FRE7241 Algorithmic Portfolio Management

Lecture#2, Spring 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

March 26, 2024

Kernel Density of Asset Returns

The kernel density is proportional to the number of data points close to a given point.

The kernel density is analogous to a histogram, but it provides more detailed information about the distribution of the data.

The smoothing kernel $K(x)$ is a symmetric function which decreases with the distance x.

The kernel density d_r at a point r is equal to the sum over the kernel function $K(x)$:

$$d_r = \sum_{j=1}^{n} K(r - r_j)$$

The function `density()` calculates a kernel estimate of the probability density for a sample of data.

The parameter *smoothing bandwidth* is the standard deviation of the smoothing kernel $K(x)$.

The function `density()` returns a vector of densities at equally spaced points, not for the original data points.

The function `approx()` interpolates a vector of data into another vector.

```
> library(rutils)  # Load package rutils
> # Calculate VTI percentage returns
> retp <- rutils::etfenv$returns$VTI
> retp <- drop(coredata(na.omit(retp)))
> nrows <- NROW(retp)
> # Mean and standard deviation of returns
> c(mean(retp), sd(retp))
> # Calculate the smoothing bandwidth as the MAD of returns 10 point
> retp <- sort(retp)
> bwidth <- 10*mad(rutils::diffit(retp, lagg=10))
> # Calculate the kernel density
> densv <- sapply(1:nrows, function(it) {
+   sum(dnorm(retp-retp[it], sd=bwidth))
+ })  # end sapply
> madv <- mad(retp)
> plot(retp, densv, xlim=c(-5*madv, 5*madv),
+      t="l", col="blue", lwd=3,
+      xlab="returns", ylab="density",
+      main="Density of VTI Returns")
> # Calculate the kernel density using density()
> densv <- density(retp, bw=bwidth)
> NROW(densv$y)
> x11(width=6, height=5)
> plot(densv, xlim=c(-5*madv, 5*madv),
+      xlab="returns", ylab="density",
+      col="blue", lwd=3, main="Density of VTI Returns")
> # Interpolate the densv vector into returns
> densv <- approx(densv$x, densv$y, xout=retp)
> all.equal(densv$x, retp)
> plot(densv, xlim=c(-5*madv, 5*madv),
+      xlab="returns", ylab="density",
+      t="l", col="blue", lwd=3,
+      main="Density of VTI Returns")
```

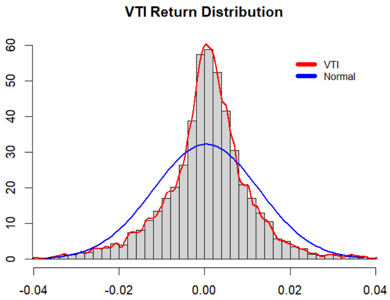
Distribution of Asset Returns

Asset returns are usually not normally distributed and they exhibit *leptokurtosis* (large kurtosis, or fat tails).

The function `hist()` calculates and plots a histogram, and returns its data *invisibly*.

The parameter `breaks` is the number of cells of the histogram.

The function `lines()` draws a line through specified points.




```
> # Plot histogram
> histp <- hist(retp, breaks=100, freq=FALSE,
+   xlim=c(-5*madv, 5*madv), xlab="", ylab="",
+   main="VTI Return Distribution")
> # Draw kernel density of histogram
> lines(densv, col="red", lwd=2)
> # Add density of normal distribution
> curve(expr=dnorm(x, mean=mean(retp), sd=sd(retp)),
+ add=TRUE, lwd=2, col="blue")
> # Add legend
> legend("topright", inset=0.05, cex=0.8, title=NULL,
+  leg=c("VTI", "Normal"), bty="n", y.intersp=0.4,
+  lwd=6, bg="white", col=c("red", "blue"))
```

The Quantile-Quantile Plot

A *Quantile-Quantile* (*Q-Q*) plot is a plot of points with the same *quantiles*, from two probability distributions.

If the two distributions are similar then all the points in the *Q-Q* plot lie along the diagonal.

The *VTI Q-Q* plot shows that the *VTI* return distribution has fat tails.

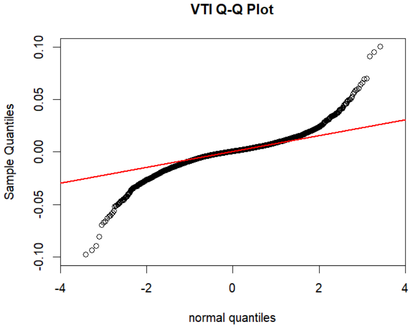
The *p*-value of the *Shapiro-Wilk* test is very close to zero, which shows that the *VTI* returns are very unlikely to be normal.

The function `shapiro.test()` performs the *Shapiro-Wilk* test of normality.

The function `qqnorm()` produces a normal *Q-Q* plot.

The function `qqline()` fits a line to the normal quantiles.

```
> # Create normal Q-Q plot
> qqnorm(retp, ylim=c(-0.1, 0.1), main="VTI Q-Q Plot",
+  xlab="Normal Quantiles")
> # Fit a line to the normal quantiles
> qqline(retp, col="red", lwd=2)
> # Perform Shapiro-Wilk test
> shapiro.test(retp)
```



Boxplots of Distributions of Values

Box-and-whisker plots (*boxplots*) are graphical representations of a distribution of values.

The bottom and top box edges (*hinges*) are equal to the first and third quartiles, and the *box* width is equal to the interquartile range (*IQR*).

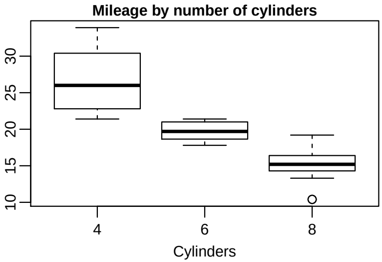
The nominal range is equal to 1.5 times the *IQR* above and below the box *hinges*.

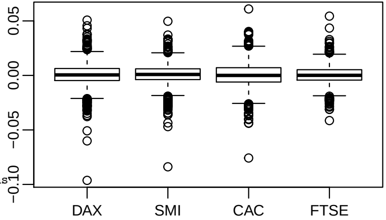
The *whiskers* are dashed vertical lines representing values beyond the first and third quartiles, but within the nominal range.

The *whiskers* end at the last values within the nominal range, while the open circles represent outlier values beyond the nominal range.

The function `boxplot()` has two `methods`: one for `formula` objects (for categorical variables), and another for `data frames`.

```
> # Boxplot method for formula
> boxplot(formula=mpg ~ cyl, data=mtcars,
+   main="Mileage by number of cylinders",
+   xlab="Cylinders", ylab="Miles per gallon")
> # Boxplot method for data frame of EuStockMarkets percentage returns
> boxplot(x=diff(log(EuStockMarkets)))
```





Higher Moments of Asset Returns

The estimators of moments of a probability distribution are given by:

Sample mean: $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

Sample variance: $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$

With their expected values equal to the population mean and standard deviation:
$\mathbb{E}[\bar{x}] = \mu$ and $\mathbb{E}[\hat{\sigma}] = \sigma$

The sample skewness (third moment):

$$\varsigma = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{\hat{\sigma}})^3$$

The sample kurtosis (fourth moment):

$$\kappa = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{\hat{\sigma}})^4$$

The normal distribution has skewness equal to 0 and kurtosis equal to 3.

Stock returns typically have negative skewness and kurtosis much greater than 3.

```
> # Calculate VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Number of observations
> nrows <- NROW(retp)
> # Mean of VTI returns
> retm <- mean(retp)
> # Standard deviation of VTI returns
> stdev <- sd(retp)
> # Skewness of VTI returns
> nrows/((nrows-1)*(nrows-2))*sum(((retp - retm)/stdev)^3)
> # Kurtosis of VTI returns
> nrows*(nrows+1)/((nrows-1)^3)*sum(((retp - retm)/stdev)^4)
> # Random normal returns
> retp <- rnorm(nrows, sd=stdev)
> # Mean and standard deviation of random normal returns
> retm <- mean(retp)
> stdev <- sd(retp)
> # Skewness of random normal returns
> nrows/((nrows-1)*(nrows-2))*sum(((retp - retm)/stdev)^3)
> # Kurtosis of random normal returns
> nrows*(nrows+1)/((nrows-1)^3)*sum(((retp - retm)/stdev)^4)
```

Functions for Calculating Skew and Kurtosis

R provides an easy way for users to write functions.

The function calc_skew() calculates the skew of returns, and calc_kurt() calculates the kurtosis.

Functions return the value of the last expression that is evaluated.

```
> # calc_skew() calculates skew of returns
> calc_skew <- function(retp) {
+   retp <- na.omit(retp)
+   sum(((retp - mean(retp))/sd(retp))^3)/NROW(retp)
+ }  # end calc_skew
> # calc_kurt() calculates kurtosis of returns
> calc_kurt <- function(retp) {
+   retp <- na.omit(retp)
+   sum(((retp - mean(retp))/sd(retp))^4)/NROW(retp)
+ }  # end calc_kurt
> # Calculate skew and kurtosis of VTI returns
> calc_skew(retp)
> calc_kurt(retp)
> # calc_mom() calculates the moments of returns
> calc_mom <- function(retp, moment=3) {
+   retp <- na.omit(retp)
+   sum(((retp - mean(retp))/sd(retp))^moment)/NROW(retp)
+ }  # end calc_mom
> # Calculate skew and kurtosis of VTI returns
> calc_mom(retp, moment=3)
> calc_mom(retp, moment=4)
```

Standard Errors of Estimators

Statistical estimators are functions of samples (which are random variables), and therefore are themselves *random variables*.

The *standard error* (SE) of an estimator is defined as its *standard deviation* (not to be confused with the *population standard deviation* of the underlying random variable).

For example, the *standard error* of the estimator of the mean is equal to:

$$\sigma_{\mu} = \frac{\sigma}{\sqrt{n}}$$

Where σ is the *population standard deviation* (which is usually unkown).

The *estimator* of this *standard error* is equal to:

$$SE_{\mu} = \frac{\hat{\sigma}}{\sqrt{n}}$$

where: $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$ is the sample standard deviation (the estimator of the population standard deviation).

```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> nrows <- 1000
> datav <- rnorm(nrows)
> # Sample mean
> mean(datav)
> # Sample standard deviation
> sd(datav)
> # Standard error of sample mean
> sd(datav)/sqrt(nrows)
```

Normal (Gaussian) Probability Distribution

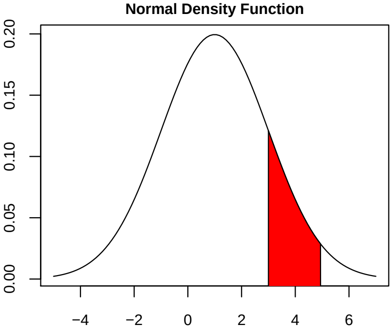
The *Normal (Gaussian)* probability density function is given by:

$$\phi(x, \mu, \sigma) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

The *Standard Normal* distribution $\phi(0, 1)$ is a special case of the *Normal* $\phi(\mu, \sigma)$ with $\mu = 0$ and $\sigma = 1$.

The function `dnorm()` calculates the *Normal* probability density.

