FRE7241 Algorithmic Portfolio Management

Lecture#2, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

September 13, 2022

NYU | TANDON SCHOOL OF ENGINEERING

The *ETF* Database

Exchange-traded Funds (*ETFs*) are funds which invest in portfolios of assets, such as stocks, commodities, or bonds.

ETFs are shares in portfolios of assets, and they are traded just like stocks.

ETFs provide investors with convenient, low cost, and liquid instruments to invest in various portfolios of assets.

The file `etf_list.csv` contains a database of exchange-traded funds (*ETFs*) and exchange traded notes (*ETNs*).

We will select a portfolio of *ETFs* for illustrating various investment strategies.

```
> # Select ETF symbols for asset allocation
> symbolv <- c("VTI", "VEU", "EEM", "XLY", "XLP", "XLE", "XLF",
+   "XLV", "XLI", "XLB", "XLK", "XLU", "VYM", "IVW", "IWB", "IWD",
+   "IWF", "IEF", "TLT", "VNQ", "DBC", "GLD", "USO", "VXX", "SVXY",
+   "MTUM", "IVE", "VLUE", "QUAL", "VTV", "USMV")
> # Read etf database into data frame
> etflist <- read.csv(file="/Users/jerzy/Develop/lecture_slides/data
> rownames(etflist) <- etflist$Symbol
> # Select from etflist only those ETF's in symbolv
> etflist <- etflist[symbolv, ]
> # Shorten names
> etfnames <- sapply(etflist$Name, function(name) {
+   namesvplit <- strsplit(name, split=" ")[[1]]
+   namesvplit <- namesvplit[c(-1, -NROW(namesvplit))]
+   name_match <- match("Select", namesvplit)
+   if (!is.na(name_match))
+     namesvplit <- namesvplit[-name_match]
+   paste(namesvplit, collapse=" ")
+ })  # end sapply
> etflist$Name <- etfnames
> etflist["IEF", "Name"] <- "10 year Treasury Bond Fund"
> etflist["TLT", "Name"] <- "20 plus year Treasury Bond Fund"
> etflist["XLY", "Name"] <- "Consumer Discr. Sector Fund"
> etflist["EEM", "Name"] <- "Emerging Market Stock Fund"
> etflist["MTUM", "Name"] <- "Momentum Factor Fund"
> etflist["SVXY", "Name"] <- "Short VIX Futures"
> etflist["VXX", "Name"] <- "Long VIX Futures"
> etflist["DBC", "Name"] <- "Commodity Futures Fund"
> etflist["USO", "Name"] <- "WTI Oil Futures Fund"
> etflist["GLD", "Name"] <- "Physical Gold Fund"
```

ETF Portfolio for Investment Strategies

The portfolio contains *ETFs* representing different *industry sectors* and *investment styles*.

The *ETFs* with names *X** represent industry *sector funds* (energy, financial, etc.)

The *ETFs* with names *I** represent *style funds* (value, growth, size).

IWB is the Russell 1000 small-cap fund.

MTUM is an *ETF* which owns a stock portfolio representing the *momentum factor*.

DBC is an *ETF* providing the total return on a portfolio of commodity futures.

VXX is an *ETN* providing the total return of *long VIX* futures contracts (specifically the *S&P* VIX Short-Term Futures Index).

VXX is *bearish* because it's *long* VIX futures, and the VIX *rises* when stock prices *drop*.

SVXY is an *ETF* providing the total return of *short VIX* futures contracts.

SVXY is *bullish* because it's *short* VIX futures, and the VIX *drops* when stock prices *rise*.

Symbol	Name	Fund.Type
VTI	Total Stock Market	US Equity ETF
VEU	FTSE All World Ex US	Global Equity ETF
EEM	Emerging Market Stock Fund	Global Equity ETF
XLY	Consumer Discr. Sector Fund	US Equity ETF
XLP	Consumer Staples Sector Fund	US Equity ETF
XLE	Energy Sector Fund	US Equity ETF
XLF	Financial Sector Fund	US Equity ETF
XLV	Health Care Sector Fund	US Equity ETF
XLI	Industrial Sector Fund	US Equity ETF
XLB	Materials Sector Fund	US Equity ETF
XLK	Technology Sector Fund	US Equity ETF
XLU	Utilities Sector Fund	US Equity ETF
VYM	Large-cap Value	US Equity ETF
IVW	S&P 500 Growth Index Fund	US Equity ETF
IWB	Russell 1000	US Equity ETF
IWD	Russell 1000 Value	US Equity ETF
IWF	Russell 1000 Growth	US Equity ETF
IEF	10 year Treasury Bond Fund	US Fixed Income ETF
TLT	20 plus year Treasury Bond Fund	US Fixed Income ETF
VNQ	REIT ETF - DNQ	US Equity ETF
DBC	Commodity Futures Fund	Commodity Based ETF
GLD	Physical Gold Fund	Commodity Based ETF
USO	WTI Oil Futures Fund	Commodity Based ETF
VXX	Long VIX Futures	Commodity Based ETN
SVXY	Short VIX Futures	Commodity Based ETF
MTUM	Momentum Factor Fund	US Equity ETF
IVE	S&P 500 Value Index Fund	US Equity ETF
VLUE	MSCI USA Value Factor	US Equity ETF
QUAL	MSCI USA Quality Factor	US Equity ETF
VTV	Value	US Equity ETF
USMV	MSCI USA Minimum Volatility Fund	US Equity ETF

Exchange Traded Notes (*ETNs*)

ETNs are similar to *ETFs*, with the difference that *ETFs* are shares in a fund which owns the underlying assets, while *ETNs* are notes from issuers which promise payouts according to a formula tied to the underlying asset.

ETFs are similar to mutual funds, while *ETNs* are similar to corporate bonds.

ETNs are technically unsecured corporate debt, but instead of fixed coupons, they promise to provide returns on a market index or futures contract.

The *ETN* issuer promises the payout and is responsible for tracking the index.

The *ETN* investor has counterparty credit risk to the *ETN* issuer.

Downloading ETF Prices Using Package *quantmod*

The function `getSymbols()` downloads time series data into the specified *environment*.

`getSymbols()` downloads the daily *OHLC* prices and trading volume (Open, High, Low, Close, Adjusted, Volume).

`getSymbols()` creates objects in the specified *environment* from the input strings (names), and assigns the data to those objects, without returning them as a function value, as a *side effect*.

If the argument `"auto.assign"` is set to `FALSE`, then `getSymbols()` returns the data, instead of assigning it silently.

Yahoo data quality deteriorated significantly in 2017, and *Google* data quality is also poor, leaving *Tiingo* and *Alpha Vantage* as the only major providers of free daily *OHLC* stock prices.

But Quandl doesn't provide free *ETF* prices, leaving *Alpha Vantage* as the best provider of free daily *ETF* prices.

```
> # Select ETF symbols for asset allocation
> symbolv <- c("VTI", "VEU", "EEM", "XLY", "XLP", "XLE", "XLF",
+   "XLV", "XLI", "XLB", "XLK", "XLU", "VYM", "IVW", "IWB", "IWD",
+   "IWF", "IEF", "TLT", "VNQ", "DBC", "GLD", "USO", "VXX", "SVXY",
+   "MTUM", "IVE", "VLUE", "QUAL", "VTV", "USMV")
> library(rutils)  # Load package rutils
> etfenv <- new.env()  # New environment for data
> # Boolean vector of symbols already downloaded
> isdownloaded <- symbolv %in% ls(etfenv)
> # Download data for symbolv using single command - creates pacing
> getSymbols.av(symbolv, adjust=TRUE, env=etfenv,
+   output.size="full", api.key="T7JPW54ES8G75310")
> # Download data from Alpha Vantage using while loop
> nattempts <- 0  # number of download attempts
> while (((sum(!isdownloaded)) > 0) & (nattempts<10)) {
+   # Download data and copy it into environment
+   nattempts <- nattempts + 1
+   cat("Download attempt = ", nattempts, "\n")
+   for (symbol in na.omit(symbolv[!isdownloaded][1:5])) {
+     cat("Processing: ", symbol, "\n")
+     tryCatch(  # With error handler
+ quantmod::getSymbols.av(symbol, adjust=TRUE, env=etfenv, auto.assi
+ # Error handler captures error condition
+ error=function(error_cond) {
+   print(paste("error handler: ", error_cond))
+ },  # end error handler
+ finally=print(paste("symbol=", symbol))
+     )  # end tryCatch
+   }  # end for
+   # Update vector of symbols already downloaded
+   isdownloaded <- symbolv %in% ls(etfenv)
+   cat("Pausing 1 minute to avoid pacing...\n")
+   Sys.sleep(65)
+ }  # end while
> # Download all symbolv using single command - creates pacing erro
> # quantmod::getSymbols.av(symbolv, env=etfenv, adjust=TRUE, from="
```

Inspecting ETF Prices in an Environment

The function `get()` retrieves objects that are referenced using character strings, instead of their names.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

```
> ls(etfenv)  # List files in etfenv
> # Get class of object in etfenv
> class(get(x=symbolv[1], envir=etfenv))
> # Another way
> class(etfenv$VTI)
> colnames(etfenv$VTI)
> # Get first 3 rows of data
> head(etfenv$VTI, 3)
> # Get last 11 rows of data
> tail(etfenv$VTI, 11)
> # Get class of all objects in etfenv
> eapply(etfenv, class)
> # Get class of all objects in R workspace
> lapply(ls(), function(ob_ject) class(get(ob_ject)))
> # Get end dates of all objects in etfenv
> as.Date(sapply(etfenv, end))
```

Adjusting Stock Prices Using Package *quantmod*

Traded stock and bond prices experience jumps after splits and dividends, and must be adjusted to account for them.

