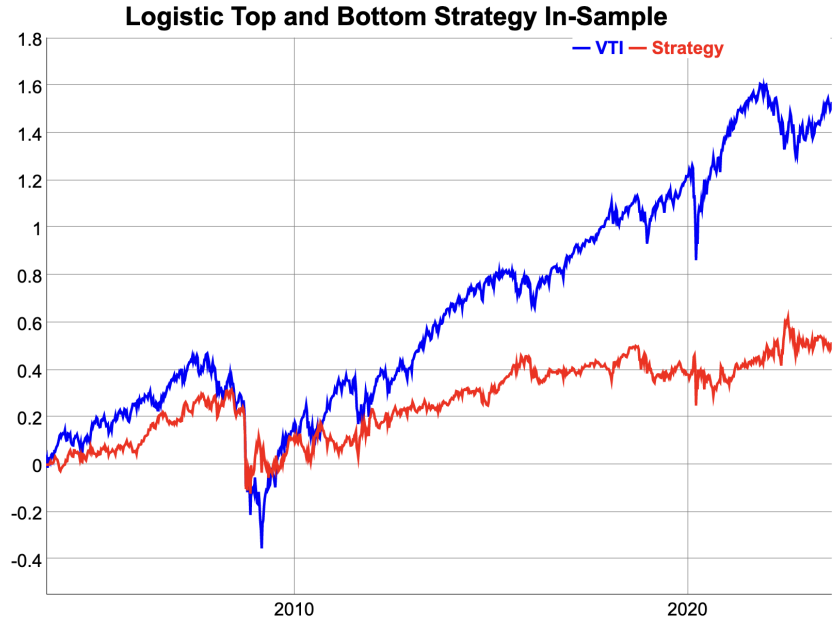
# Logistic Tops and Bottoms Strategy In-sample

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

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

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

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

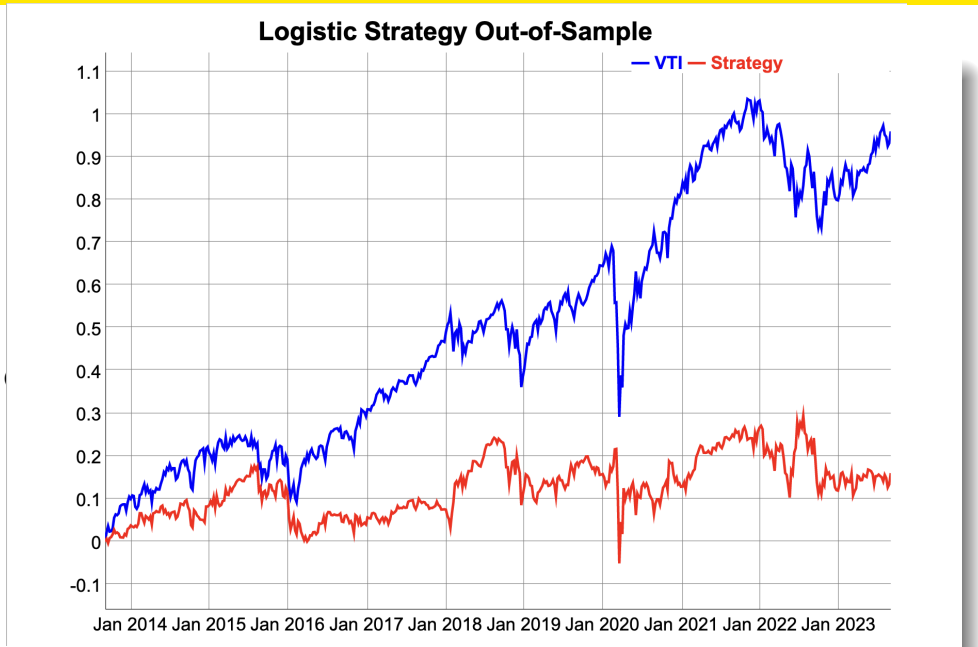



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Top and Bottom Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

# Logistic Tops and Bottoms Strategy Out-of-Sample

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

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




```
> # Simulate in-sample VTI strategy
> topsav <- HighFreq::roll_sum(matrix(topf), 5)/5
> botsav <- HighFreq::roll_sum(matrix(botf), 5)/5
> posv <- (botsav - topsav)
> posv <- rutils::lagit(posv)
> pnls <- retp[outsample, ]*posv
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of in-sample VTI strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Logistic Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

# Limitations of Backtest Simulations

A *backtest* is a simulation of a trading strategy on historical data.

Backtest simulations are useful for determining the relative advantages of different strategy features and parameters.

But the simulated strategy pnls may not be realistic for several reasons:

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

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

But the backtest simulations are not predicting future profits.

# Daily Mean Reverting Strategy

The lag $k$ *autocorrelation* of a time series of returns $r_t$ is equal to:

$$\rho_k = \frac{\sum_{t=k+1}^{n}(r_t - \bar{r})(r_{t-k} - \bar{r})}{(n-k)\,\sigma^2}$$

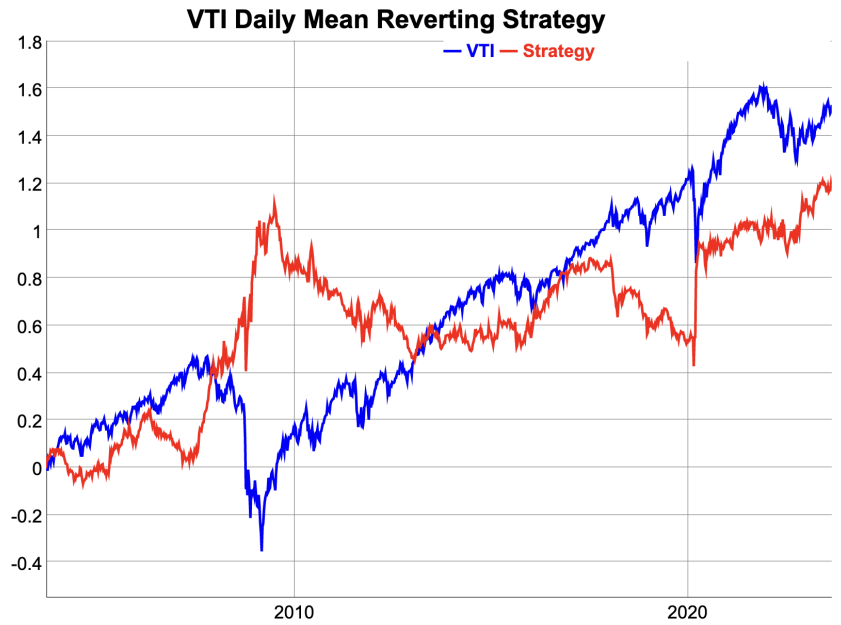
The function `rutils::plot_acf()` calculates and plots the autocorrelations of a time series.

Daily stock returns often exhibit some negative autocorrelations.

The daily mean reverting strategy buys or sells short \$1 of stock at the end of each day (depending on the sign of the previous daily return), and holds the position until the next day.

If the previous daily return was positive, it sells short \$1 of stock. If the previous daily return was negative, it buys \$1 of stock.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the autocorrelations of VTI daily returns
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy
> posv <- rutils::lagit(sign(retp), lagg=1)
> pnls <- (-retp*posv)
```



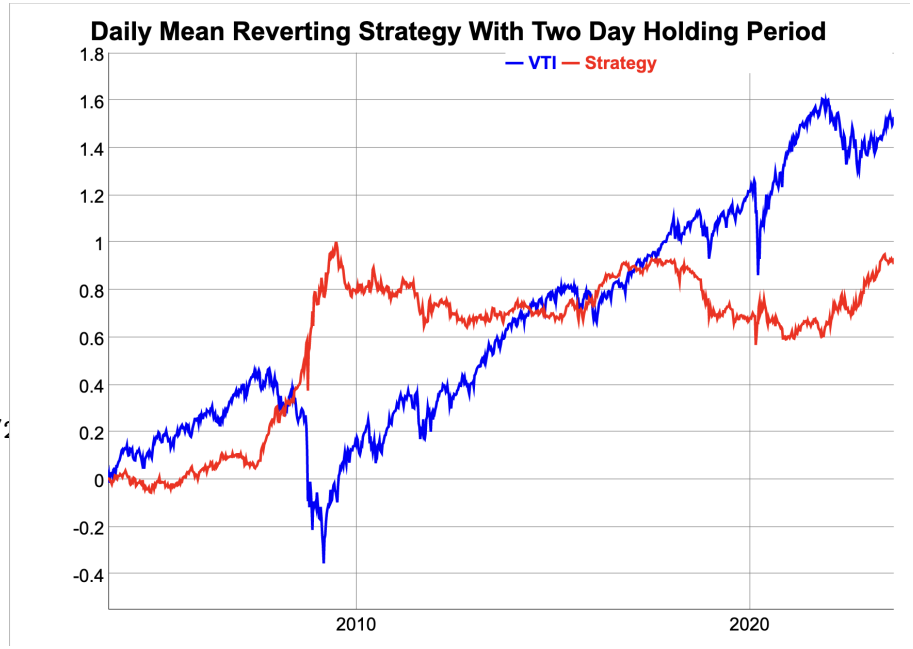

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

# Daily Mean Reverting Strategy With a Holding Period

The daily mean reverting strategy can be improved by combining the daily returns from the previous two days. This is equivalent to holding the position for two days, instead of rolling it daily.