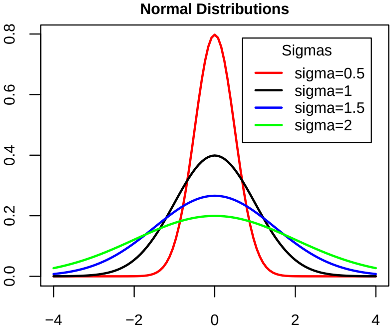
```
> xvar <- seq(-5, 7, length=100)
> yvar <- dnorm(xvar, mean=1.0, sd=2.0)
> plot(xvar, yvar, type="l", lty="solid", xlab="", ylab="")
> title(main="Normal Density Function", line=0.5)
> startp <- 3; endd <- 5  # Set lower and upper bounds
> # Set polygon base
> subv <- ((xvar >= startp) & (xvar <= endd))
> polygon(c(startp, xvar[subv], endd),  # Draw polygon
+   c(-1, yvar[subv], -1), col="red")
```



Normal (Gaussian) Probability Distributions

Plots of several *Normal* distributions with different values of σ, using the function curve() for plotting functions given by their name.

```
> sigmavs <- c(0.5, 1, 1.5, 2)  # Sigma values
> # Create plot colors
> colorv <- c("red", "black", "blue", "green")
> # Create legend labels
> labelv <- paste("sigma", sigmavs, sep="=")
> for (it in 1:4) {  # Plot four curves
+   curve(expr=dnorm(x, sd=sigmavs[it]),
+   xlim=c(-4, 4), xlab="", ylab="", lwd=2,
+   col=colorv[it], add=as.logical(it-1))
+ }  # end for
> # Add title
> title(main="Normal Distributions", line=0.5)
> # Add legend
> legend("topright", inset=0.05, title="Sigmas", y.intersp=0.4,
+  labelv, cex=0.8, lwd=2, lty=1, bty="n", col=colorv)
```



Student's *t-distribution*

Let $z_1, \ldots, z_\nu$ be independent standard normal random variables, with sample mean: $\bar{z} = \frac{1}{\nu} \sum_{i=1}^{\nu} z_i$ ($\mathbb{E}[\bar{z}] = \mu$) and sample variance: $\hat{\sigma}^2 = \frac{1}{\nu - 1} \sum_{i=1}^{\nu} (z_i - \bar{z})^2$

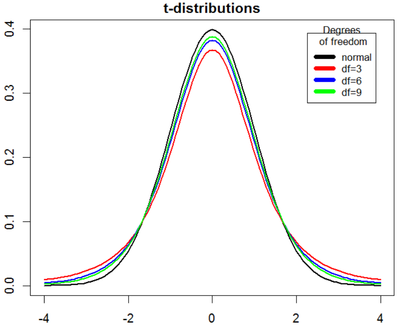
Then the random variable (*t-ratio*):

$$t = \frac{\bar{z} - \mu}{\hat{\sigma} / \sqrt{\nu}}$$

Follows the *t-distribution* with ν degrees of freedom, with the probability density function:

$$f(t) = \frac{\Gamma((\nu + 1)/2)}{\sqrt{\pi \nu}\, \Gamma(\nu/2)} \, (1 + t^2/\nu)^{-(\nu+1)/2}$$

```
> degf <- c(3, 6, 9)  # Df values
> colorv <- c("black", "red", "blue", "green")
> labelv <- c("normal", paste("df", degf, sep="="))
> # Plot a Normal probability distribution
> curve(expr=dnorm, xlim=c(-4, 4), xlab="", ylab="", lwd=2)
> for (it in 1:3) {  # Plot three t-distributions
+   curve(expr=dt(x, df=degf[it]), xlab="", ylab="",
+ lwd=2, col=colorv[it+1], add=TRUE)
+ }  # end for
```




```
> # Add title
> title(main="t-distributions", line=0.5)
> # Add legend
> legend("topright", inset=0.05, bty="n", y.intersp=0.4,
+        title="Degrees\n of freedom", labelv,
+        cex=0.8, lwd=6, lty=1, col=colorv)
```

Mixture Models of Returns

Mixture models are produced by randomly sampling data from different distributions.

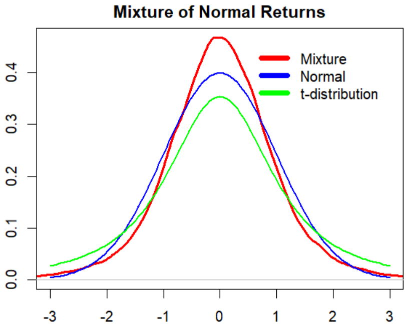
The mixture of two normal distributions with different variances produces a distribution with *leptokurtosis* (large kurtosis, or fat tails).

Student's *t-distribution* has fat tails because the sample variance in the denominator of the *t-ratio* is variable.

The time-dependent volatility of asset returns is referred to as *heteroskedasticity*.

Random processes with *heteroskedasticity* can be considered a type of mixture model.

The *heteroskedasticity* produces *leptokurtosis* (large kurtosis, or fat tails).




```
> # Mixture of two normal distributions with sd=1 and sd=2
> nrows <- 1e5
> retp <- c(rnorm(nrows/2), 2*rnorm(nrows/2))
> retp <- (retp-mean(retp))/sd(retp)
> # Kurtosis of normal
> calc_kurt(rnorm(nrows))
> # Kurtosis of mixture
> calc_kurt(retp)
> # Or
> nrows*sum(retp^4)/(nrows-1)^2
```

```
> # Plot the distributions
> plot(density(retp), xlab="", ylab="",
+   main="Mixture of Normal Returns",
+   xlim=c(-3, 3), type="l", lwd=3, col="red")
> curve(expr=dnorm, lwd=2, col="blue", add=TRUE)
> curve(expr=dt(x, df=3), lwd=2, col="green", add=TRUE)
> # Add legend
> legend("topright", inset=0.05, lty=1, lwd=6, bty="n",
+   legend=c("Mixture", "Normal", "t-distribution"), y.intersp=0.4,
+   col=c("red", "blue", "green"))
```

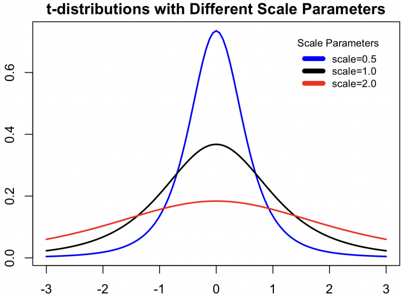
Non-standard Student's *t-distribution*

The non-standard Student's *t-distribution* has the probability density function:

$$f(t) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\sigma\,\Gamma(\nu/2)}\,(1+(\frac{t-\mu}{\sigma})^2/\nu)^{-(\nu+1)/2}$$

It has non-zero mean equal to the location parameter μ, and a standard deviation proportional to the scale parameter σ.

```
> dev.new(width=6, height=5, noRStudioGD=TRUE)
> # x11(width=6, height=5)
> # Define density of non-standard t-distribution
> tdistr <- function(x, dfree, locv=0, scalev=1) {
+   dt((x-locv)/scalev, df=dfree)/scalev
+ }  # end tdistr
> # Or
> tdistr <- function(x, dfree, locv=0, scalev=1) {
+   gamma((dfree+1)/2)/(sqrt(pi*dfree)*gamma(dfree/2)*scalev)*
+     (1+((x-locv)/scalev)^2/dfree)^(-(dfree+1)/2)
+ }  # end tdistr
> # Calculate vector of scale values
> scalev <- c(0.5, 1.0, 2.0)
> colorv <- c("blue", "black", "red")
> labelv <- paste("scale", format(scalev, digits=2), sep="=")
> # Plot three t-distributions
> for (it in 1:3) {
+   curve(expr=tdistr(x, dfree=3, scalev=scalev[it]), xlim=c(-3, 3),
+ xlab="", ylab="", lwd=2, col=colorv[it], add=(it>1))
+ }  # end for
```




```
> # Add title
> title(main="t-distributions with Different Scale Parameters", line
> # Add legend
> legend("topright", inset=0.05, bty="n", title="Scale Parameters",
+        cex=0.8, lwd=6, lty=1, col=colorv, y.intersp=0.4)
```

The *Shapiro-Wilk* Test of Normality

The *Shapiro-Wilk* test is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ is from a normally distributed population.

The test statistic is equal to:

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Where the: $\{a_1, \ldots, a_n\}$ are proportional to the *order statistics* of random variables from the normal distribution.

$x_{(k)}$ is the k-th *order statistic*, and is equal to the k-th smallest value in the sample: $\{x_1, \ldots, x_n\}$.

The *Shapiro-Wilk* statistic follows its own distribution, and is less than or equal to 1.

The *Shapiro-Wilk* statistic is close to 1 for samples from normal distributions.

The p-value for *VTI* returns is extremely small, and we conclude that the *null hypothesis* is `FALSE`, and the *VTI* returns are not from a normally distributed population.

The *Shapiro-Wilk* test is not reliable for large sample sizes, so it's limited to less than 5000 sample size.

```
> # Calculate VTI percentage returns
> library(rutils)
> retp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))[1:499]
> # Reduce number of output digits
> ndigits <- options(digits=5)
> # Shapiro-Wilk test for normal distribution
> nrows <- NROW(retp)
> shapiro.test(rnorm(nrows))

Shapiro-Wilk normality test

data:  rnorm(nrows)
W = 0.995, p-value = 0.11
> # Shapiro-Wilk test for VTI returns
> shapiro.test(retp)

Shapiro-Wilk normality test

data:  retp
W = 0.991, p-value = 0.0029
> # Shapiro-Wilk test for uniform distribution
> shapiro.test(runif(nrows))

Shapiro-Wilk normality test

data:  runif(nrows)
W = 0.952, p-value = 1.3e-11
> # Restore output digits
> options(digits=ndigits$digits)
```

The *Jarque-Bera* Test of Normality

The *Jarque-Bera* test is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ is from a normally distributed population.

The test statistic is equal to:

$$JB = \frac{n}{6}(\varsigma^2 + \frac{1}{4}(\kappa - 3)^2)$$

Where the *skewness* and *kurtosis* are defined as:

$$\varsigma = \frac{1}{n} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{\hat{\sigma}})^3 \qquad \kappa = \frac{1}{n} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{\hat{\sigma}})^4$$

The *Jarque-Bera* statistic asymptotically follows the *chi-squared* distribution with 2 degrees of freedom.

The *Jarque-Bera* statistic is small for samples from normal distributions.

The *p*-value for *VTI* returns is extremely small, and we conclude that the *null hypothesis* is `FALSE`, and the *VTI* returns are not from a normally distributed population.

```
> library(tseries)  # Load package tseries
> # Jarque-Bera test for normal distribution
> jarque.bera.test(rnorm(nrows))

Jarque Bera Test

data:  rnorm(nrows)
X-squared = 4, df = 2, p-value = 0.1
> # Jarque-Bera test for VTI returns
> jarque.bera.test(retp)

Jarque Bera Test

data:  retp
X-squared = 22, df = 2, p-value = 2e-05
> # Jarque-Bera test for uniform distribution
> jarque.bera.test(runif(NROW(retp)))

Jarque Bera Test

data:  runif(NROW(retp))
X-squared = 27, df = 2, p-value = 1e-06
```

The *Kolmogorov-Smirnov* Test for Probability Distributions

The *Kolmogorov-Smirnov* test *null hypothesis* is that two samples: $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ were obtained from the same probability distribution.

The *Kolmogorov-Smirnov* statistic depends on the maximum difference between two empirical cumulative distribution functions (cumulative frequencies):

$$D = \sup_i |P(x_i) - P(y_i)|$$

The function `ks.test()` performs the *Kolmogorov-Smirnov* test and returns the statistic and its *p*-value *invisibly*.