The function `adjustOHLC()` adjusts *OHLC* prices.

The function `get()` retrieves objects that are referenced using character strings, instead of their names.

The function `assign()` assigns a value to an object in a specified *environment*, by referencing it using a character string (name).

The functions `get()` and `assign()` allow retrieving and assigning values to objects that are referenced using character strings.

The function `mget()` accepts a vector of strings and returns a list of the corresponding objects extracted from an *environment*.

If the argument `"adjust"` in function `getSymbols()` is set to `TRUE`, then `getSymbols()` returns adjusted data.

```
> # Check of object is an OHLC time series
> is.OHLC(etfenv$VTI)
> # Adjust single OHLC object using its name
> etfenv$VTI <- adjustOHLC(etfenv$VTI, use.Adjusted=TRUE)
>
> # Adjust OHLC object using string as name
> assign(symbolv[1], adjustOHLC(
+     get(x=symbolv[1], envir=etfenv), use.Adjusted=TRUE),
+   envir=etfenv)
>
> # Adjust objects in environment using vector of strings
> for (symbol in ls(etfenv)) {
+   assign(symbol,
+    adjustOHLC(get(symbol, envir=etfenv), use.Adjusted=TRUE),
+    envir=etfenv)
+ }  # end for
```

Extracting Time Series from Environments

The function `mget()` accepts a vector of strings and returns a list of the corresponding objects extracted from an *environment*.

The extractor (accessor) functions from package *quantmod*: `Cl()`, `Vo()`, etc., extract columns from *OHLC* data.

A list of *xts* series can be flattened into a single *xts* series using the function `do.call()`.

The function `do.call()` executes a function call using a function name and a list of arguments.

`do.call()` passes the list elements individually, instead of passing the whole list as one argument.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

Time series can also be extracted from an *environment* by coercing it into a `list`, and then subsetting and merging it into an *xts* series using the function `do.call()`.

```
> library(rutils)  # Load package rutils
> # Define ETF symbols
> symbolv <- c("VTI", "VEU", "IEF", "VNQ")
> # Extract symbolv from rutils::etfenv
> pricets <- mget(symbolv, envir=rutils::etfenv)
> # pricets is a list of xts series
> class(pricets)
> class(pricets[[1]])
> tail(pricets[[1]])
> # Extract close prices
> pricets <- lapply(pricets, quantmod::Cl)
> # Collapse list into time series the hard way
> prices2 <- cbind(pricets[[1]], pricets[[2]], pricets[[3]], pricets
> class(prices2)
> dim(prices2)
> # Collapse list into time series using do.call()
> pricets <- do.call(cbind, pricets)
> all.equal(prices2, pricets)
> class(pricets)
> dim(pricets)
> # Or extract and cbind in single step
> pricets <- do.call(cbind, lapply(
+   mget(symbolv, envir=rutils::etfenv), quantmod::Cl))
> # Or extract and bind all data, subset by symbolv
> pricets <- lapply(symbolv, function(symbol) {
+     quantmod::Cl(get(symbol, envir=rutils::etfenv))
+ })  # end lapply
> # Or loop over etfenv without anonymous function
> pricets <- do.call(cbind,
+   lapply(as.list(rutils::etfenv)[symbolv], quantmod::Cl))
> # Same, but works only for OHLC series - produces error
> pricets <- do.call(cbind,
+   eapply(rutils::etfenv, quantmod::Cl)[symbolv])
```

Managing Time Series

Time series columns can be renamed, and then saved into `.csv` files.

The function `strsplit()` splits the elements of a character vector.

The package *zoo* contains functions `write.zoo()` and `read.zoo()` for writing and reading *zoo* time series from `.txt` and `.csv` files.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

The function `assign()` assigns a value to an object in a specified *environment*, by referencing it using a character string (name).

The function `save()` writes objects to compressed binary `.RData` files.

```
> # Column names end with ".Close"
> colnames(pricets)
> strsplit(colnames(pricets), split="[.]")
> do.call(rbind, strsplit(colnames(pricets), split="[.]"))
> do.call(rbind, strsplit(colnames(pricets), split="[.]"))[, 1]
> # Drop ".Close" from colnames
> colnames(pricets) <- rutils::get_name(colnames(pricets))
> # Or
> # colnames(pricets) <- do.call(rbind,
> #   strsplit(colnames(pricets), split="[.]"))[, 1]
> tail(pricets, 3)
> # Which objects in global environment are class xts?
> unlist(eapply(globalenv(), is.xts))
> # Save xts to csv file
> write.zoo(pricets,
+   file="/Users/jerzy/Develop/lecture_slides/data/etf_series.csv",
> # Copy prices into etfenv
> etfenv$pricets <- pricets
> # Or
> assign("pricets", pricets, envir=etfenv)
> # Save to .RData file
> save(etfenv, file="etf_data.RData")
```

Calculating Percentage Returns from Close Prices

The function `quantmod::dailyReturn()` calculates the percentage daily returns from the *Close* prices.

The `lapply()` and `sapply()` functionals perform a loop over the columns of *zoo* and *xts* series.

```
> # Extract VTI prices
> pricets <- etfenv$prices[ ,"VTI"]
> pricets <- na.omit(pricets)
> # Calculate percentage returns "by hand"
> pricel <- as.numeric(pricets)
> pricel <- c(pricel[1], pricel[-NROW(pricel)])
> pricel <- xts(pricel, zoo::index(pricets))
> retsp <- (pricets-pricel)/pricel
> # Calculate percentage returns using dailyReturn()
> retsd <- quantmod::dailyReturn(pricets)
> head(cbind(retsd, retsp))
> all.equal(retsd, retsp, check.attributes=FALSE)
> # Calculate returns for all prices in etfenv$prices
> retsp <- lapply(etfenv$prices, function(xtsv) {
+   retsd <- quantmod::dailyReturn(na.omit(xtsv))
+   colnames(retsd) <- names(xtsv)
+   retsd
+ })  # end lapply
> # "retsp" is a list of xts
> class(retsp)
> class(retsp[[1]])
> # Flatten list of xts into a single xts
> retsp <- do.call(cbind, retsp)
> class(retsp)
> dim(retsp)
> # Copy retsp into etfenv and save to .RData file
> # assign("retsp", retsp, envir=etfenv)
> etfenv$retsp <- retsp
> save(etfenv, file="/Users/jerzy/Develop/lecture_slides/data/etf_da
```

Managing Data Inside Environments

The function `as.environment()` coerces objects (listv) into an environment.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

The function `mget()` accepts a vector of strings and returns a list of the corresponding objects extracted from an *environment*.

```
> library(rutils)
> startd <- "2012-05-10"; endd <- "2013-11-20"
> # Select all objects in environment and return as environment
> new_env <- as.environment(eapply(etfenv, "[",
+             paste(startd, endd, sep="/")))
> # Select only symbolv in environment and return as environment
> new_env <- as.environment(
+   lapply(as.list(etfenv)[symbolv], "[",
+    paste(startd, endd, sep="/")))
> # Extract and cbind Close prices and return to environment
> assign("prices", rutils::do_call(cbind,
+          lapply(ls(etfenv), function(symbol) {
+            xtsv <- quantmod::Cl(get(symbol, etfenv))
+            colnames(xtsv) <- symbol
+            xtsv
+          })), envir=new_env)
> # Get sizes of OHLC xts series in etfenv
> sapply(mget(symbolv, envir=etfenv), object.size)
> # Extract and cbind adjusted prices and return to environment
> colname <- function(xtsv)
+   strsplit(colnames(xtsv), split="[.]")[[1]][1]
> assign("prices", rutils::do_call(cbind,
+          lapply(mget(etfenv$symbolv, envir=etfenv),
+                 function(xtsv) {
+                   xtsv <- Ad(xtsv)
+                   colnames(xtsv) <- colname(xtsv)
+                   xtsv
+          })), envir=new_env)
```

Loading Stock Tickers

The file `sp500_constituents.csv` contains a *data frame* of *S&P500* constituents.

The stock tickers are stored in the column `"Ticker"`.

The *data frame* contains duplicate tickers, which must be removed.

Some tickers (like "BRK.B" and "BF.B") are not valid symbols in *Tiingo*, so they must be renamed.

```
> # Load data frame of S&P500 constituents from CSV file
> sp500 <- read.csv(file="/Users/jerzy/Develop/lecture_slides/data/s
> # Inspect data frame of S&P500 constituents
> dim(sp500)
> colnames(sp500)
> # Extract tickers from the column Ticker
> symbolv <- sp500$Ticker
> # Get duplicate tickers
> tablev <- table(symbolv)
> duplicates <- tablev[tablev>1]
> duplicates <- names(duplicates)
> # Get duplicate records (rows) of sp500
> sp500[symbolv %in% duplicates, ]
> # Get unique tickers
> symbolv <- unique(symbolv)
> # Find index of ticker "BRK.B"
> which(symbolv=="BRK.B")
> # Rename "BRK.B" to "BRK-B" and "BF.B" to "BF-B"
> symbolv[which(symbolv=="BRK.B")] <- "BRK-B"
> symbolv[which(symbolv=="BF.B")] <- "BF-B"
```

Downloading Stock Time Series From *Tiingo*

Yahoo data quality deteriorated significantly in 2017, and *Google* data quality is also poor, leaving *Tiingo*, *Alpha Vantage*, and Quandl as the only major providers of free daily *OHLC* stock prices.