The daily mean reverting strategy with a holding period performs better than the simple daily strategy because of risk diversification.

```
> # Simulate mean reverting strategy with two day holding period
> posv <- rutils::lagit(rutils::roll_sum(sign(retp), look_back=2))/2
> pnls <- (-retp*posv)
```



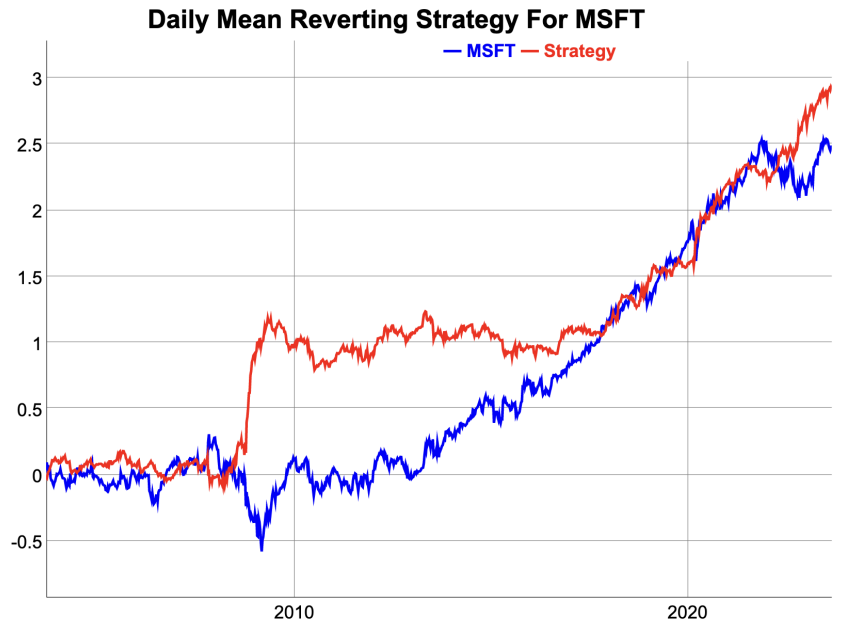

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy With Two Day Holding Period"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# Daily Mean Reverting Strategy For Stocks

Some daily stock returns exhibit stronger negative autocorrelations than ETFs.

But the daily mean reverting strategy doesn't perform well for many stocks.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns
> retp <- na.omit(retstock$MSFT)
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy with two day holding period
> posv <- rutils::lagit(rutils::roll_sum(sign(retp), look_back=2))/2
> pnls <- (-retp*posv)
```




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

# Daily Mean Reverting Strategy For All Stocks

The combined daily mean reverting strategy for all *S&P500* stocks performed well prior to and during the 2008 financial crisis, but was flat afterwards.

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

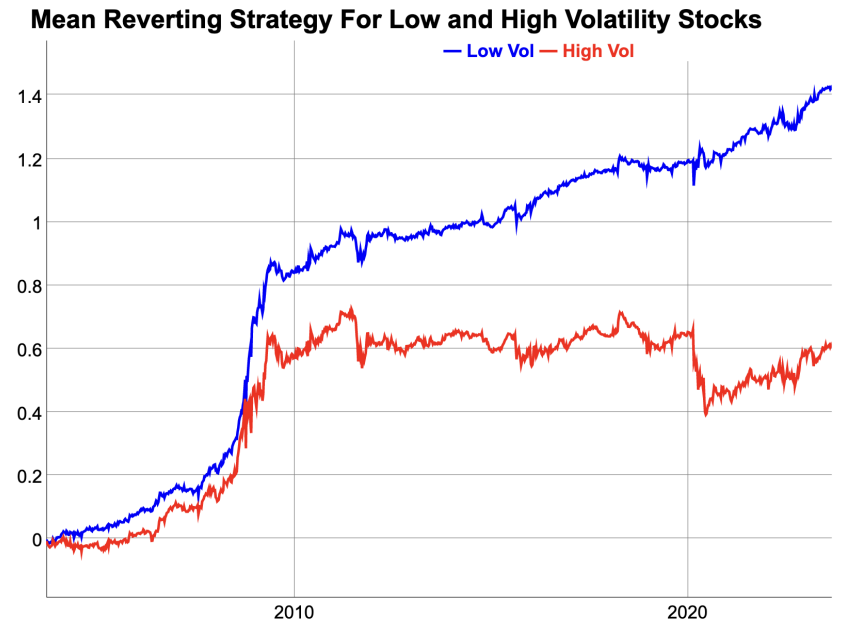
```
> # Simulate mean reverting strategy for all S&P500 stocks
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(retstock, function(retp) {
+   retp <- na.omit(retp)
+   posv <- rutils::roll_sum(sign(retp), look_back=2)/2
+   posv <- rutils::lagit(posv)
+   pnls <- (-retp*posv)
+   pnls
+ }, mc.cores=ncores)  # end mclapply
> pnls <- do.call(cbind, pnll)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> # Calculate the average returns of all S&P500 stocks
> datev <- zoo::index(retstock)
> indeks <- rowMeans(retstock, na.rm=TRUE)
```


```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("All Stocks", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy For All Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# Mean Reverting Strategy For Low and High Volatility Stocks

The daily mean reverting strategy performs better for low volatility stocks than for high volatility stocks.

```
> # Calculate the stock volatilities
> stdev <- mclapply(retstock, function(retp) {
+   sd(na.omit(retp))
+ }, mc.cores=ncores)  # end mclapply
> stdev <- do.call(c, stdev)
> # Calculate the median volatility
> medianv <- median(stdev)
> # Calculate the pnls for low volatility stocks
> pnlovol <- do.call(cbind, pnll[stdev < medianv])
> pnlovol <- rowMeans(pnlovol, na.rm=TRUE)
> # Calculate the pnls for high volatility stocks
> pnlhivol <- do.call(cbind, pnll[stdev >= medianv])
> pnlhivol <- rowMeans(pnlhivol, na.rm=TRUE)
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(pnlovol, pnlhivol)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Low Vol", "High Vol")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Mean Reverting Strategy For Low and High Volatility Stocks
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# The EWMA Mean-Reversion Strategy

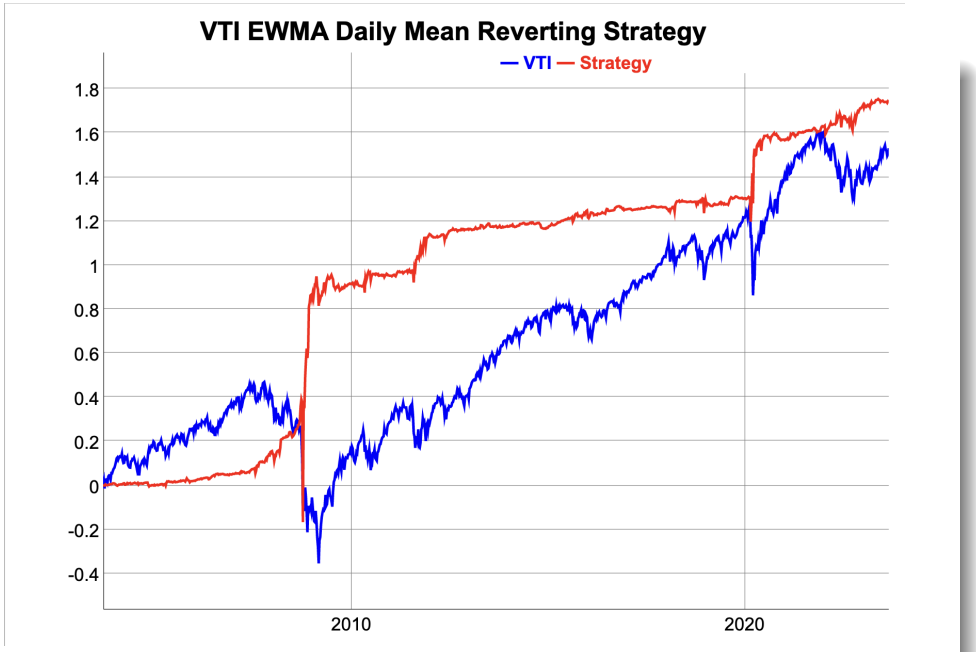
The *EWMA* mean-reversion strategy holds both long stock positions and short positions proportional to the trailing *EWMA* of past returns.

The strategy adjusts its stock position at the end of each day, just before the close of the market.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The problem is that the strategy makes profits mostly in periods of high volatility, but otherwise it's not very profitable.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the EWMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retp, lambda=0.1)
> # Calculate the positions and pnls
> posv <- retma
> # posv <- sign(retma)
> posv <- rutils::lagit(posv, lagg=1)
> pnls <- (-retp*posv)
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




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

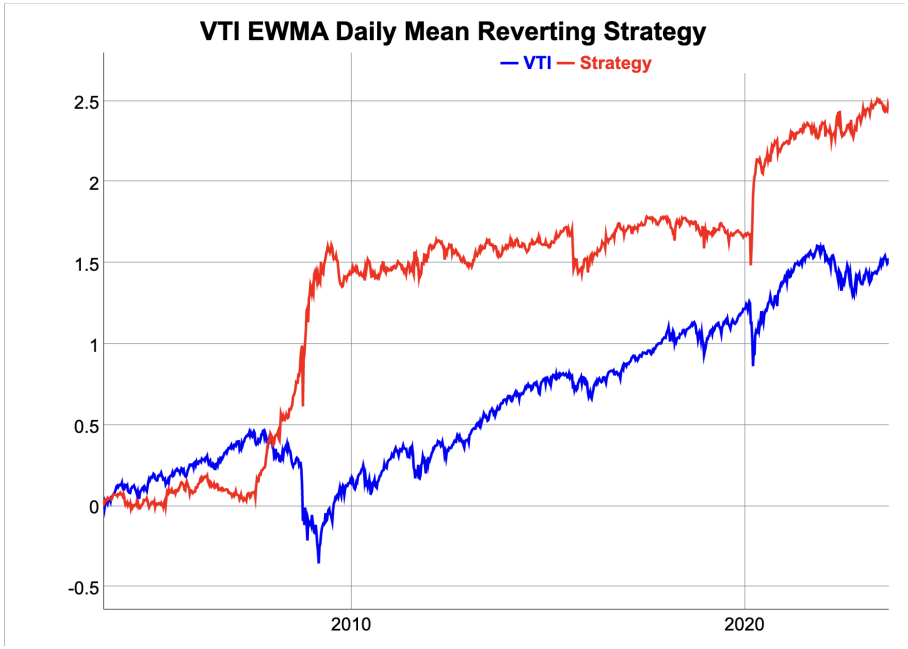
# The EWMA Mean-Reversion Strategy Scaled By Volatility

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

Scaling the returns by their trailing volatilities reduces the profits in periods of high volatility, but doesn't improve profits in periods of low volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the trailing volatility
> volat <- HighFreq::run_var(retp, lambda=0.5)
> volat <- sqrt(volat)
> # Scale the returns by their trailing volatility
> retsc <- ifelse(volat > 0, retp/volat, 0)
> # Calculate the EWMA returns
> retma <- HighFreq::run_mean(retsc, lambda=0.1)
> # Calculate the positions and pnls
> posv <- retma
> # posv <- sign(retma)
> posv <- rutils::lagit(posv, lagg=1)
> pnls <- (-retp*posv)
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




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

# Autoregressive Model of Stock Returns

The stock returns $r_t$ can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term $\varphi_0$:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

The *residuals* $\varepsilon_t$ are assumed to be normally distributed, independent, and stationary.

The autoregressive model can be written in matrix form as:

$$\mathbf{r} = \varphi \mathbb{P} + \varepsilon$$

Where $\varphi = \{\varphi_0, \varphi_1, \varphi_2, \ldots \varphi_n\}$ is the vector of autoregressive coefficients.

The *autoregressive* model is equivalent to *multivariate* linear regression, with the *response* equal to the returns $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ equal to the lags of the returns.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> # Add constant column for intercept coefficient phi0
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
```