The second argument to `ks.test()` can be either a `numeric` vector of data values, or a name of a cumulative distribution function.

The *Kolmogorov-Smirnov* test can be used as a *goodness of fit* test, to test if a set of observations fits a probability distribution.

```
> # KS test for normal distribution
> ks_test <- ks.test(rnorm(100), pnorm)
> ks_test$p.value
> # KS test for uniform distribution
> ks.test(runif(100), pnorm)
> # KS test for two shifted normal distributions
> ks.test(rnorm(100), rnorm(100, mean=0.1))
> ks.test(rnorm(100), rnorm(100, mean=1.0))
> # KS test for two different normal distributions
> ks.test(rnorm(100), rnorm(100, sd=2.0))
> # KS test for VTI returns vs normal distribution
> retp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))
> retp <- (retp - mean(retp))/sd(retp)
> ks.test(retp, pnorm)
```

Chi-squared Distribution

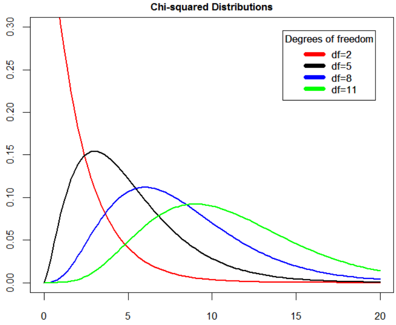
Let $z_1, \ldots, z_k$ be independent standard *Normal* random variables.

Then the random variable $X = \sum_{i=1}^{k} z_i^2$ is distributed according to the *Chi-squared* distribution with k degrees of freedom: $X \sim \chi_k^2$, and its probability density function is given by:

$$f(x) = \frac{x^{k/2-1}\, e^{-x/2}}{2^{k/2}\, \Gamma(k/2)}$$

The *Chi-squared* distribution with k degrees of freedom has mean equal to k and variance equal to $2k$.

```
> # Degrees of freedom
> degf <- c(2, 5, 8, 11)
> # Plot four curves in loop
> colorv <- c("red", "black", "blue", "green")
> for (it in 1:4) {
+   curve(expr=dchisq(x, df=degf[it]),
+   xlim=c(0, 20), ylim=c(0, 0.3),
+   xlab="", ylab="", col=colorv[it],
+   lwd=2, add=as.logical(it-1))
+ }  # end for
```




```
> # Add title
> title(main="Chi-squared Distributions", line=0.5)
> # Add legend
> labelv <- paste("df", degf, sep="=")
> legend("topright", inset=0.05, bty="n", y.intersp=0.4,
+        title="Degrees of freedom", labelv,
+        cex=0.8, lwd=6, lty=1, col=colorv)
```

The *Chi-squared* Test for the Goodness of Fit

Goodness of Fit tests are designed to test if a set of observations fits an assumed theoretical probability distribution.

The *Chi-squared* test tests if a frequency of counts fits the specified distribution.

The *Chi-squared* statistic is the sum of squared differences between the observed frequencies o_i and the theoretical frequencies p_i:

$$\chi^2 = N \sum_{i=1}^{n} \frac{(o_i - p_i)^2}{p_i}$$

Where N is the total number of observations.

The *null hypothesis* is that the observed frequencies are consistent with the theoretical distribution.

The function `chisq.test()` performs the *Chi-squared* test and returns the statistic and its p-value *invisibly*.

The parameter `breaks` in the function `hist()` should be chosen large enough to capture the shape of the frequency distribution.

```
> # Observed frequencies from random normal data
> histp <- hist(rnorm(1e3, mean=0), breaks=100, plot=FALSE)
> countsn <- histp$counts
> # Theoretical frequencies
> countst <- rutils::diffit(pnorm(histp$breaks))
> # Perform Chi-squared test for normal data
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
> # Return p-value
> chisq_test <- chisq.test(x=countsn, p=countst, rescale.p=TRUE, si
> chisq_test$p.value
> # Observed frequencies from shifted normal data
> histp <- hist(rnorm(1e3, mean=2), breaks=100, plot=FALSE)
> countsn <- histp$counts/sum(histp$counts)
> # Theoretical frequencies
> countst <- rutils::diffit(pnorm(histp$breaks))
> # Perform Chi-squared test for shifted normal data
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
> # Calculate histogram of VTI returns
> histp <- hist(retp, breaks=100, plot=FALSE)
> countsn <- histp$counts
> # Calculate cumulative probabilities and then difference them
> countst <- pt((histp$breaks-locv)/scalev, df=2)
> countst <- rutils::diffit(countst)
> # Perform Chi-squared test for VTI returns
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
```

The Likelihood Function of Student's *t-distribution*

The non-standard Student's *t-distribution* is:

$$f(t) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\sigma\,\Gamma(\nu/2)} \, (1 + (\frac{t-\mu}{\sigma})^2/\nu)^{-(\nu+1)/2}$$

It has non-zero mean equal to the location parameter μ, and a standard deviation proportional to the scale parameter σ.

The negative logarithm of the probability density is equal to:

$$-\log(f(t)) = -\log(\frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\Gamma(\nu/2)}) + \log(\sigma) + \frac{\nu+1}{2}\,\log(1 + (\frac{t-\mu}{\sigma})^2/\nu)$$

The *likelihood* function $\mathcal{L}(\theta|\bar{x})$ is a function of the model parameters θ, given the observed values $\bar{x}$, under the model's probability distribution $f(x|\theta)$:

$$\mathcal{L}(\theta|x) = \prod_{i=1}^{n} f(x_i|\theta)$$

```
> # Objective function from function dt()
> likefun <- function(par, dfree, data) {
+   -sum(log(dt(x=(data-par[1])/par[2], df=dfree)/par[2]))
+ }  # end likefun
> # Demonstrate equivalence with log(dt())
> likefun(c(1, 0.5), 2, 2:5)
> -sum(log(dt(x=(2:5-1)/0.5, df=2)/0.5))
> # Objective function is negative log-likelihood
> likefun <- function(par, dfree, data) {
+   sum(-log(gamma((dfree+1)/2)/(sqrt(pi*dfree)*gamma(dfree/2))) +
+     log(par[2]) + (dfree+1)/2*log(1+((data-par[1])/par[2])^2/dfree
+ }  # end likefun
```

The *likelihood* function measures how *likely* are the parameters, given the observed values $\bar{x}$.

The *maximum-likelihood* estimate (*MLE*) of the parameters are those that maximize the *likelihood* function:

$$\theta_{MLE} = \underset{\theta}{\arg\max}\, \mathcal{L}(\theta|x)$$

In practice the logarithm of the *likelihood* $\log(\mathcal{L})$ is maximized, instead of the *likelihood* itself.

Fitting Asset Returns into Student's *t-distribution*

The function fitdistr() from package *MASS* fits a univariate distribution to a sample of data, by performing *maximum likelihood* optimization.

The function fitdistr() performs a *maximum likelihood* optimization to find the non-standardized Student's *t-distribution* location and scale parameters.

```
> # Calculate VTI percentage returns
> retp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))
> # Fit VTI returns using MASS::fitdistr()
> fitobj <- MASS::fitdistr(retp, densfun="t", df=3)
> summary(fitobj)
> # Fitted parameters
> fitobj$estimate
> locv <- fitobj$estimate[1]
> scalev <- fitobj$estimate[2]
> locv; scalev
> # Standard errors of parameters
> fitobj$sd
> # Log-likelihood value
> fitobj$value
> # Fit distribution using optim()
> initp <- c(mean=0, scale=0.01)  # Initial parameters
> fitobj <- optim(par=initp,
+   fn=likefun, # Log-likelihood function
+   data=retp,
+   dfree=3, # Degrees of freedom
+   method="L-BFGS-B", # Quasi-Newton method
+   upper=c(1, 0.1), # Upper constraint
+   lower=c(-1, 1e-7)) # Lower constraint
> # Optimal parameters
> locv <- fitobj$par["mean"]
> scalev <- fitobj$par["scale"]
> locv; scalev
```

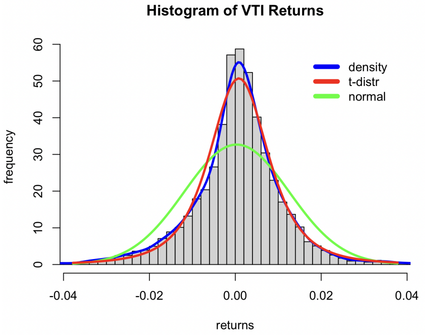
The Student's *t-distribution* Fitted to Asset Returns

Asset returns typically exhibit *negative skewness* and *large kurtosis* (leptokurtosis), or fat tails.

Stock returns fit the non-standard *t-distribution* with 3 degrees of freedom quite well.

The function `hist()` calculates and plots a histogram, and returns its data *invisibly*.

The parameter `breaks` is the number of cells of the histogram.




```
> dev.new(width=6, height=5, noRStudioGD=TRUE)
> # x11(width=6, height=5)
> # Plot histogram of VTI returns
> madv <- mad(retp)
> histp <- hist(retp, col="lightgrey",
+   xlab="returns", breaks=100, xlim=c(-5*madv, 5*madv),
+   ylab="frequency", freq=FALSE, main="Histogram of VTI Returns")
> lines(density(retp, adjust=1.5), lwd=3, col="blue")
> # Plot the Normal probability distribution
> curve(expr=dnorm(x, mean=mean(retp),
+   sd=sd(retp)), add=TRUE, lwd=3, col="green")
> # Define non-standard t-distribution
> tdistr <- function(x, dfree, locv=0, scalev=1) {
+   dt((x-locv)/scalev, df=dfree)/scalev
+ }  # end tdistr
> # Plot t-distribution function
> curve(expr=tdistr(x, dfree=3, locv=locv, scalev=scalev), col="red", lwd=3, add=TRUE)
> # Add legend
> legend("topright", inset=0.05, bty="n", y.intersp=0.4,
+   leg=c("density", "t-distr", "normal"),
+   lwd=6, lty=1, col=c("blue", "red", "green"))
```

Goodness of Fit of Student's *t-distribution* Fitted to Asset Returns

The *Q-Q* plot illustrates the relative distributions of two samples of data.

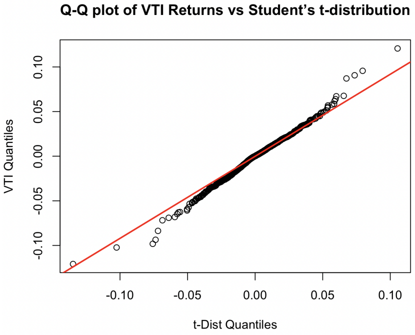
The *Q-Q* plot shows that stock returns fit the non-standard *t-distribution* with 3 degrees of freedom quite well.

The function qqplot() produces a *Q-Q* plot for two samples of data.

The function ks.test() performs the *Kolmogorov-Smirnov* test for the similarity of two distributions.

The *null hypothesis* of the *Kolmogorov-Smirnov* test is that the two samples were obtained from the same probability distribution.