But Quandl doesn't provide free *ETF* prices, while *Tiingo* does.

The function `getSymbols()` has a *method* for downloading time series data from *Tiingo*, called `getSymbols.tiingo()`.

Users must first obtain a *Tiingo API key*, and then pass it in `getSymbols.tiingo()` calls: https://www.tiingo.com/

Note that the data are downloaded as `xts` time series, with a date-time index of class `POSIXct` (not `Date`).

```
> # Load package rutils
> library(rutils)
> # Create new environment for data
> sp500env <- new.env()
> # Boolean vector of symbols already downloaded
> isdownloaded <- symbolv %in% ls(sp500env)
> # Download in while loop from Tiingo and copy into environment
> nattempts <- 0  # Number of download attempts
> while (((sum(!isdownloaded)) > 0) & (nattempts<3)) {
+   # Download data and copy it into environment
+   nattempts <- nattempts + 1
+   cat("Download attempt = ", nattempts, "\n")
+   for (symbol in symbolv[!isdownloaded]) {
+     cat("processing: ", symbol, "\n")
+     tryCatch(  # With error handler
+ quantmod::getSymbols(symbol, src="tiingo", adjust=TRUE, auto.assig
+            from="1990-01-01", env=sp500env, api.key="j84ac2b9c5bde
+ # Error handler captures error condition
+ error=function(error_cond) {
+   print(paste("error handler: ", error_cond))
+ },  # end error handler
+ finally=print(paste("symbol=", symbol))
+     )  # end tryCatch
+   }  # end for
+   # Update vector of symbols already downloaded
+   isdownloaded <- symbolv %in% ls(sp500env)
+   Sys.sleep(2)  # Wait 2 seconds until next attempt
+ }  # end while
> class(sp500env$AAPL)
> class(zoo::index(sp500env$AAPL))
> tail(sp500env$AAPL)
> symbolv[!isdownloaded]
```

Coercing Date-time Indices

The date-time indices of the *OHLC* stock prices are in the `POSIXct` format suitable for intraday prices, not daily prices.

The function `as.Date()` coerces `POSIXct` objects into `Date` objects.

The function `get()` retrieves objects that are referenced using character strings, instead of their names.

The function `assign()` assigns a value to an object in a specified *environment*, by referencing it using a character string (name).

The functions `get()` and `assign()` allow retrieving and assigning values to objects that are referenced using character strings.

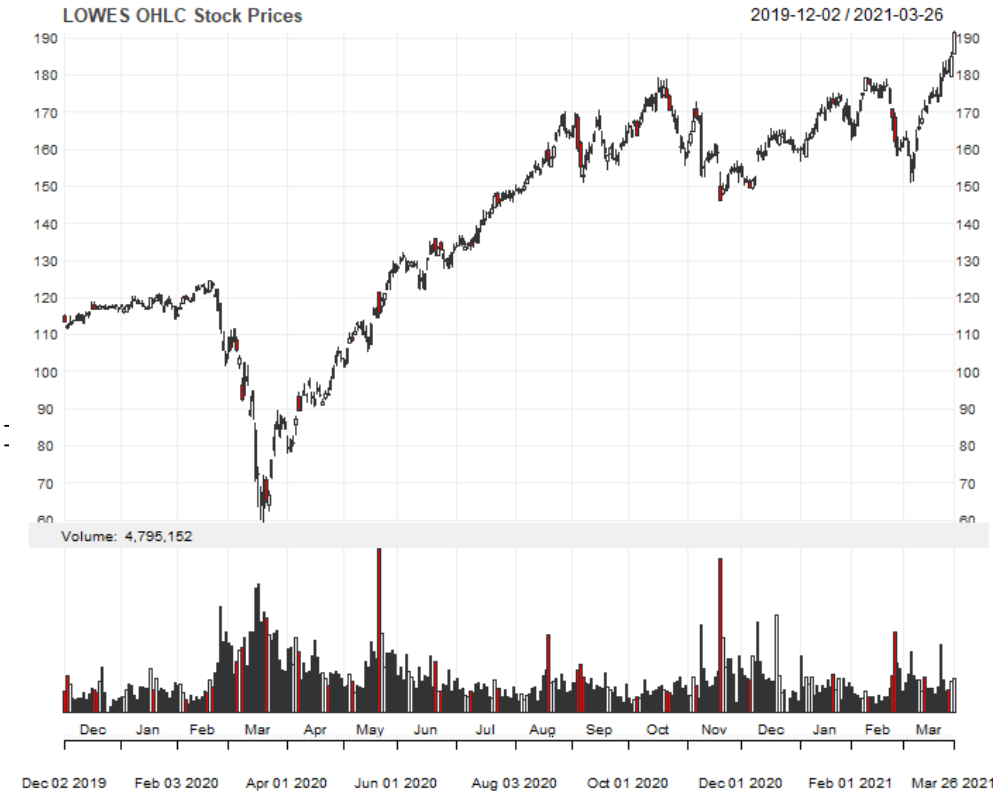
```
> # The date-time index of AAPL is POSIXct
> class(zoo::index(sp500env$AAPL))
> # Coerce the date-time index of AAPL to Date
> zoo::index(sp500env$AAPL) <- as.Date(zoo::index(sp500env$AAPL))
> # Coerce all the date-time indices to Date
> for (symbol in ls(sp500env)) {
+   ohlc <- get(symbol, envir=sp500env)
+   zoo::index(ohlc) <- as.Date(zoo::index(ohlc))
+   assign(symbol, ohlc, envir=sp500env)
+ }  # end for
```

Managing Exceptions in Stock Symbols

The column names for symbol "LOW" (Lowe's company) must be renamed for the extractor function quantmod::Lo() to work properly.

Tickers which contain a dot in their name (like "BRK.B") are not valid symbols in R, so they must be downloaded separately and renamed.

```
> # "LOW.Low" is a bad column name
> colnames(sp500env$LOW)
> strsplit(colnames(sp500env$LOW), split="[.]")
> do.call(cbind, strsplit(colnames(sp500env$LOW), split="[.]"))
> do.call(cbind, strsplit(colnames(sp500env$LOW), split="[.]"))[2,
> # Extract proper names from column names
> namesv <- rutils::get_name(colnames(sp500env$LOW), field=2)
> # Or
> # namesv <- do.call(rbind, strsplit(colnames(sp500env$LOW),
> #                                   split="[.]"))[, 2]
> # Rename "LOW" colnames to "LOWES"
> colnames(sp500env$LOW) <- paste("LO_WES", namesv, sep=".")
> sp500env$LOWES <- sp500env$LOW
> rm(LOW, envir=sp500env)
> # Rename BF-B colnames to "BFB"
> colnames(sp500env$"BF-B") <- paste("BFB", namesv, sep=".")
> sp500env$BFB <- sp500env$"BF-B"
> rm("BF-B", envir=sp500env)
> # Rename BRK-B colnames
> sp500env$BRKB <- sp500env$`BRK-B`
> rm(`BRK-B`, envir=sp500env)
> colnames(sp500env$BRKB) <- gsub("BRK-B", "BRKB", colnames(sp500e
> # Save OHLC prices to .RData file
> save(sp500env, file="/Users/jerzy/Develop/lecture_slides/data/sp500.RData")
> # Download "BRK.B" separately with auto.assign=FALSE
> # BRKB <- quantmod::getSymbols("BRK-B", auto.assign=FALSE, src="tiingo", adjust=TRUE, from="1990-01-01", api.key="j84ac2b9c5bde2d68e3
> # colnames(BRKB) <- paste("BRKB", namesv, sep=".")
> # sp500env$BRKB <- BRKB
```

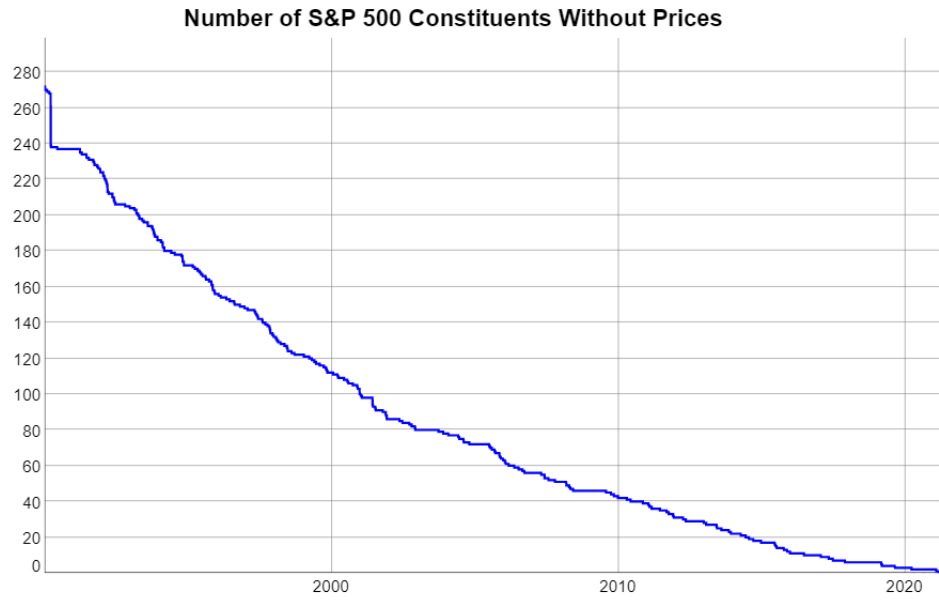



```
> # Plot OHLC candlestick chart for LOWES
> chart_Series(x=sp500env$LOWES["2019-12/"],
+   TA="add_Vo()", name="LOWES OHLC Stock Prices")
> # Plot dygraph
> dygraphs::dygraph(sp500env$LOWES["2019-12/", -5], main="LOWES OHL
+   dyCandlestick()
```

S&P500 Stock Index Constituent Prices

The file sp500.RData contains the *environment* sp500_env with *OHLC* prices and trading volumes of *S&P500* stock index constituents.