# Forecasting Stock Returns Using Autoregressive Models

The fitted autoregressive coefficients $\varphi$ are equal to the *response* $\mathbf{r}$ multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

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

The *in-sample* autoregressive forecasts of the returns are calculated by multiplying the predictor matrix by the fitted AR coefficients:

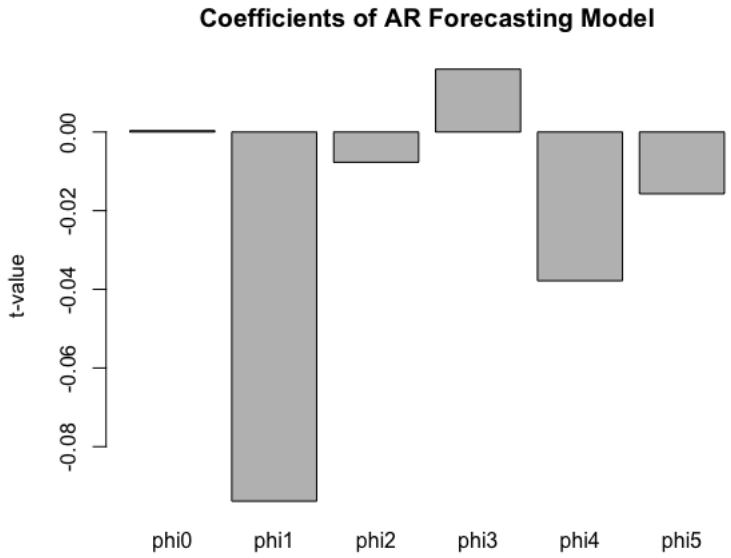
$$f_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n}$$

For *VTI* returns, the intercept coefficient $\varphi_0$ has a small positive value, while the first autoregressive coefficient $\varphi_1$ has a small negative value.

This means that the autoregressive forecasting model is a combination of a static long stock position, plus a mean-reverting model which switches its stock position to the reverse of the previous day's return.

The function `MASS::ginv()` calculates the generalized inverse of a matrix.

```
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
```




```
> # Plot the AR coefficients
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> barplot(coeff ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficients of AR Forecasting Model")
```

# The t-values of the Autoregressive Coefficients

The forecast residuals are equal to the differences between the return forecasts minus the actual returns: $\varepsilon = f_t - r_t$

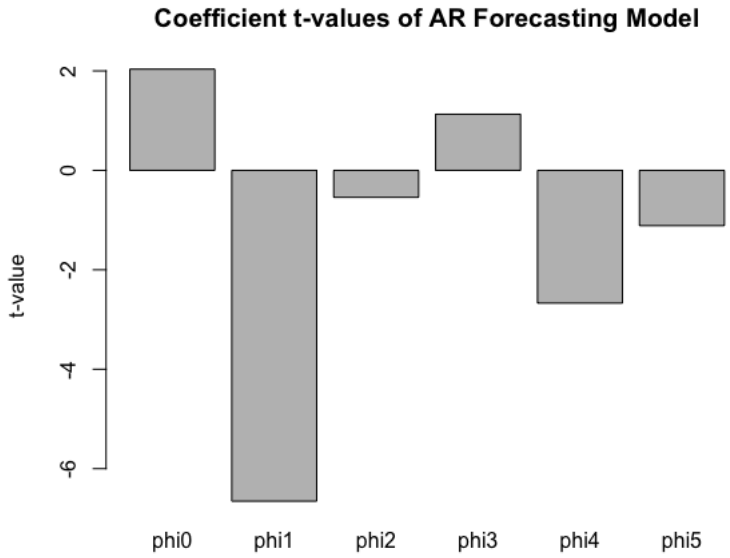
The variance of the autoregressive coefficients $\sigma^2_{\varphi}$ is equal to the variance of the forecast residuals $\sigma^2_{\varepsilon}$ divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma^2_{\varphi} = \sigma^2_{\varepsilon}(\mathbb{P}^T\mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_{\varphi}}$$

The intercept coefficient $\varphi_0$ and the first autoregressive coefficient $\varphi_1$ have statistically significant t-values.




```
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecast
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
> # Calculate the variance of the residuals
> vars <- sum(resids^2)/(nrows-NROW(coeff))
```

```
> # Calculate the predictor matrix squared
> predm2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covar <- vars*MASS::ginv(predm2)
> coeffsd <- sqrt(diag(covar))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coeffsd)
> coeffn <- paste0("phi", 0:(NROW(coefft)-1))
> # Plot the t-values of the AR coefficients
> barplot(coefft ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficient t-values of AR Forecasting Model")
```

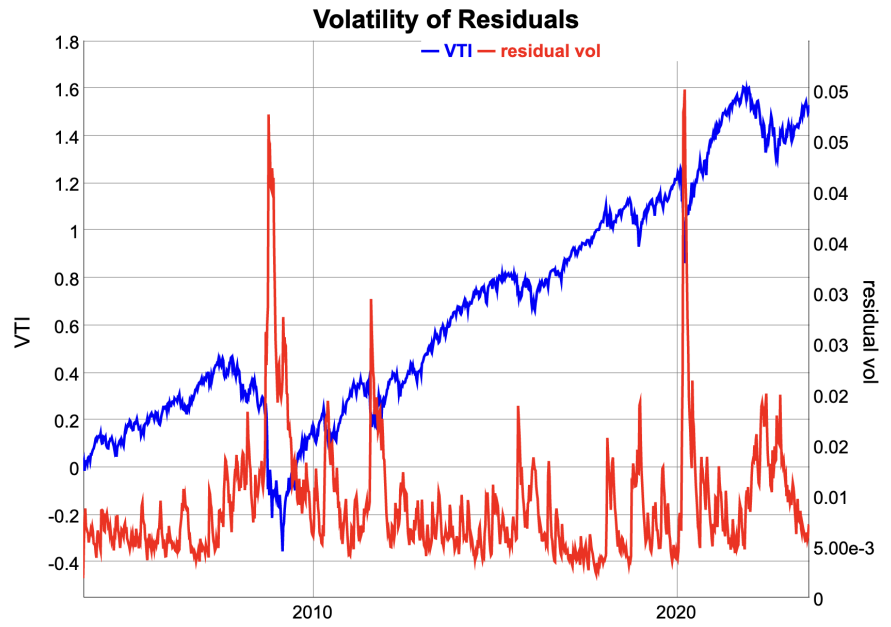
# Residuals of Autoregressive Forecasting Model

The autoregressive model assumes stationary returns and residuals, with similar volatility over time.

In reality stock volatility is highly time dependent, so the volatility of the residuals is also time dependent.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the trailing volatility of the residuals
> residv <- sqrt(HighFreq::run_var(resids, lambda=0.9))
```