The *Kolmogorov-Smirnov* test rejects the *null hypothesis* that stock returns follow closely the non-standard *t-distribution* with 3 degrees of freedom.




```
> # Calculate sample from non-standard t-distribution with df=3
> tdata <- scalev*rt(NROW(retp), df=3) + locv
> # Q-Q plot of VTI Returns vs non-standard t-distribution
> qqplot(tdata, retp, xlab="t-Dist Quantiles", ylab="VTI Quantiles"
+        main="Q-Q plot of VTI Returns vs Student's t-distribution"
> # Calculate quartiles of the distributions
> probs <- c(0.25, 0.75)
> qrets <- quantile(retp, probs)
> qtdata <- quantile(tdata, probs)
> # Calculate slope and plot line connecting quartiles
> slope <- diff(qrets)/diff(qtdata)
> intercept <- qrets[1]-slope*qtdata[1]
> abline(intercept, slope, lwd=2, col="red")
```

```
> # KS test for VTI returns vs t-distribution data
> ks.test(retp, tdata)
> # Define cumulative distribution of non-standard t-distribution
> ptdistr <- function(x, dfree, locv=0, scalev=1) {
+   pt((x-locv)/scalev, df=dfree)
+ }  # end ptdistr
> # KS test for VTI returns vs cumulative t-distribution
> ks.test(sample(retp, replace=TRUE), ptdistr, dfree=3, locv=locv, s
```

Leptokurtosis Fat Tails of Asset Returns

The probability under the *normal* distribution decreases exponentially for large values of x:

$$\phi(x) \propto e^{-x^2/2\sigma^2} \qquad (as\ |x| \to \infty)$$

This is because a normal variable can be thought of as the sum of a large number of independent binomial variables of equal size.

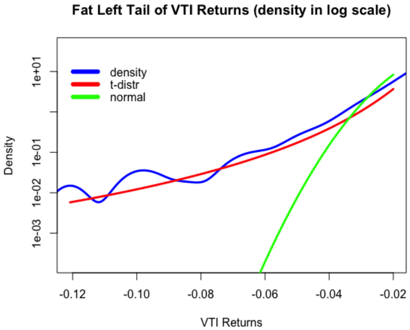
So large values are produced only when all the contributing binomial variables are of the same sign, which is very improbable, so it produces extremely low tail probabilities (thin tails),

But in reality, the probability of large negative asset returns decreases much slower, as the negative power of the returns (fat tails).

The probability under Student's *t-distribution* decreases as a power for large values of x:

$$f(x) \propto |x|^{-(\nu+1)} \qquad (as\ |x| \to \infty)$$

This is because a *t-variable* can be thought of as the sum of normal variables with different volatilities (different sizes).



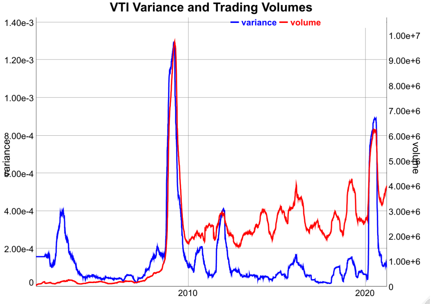

```
> # Plot log density of VTI returns
> plot(density(retp, adjust=4), xlab="VTI Returns", ylab="Density",
+      main="Fat Left Tail of VTI Returns (density in log scale)",
+      type="l", lwd=3, col="blue", xlim=c(min(retp), -0.02), log="y
> # Plot t-distribution function
> curve(expr=dt((x-locv)/scalev, df=3)/scalev, lwd=3, col="red", add
> # Plot the Normal probability distribution
> curve(expr=dnorm(x, mean=mean(retp), sd=sd(retp)), lwd=3, col="gre
> # Add legend
> legend("topleft", inset=0.01, bty="n", y.intersp=c(0.25, 0.25, 0.2
+   legend=c("density", "t-distr", "normal"), y.intersp=0.4,
+   lwd=6, lty=1, col=c("blue", "red", "green"))
```

Trading Volumes

The average trading volumes have increased significantly since the 2008 crisis, mostly because of high frequency trading (HFT).

Higher levels of volatility coincide with higher *trading volumes*.

The time-dependent volatility of asset returns (*heteroskedasticity*) produces their fat tails (*leptokurtosis*).




```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> closep <- drop(coredata(quantmod::Cl(ohlc)))
> retp <- rutils::diffit(log(closep))
> volumv <- coredata(quantmod::Vo(ohlc))
> # Calculate trailing variance
> lookb <- 121
> varv <- HighFreq::roll_var_ohlc(log(ohlc), method="close", lookb=lookb, scale=FALSE)
> varv[1:lookb, ] <- varv[lookb+1, ]
> # Calculate trailing average volume
> volumr <- HighFreq::roll_sum(volumv, lookb=lookb)/lookb
> # dygraph plot of VTI variance and trading volumes
> datav <- xts::xts(cbind(varv, volumr), zoo::index(ohlc))
> colnamev <- c("variance", "volume")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav, main="VTI Variance and Trading Volumes") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], strokeWidth=2, axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, axis="y2", col="red") %>%
+   dyLegend(show="always", width=500)
```

Asset Returns in Trading Time

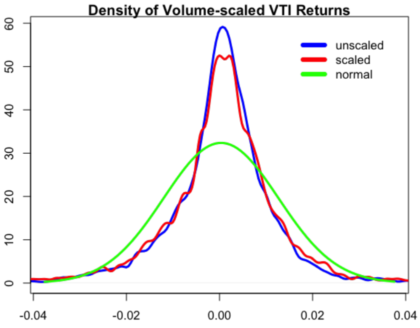
The time-dependent volatility of asset returns (*heteroskedasticity*) produces their fat tails (*leptokurtosis*).

If asset returns were measured at fixed intervals of *trading volumes* (*trading time* instead of clock time), then the volatility would be lower and less time-dependent.

The asset returns can be adjusted to *trading time* by dividing them by the *square root of the trading volumes*, to obtain scaled returns over equal trading volumes.

The scaled returns have a more positive *skewness* and a smaller *kurtosis* than unscaled returns.

```
> # Scale the returns using volume clock to trading time
> retsc <- ifelse(volumv > 0, sqrt(volumr)*retp/sqrt(volumv), 0)
> retsc <- sd(retp)*retsc/sd(retsc)
> # retsc <- ifelse(volumv > 1e4, retp/volumv, 0)
> # Calculate moments of scaled returns
> nrows <- NROW(retp)
> sapply(list(retp=retp, retsc=retsc),
+   function(rets) {sapply(c(skew=3, kurt=4),
+      function(x) sum((rets/sd(rets))^x)/nrows)
+ })  # end sapply
```




```
> # x11(width=6, height=5)
> dev.new(width=6, height=5, noRStudioGD=TRUE)
> par(mar=c(3, 3, 2, 1), oma=c(1, 1, 1, 1))
> # Plot densities of SPY returns
> madv <- mad(retp)
> # bwidth <- mad(rutils::diffit(retp))
> plot(density(retp, bw=madv/10), xlim=c(-5*madv, 5*madv),
+      lwd=3, mgp=c(2, 1, 0), col="blue",
+      xlab="returns (standardized)", ylab="frequency",
+      main="Density of Volume-scaled VTI Returns")
> lines(density(retsc, bw=madv/10), lwd=3, col="red")
> curve(expr=dnorm(x, mean=mean(retp), sd=sd(retp)),
+ add=TRUE, lwd=3, col="green")
> # Add legend
> legend("topright", inset=0.05, bty="n", y.intersp=0.4,
```

The Median Absolute Deviation Estimator of Dispersion

The *Median Absolute Deviation* (*MAD*) is a nonparametric measure of dispersion (variability), defined using the median instead of the mean:

$$\mathrm{MAD} = \mathrm{median}(\mathrm{abs}(x_i - \mathrm{median}(\mathbf{x})))$$

The advantage of *MAD* is that it's always well defined, even for data that has infinite variance.

The *MAD* for normally distributed data is equal to $\Phi^{-1}(0.75) \cdot \hat{\sigma} = 0.6745 \cdot \hat{\sigma}$.

The function `mad()` calculates the *MAD* and divides it by $\Phi^{-1}(0.75)$ to make it comparable to the standard deviation.

For normally distributed data the *MAD* has a larger standard error than the standard deviation.

```
> # Simulate normally distributed data
> nrows <- 1000
> datav <- rnorm(nrows)
> sd(datav)
> mad(datav)
> median(abs(datav - median(datav)))
> median(abs(datav - median(datav)))/qnorm(0.75)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:10000, function(x) {
+   samplev <- datav[sample.int(nrows, replace=TRUE)]
+   c(sd=sd(samplev), mad=mad(samplev))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped variance
> head(bootd)
> sum(is.na(bootd))
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # Initialize compute cluster
> bootd <- parLapply(compclust, 1:10000,
+   function(x, datav) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, datav=datav)  # end parLapply
> # Parallel bootstrap under Mac-OSX or Linux
> bootd <- mclapply(1:10000, function(x) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> stopCluster(compclust)  # Stop R processes over cluster
> bootd <- rutils::do_call(rbind, bootd)
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

The Median Absolute Deviation of Asset Returns

For normally distributed data the *MAD* has a larger standard error than the standard deviation.

But for distributions with fat tails (like asset returns), the standard deviation has a larger standard error than the *MAD*.

The *bootstrap* procedure performs a loop, which naturally lends itself to parallel computing.

The function `makeCluster()` starts running `R` processes on several CPU cores under *Windows*.

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

The `R` processes started by `makeCluster()` don't inherit any data from the parent `R` process.

Therefore the required data must be either passed into `parLapply()` via the dots `"..."` argument, or by calling the function `clusterExport()`.

The function `mclapply()` performs loops using parallel computing on several CPU cores under *Mac-OSX* or *Linux*.

The function `stopCluster()` stops the `R` processes running on several CPU cores.

```
> # Calculate VTI returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> sd(retp)
> mad(retp)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:10000, function(x) {
+   samplev <- retp[sample.int(nrows, replace=TRUE)]
+   c(sd=sd(samplev), mad=mad(samplev))
+ })  # end sapply
> bootd <- t(bootd)
> # Means and standard errors from bootstrap
> 100*apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # Initialize compute cluster
> clusterExport(compclust, c("nrows", "returns"))
> bootd <- parLapply(compclust, 1:10000,
+   function(x) {
+     samplev <- retp[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   })  # end parLapply
> # Parallel bootstrap under Mac-OSX or Linux
> bootd <- mclapply(1:10000, function(x) {
+     samplev <- retp[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> stopCluster(compclust)  # Stop R processes over cluster
> bootd <- rutils::do_call(rbind, bootd)
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

The Downside Deviation of Asset Returns

Some investors argue that positive returns don't represent risk, only those returns less than the target rate of return r_t.

The *Downside Deviation* (semi-deviation) σ_d is equal to the standard deviation of returns less than the target rate of return r_t:

$$\sigma_d = \sqrt{\frac{1}{n} \sum_{i=1}^{n} ([r_i - r_t]_-)^2}$$

The function `DownsideDeviation()` from package *PerformanceAnalytics* calculates the downside deviation, for either the full time series (`method="full"`) or only for the subseries less than the target rate of return r_t (`method="subset"`).