The *S&P500* stock index constituent data is of poor quality before 2000, so we'll mostly use the data after the year 2000.



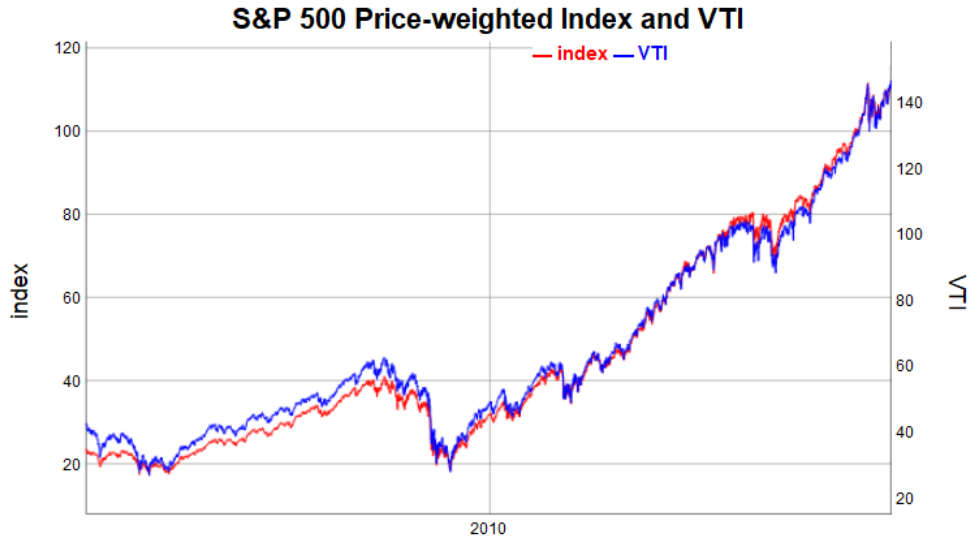

```
> # Load S&P500 constituent stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500.RData")
> pricets <- eapply(sp500env, quantmod::Cl)
> pricets <- rutils::do_call(cbind, pricets)
> # Carry forward non-NA prices
> pricets <- zoo::na.locf(pricets, na.rm=FALSE)
> # Drop ".Close" from column names
> colnames(pricets[, 1:4])
> colnames(pricets) <- rutils::get_name(colnames(pricets))
> # Or
> # colnames(pricets) <- do.call(rbind,
> #   strsplit(colnames(pricets), split="[.]"))[, 1]
> # Calculate percentage returns of the S&P500 constituent stocks
> # retsp <- xts::diff.xts(log(pricets))
> retsp <- xts::diff.xts(pricets)/
+   rutils::lagit(pricets, pad_zeros=FALSE)
> set.seed(1121)
> samplev <- sample(NCOL(retsp), s=100, replace=FALSE)
> prices100 <- pricets[, samplev]
> returns100 <- retsp[, samplev]
> save(pricets, prices100,
+   file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.RData")
> save(retsp, returns100,
+   file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData")
```

```
> # Calculate number of constituents without prices
> datav <- rowSums(is.na(pricets))
> datav <- xts::xts(datav, order.by=zoo::index(pricets))
> dygraphs::dygraph(datav, main="Number of S&P500 Constituents Witho
+   dyOptions(colors="blue", strokeWidth=2)
```

S&P500 Stock Portfolio Index

The price-weighted index of *S&P500* constituents closely follows the VTI *ETF*.

```
> # Calculate price weighted index of constituent
> ncols <- NCOL(pricets)
> pricets <- zoo::na.locf(pricets, fromLast=TRUE)
> indeks <- xts(rowSums(pricets)/ncols, zoo::index(pricets))
> colnames(indeks) <- "index"
> # Combine index with VTI
> datav <- cbind(indeks[zoo::index(etfenv$VTI)], etfenv$VTI[, 4])
> colnamev <- c("index", "VTI")
> colnames(datav) <- colnamev
> # Plot index with VTI
> dygraphs::dygraph(datav,
+   main="S&P 500 Price-weighted Index and VTI") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="red") %>%
+   dySeries(name=colnamev[2], axis="y2", col="blue")
```



Writing Time Series To Files

The data from *Tiingo* is downloaded as `xts` time series, with a date-time index of class `POSIXct` (not `Date`).

The function `save()` writes objects to compressed binary `.RData` files.

The easiest way to share data between `R` and `Excel` is through `.csv` files.

The package *zoo* contains functions `write.zoo()` and `read.zoo()` for writing and reading *zoo* time series from `.txt` and `.csv` files.

The function `data.table::fread()` reads from `.csv` files over 6 times faster than the function `read.csv()`!

The function `data.table::fwrite()` writes to `.csv` files over 12 times faster than the function `write.csv()`, and 278 times faster than function `cat()`!

```
> # Save the environment to compressed .RData file
> dir_name <- "/Users/jerzy/Develop/lecture_slides/data/"
> save(sp500env, file=paste0(dir_name, "sp500.RData"))
> # Save the ETF prices into CSV files
> dir_name <- "/Users/jerzy/Develop/lecture_slides/data/SP500/"
> for (symbol in ls(sp500env)) {
+   zoo::write.zoo(sp500env$symbol, file=paste0(dir_name, symbol, "
+ }  # end for
> # Or using lapply()
> file_names <- lapply(ls(sp500env), function(symbol) {
+   xtsv <- get(symbol, envir=sp500env)
+   zoo::write.zoo(xtsv, file=paste0(dir_name, symbol, ".csv"))
+   symbol
+ })  # end lapply
> unlist(file_names)
> # Or using eapply() and data.table::fwrite()
> file_names <- eapply(sp500env , function(xtsv) {
+   file_name <- rutils::get_name(colnames(xtsv)[1])
+   data.table::fwrite(data.table::as.data.table(xtsv), file=paste0
+   file_name
+ })  # end eapply
> unlist(file_names)
```

Reading Time Series from Files

The function `load()` reads data from `.RData` files, and *invisibly* returns a vector of names of objects created in the workspace.

The function `Sys.glob()` listv files matching names obtained from wildcard expansion.

The easiest way to share data between `R` and `Excel` is through `.csv` files.

The function `as.Date()` parses `character` strings, and coerces `numeric` and `POSIXct` objects into `Date` objects.

The function `data.table::setDF()` coerces a *data table* object into a *data frame* using a *side effect*, without making copies of data.

The function `data.table::fread()` reads from `.csv` files over 6 times faster than the function `read.csv()`!

```
> # Load the environment from compressed .RData file
> dir_name <- "/Users/jerzy/Develop/lecture_slides/data/"
> load(file=paste0(dir_name, "sp500.RData"))
> # Get all the .csv file names in the directory
> dir_name <- "/Users/jerzy/Develop/lecture_slides/data/SP500/"
> file_names <- Sys.glob(paste0(dir_name, "*.csv"))
> # Create new environment for data
> sp500env <- new.env()
> for (file_name in file_names) {
+   xtsv <- xts::as.xts(zoo::read.csv.zoo(file_name))
+   symbol <- rutils::get_name(colnames(xtsv)[1])
+   # symbol <- strsplit(colnames(xtsv), split="[.]")[[1]][1]
+   assign(symbol, xtsv, envir=sp500env)
+ }  # end for
> # Or using fread()
> for (file_name in file_names) {
+   xtsv <- data.table::fread(file_name)
+   data.table::setDF(xtsv)
+   xtsv <- xts::xts(xtsv[, -1], as.Date(xtsv[, 1]))
+   symbol <- rutils::get_name(colnames(xtsv)[1])
+   assign(symbol, xtsv, envir=sp500env)
+ }  # end for
```

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

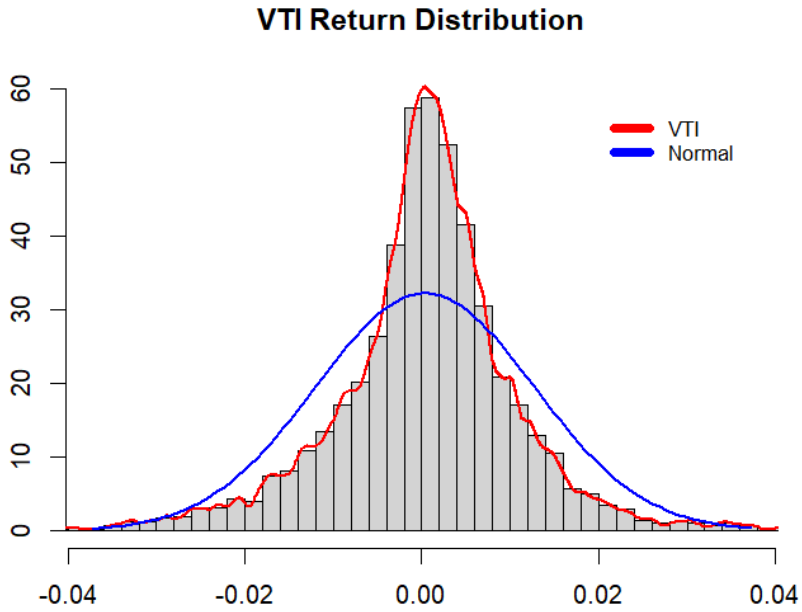
Distribution of Asset Returns

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(retsp, breaks=100, freq=FALSE,
+   xlim=c(-5*madv, 5*madv), xlab="", ylab="",
+   main="VTI Return Distribution")
> # Draw kernel density of histogram
> lines(densityv, col="red", lwd=2)
> # Add density of normal distribution
> curve(expr=dnorm(x, mean=mean(retsp), sd=sd(retsp)),
+ add=TRUE, lwd=2, col="blue")
> # Add legend
> legend("topright", inset=0.05, cex=0.8, title=NULL,
+  leg=c("VTI", "Normal"), bty="n",
+  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.