```
> # Plot dygraph of volatility of residuals
> datav <- cbind(cumsum(retp), residv)
> colnames(datav) <- c("VTI", "residual vol")
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endd], main="Volatility of Residuals") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="residual vol", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name="residual vol", axis="y2", strokeWidth=2, col="red
+   dyLegend(show="always", width=300)
```

# Autoregressive Strategy In-Sample

The first step in strategy development is optimizing it in-sample, even though in practice it can't be implemented. Because a strategy can't perform well out-of-sample if it doesn't perform well in-sample.

The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

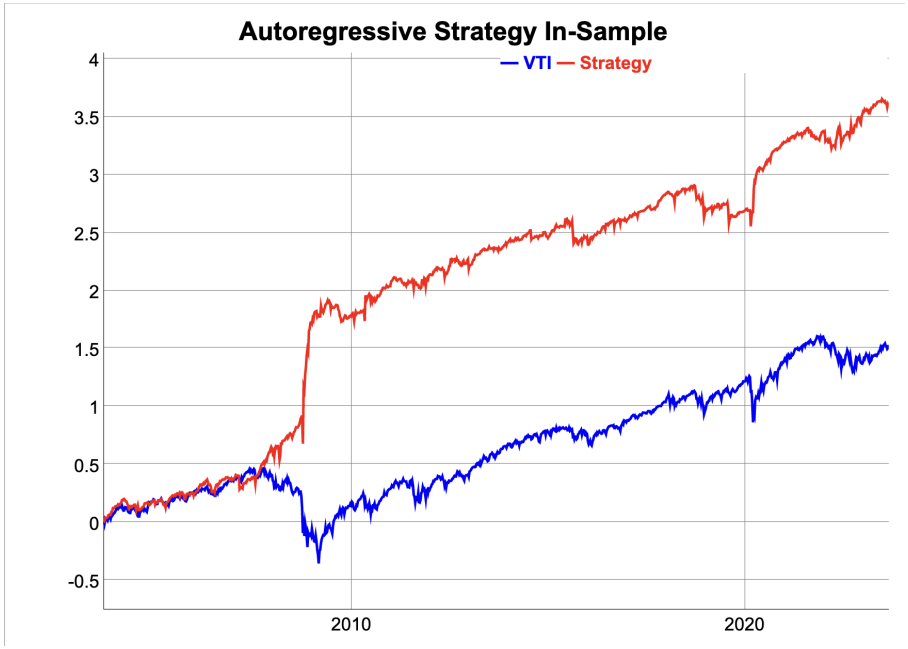
The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and very high risk.

The leverage can be reduced by scaling (dividing) the forecasts by their trailing volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Scale the forecasts by their volatility
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.2))
> fcastsc <- ifelse(fcastv > 0, fcasts/fcastv, 0)
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcastsc
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




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

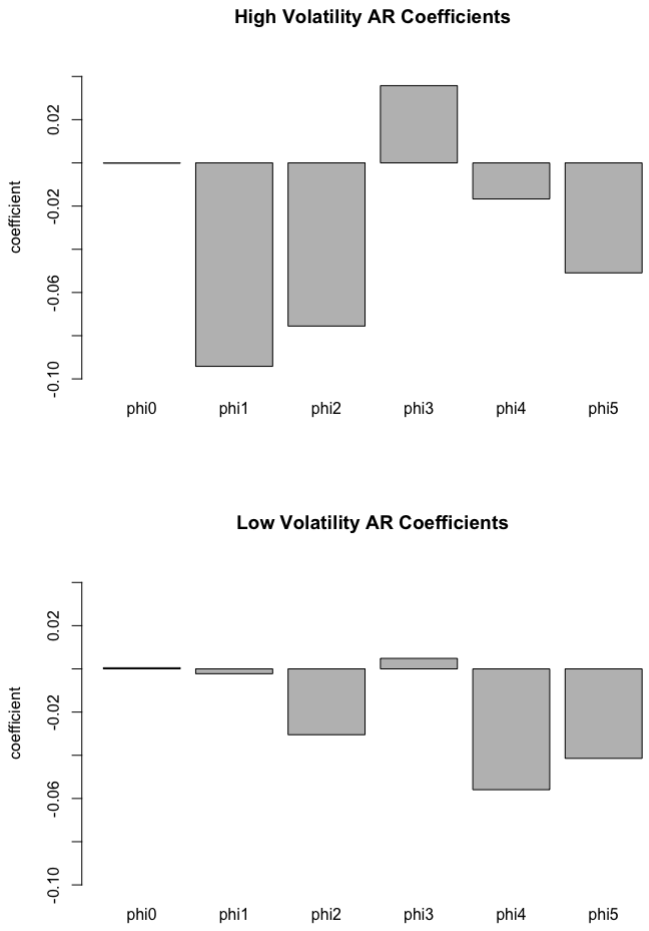
# Autoregressive Coefficients in Periods of Low and High Volatility

The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.

The autoregressive coefficients in periods of high volatility are very different from those under low volatility.

In periods of high volatility, there are larger negative autocorrelations than in low volatility.

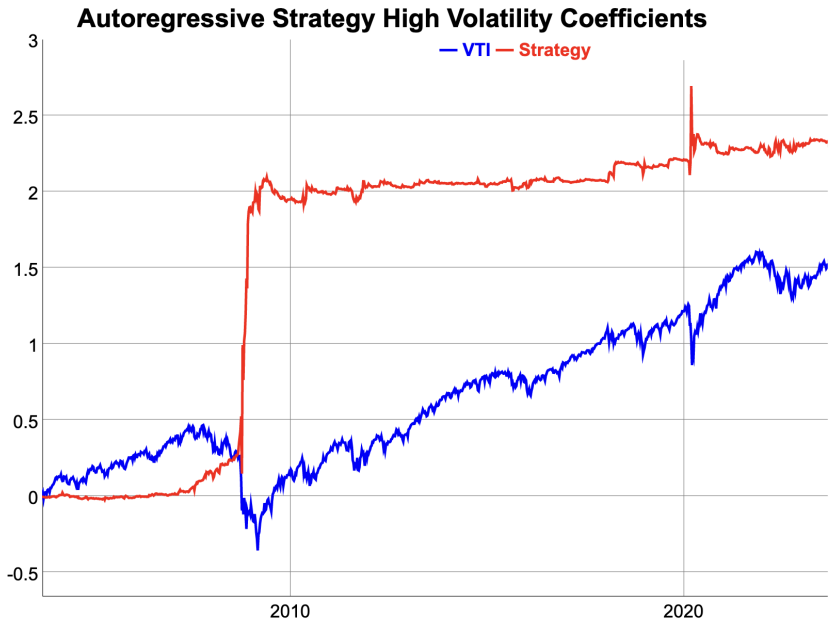
```
> # Calculate the high volatility AR coefficients
> respv <- retp["2008/2011"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffh <- drop(predinv %*% respv)
> coeffn <- paste0("phi", 0:(NROW(coeffh)-1))
> barplot(coeffh ~ coeffn, main="High Volatility AR Coefficients",
+   col="grey", xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
> # Calculate the low volatility AR coefficients
> respv <- retp["2012/2019"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffl <- drop(predinv %*% respv)
> barplot(coeffl ~ coeffn, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```



# Performance of Low and High Volatility Autoregressive Coefficients

The autoregressive coefficients obtained from periods of high volatility are overfitted and only perform well in periods of high volatility. Similarly the low volatility coefficients.

```
> # Calculate the pnls for the high volatility AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retp)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> fcasts <- predm %*% coeffh
> pnlh <- retp*fcasts
> pnlh <- pnlh*sd(retp[retp<0])/sd(pnlh[pnlh<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnlh)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy High Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Calculate the pnls for the low volatility AR coefficients
> fcasts <- predm %*% coeffl
> pnll <- retp*fcasts
> pnll <- pnll*sd(retp[retp<0])/sd(pnll[pnll<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnll)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Low Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```




# The *Winsor* Function

Some models produce very large dollar allocations, leading to large portfolio leverage (dollars invested divided by the capital).

The *winsor function* maps the *model weight* $w$ into the dollar amount for investment. The hyperbolic tangent function can serve as a winsor function:

$$W(x) = \frac{\exp(\lambda w) - \exp(-\lambda w)}{\exp(\lambda w) + \exp(-\lambda w)}$$

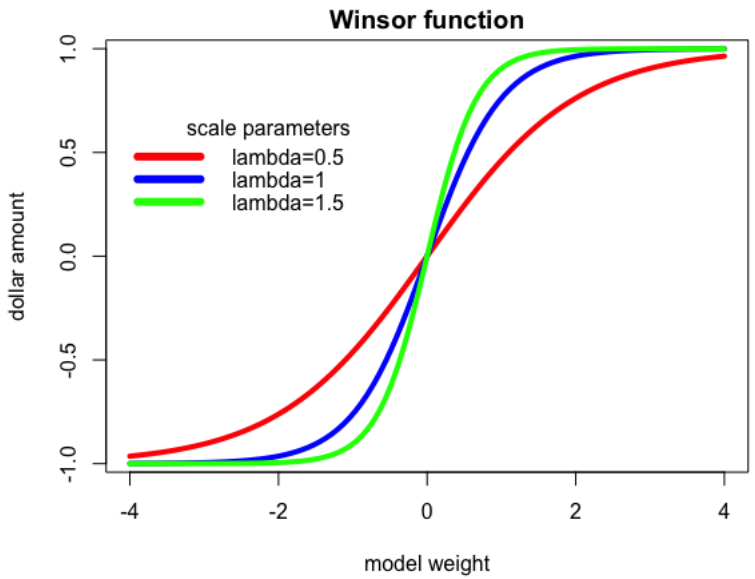
Where $\lambda$ is the scale parameter.

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

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

For very small values of the scale parameter $\lambda$, the invested dollar amount is linear for a wide range of *model weights*. So the strategy is mostly invested in dollar amounts proportional to the *model weights*.

For very large values of the scale parameter $\lambda$, the invested dollar amount jumps from $-1\$$ for negative *model weights* to $+1\$$ for positive *model weight* values. So the strategy is invested in either $-1\$$ or $+1\$$ dollar amounts.



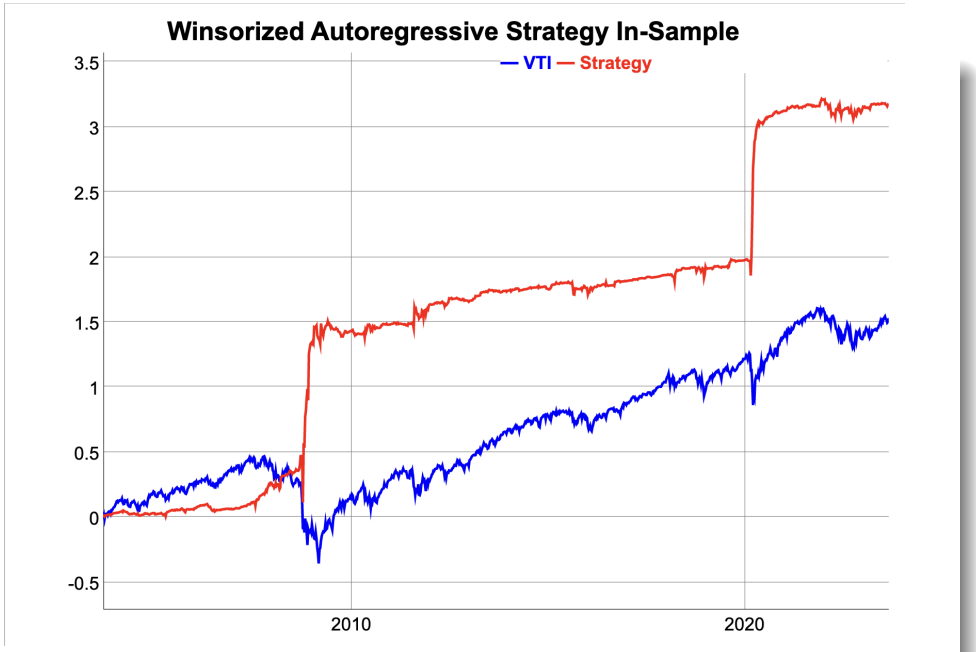

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambda) tanh(lambda*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="model weight", ylab="dollar amount",
+ col=colorv[indeks], add=(indeks>1))
+ }  # end for
> # Add title and legend
> title(main="Winsor function", line=0.5)
> legend("topleft", title="scale parameters\n",
+    paste("lambda", lambdav, sep="="), inset=0.0, cex=1.0,
+    lwd=6, bty="n", y.intersp=0.3, lty=1, col=colorv)
```