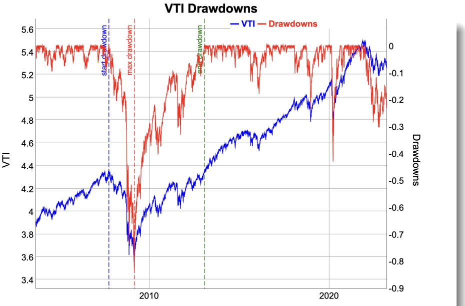
```
> library(PerformanceAnalytics)
> # Define target rate of return of 50 bps
> targetr <- 0.005
> # Calculate the full downside returns
> retsub <- (retp - targetr)
> retsub <- ifelse(retsub < 0, retsub, 0)
> nrows <- NROW(retsub)
> # Calculate the downside deviation
> all.equal(sqrt(sum(retsub^2)/nrows),
+   drop(DownsideDeviation(retp, MAR=targetr, method="full")))
> # Calculate the subset downside returns
> retsub <- (retp - targetr)
> retsub <- retsub[retsub < 0]
> nrows <- NROW(retsub)
> # Calculate the downside deviation
> all.equal(sqrt(sum(retsub^2)/nrows),
+   drop(DownsideDeviation(retp, MAR=targetr, method="subset")))
```

Drawdown Risk

A *drawdown* is the drop in prices from their historical peak, and is equal to the difference between the prices minus the cumulative maximum of the prices.

Drawdown risk determines the risk of liquidation due to stop loss limits.

```
> # Calculate time series of VTI drawdowns
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> drawdns <- (closep - cummax(closep))
> # Extract the date index from the time series closep
> datev <- zoo::index(closep)
> # Calculate the maximum drawdown date and depth
> indexmin <- which.min(drawdns)
> datemin <- datev[indexmin]
> maxdd <- drawdns[datemin]
> # Calculate the drawdown start and end dates
> startd <- max(datev[(datev < datemin) & (drawdns == 0)])
> endd <- min(datev[(datev > datemin) & (drawdns == 0)])
> # dygraph plot of VTI drawdowns
> datav <- cbind(closep, drawdns)
> colnamev <- c("VTI", "Drawdowns")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav, main="VTI Drawdowns") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2],
+    valueRange=(1.2*range(drawdns)+0.1), independentTicks=TRUE) %
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red") %>%
+   dyEvent(startd, "start drawdown", col="blue") %>%
+   dyEvent(datemin, "max drawdown", col="red") %>%
+   dyEvent(endd, "end drawdown", col="green")
```




```
> # Plot VTI drawdowns using package quantmod
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("blue")
> x11(width=6, height=5)
> quantmod::chart_Series(x=closep, name="VTI Drawdowns", theme=plot_
> xval <- match(startd, datev)
> yval <- max(closep)
> abline(v=xval, col="blue")
> text(x=xval, y=0.95*yval, "start drawdown", col="blue", cex=0.9)
> xval <- match(datemin, datev)
> abline(v=xval, col="red")
> text(x=xval, y=0.9*yval, "max drawdown", col="red", cex=0.9)
> xval <- match(endd, datev)
> abline(v=xval, col="green")
> text(x=xval, y=0.85*yval, "end drawdown", col="green", cex=0.9)
```

Drawdown Risk Using PerformanceAnalytics::table.Drawdowns()

The function table.Drawdowns() from package *PerformanceAnalytics* calculates a data frame of drawdowns.

```
> library(xtable)
> library(PerformanceAnalytics)
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> retp <- rutils::diffit(closep)
> # Calculate table of VTI drawdowns
> tablev <- PerformanceAnalytics::table.Drawdowns(retp, geometric=FALSE)
> # Convert dates to strings
> tablev <- cbind(sapply(tablev[, 1:3], as.character), tablev[, 4:7])
> # Print table of VTI drawdowns
> print(xtable(tablev), comment=FALSE, size="tiny", include.rownames=FALSE)
```

From	Trough	To	Depth	Length	To Trough	Recovery
2007-10-10	2009-03-09	2012-03-13	-0.57	1115.00	355.00	760.00
2001-06-06	2002-10-09	2004-11-04	-0.45	858.00	336.00	522.00
2020-02-20	2020-03-23	2020-08-12	-0.18	122.00	23.00	99.00
2022-01-04	2022-10-12		-0.10	473.00	195.00	
2018-09-21	2018-12-24	2019-04-23	-0.10	146.00	65.00	81.00

The Loss Distribution of Asset Returns

The distribution of returns has a long left tail of negative returns representing the risk of loss.

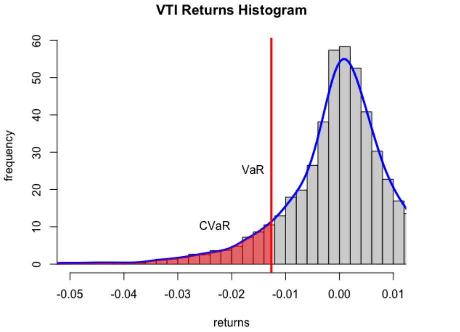
The *Value at Risk* (VaR) is equal to the quantile of returns corresponding to a given confidence level α.

The *Conditional Value at Risk* (CVaR) is equal to the average of negative returns less than the VaR.

The function `hist()` calculates and plots a histogram, and returns its data *invisibly*.

The function `density()` calculates a kernel estimate of the probability density for a sample of data.

```
> # Calculate VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> confl <- 0.1
> varisk <- quantile(retp, confl)
> cvar <- mean(retp[retp <= varisk])
> # Plot histogram of VTI returns
> x11(width=6, height=5)
> par(mar=c(3, 2, 1, 0), oma=c(0, 0, 0, 0))
> histp <- hist(retp, col="lightgrey",
+   xlab="returns", ylab="frequency", breaks=100,
+   xlim=c(-0.05, 0.01), freq=FALSE, main="VTI Returns Histogram")
> # Calculate density
> densv <- density(retp, adjust=1.5)
```




```
> # Plot density
> lines(densv, lwd=3, col="blue")
> # Plot line for VaR
> abline(v=varisk, col="red", lwd=3)
> text(x=varisk, y=25, labels="VaR", lwd=2, pos=2)
> # Plot polygon shading for CVaR
> text(x=1.5*varisk, y=10, labels="CVaR", lwd=2, pos=2)
> varmax <- -0.06
> rangev <- (densv$x < varisk) &  (densv$x > varmax)
> polygon(c(varmax, densv$x[rangev], varisk),
+   c(0, densv$y[rangev], 0), col=rgb(1, 0, 0,0.5), border=NA)
```

Value at Risk (*VaR*)

The *Value at Risk* (VaR) is equal to the quantile of returns corresponding to a given confidence level α:

$$\alpha = \int_{-\infty}^{\mathrm{VaR}(\alpha)} \mathsf{f}(r) \, \mathrm{d}r$$

Where $\mathsf{f}(r)$ is the probability density (distribution) of returns.

At a high confidence level, the value of VaR is subject to estimation error, and various numerical methods are used to approximate it.

The function `quantile()` calculates the sample quantiles. It uses interpolation to improve the accuracy. Information about the different interpolation methods can be found by typing `?quantile`.

A simpler but less accurate way of calculating the quantile is by sorting and selecting the data closest to the quantile.

The function `VaR()` from package *PerformanceAnalytics* calculates the *Value at Risk* using several different methods.

```
> # Calculate VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> confl <- 0.05
> # Calculate VaR approximately by sorting
> sortv <- sort(as.numeric(retp))
> cutoff <- round(confl*nrows)
> varisk <- sortv[cutoff]
> # Calculate VaR as quantile
> varisk <- quantile(retp, probs=confl)
> # PerformanceAnalytics VaR
> PerformanceAnalytics::VaR(retp, p=(1-confl), method="historical")
> all.equal(unname(varisk),
+   as.numeric(PerformanceAnalytics::VaR(retp,
+   p=(1-confl), method="historical")))
```

Conditional Value at Risk (*CVaR*)

The *Conditional Value at Risk* (CVaR) is equal to the average of negative returns less than the VaR:

$$\mathrm{CVaR} = \frac{1}{\alpha} \int_0^{\alpha} \mathrm{VaR}(p) \, \mathrm{d}p$$

The *Conditional Value at Risk* is also called the *Expected Shortfall* (*ES*), or the Expected Tail Loss (*ETL*).

The function ETL() from package *PerformanceAnalytics* calculates the *Conditional Value at Risk* using several different methods.

```
> # Calculate VaR as quantile
> varisk <- quantile(retp, confl)
> # Calculate CVaR as expected loss
> cvar <- mean(retp[retp <= varisk])
> # PerformanceAnalytics VaR
> PerformanceAnalytics::ETL(retp, p=(1-confl), method="historical")
> all.equal(unname(cvar),
+   as.numeric(PerformanceAnalytics::ETL(retp,
+     p=(1-confl), method="historical")))
```

Risk and Return Statistics

The function `table.Stats()` from package *PerformanceAnalytics* calculates a data frame of risk and return statistics of the return distributions.

```
> # Calculate the risk-return statistics
> riskstats <-
+   PerformanceAnalytics::table.Stats(rutils::etfenv$returns)
> class(riskstats)
> # Transpose the data frame
> riskstats <- as.data.frame(t(riskstats))
> # Add Name column
> riskstats$Name <- rownames(riskstats)
> # Add Sharpe ratio column
> riskstats$"Arithmetic Mean" <-
+   sapply(rutils::etfenv$returns, mean, na.rm=TRUE)
> riskstats$Sharpe <-
+   sqrt(252)*riskstats$"Arithmetic Mean"/riskstats$Stdev
> # Sort on Sharpe ratio
> riskstats <- riskstats[order(riskstats$Sharpe, decreasing=TRUE), ]
```

	Sharpe	Skewness	Kurtosis
USMV	0.779	-0.857	21.21
QUAL	0.650	-0.509	12.74
MTUM	0.593	-0.675	11.85
IEF	0.472	0.056	2.59
VLUE	0.444	-0.950	17.06
XLV	0.435	0.071	10.06
GLD	0.425	-0.306	6.16
VTV	0.407	-0.659	13.69
VTI	0.406	-0.375	10.65
XLP	0.392	-0.120	8.67
VYM	0.386	-0.672	14.48
XLY	0.382	-0.356	6.56
XLI	0.366	-0.375	7.48
IWB	0.354	-0.385	9.91
IWD	0.336	-0.483	12.54
IVW	0.335	-0.296	8.33
XLU	0.330	0.001	11.82
IVE	0.325	-0.475	10.01
QQQ	0.321	-0.025	6.38
XLB	0.321	-0.366	5.28
XLK	0.313	0.074	6.60
IWF	0.304	-0.650	30.46
EEM	0.283	0.025	15.51
XLE	0.263	-0.529	12.41
TLT	0.260	-0.012	3.59
AIEQ	0.231	-0.701	6.95
VNQ	0.227	-0.531	17.90
SVXY	0.166	-18.142	656.67
XLF	0.153	-0.121	14.04
VEU	0.134	-0.501	11.60
DBC	0.026	-0.493	3.28
USO	-0.308	-1.139	14.12
VXX	-1.191	1.109	5.08

Investor Risk and Return Preferences

Investors typically prefer larger *odd moments* of the return distribution (mean, skewness), and smaller *even moments* (variance, kurtosis).

But positive skewness is often associated with lower returns, which can be observed in the *VIX* volatility ETFs, *VXX* and *SVXY*.

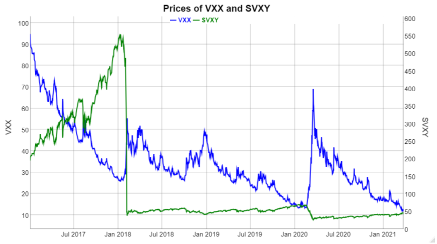
The *VXX* ETF is long the *VIX* index (effectively long an option), so it has positive skewness and small kurtosis, but negative returns (it's short market risk).

Since the *VXX* is effectively long an option, it pays option premiums so it has negative returns most of the time, with isolated periods of positive returns when markets drop.

The *SVXY* ETF is short the *VIX* index, so it has negative skewness and large kurtosis, but positive returns (it's long market risk).