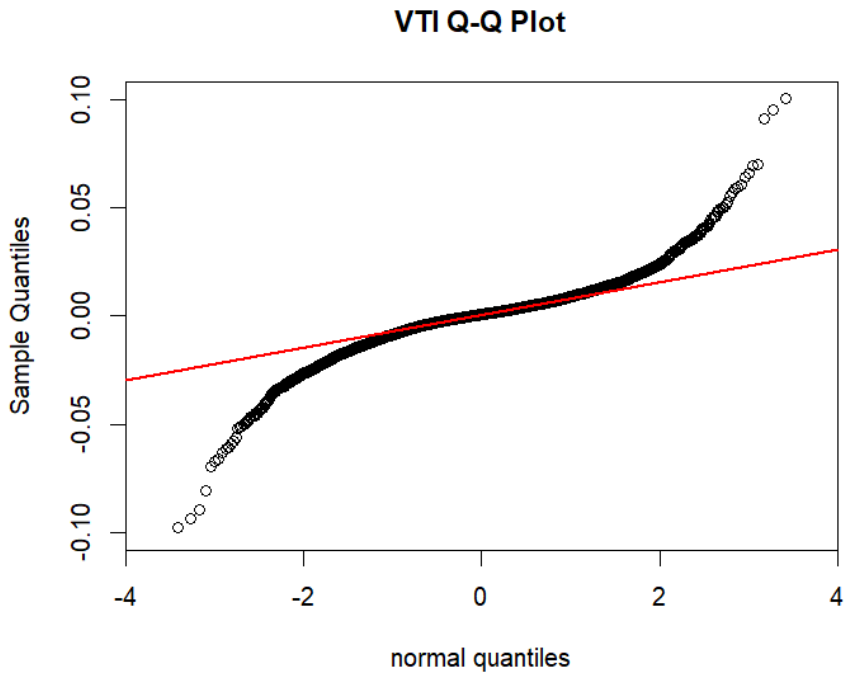
The *VTI Q-Q* plot shows that the *VTI* return distribution has fat tails.

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

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*).

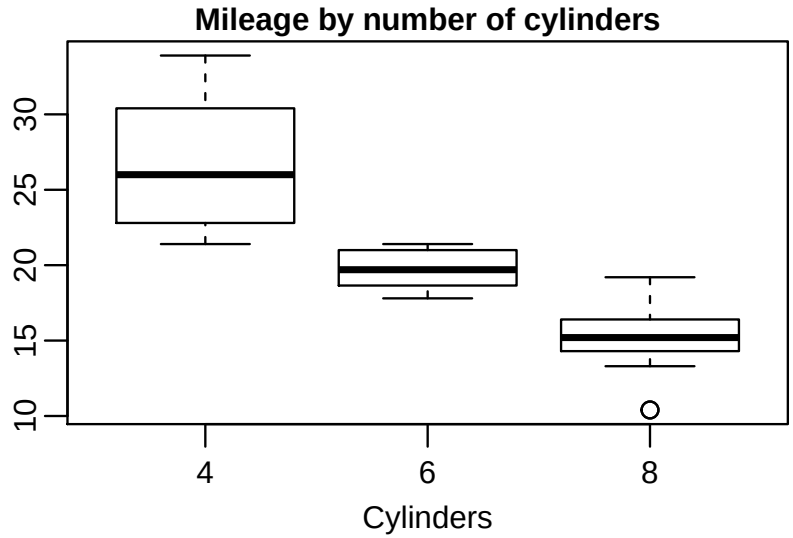
The nominal range is equal to 1.5 times the *IQR* above and below the box *hinges*.

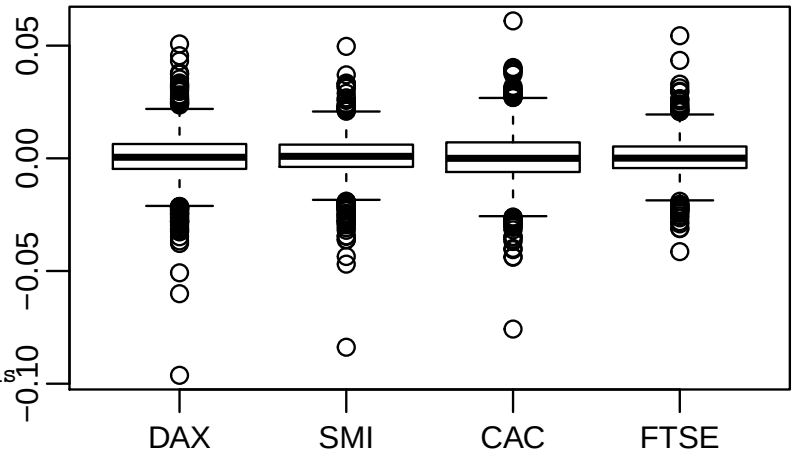
The *whiskers* are dashed vertical lines representing values beyond the first and third quartiles, but within the nominal range.

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):

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

The sample kurtosis (fourth moment):