# Winsorized Autoregressive Strategy

The performance of the autoregressive strategy can be improved by fitting its coefficients using the *winsorized returns*, to reduce the effect of time-dependent volatility.

The performance can also be improved by *winsorizing* the forecasts, by reducing the leverage due to very large forecasts.

```
> # Winsorize the VTI returns
> retw <- winsorfun(retp/0.01, lambda=0.1)
> # Define the response and predictor matrices
> predm <- lapply(1:orderp, rutils::lagit, input=retw)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retw
> # Calculate the in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Winsorize the forecasts
> # fcasts <- winsorfun(fcasts/mad(fcasts), lambda=1.5)
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




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

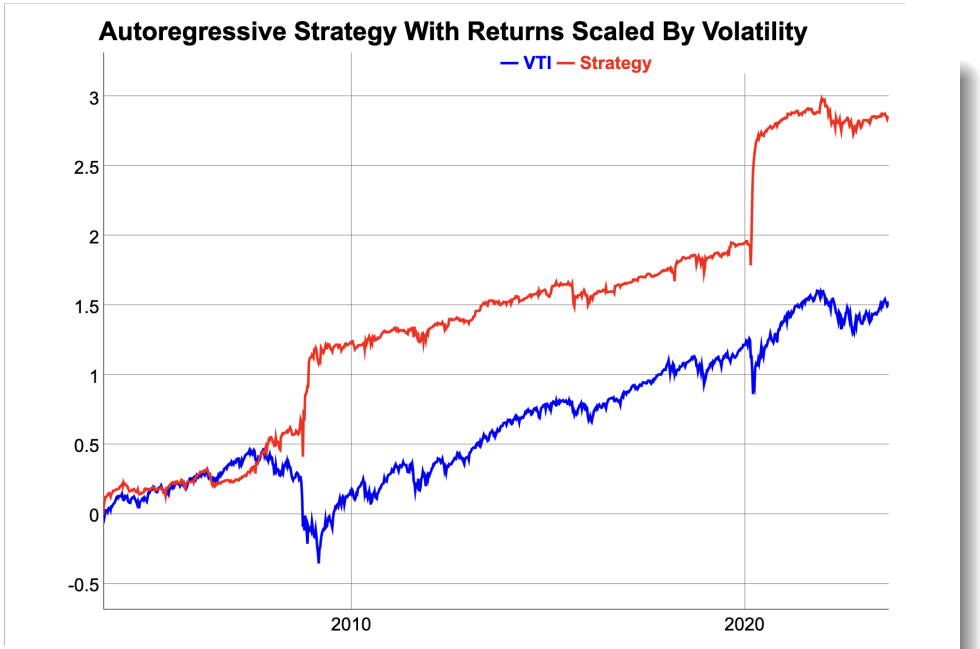
# Autoregressive Strategy With Returns Scaled By Volatility

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by their trailing volatility.

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Scale the returns by their trailing volatility
> varv <- HighFreq::run_var(retp, lambda=0.99)
> retsc <- ifelse(varv > 0, retp/sqrt(varv), 0)
> # Calculate the AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retsc)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retsc
> # Calculate the in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```



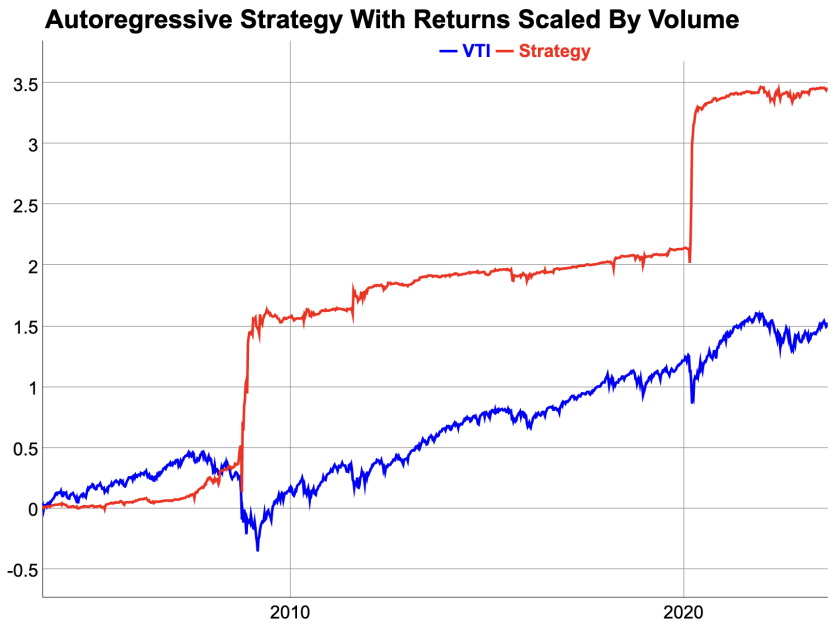

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Returns Scaled By Volatility"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# Autoregressive Strategy in Trading Time

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by the trading volumes.

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns in *trading time*, to account for time-dependent volatility.

```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> datev <- zoo::index(ohlc)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(log(closep))
> volumv <- quantmod::Vo(ohlc)
> # Calculate trailing average volume
> volumr <- HighFreq::run_mean(volumv, lambda=0.25)
> # Scale the returns using volume clock to trading time
> retsc <- ifelse(volumv > 0, volumr*retp/volumv, 0)
> # Calculate the AR coefficients
> respv <- retsc
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




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

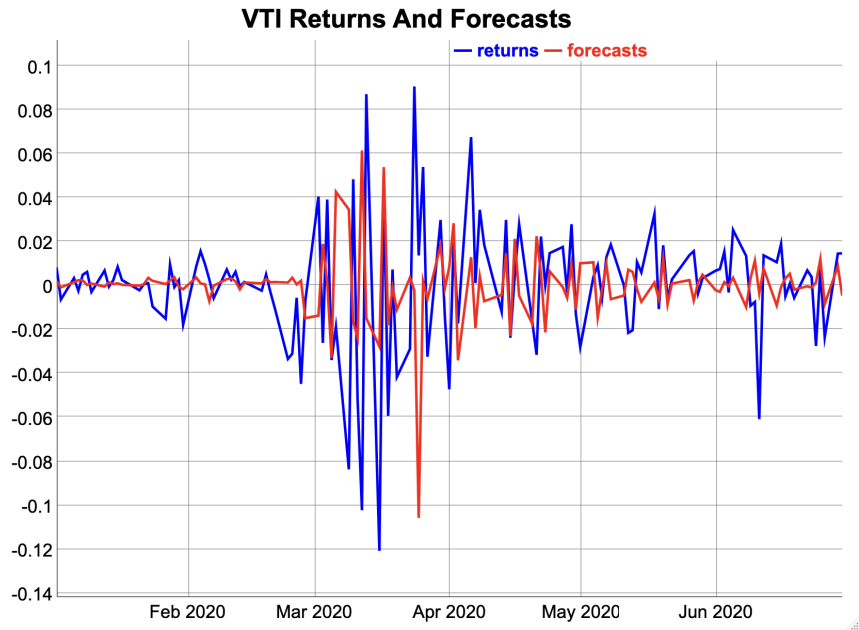
# Mean Squared Error of the Autoregressive Forecasting Model

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

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

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

```
> # Calculate the correlation between forecasts and returns
> cor(fcasts, retp)
> # Calculate the forecasting errors
> errorf <- (fcasts - retp)
> # Mean squared error
> mean(errorf^2)
```




```
> # Plot the forecasts
> datav <- cbind(retp, fcasts)["2020-01/2020-06"]
> colnames(datav) <- c("returns", "forecasts")
> dygraphs::dygraph(datav,
+   main="VTI Returns And Forecasts") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

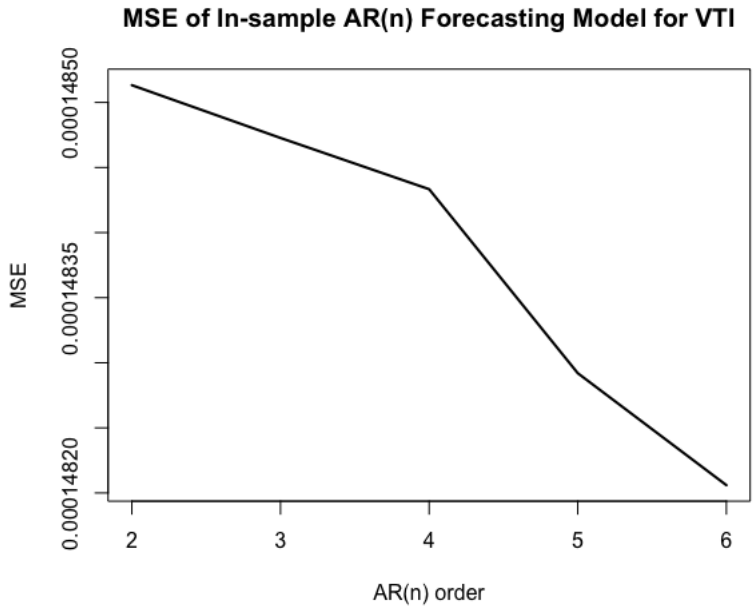
# In-sample Order Selection of Autoregressive Forecasting

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

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

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