Since the *SVXY* is effectively short an option, it earns option premiums so it has positive returns most of the time, but it suffers sharp losses when markets drop.

	Sharpe	Skewness	Kurtosis
VXX	-1.191	1.11	5.08
SVXY	0.166	-18.14	656.67




```
> # dygraph plot of VXX versus SVXY
> pricev <- na.omit(rutils::etfenv$prices[, c("VXX", "SVXY")])
> pricev <- pricev["2017/"]
> colnamev <- c("VXX", "SVXY")
> colnames(pricev) <- colnamev
> dygraphs::dygraph(pricev, main="Prices of VXX and SVXY") %>%
+   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="green'
+   dyLegend(show="always", width=300) %>% dyLegend(show="always", w
+   dyLegend(show="always", width=300)
```

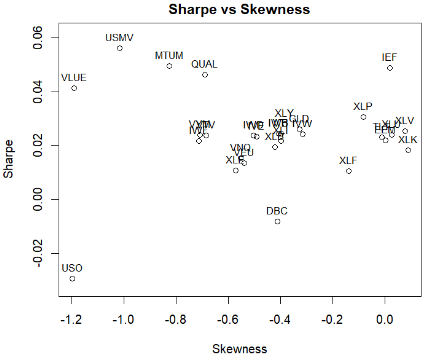
Skewness and Return Tradeoff

Similarly to the *VXX* and *SVXY*, for most other ETFs positive skewness is often associated with lower returns.

Some of the exceptions are bond ETFs (like *IEF*), which have both non-negative skewness and positive returns.

Another exception are commodity ETFs (like *USO* oil), which have both negative skewness and negative returns.

```
> # Remove VIX volatility ETF data
> riskstats <- riskstats[-match(c("VXX", "SVXY"), riskstats$Name),
> # Plot scatterplot of Sharpe vs Skewness
> plot(Sharpe ~ Skewness, data=riskstats,
+      ylim=1.1*range(riskstats$Sharpe),
+      main="Sharpe vs Skewness")
> # Add labels
> text(x=riskstats$Skewness, y=riskstats$Sharpe,
+     labels=riskstats$Name, pos=3, cex=0.8)
> # Plot scatterplot of Kurtosis vs Skewness
> x11(width=6, height=5)
> par(mar=c(4, 4, 2, 1), oma=c(0, 0, 0, 0))
> plot(Kurtosis ~ Skewness, data=riskstats,
+      ylim=c(1, max(riskstats$Kurtosis)),
+      main="Kurtosis vs Skewness")
> # Add labels
> text(x=riskstats$Skewness, y=riskstats$Kurtosis,
+     labels=riskstats$Name, pos=1, cex=0.5)
```



Risk-adjusted Return Measures

The *Sharpe ratio* $\mathrm{S_r}$ is equal to the excess returns (in excess of the risk-free rate r_f) divided by the standard deviation σ of the returns:

$$\mathrm{S_r} = \frac{E[r - r_f]}{\sigma}$$

The *Sortino ratio* $\mathrm{So_r}$ is equal to the excess returns divided by the *downside deviation* σ_d (standard deviation of returns that are less than a target rate of return r_t):

$$\mathrm{So_r} = \frac{E[r - r_t]}{\sigma_d}$$

The *Calmar ratio* $\mathrm{C_r}$ is equal to the excess returns divided by the *maximum drawdown* DD of the returns:

$$\mathrm{C_r} = \frac{E[r - r_f]}{\mathrm{DD}}$$

The *Dowd ratio* $\mathrm{D_r}$ is equal to the excess returns divided by the *Value at Risk* (VaR) of the returns:

$$\mathrm{D_r} = \frac{E[r - r_f]}{\mathrm{VaR}}$$

The *Conditional Dowd ratio* $\mathrm{Dc_r}$ is equal to the excess returns divided by the *Conditional Value at Risk* (CVaR) of the returns:

$$\mathrm{Dc_r} = \frac{E[r - r_f]}{\mathrm{CVaR}}$$

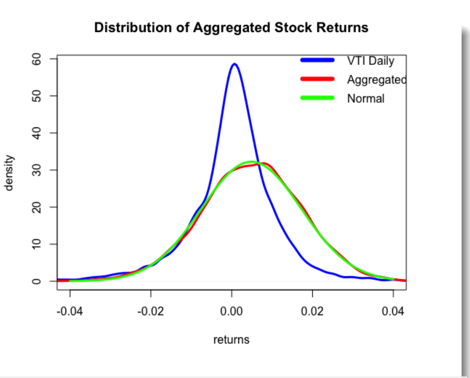
```
> library(PerformanceAnalytics)
> retp <- rutils::etfenv$returns[, c("VTI", "IEF")]
> retp <- na.omit(retp)
> # Calculate the Sharpe ratio
> confl <- 0.05
> PerformanceAnalytics::SharpeRatio(retp, p=(1-confl),
+   method="historical")
> # Calculate the Sortino ratio
> PerformanceAnalytics::SortinoRatio(retp)
> # Calculate the Calmar ratio
> PerformanceAnalytics::CalmarRatio(retp)
> # Calculate the Dowd ratio
> PerformanceAnalytics::SharpeRatio(retp, FUN="VaR",
+   p=(1-confl), method="historical")
> # Calculate the Dowd ratio from scratch
> varisk <- sapply(retp, quantile, probs=confl)
> -sapply(retp, mean)/varisk
> # Calculate the Conditional Dowd ratio
> PerformanceAnalytics::SharpeRatio(retp, FUN="ES",
+   p=(1-confl), method="historical")
> # Calculate the Conditional Dowd ratio from scratch
> cvar <- sapply(retp, function(x) {
+   mean(x[x < quantile(x, confl)])
+ })
> -sapply(retp, mean)/cvar
```

Risk of Aggregated Stock Returns

Stock returns aggregated over longer holding periods are closer to normally distributed, and their skewness, kurtosis, and tail risks are significantly lower than for daily returns.

Stocks become less risky over longer holding periods, so investors may choose to own a higher percentage of stocks, provided they hold them for a longer period of time.

```
> # Calculate VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> # Bootstrap aggregated monthly VTI returns
> holdp <- 22
> reta <- sqrt(holdp)*sapply(1:nrows, function(x) {
+     mean(retp[sample.int(nrows, size=holdp, replace=TRUE)])
+ })  # end sapply
> # Calculate mean, standard deviation, skewness, and kurtosis
> datav <- cbind(retp, reta)
> colnames(datav) <- c("VTI", "Agg")
> sapply(datav, function(x) {
+   # Standardize the returns
+   meanv <- mean(x); stdev <- sd(x); x <- (x - meanv)/stdev
+   c(mean=meanv, stdev=stdev, skew=mean(x^3), kurt=mean(x^4))
+ })  # end sapply
> # Calculate the Sharpe and Dowd ratios
> confl <- 0.02
> ratiom <- sapply(datav, function(x) {
+   stdev <- sd(x)
+   varisk <- unname(quantile(x, probs=confl))
+   cvar <- mean(x[x < varisk])
+   mean(x)/c(Sharpe=stdev, Dowd=-varisk, DowdC=-cvar)
+ })  # end sapply
> # Annualize the daily risk
> ratiom[, 1] <- sqrt(22)*ratiom[, 1]
```




```
> # Plot the densities of returns
> plot(density(retp), t="l", lwd=3, col="blue",
+      xlab="returns", ylab="density", xlim=c(-0.04, 0.04),
+      main="Distribution of Aggregated Stock Returns")
> lines(density(reta), t="l", col="red", lwd=3)
> curve(expr=dnorm(x, mean=mean(reta), sd=sd(reta)), col="green", lw
> legend("topright", legend=c("VTI Daily", "Aggregated", "Normal"),
+  inset=-0.1, bg="white", lty=1, lwd=6, col=c("blue", "red", "greer
```

Tests for Market Timing Skill

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

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

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

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

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

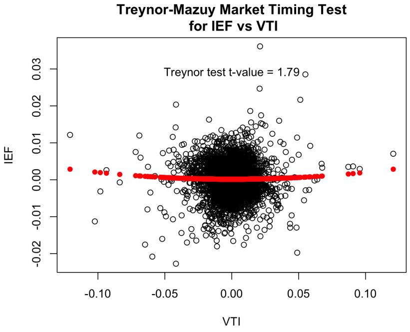
If the coefficient γ is statistically significant, then it's very likely due to *market timing* skill.

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

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

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

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




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

Calculating Asset Returns

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

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

The initial returns are all equal to zero.

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

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

Compounding Asset Returns

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

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

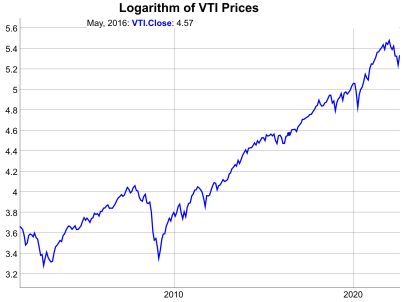
The sum of the percentage returns (without compounding): $\sum_{i=1}^{n} r_i^p$ represents the wealth path from owning a *fixed dollar amount* of stock.

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

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

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

```
> # Set the initial dollar returns
> retd[1, ] <- pricev[1, ]
> # Calculate the prices from dollar returns
> pricen <- cumsum(retd)
> all.equal(pricen, pricev)
> # Compound the percentage returns
> pricen <- cumprod(1 + retp)
> # Set the initial prices
> pricesi <- as.numeric(pricev[1, ])
> pricen <- lapply(1:NCOL(pricen), function (i) pricesi[i]*pricen[, i])
> pricen <- rutils::do_call(cbind, pricen)
> # pricen <- t(t(pricen)*pricesi)
> all.equal(pricen, pricev, check.attributes=FALSE)
```




```
> # Plot log VTI prices
> endd <- rutils::calc_endpoints(rutils::etfenv$VTI, interval="weeks
> dygraphs::dygraph(log(quantmod::Cl(rutils::etfenv$VTI)[endd]),
+   main="Logarithm of VTI Prices") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Funding Costs of Single Asset Rebalancing

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

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

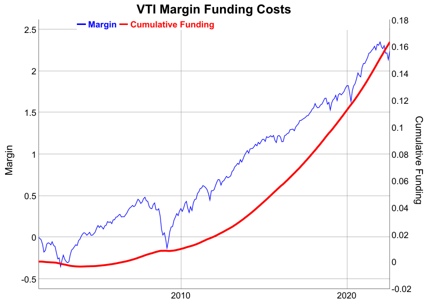
The cash in the *margin account* can be positive (accumulated profits) or negative (losses).

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

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

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

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




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

Transaction Costs of Trading

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

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

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

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

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

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

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

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

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

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

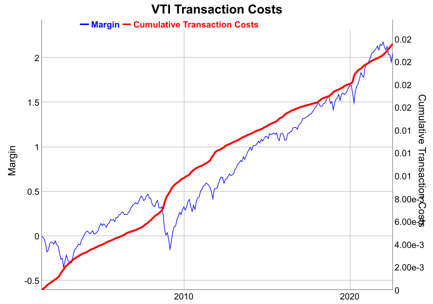
Transaction Costs of Single Asset Rebalancing

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

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

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

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