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

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
> retsp <- na.omit(rutils::etfenv$returns$VTI)
> # Number of observations
> nrows <- NROW(retsp)
> # Mean of VTI returns
> retsm <- mean(retsp)
> # Standard deviation of VTI returns
> sdrets <- sd(retsp)
> # Skewness of VTI returns
> nrows/((nrows-1)*(nrows-2))*
+   sum(((retsp - retsm)/sdrets)^3)
> # Kurtosis of VTI returns
> nrows*(nrows+1)/((nrows-1)^3)*
+   sum(((retsp - retsm)/sdrets)^4)
> # Random normal returns
> retsp <- rnorm(nrows, sd=sdrets)
> # Mean and standard deviation of random normal returns
> retsm <- mean(retsp)
> sdrets <- sd(retsp)
> # Skewness of random normal returns
> nrows/((nrows-1)*(nrows-2))*
+   sum(((retsp - retsm)/sdrets)^3)
> # Kurtosis of random normal returns
> nrows*(nrows+1)/((nrows-1)^3)*
+   sum(((retsp - retsm)/sdrets)^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(retsp) {
+   retsp <- na.omit(retsp)
+   sum(((retsp - mean(retsp))/sd(retsp))^3)/NROW(retsp)
+ }  # end calc_skew
> # calc_kurt() calculates kurtosis of returns
> calc_kurt <- function(retsp) {
+   retsp <- na.omit(retsp)
+   sum(((retsp - mean(retsp))/sd(retsp))^4)/NROW(retsp)
+ }  # end calc_kurt
> # Calculate skew and kurtosis of VTI returns
> calc_skew(retsp)
> calc_kurt(retsp)
> # calcmom() calculates the moments of returns
> calcmom <- function(retsp, moment=3) {
+   retsp <- na.omit(retsp)
+   sum(((retsp - mean(retsp))/sd(retsp))^moment)/NROW(retsp)
+ }  # end calcmom
> # Calculate skew and kurtosis of VTI returns
> calcmom(retsp, moment=3)
> calcmom(retsp, 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).

```
> set.seed(1121)  # Reset random number generator
> # 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

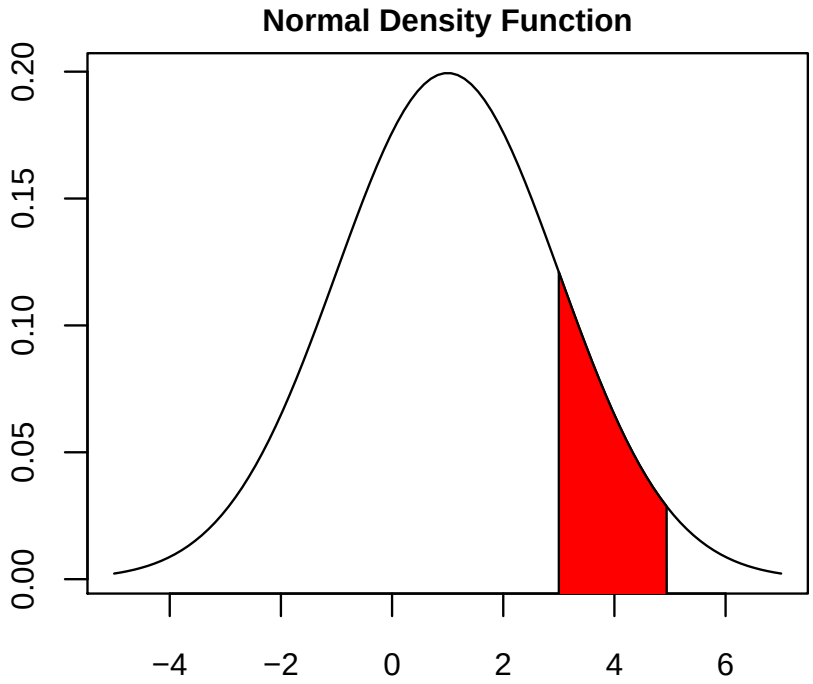
The *Normal (Gaussian)* probability density function is given by:

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

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.

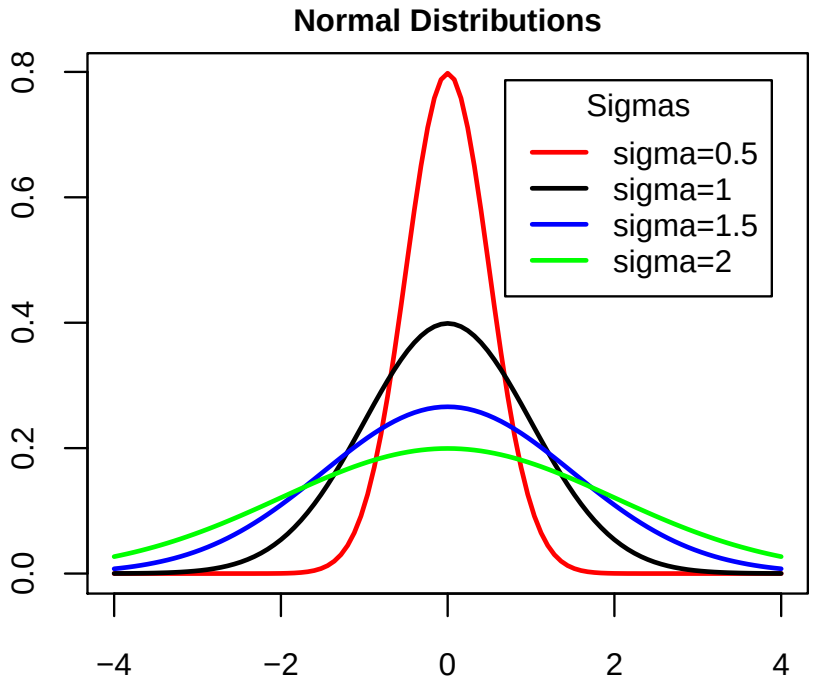
```
> xvar <- seq(-5, 7, length=100)
> yvar <- dnorm(xvar, mean=1.0, sd=2.0)
> plot(xvar, yvar, type="l", lty="solid", xlab="", ylab="")
> title(main="Normal Density Function", line=0.5)
> startp <- 3; endp <- 5  # Set lower and upper bounds
> # Set polygon base
> subv <- ((xvar >= startp) & (xvar <= endp))
> polygon(c(startp, xvar[subv], endp),  # 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
> colors <- 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=colors[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",
+  labelv, cex=0.8, lwd=2, lty=1, bty="n", col=colors)
```



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: $\hat{\sigma}^2 = \frac{1}{\nu-1}\sum_{i=1}^{\nu}(z_i - \bar{z})^2$

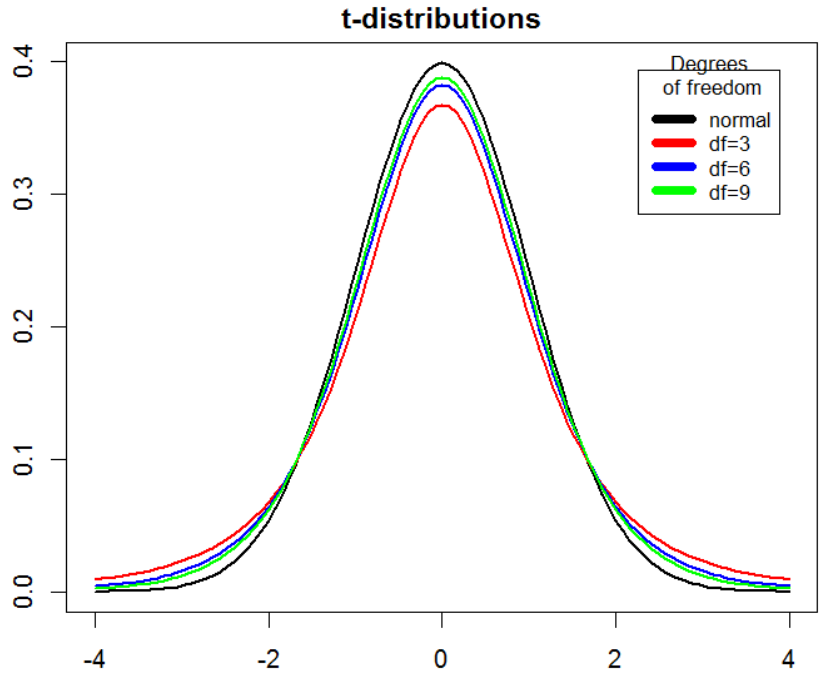
Then the random variable (*t-ratio*):

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

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

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

```
> degf <- c(3, 6, 9)  # Df values
> colors <- 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=colors[it+1], add=TRUE)
+ }  # end for
```




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

Mixture Models of Returns

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

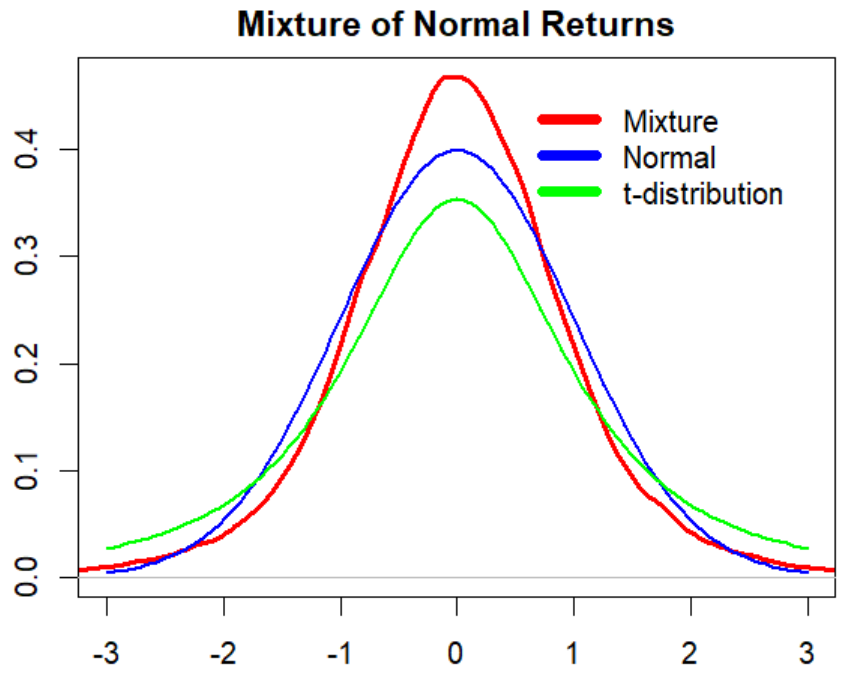
The mixture of two normal distributions with different variances produces a distribution with *leptokurtosis* (large kurtosis, or fat tails).

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
> retsp <- c(rnorm(nrows/2), 2*rnorm(nrows/2))
> retsp <- (retsp-mean(retsp))/sd(retsp)
> # Kurtosis of normal
> calc_kurt(rnorm(nrows))
> # Kurtosis of mixture
> calc_kurt(retsp)
> # Or
> nrows*sum(retsp^4)/(nrows-1)^2
```

```
> # Plot the distributions
> plot(density(retsp), 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"),
+   col=c("red", "blue", "green"))
```

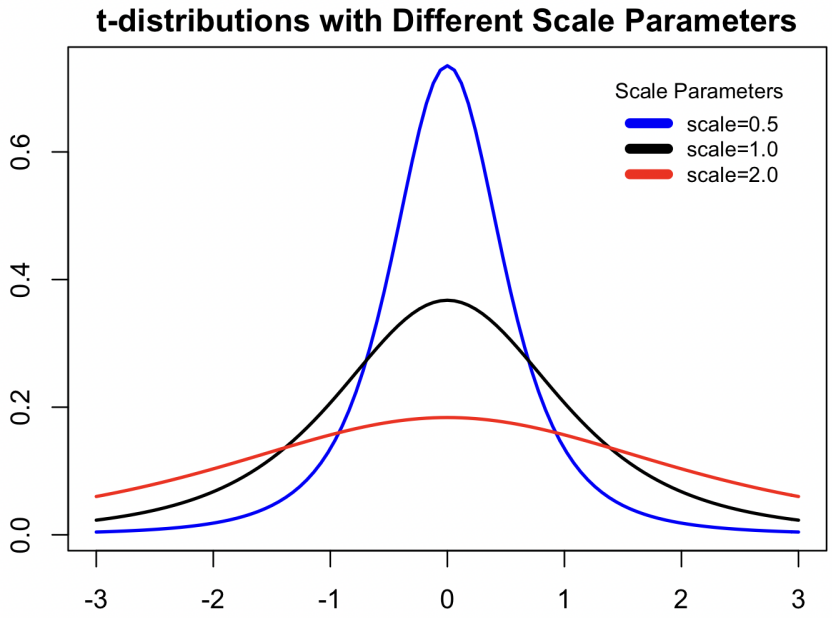
Non-standard Student's *t-distribution*

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)
> colors <- 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=colors[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=colors)
```

The *Shapiro-Wilk* Test of Normality

The *Shapiro-Wilk* test is designed to test the *null hypothesis* that a sample: $\{x1, \ldots, x_n\}$ 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: $\{a1, \ldots, a_n\}$ 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: $\{x1, \ldots, x_n\}$.

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)
> retsp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))[1:4999]
> # Reduce number of output digits
> ndigits <- options(digits=5)
> # Shapiro-Wilk test for normal distribution
> nrows <- NROW(retsp)
> shapiro.test(rnorm(nrows))

Shapiro-Wilk normality test

data:  rnorm(nrows)
W = 1, p-value = 0.24
> # Shapiro-Wilk test for VTI returns
> shapiro.test(retsp)

Shapiro-Wilk normality test

data:  retsp
W = 0.886, p-value <2e-16
> # Shapiro-Wilk test for uniform distribution
> shapiro.test(runif(nrows))

Shapiro-Wilk normality test

data:  runif(nrows)
W = 0.955, p-value <2e-16
> # 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: $\{x1, \ldots, x_n\}$ 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 = 1, df = 2, p-value = 0.5
> # Jarque-Bera test for VTI returns
> jarque.bera.test(retsp)

Jarque Bera Test

data:  retsp
X-squared = 28386, df = 2, p-value <2e-16
> # Jarque-Bera test for uniform distribution
> jarque.bera.test(runif(NROW(retsp)))

Jarque Bera Test

data:  runif(NROW(retsp))
X-squared = 301, df = 2, p-value <2e-16
```