```
> # Calculate the forecasts as function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[, 1:ordern])
+   coeff <- predinv %*% respv
+   # Calculate the in-sample forecasts of VTI
+   drop(predm[, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```




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

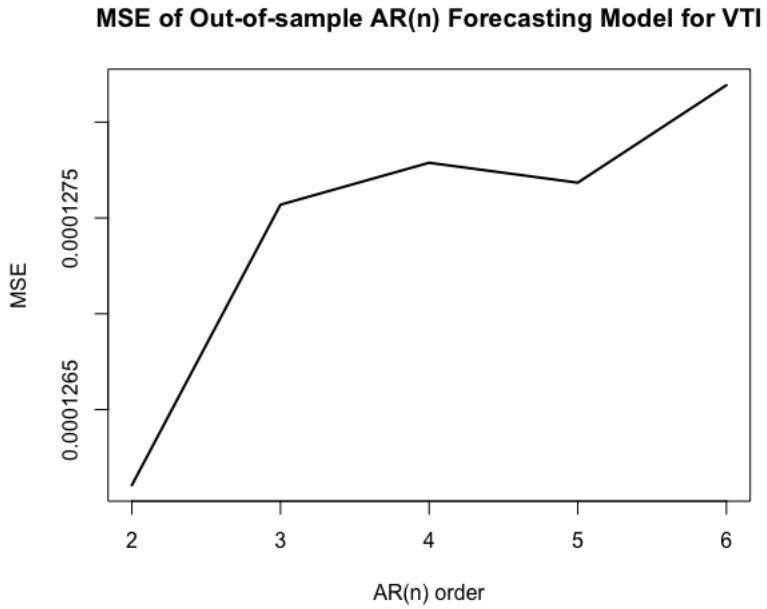
# Out-of-Sample Forecasting Using Autoregressive Models

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

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

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

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




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

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

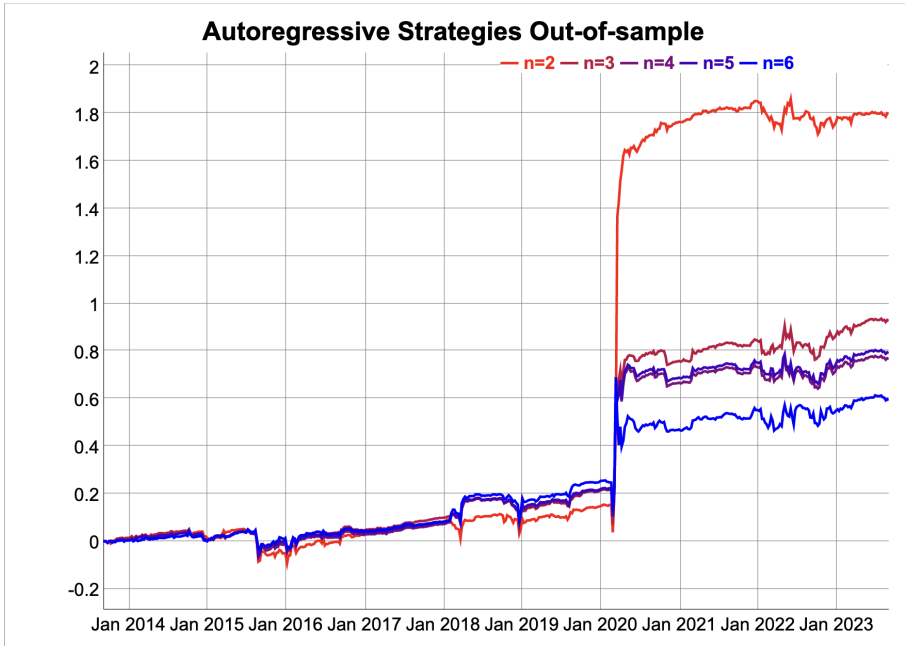
# Out-of-Sample Autoregressive Strategy

The autoregressive strategy invests a dollar amount of *VTI* proportional to the AR forecasts.

The out-of-sample, risk-adjusted performance of the autoregressive strategy is better for a smaller order parameter *n* of the *AR(n)* model.

The optimal order parameter is `n = 2`, with a positive intercept coefficient $\varphi_0$ (since the average *VTI* returns were positive), and a negative coefficient $\varphi_1$ (because of strong negative autocorrelations in periods of high volatility).

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




```
> # Calculate the optimal AR coefficients
> predinv <- MASS::ginv(predm[insample, 1:2])
> coeff <- drop(predinv %*% respv[insample])
> # Calculate the out-of-sample PnLs
> pnls <- lapply(fcasts, function(fcast) {
+   pnls <- fcast*retp[outsample]
+   pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
+ })  # end lapply
> pnls <- rutils::do_call(cbind, pnls)
> colnames(pnls) <- names(fcasts)
```

```
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls))
> colnamev <- colnames(pnls)
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Autoregressive Strategies Out-of-sample") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

# Rolling Autoregressive Forecasting Model

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

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

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

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

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

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

```
> # Perform rolling forecasting
> look_back <- 100
> fcasts <- sapply((look_back+1):nrows, function(tday) {
+   # Define rolling look-back range
+   startp <- max(1, tday-look_back)
+   # Or expanding look-back range
+   # startp <- 1
+   rangev <- startp:(tday-1) # In-sample range
+   # Invert the predictor matrix
+   predinv <- MASS::ginv(predm[rangev, ])
+   # Calculate the fitted AR coefficients
+   coeff <- predinv %*% respv[rangev]
+   # Calculate the out-of-sample forecast
+   predm[tday, ] %*% coeff
+ })  # end sapply
> # Add warmup period
> fcasts <- c(rep(0, look_back), fcasts)
```

# Backtesting Function for the Forecasting Model

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

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

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

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

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

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

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

```
> # Define backtesting function
> sim_fcasts <- function(look_back=100, ordern=5, fixedlb=TRUE) {
+   # Perform rolling forecasting
+   fcasts <- sapply((look_back+1):nrows, function(tday) {
+     # Define rolling look-back range
+     if (fixedlb)
+ startp <- max(1, tday-look_back) # Fixed look-back
+     else
+ startp <- 1 # Expanding look-back
+     rangev <- startp:(tday-1) # In-sample range
+     # Invert the predictor matrix
+     predinv <- MASS::ginv(predm[rangev, 1:ordern])
+     # Calculate the fitted AR coefficients
+     coeff <- predinv %*% respv[rangev]
+     # Calculate the out-of-sample forecast
+     predm[tday, 1:ordern] %*% coeff
+   })  # end sapply
+   # Add warmup period
+   fcasts <- c(rep(0, look_back), fcasts)
+ }  # end sim_fcasts
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(look_back=100, ordern=5)
> c(mse=mean((fcasts - retp)^2), cor=cor(retp, fcasts))
```

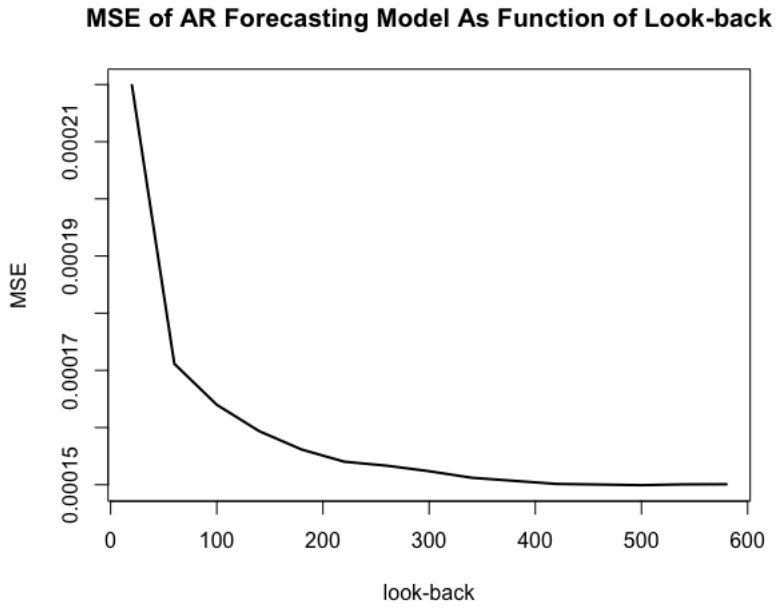
# Forecasting Dependence On the Look-back Interval

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

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

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

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




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> look_backs <- seq(20, 600, 40)
> fcasts <- parLapply(cluster, look_backs, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(look_backs, sim_fcasts, ordern=6, mc.cores=nc
```