```
> # bidask equal to 1 bp for liquid ETFs
> bidask <- 0.001
> # Cumulative transaction costs
> costs <- bidask*cumsum(abs(retp))/2
> # Subtract transaction costs from margin account
> marginv <- cumsum(retp)
> marginv <- (marginv - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, costs)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endd], main="VTI Transaction Costs") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=200)
```

Combining the Returns of Multiple Assets

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

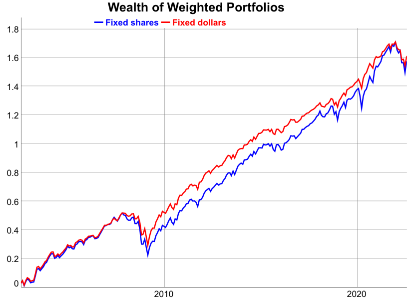
Multiplying the weights times the percentage returns is equivalent to investing in *fixed dollar amounts of stock* proportional to the weights (aka *Fixed Dollar Allocation* or FDA).

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

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

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

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




```
> # Wealth of fixed dollars (with rebalancing)
> wealthfd <- cumsum(retp %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(log(wealthfs), wealthfd)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Fixed dollars")
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> colnamev <- colnames(wealthv)
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(wealthv[endd], main="Wealth of Weighted Portfoli
+   dySeries(name=colnamev[1], col="blue", strokeWidth=2) %>%
+   dySeries(name=colnamev[2], col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Transaction Costs of Weighted Portfolio Rebalancing

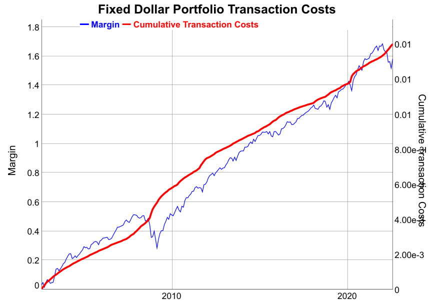
Maintaining a *fixed dollar allocation* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

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

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

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

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




```
> # Margin account for fixed dollars (with rebalancing)
> marginv <- cumsum(retp %*% weightv)
> # Cumulative transaction costs
> costs <- bidask*cumsum(abs(retp) %*% weightv)/2
> # Subtract transaction costs from margin account
> marginv <- (marginv - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, costs)
> datav <- xts::xts(datav, datev)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endd], main="Fixed Dollar Portfolio Transa
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=200)
```

Proportional Dollar Allocations

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

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

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

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

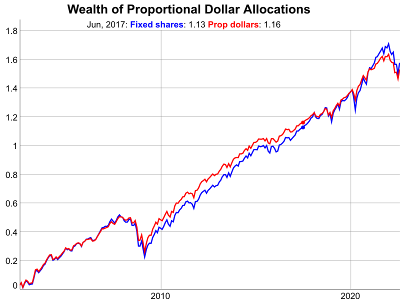
The total portfolio wealth at time t is equal to the wealth at time $t-1$ multiplied by the weighted returns: $w_t = w_{t-1}(1 + \bar{r}_t)$.

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

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

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

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




```r
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- cumsum(retd %*% (weightv/pricesi))
> # Or compound the percentage returns
> wealthfs <- cumprod(1 + retp) %*% weightv
> # Wealth of proportional allocations (with rebalancing)
> wealthpd <- cumprod(1 + retp %*% weightv)
> wealthv <- cbind(wealthfs, wealthpd)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Prop dollars")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Proportional Dollar Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Transaction Costs With Proportional Dollar Allocations

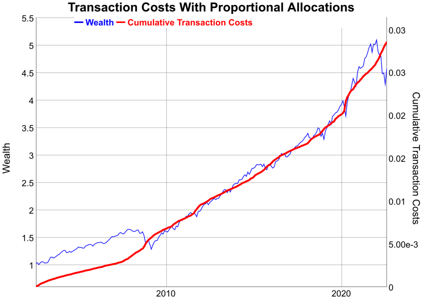
In each period the stocks must be rebalanced to maintain the proportional dollar allocations.

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

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

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

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




```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealthpd, costs)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Wealth", "Cumulative Transaction Costs")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv[endd],
+   main="Transaction Costs With Proportional Allocations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=200)
```

Proportional Target Allocation Strategy

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

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

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

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

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

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

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

```
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- drop(apply(retp, 2, function(x) cumprod(1 + x)) %*% we
> # Wealth of proportional dollar allocations (with rebalancing)
> wealthpd <- cumprod(1 + retp %*% weightv) - 1
> # Wealth of proportional target allocation (with rebalancing)
> retp <- zoo::coredata(retp)
> threshv <- 0.05
> wealthv <- matrix(nrow=NROW(retp), ncol=2)
> colnames(wealthv) <- colnames(retp)
> wealthv[1, ] <- weightv
> for (it in 2:NROW(retp)) {
+   # Accrue wealth without rebalancing
+   wealthv[it, ] <- wealthv[it-1, ]*(1 + retp[it, ])
+   # Rebalance if wealth allocations differ from weights
+   if (sum(abs(wealthv[it, ] - sum(wealthv[it, ])*weightv))/sum(wea
+     # cat("Rebalance at:", it, "\n")
+     wealthv[it, ] <- sum(wealthv[it, ])*weightv
+   } # end if
+ } # end for
> wealthv <- rowSums(wealthv) - 1
> wealthv <- cbind(wealthpd, wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Proportional Allocations", "Proportional T
> dygraphs::dygraph(wealthv, main="Wealth of Proportional Target All
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

draft: Stock Index Weighting Methods

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

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

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

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

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

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

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

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

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

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

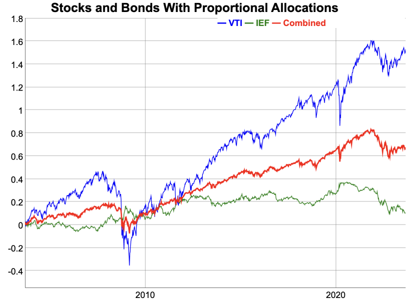
Stock and Bond Portfolio With Proportional Dollar Allocations

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

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

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

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




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

```
> # Wealth of proportional allocations
> wealthv <- cumsum(retp)
> # Calculate the a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> # Plot cumulative log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Stocks and Bonds With Proportional Allocations") %>%
+   dyOptions(colors=c("blue", "green", "blue", "red")) %>%
+   dySeries("Combined", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

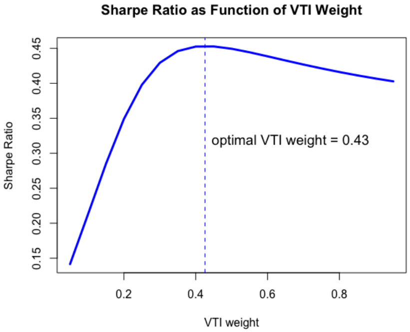
Optimal Stock and Bond Portfolio Allocations

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

Using the past 20 years of data, the optimal *VTI* weight is about 0.43.

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

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




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

Simulating Wealth Scenarios Using Bootstrap

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

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

For sampling from rows of data, it's better to convert time series to matrices.

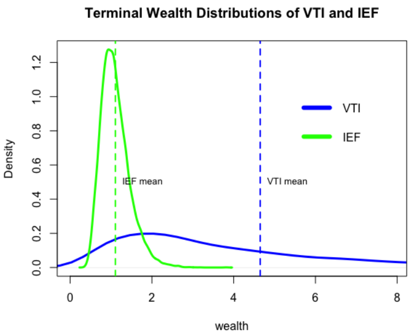
```
> # Coerce the returns from xts time series to matrix
> retp <- zoo::coredata(retp[, 1:2])
> nrows <- NROW(retp)
> # Bootstrap the returns and Calculate the a list of random returns
> nboot <- 1e4
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> # Perform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

The Distributions of Terminal Wealth From Bootstrap

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

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

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



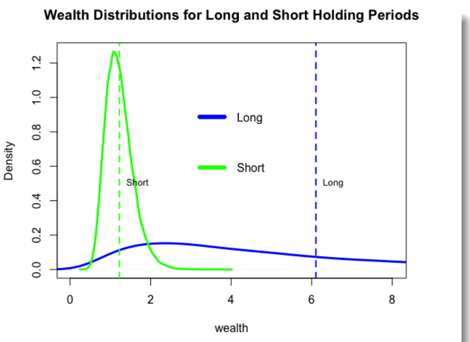

```
> # Plot the densities of the terminal wealths of VTI and IEF
> vtim <- mean(vtiw); iefm <- mean(iefw)
> vtid <- density(vtiw); iefd <- density(iefw)
> plot(vtid, col="blue", lwd=3, xlab="wealth",
+      xlim=c(0, 2*max(iefd$x)), ylim=c(0, max(iefd$y)),
+      main="Terminal Wealth Distributions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

The Distribution of Stock Wealth and Holding Period

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

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

```
> # Calculate the distributions of stock wealth
> holdv <- nrows*seq(0.1, 1.0, 0.1)
> wealthm <- mclapply(bootd, function(retp) {
+   sapply(holdv, function(holdp) {
+     prod(1 + retp[1:holdp, "VTI"])
+   })  # end sapply
+ }, mc.cores=ncores)  # end mclapply
> wealthm <- do.call(rbind, wealthm)
> dim(wealthm)
```




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

Risk-adjusted Stock Wealth and Holding Period

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

The risk-adjusted wealth measure is equal to the mean wealth divided by the downside risk.

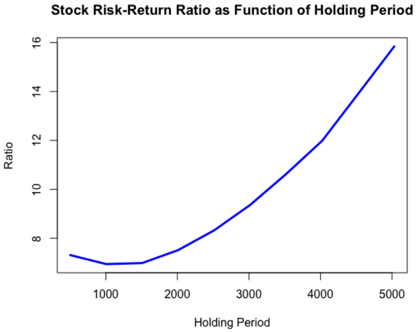
U.S. stocks in the last 40 years have had higher risk-adjusted wealth for longer holding periods.

The risk-adjusted wealth is also higher for very short holding periods because the risk is low - there's not enough time for the wealth to significantly drop below par (1).

The risk increases for intermediate holding periods, so the risk-adjusted wealth drops.

The mean wealth increases for longer holding periods, so the risk-adjusted wealth also increases.