The *Kolmogorov-Smirnov* Test for Probability Distributions

The *Kolmogorov-Smirnov* test *null hypothesis* is that two samples: $\{x1, \ldots, x_n\}$ and $\{y1, \ldots, y_n\}$ 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$p.value
> # KS test for uniform distribution
> ks.test(runif(100), pnorm)
> # KS test for two shifted normal distributions
> ks.test(rnorm(100), rnorm(100, mean=0.1))
> ks.test(rnorm(100), rnorm(100, mean=1.0))
> # KS test for two different normal distributions
> ks.test(rnorm(100), rnorm(100, sd=2.0))
> # KS test for VTI returns vs normal distribution
> retsp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))
> retsp <- (retsp - mean(retsp))/sd(retsp)
> ks.test(retsp, pnorm)
```

Chi-squared Distribution

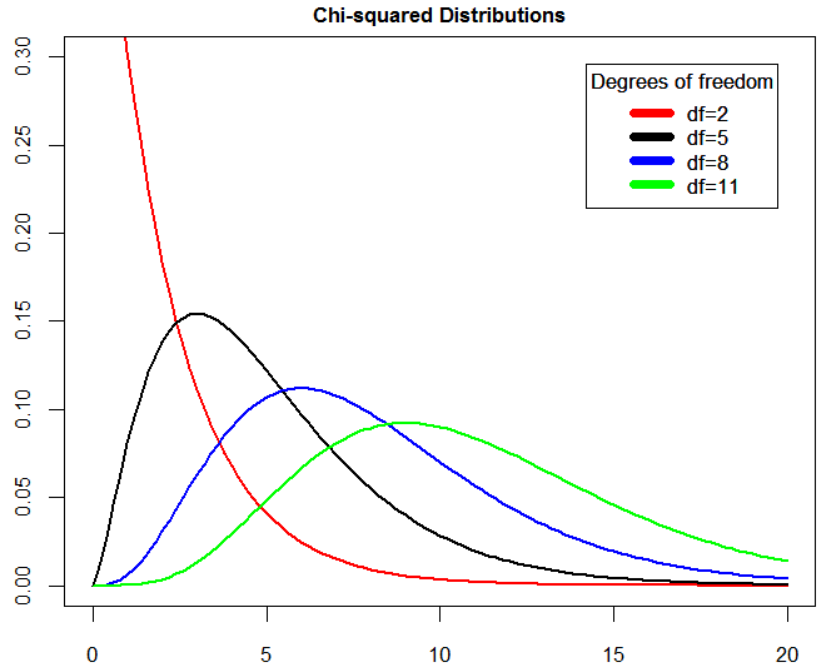
Let $z1, \ldots, z_k$ be independent standard *Normal* random variables.

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
> colors <- 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=colors[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",
+        title="Degrees of freedom", labelv,
+        cex=0.8, lwd=6, lty=1, col=colors)
```

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, sim
> 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(retsp, 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:

$$\theta_{MLE} = \arg\max_{\theta} \mathcal{L}(\theta|x)$$

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

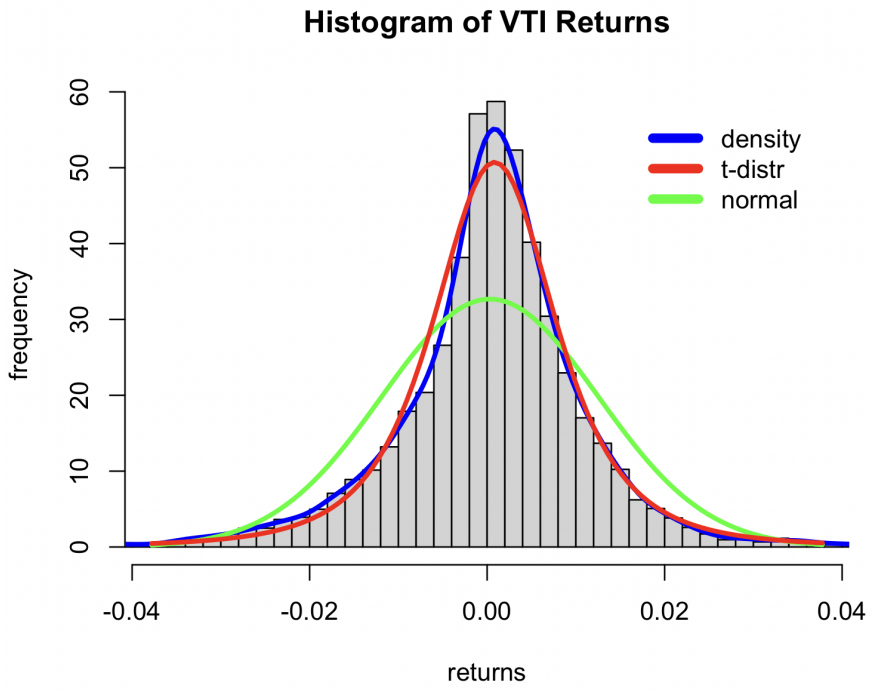
The Student's *t-distribution* Fitted to Asset Returns

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(retsp)
> histp <- hist(retsp, col="lightgrey",
+   xlab="returns", breaks=100, xlim=c(-5*madv, 5*madv),
+   ylab="frequency", freq=FALSE, main="Histogram of VTI Returns")
> lines(density(retsp, adjust=1.5), lwd=3, col="blue")
> # Plot the Normal probability distribution
> curve(expr=dnorm(x, mean=mean(retsp),
+   sd=sd(retsp)), 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",
+   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.

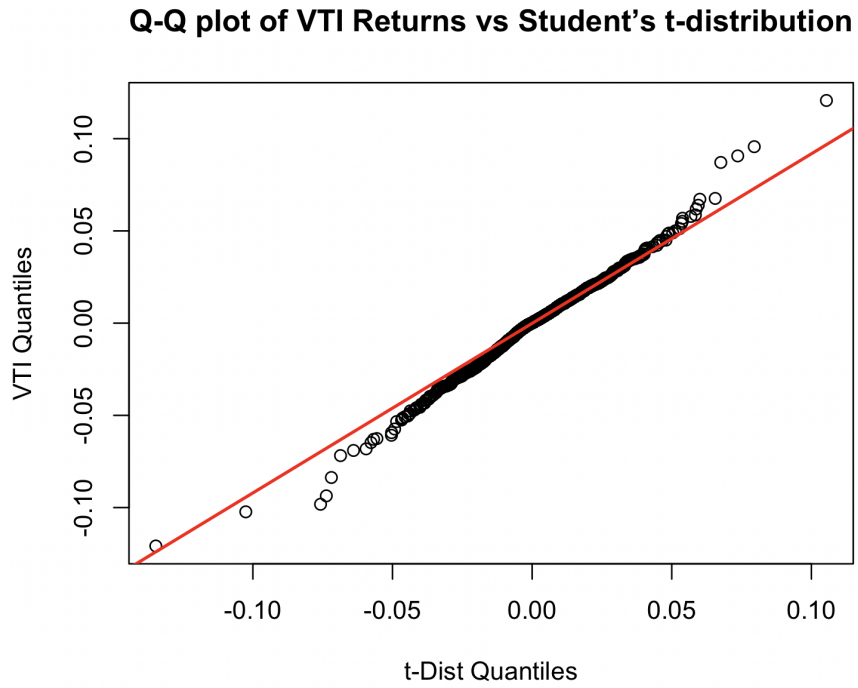
The *Q-Q* plot shows that stock returns fit the non-standard *t-distribution* with 3 degrees of freedom quite well.