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

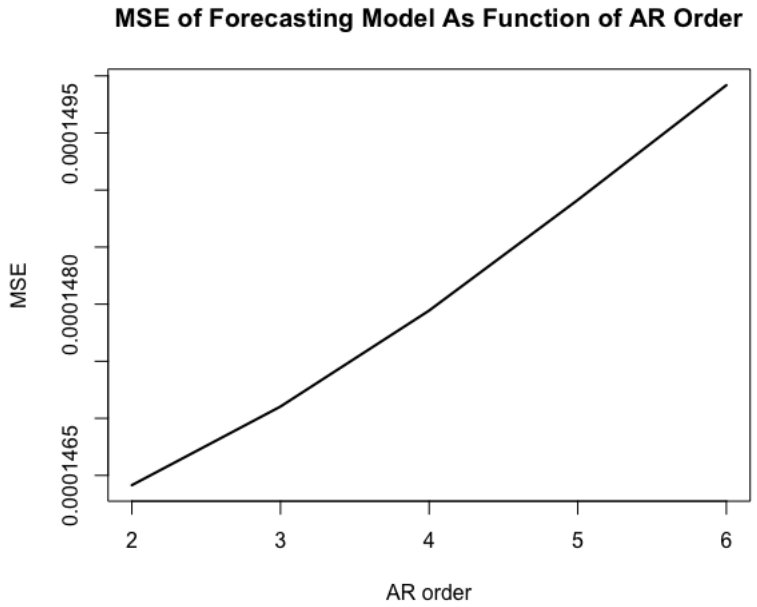
# Order Dependence With Fixed Look-back

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

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

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

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




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

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

# Autoregressive Strategy With Fixed Look-back

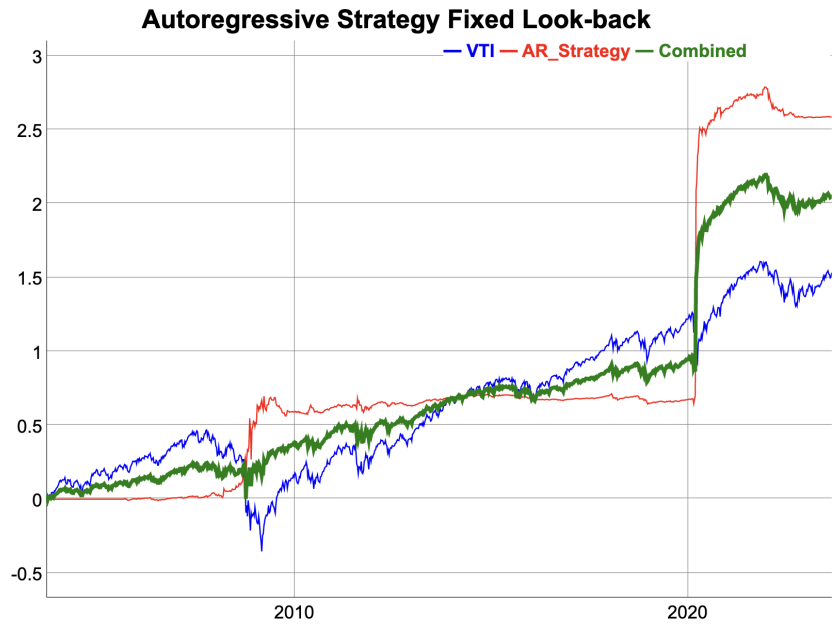
The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

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

To reduce excessive risk, a binary strategy can be used, with portfolio weights equal to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(look_back=look_back, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

# Order Dependence With Expanding Look-back

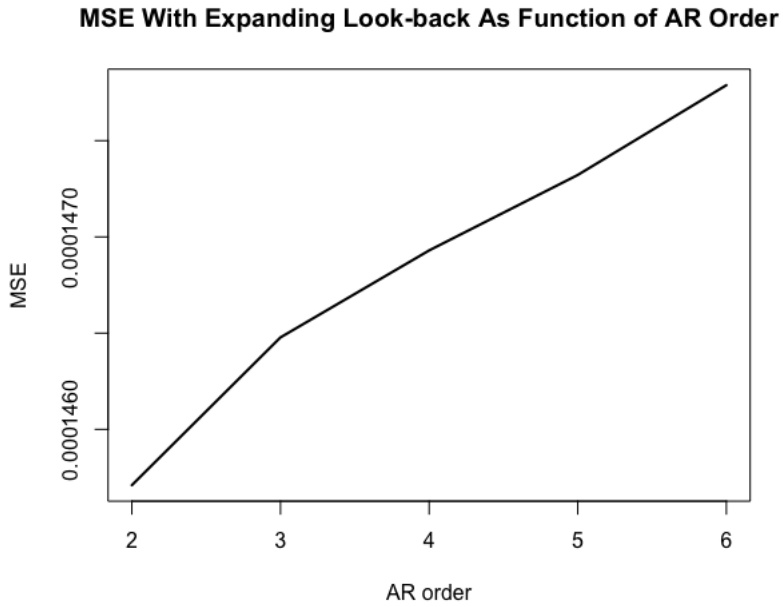
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model.

Longer look-back intervals (`look_back`) are usually better for the autoregressive forecasting model.

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

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



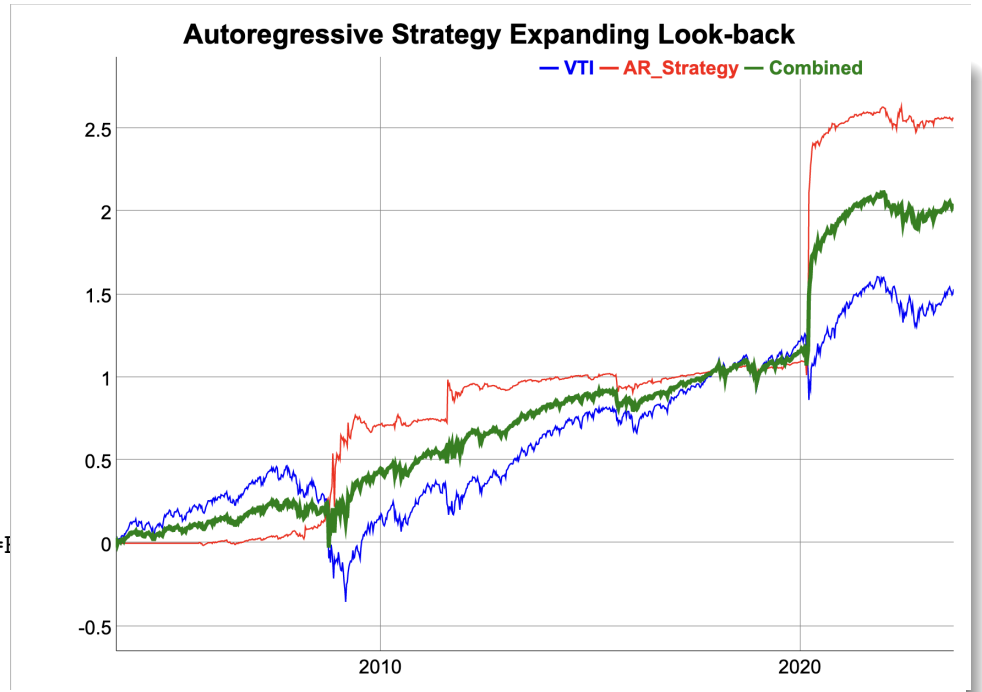

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv,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

# Autoregressive Strategy With Expanding Look-back

The model with an *expanding* look-back interval has better performance compared to the *fixed* look-back interval.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

```
> # Simulate the autoregressive forecasts with expanding look-back
> fcasts <- sim_fcasts(look_back=look_back, ordern=ordern, fixedlb=F
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

# Intraday and Overnight Stock Strategies

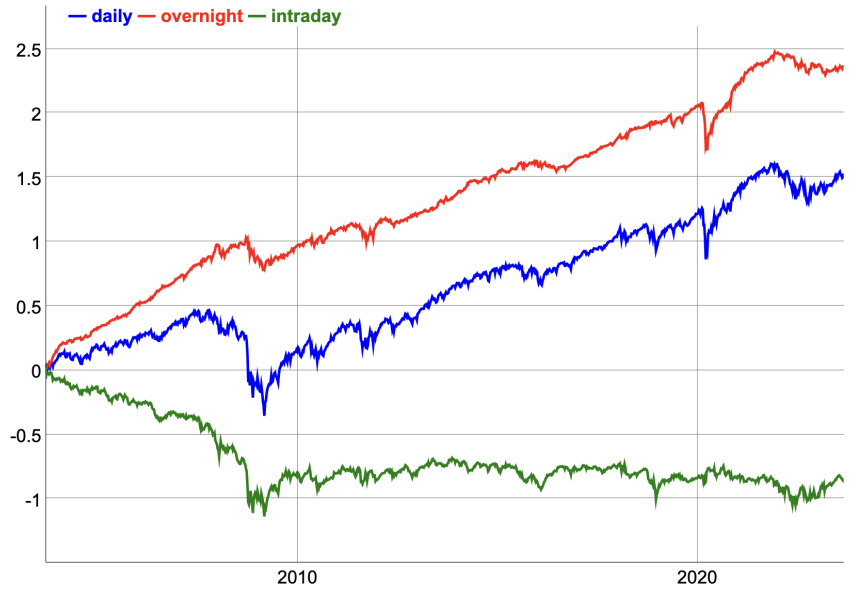
The overnight stock strategy consists of holding a long position only overnight (buying at the market close and selling at the open the next day).

The intraday stock strategy consists of holding a long position only during the daytime (buying at the market open and selling at the close the same day).

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 Market Anomaly* is not as pronounced after the 2008-2009 financial crisis.




```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> 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=600)
```

# The Bid-Ask Spread

The *bid-ask spread* is the difference between the best ask (offer) price minus the best bid price in the market.

The *bid-ask spread* can be estimated from the differences between the execution prices of consecutive buy and sell market orders (roundtrip trades).

Market orders are orders to buy or sell a stock immediately at the best available price in the market. Market orders guarantee that the trade will be executed, but they do not guarantee the execution price. Market orders are subject to the *bid-ask spread*.

Limit orders are orders to buy or sell a stock at the limit price or better (the investor sets the limit price). Limit orders do not guarantee that the trade will be executed, but they guarantee the execution price. Limit orders are placed only for a certain time when they are "live".

Market orders are executed by matching them with live limit orders through a matching engine at an exchange.

The *bid-ask spread* for many liquid ETFs is about 1 basis point. For example the *XLK ETF*

The most liquid *SPY ETF* usually trades at a *bid-ask spread* of only one tick (cent=$0.01, or about 0.2 basis points).

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.