```
> # Define the risk-adjusted wealth measure
> riskretfun <- function(wealthv) {
+   riskv <- 0.01 # Risk floor
+   if (min(wealthv) < 1) # Some wealth is below par
+     # Calculate the mean stock wealth below par
+     riskv <- mean((1-wealthv)[wealthv<1])
+   mean(wealthv)/riskv
+ }  # end riskretfun
> # Calculate the stock wealth risk-return ratios
> riskrets <- apply(wealthm, 2, riskretfun)
```



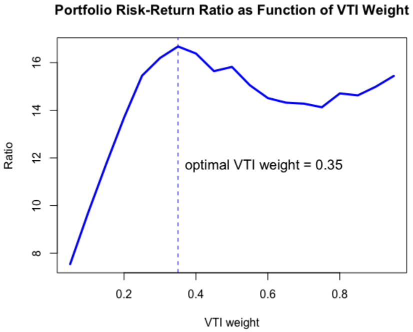

```
> # Plot the stock wealth risk-return ratios
> plot(x=holdv, y=riskrets,
+      main="Stock Risk-Return Ratio as Function of Holding Period",
+      xlab="Holding Period", ylab="Ratio",
+      t="l", lwd=3, col="blue")
```

Optimal Stock and Bond Portfolio Allocations From Bootstrap

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

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

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




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

The *All-Weather* Portfolio

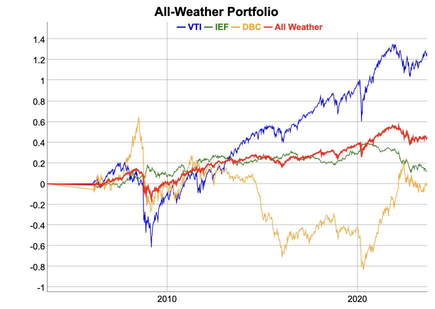
The *All-Weather* portfolio is a portfolio with proportional allocations of stocks (30%), bonds (55%), and commodities and precious metals (15%) (approximately).

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

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

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

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




```
> # Calculate the cumulative wealth from returns
> wealthv <- cumsum(retp)
> # Calculate the a vector of monthly end points
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> # dygraph all-weather wealth
> dygraphs::dygraph(wealthv[endd], main="All-Weather Portfolio") %>%
+   dyOptions(colors=c("blue", "green", "orange", "red")) %>%
+   dySeries("All Weather", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
> # Plot all-weather wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("orange", "blue", "green", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme, lwd=c(2, 2, 2, 4
+        name="All-Weather Portfolio")
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

Constant Proportion Portfolio Insurance Strategy

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

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

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

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

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

CPPI Strategy Dynamics

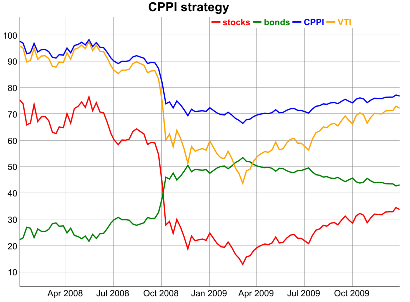
If the stock price changes and the portfolio value becomes P_t, then the dollar amount invested in stocks must be adjusted to: $C * (P_t - F)$. The amount invested in stocks changes both because the stock price changes and because of rebalancing with the zero-coupon bonds.

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

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

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

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




```
> # Simulate CPPI strategy
> for (t in 2:nrows) {
+   portfv[t] <- portfv[t-1] + stockv[t-1]*retp[t]
+   stockv[t] <- min(coeff*(portfv[t] - bfloor), portfv[t])
+   bondv[t] <- (portfv[t] - stockv[t])
+ }  # end for
> # dygraph plot of CPPI strategy
> pricev <- 100*cumprod(1 + retp)
> datav <- xts::xts(cbind(stockv, bondv, portfv, pricev), datev)
> colnames(datav) <- c("stocks", "bonds", "CPPI", "VTI")
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endd], main="CPPI strategy") %>%
+   dyOptions(colors=c("red", "green", "blue", "orange"), strokeWidt
+   dyLegend(show="always", width=200)
```

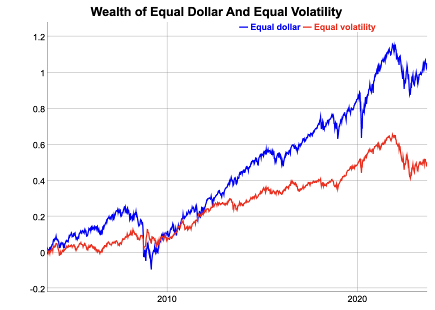
The Standardized Stock Prices

The *standardized price* is the dollar amount of stock with unit dollar volatility. The *standardized price* is equal to the ratio of the stock price divided by its dollar volatility: $\frac{p}{\sigma_d}$

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

The equal volatility portfolio has lower returns because it's overweight bonds.

```r
> # Calculate the dollar returns of VTI and IEF
> pricev <- na.omit(rutils::etfenv$prices[, c("VTI", "IEF")])
> retd <- rutils::diffit(pricev)
> # Scale the stock prices to $1 at beginning
> pricesi <- as.numeric(pricev[1, ]) # Initial stock prices
> pricesc <- pricev
> pricesc$VTI <- pricesc$VTI/pricesi[1]
> pricesc$IEF <- pricesc$IEF/pricesi[2]
> sum(pricesc[1, ])
> retsc <- rutils::diffit(pricesc)
> # Wealth of fixed number of shares (without rebalancing)
> weightv <- c(0.5, 0.5) # Buy $0.5 of each stock
> wealthed <- 1 + cumsum(retsc %*% weightv)
> # Calculate the stock prices with unit dollar volatility
> stdev <- sapply(retd, sd)
> pricesd <- pricev
> pricesd$VTI <- pricev$VTI/stdev["VTI"]
> pricesd$IEF <- pricev$IEF/stdev["IEF"]
> retsd <- rutils::diffit(pricesd)
> sapply(retsd, sd)
```




```r
> # Scale the sum of stock prices at beginning to $2
> pricesd <- 2*pricesd/sum(pricesd[1, ])
> retsd <- rutils::diffit(pricesd)
> sapply(retsd, sd)
> # Wealth of shares with equal dollar volatilities
> wealthev <- 1 + cumsum(retsd %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- xts::xts(cbind(wealthed, wealthev), zoo::index(pricev))
> colnames(wealthv) <- c("Equal dollar", "Equal volatility")
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(log(wealthv[endd]),
+   main="Wealth of Equal Dollar And Equal Volatility") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Risk Parity Strategy For Stocks and Bonds

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

If the volatility changes over time, then the standardized prices with unit dollar volatility also change over time.

So the portfolio allocations must be rebalanced to ensure that their volatilities remain equal.

The risk parity dollar returns are equal to the standardized prices times their percentage returns.

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

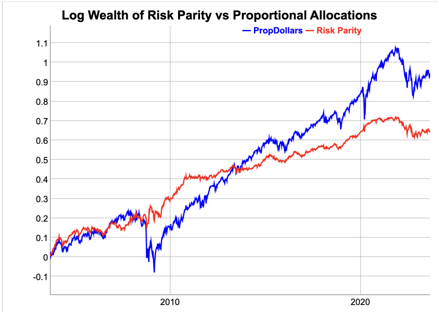
```
> # Calculate the wealth of proportional dollar allocations (with re
> retp <- retd/rutils::lagit(pricev, lagg=1, pad_zeros=FALSE)
> weightv <- c(0.5, 0.5)
> wealthpd <- cumprod(1 + retp %*% weightv)
> # Calculate the trailing dollar volatilities
> volat <- HighFreq::run_var(retd, lambda=0.2)
> volat <- sqrt(volat)
> volat <- rutils::lagit(volat)
> volat[1:2, ] <- 1
> # Calculate the standardized prices with unit dollar volatilities
> pricerp <- pricev/volat
> # Scale the sum of stock prices to $2
> pricerp <- 2*pricerp/rowSums(pricerp)
> # Calculate the risk parity dollar returns
> retrp <- retp*pricerp
> # Calculate the wealth of risk parity
> wealthrp <- 1 + cumsum(retrp %*% weightv)
```

Risk Parity Strategy Performance

The risk parity strategy for stocks and bonds has a higher Sharpe ratio, but lower absolute returns than the proportional dollar strategy.

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

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




```
> # Calculate the log wealths
> wealthv <- cbind(wealthpd, wealthrp)
> wealthv <- xts::xts(wealthv, zoo::index(pricev))
> colnames(wealthv) <- c("PropDollars", "Risk Parity")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot a dygraph of the log wealths
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(log(wealthv[endd]),
+   main="Log Wealth of Risk Parity vs Proportional Allocations") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

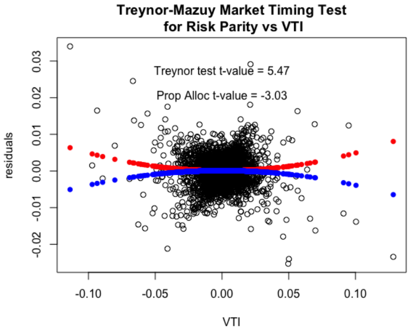
Risk Parity Strategy Market Timing Skill

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

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

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

The t-value of the proportional allocations strategy is negative, because it buys the stock as its prices drop, the opposite of risk parity.



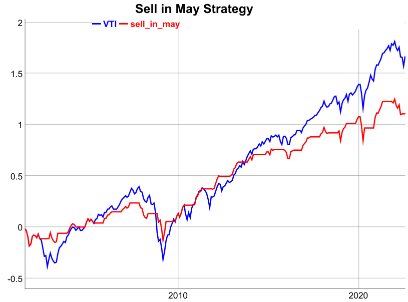

```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> retrp <- rutils::diffit(wealthv)
> retvti <- retp$VTI
> desm <- cbind(retrp, retvti, retvti^2)
> colnames(desm)[1:2] <- c("prop", "riskp")
> colnames(desm)[4] <- "treynor"
> regmod <- lm(riskp ~ VTI + treynor, data=desm)
> summary(regmod)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

```
> # Test for proportional allocations market timing of VTI using Tre
> regmod <- lm(prop ~ VTI + treynor, data=desm)
> summary(regmod)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> points.default(x=retvti, y=fitv, pch=16, col="blue")
> text(x=0.0, y=0.6*max(resids), paste("Prop Alloc t-value =", round
```

Sell in May Calendar Strategy

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

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

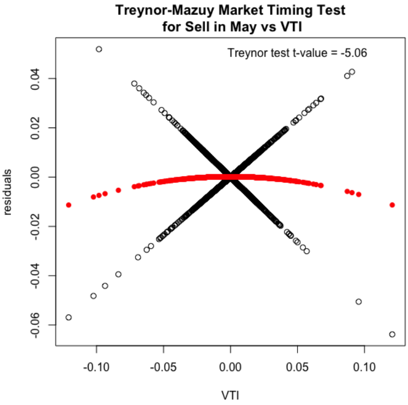



```
> # Plot wealth of Sell in May strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Sell in May Strateg
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # OR: Open x11 for plotting
> x11(width=6, height=5)
> par(mar=c(4, 4, 3, 1), oma=c(0, 0, 0, 0))
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("blue", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme, name="Sell in Ma
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+   col=plot_theme$col$line.col, bty="n")
```

Sell in May Strategy Market Timing

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

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



Overnight Market Anomaly

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

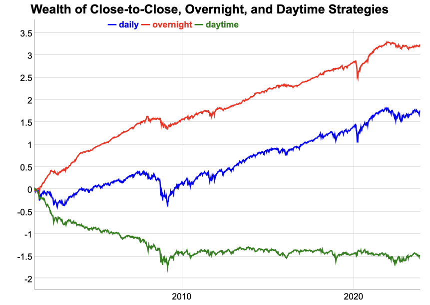
The *Overnight Market Anomaly* has been observed for many decades for most stock market indices, but not always for all stock sectors.

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

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

The *Overnight Market Anomaly* is not as pronounced after the 2008–2009 financial crisis.

```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> # Calculate the close-to-close log returns,
> # the daytime open-to-close returns
> # and the overnight close-to-open returns.
> retp <- rutils::diffit(closep)
> colnames(retp) <- "daily"
> retd <- (closep - openp)
> colnames(retd) <- "daytime"
> 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 the log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Close-to-Close, Overnight, and Daytime Strategie
+   dySeries(name="daily", strokeWidth=2, col="blue") %>%
+   dySeries(name="overnight", strokeWidth=2, col="red") %>%
+   dySeries(name="daytime", strokeWidth=2, col="green") %>%
+   dyLegend(width=500)
```

Turn of the Month Effect

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

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

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

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


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

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

Homework Assignment

Required

- Study all the lecture slides in `FRE7241_Lecture_2.pdf`, and run all the code in `FRE7241_Lecture_2.R`,
- Study *bootstrap simulation* from the files *bootstrap_technique.pdf* and *doBootstrap_primer.pdf*,

Recommended