```
> # Load the roundtrip trades
> dtable <- data.table::fread("/Users/jerzy/Develop/lecture_slides/c
> nrows <- NROW(dtable)
> class(dtable$timefill)
> # Sort the trades according to the execution time
> dtable <- dtable[order(dtable$timefill)]
> # Calculate the bid-ask spread
> priceb <- dtable$price[dtable$side == "buy"]
> pricese <- dtable$price[dtable$side == "sell"]
> bidask <- mean(priceb-pricese)
```

# EWMA Mean-Reversion Strategy For Intraday Returns

After the 2008-2009 financial crisis, the cumulative intraday stock index returns have been range-bound. So the intraday returns have more significant negative autocorrelations than overnight returns.

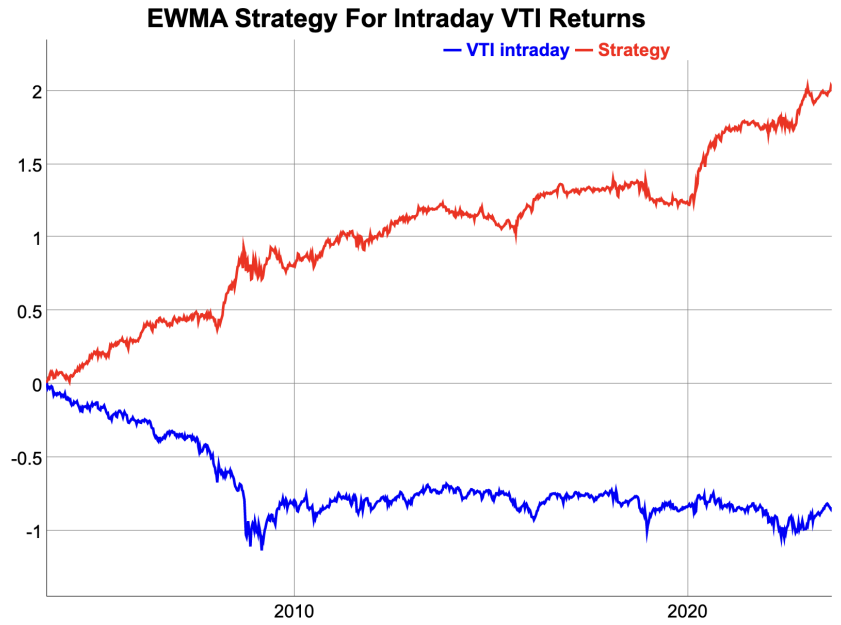
The *EWMA* mean-reversion strategy holds stock positions equal to the *sign* of the trailing *EWMA* of past returns.

The strategy adjusts its stock position at the end of each day, just before the close of the market.

An alternative strategy holds positions proportional to the trailing *EWMA* of past returns.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The limitation is that this strategy makes most of its profits in periods of high volatility, but otherwise it's profits are small.




```
> # Calculate the autocorrelations of intraday and overnight returns
> pacfl <- pacf(retd, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> pacfl <- pacf(reton, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> # Calculate the EWMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retd, lambda=0.4)
> # Calculate the positions and pnls
> posv <- rutils::lagit(sign(retma), lagg=1)
> pnls <- (-retd*posv)
```

```
> # Calculate the pnls and the transaction costs
> bidask <- 0.0001 # Bid-ask spread equal to 1 basis point
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI intraday", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+ + c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of EWMA strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="EWMA Strategy For Intraday VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

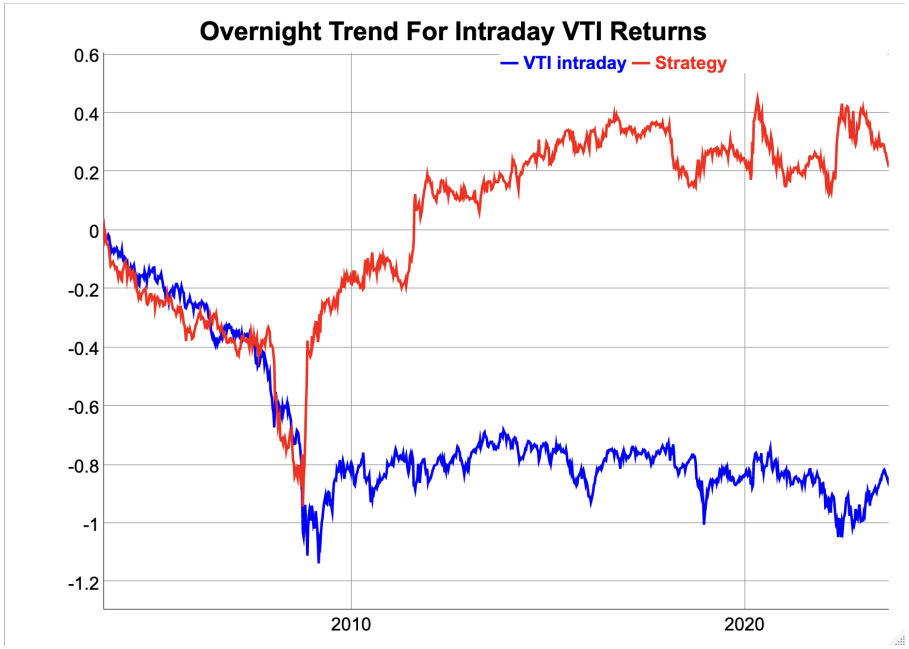
# Overnight Trend Strategy

Analysts at *JPMorgan* and at the *Federal Reserve* have observed that there is a trend in the overnight returns.

Positive overnight returns are often followed by positive intraday returns, and vice versa.

If the overnight returns were positive, then the strategy buys \$1 dollar of stock at the market open and sells it at the market close, or if the overnight returns were negative then it shorts -\$1 dollar of stock.

The strategy has performed well immediately after the 2008-2009 financial crisis, but it has waned in recent years.




```
> # Calculate the pnls and the transaction costs
> posv <- sign(reton)
> pnls <- posv*retd
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI intraday", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of EWMA strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+ main="Overnight Trend For Intraday VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# Bollinger Strategy For Intraday Returns

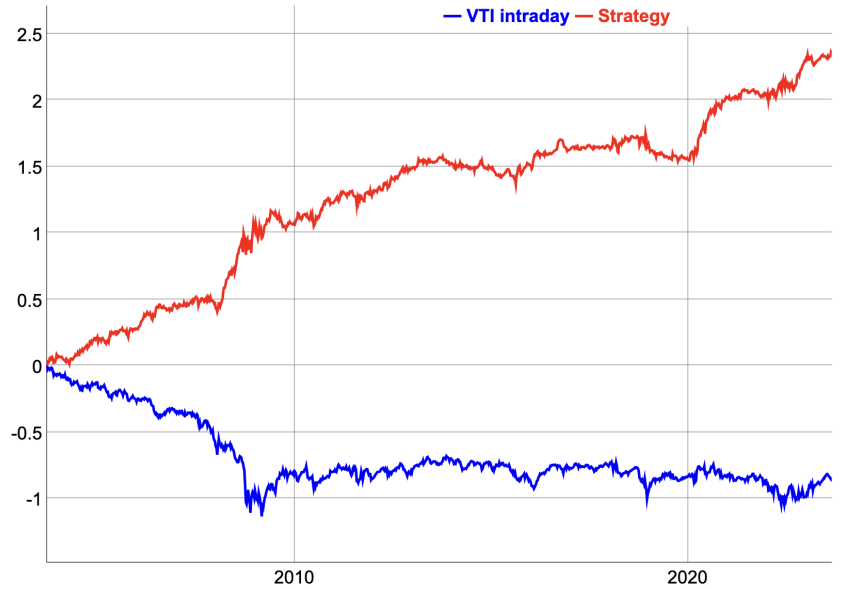
The Bollinger *z-score* for intraday returns $z_t$ , is equal to the difference between the cumulative returns $r_t$ minus their trailing mean $\bar{r}_t$, divided by their volatility $\sigma_t$:

$$z_t = \frac{r_t - \bar{r}_t}{\sigma_t}$$

The Bollinger strategy switches to \$1 dollar long stock after the *z-score* drops below the threshold of -1 (indicating the prices are cheap), or switches to -\$1 dollar short after the *z-score* exceeds the threshold of 1 (indicating the prices are rich - expensive).

The Bollinger strategy is a *mean reverting* (contrarian) strategy because it bets on the cumulative returns reverting to their mean value.

The Bollinger strategy has performed well for intraday *VTI* returns because they exhibit significant mean-reversion.




```
> # Calculate the z-scores of the cumulative intraday returns
> retc <- cumsum(retd)
> lambda <- 0.24
> retm <- rutils::lagit(HighFreq::run_mean(retc, lambda=lambda))
> retv <- sqrt(rutils::lagit(HighFreq::run_var(retc, lambda=lambda)
> zscores <- ifelse(retv > 0, (retc - retm)/retv, 0)
> # Calculate the positions from the Bollinger z-scores
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(zscores > 1, -1, posv)
> posv <- ifelse(zscores < -1, 1, posv)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv, lagg=1)
```

```
> # Calculate the pnls and the transaction costs
> pnls <- retd*posv
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # Calculate the Sharpe ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI intraday", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of intraday Bollinger strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Bollinger strategy For Intraday VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

# Homework Assignment

## Required

- Study all the lecture slides in *FRE7241_Lecture_4.pdf*, and run all the code in *FRE7241_Lecture_4.R*

## Recommended

- Download from NYU Classes and read about momentum strategies:
  *Moskowitz Time Series Momentum.pdf*
  *Bouchaud Momentum Mean Reversion Equity Returns.pdf*
  *Hurst Pedersen AQR Momentum Evidence.pdf*