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(retsp), df=3) + locv
> # Q-Q plot of VTI Returns vs non-standard t-distribution
> qqplot(tdata, retsp, 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(retsp, 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(retsp, 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(retsp, replace=TRUE), ptdistr, dfree=3, locv=locv,
```

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.

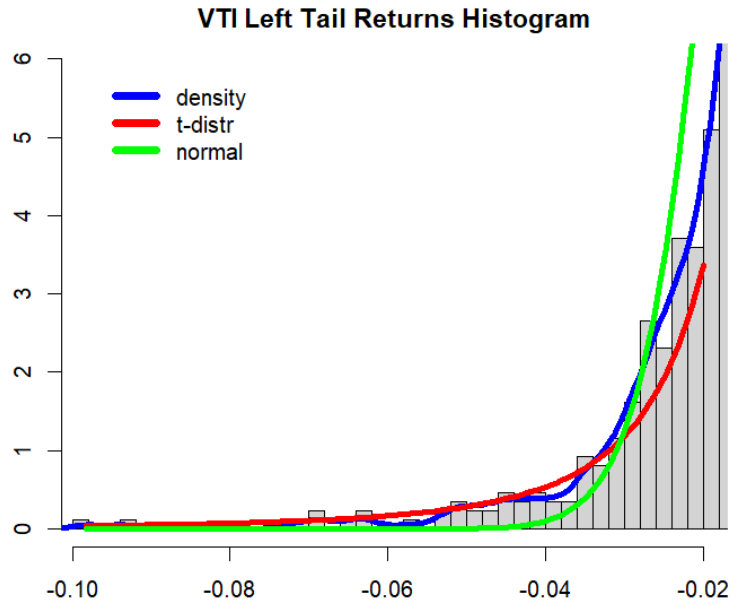
So large values are produced only when all the contributing binomial variables are of the same sign, which is very improbable, so it produces extremely low tail probabilities (thin tails),

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).



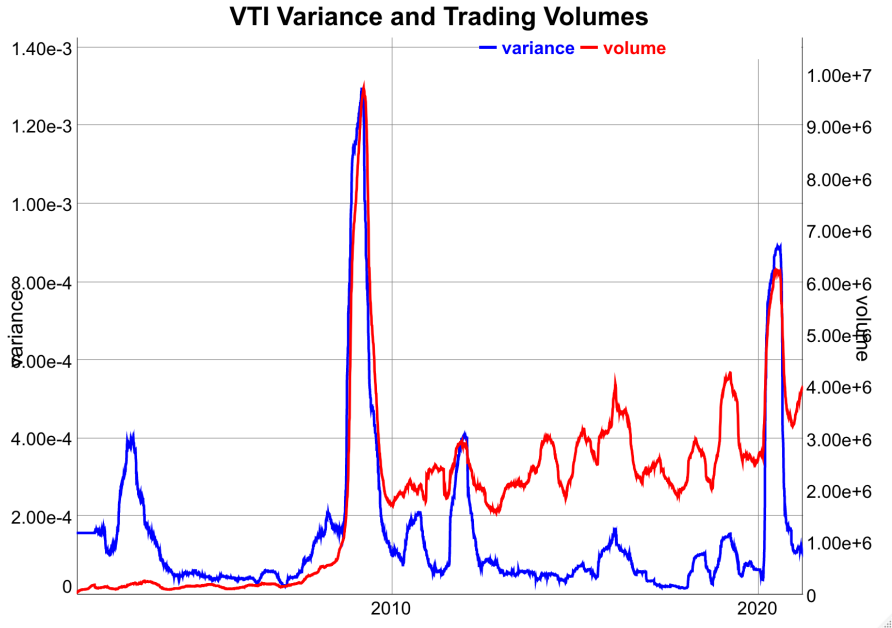

```
> # Plot histogram of VTI returns
> histp <- hist(retsp, breaks=100, plot=FALSE)
> plot(histp, xlab="retsp", ylab="frequency",
+      col="lightgrey", freq=FALSE, main="VTI Left Tail Returns Hist
+      xlim=c(min(retsp), -0.02),
+      ylim=c(0.0, histp$density[findInterval(-0.02, histp$breaks)])
> lines(density(retsp, adjust=1.5), lwd=4, col="blue")
> # Plot t-distribution function
> curve(expr=dt((x-locv)/scalev, df=2)/scalev, type="l", lwd=4, col=
> # Plot the Normal probability distribution
> curve(expr=dnorm(x, mean=mean(retsp), sd=sd(retsp)), add=TRUE, lwd
> # Add legend
> legend("topleft", inset=0.05, bty="n",
+   leg=c("density", "t-distr", "normal"),
+   lwd=6, lty=1, col=c("blue", "red", "green"))
```

Trading Volumes

The rolling 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)))
> retsp <- rutils::diffit(log(closep))
> volumes <- coredata(quantmod::Vo(ohlc))
> # Calculate rolling variance
> look_back <- 121
> variance <- HighFreq::roll_var_ohlc(log(ohlc), method="close", look_back=look_back, scale=FALSE)
> variance[1:look_back, ] <- variance[look_back+1, ]
> # Calculate rolling average volume
> volume_roll <- HighFreq::roll_vec(volumes, look_back=look_back)/look_back
> # dygraph plot of VTI variance and trading volumes
> datav <- xts::xts(cbind(variance, volume_roll), 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")
```

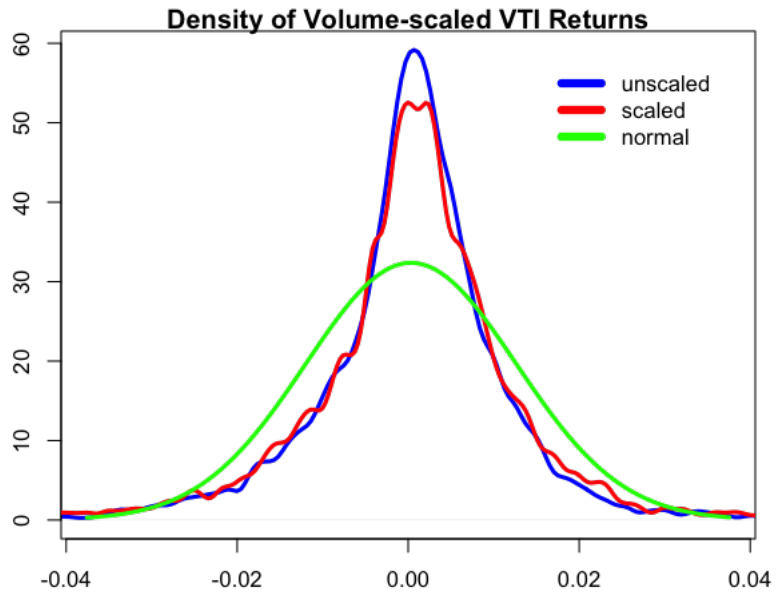
Asset Returns in Trading Time

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 returns using volume (volume clock)
> rets_scaled <- ifelse(volumes > 0,
+   sqrt(volume_roll)*retsp/sqrt(volumes), 0)
> rets_scaled <- sd(retsp)*rets_scaled/sd(rets_scaled)
> # rets_scaled <- ifelse(volumes > 1e4, retsp/volumes, 0)
> # Calculate moments of scaled returns
> nrows <- NROW(retsp)
> sapply(list(retsp=retsp, rets_scaled=rets_scaled),
+   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(retsp)
> # bwidth <- mad(rutils::diffit(retsp))
> plot(density(retsp, 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(rets_scaled, bw=madv/10), lwd=3, col="red")
> curve(expr=dnorm(x, mean=mean(retsp), sd=sd(retsp)),
+ add=TRUE, lwd=3, col="green")
> # Add legend
> legend("topright", inset=0.05, bty="n",
+   leg=c("unscaled", "scaled", "normal"),
+   lwd=6, lty=1, col=c("blue", "red", "green"))
```

Package *PerformanceAnalytics* for Risk and Performance Analysis

The package *PerformanceAnalytics* contains functions for calculating risk and performance statistics, such as the variance, skewness, kurtosis, beta, alpha, etc.

The function `data()` loads external data or listv data sets in a package.

`managers` is an *xts* time series containing monthly percentage returns of six asset managers (HAM1 through HAM6), the EDHEC Long-Short Equity hedge fund index, the S&P 500, and US Treasury 10-year bond and 3-month bill total returns.

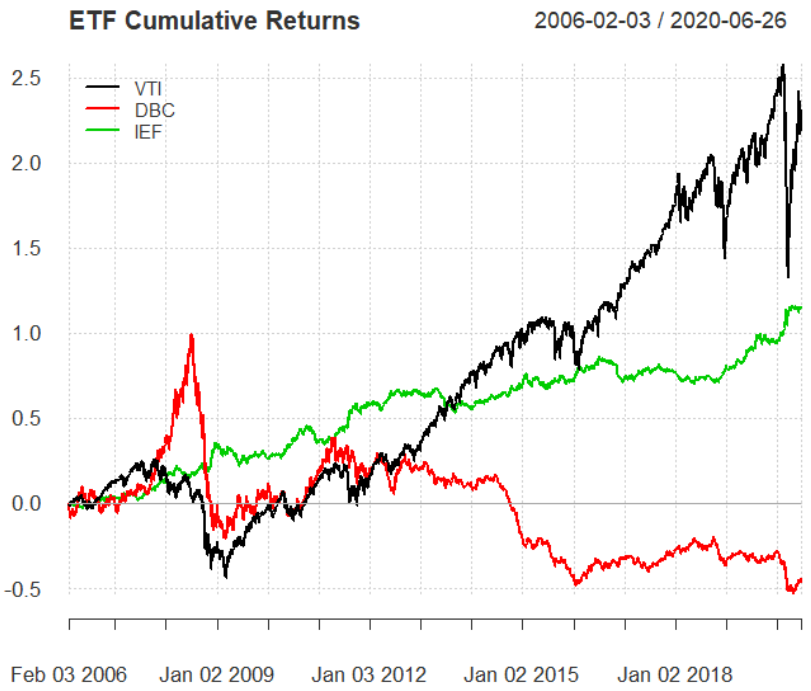
```
> # Load package PerformanceAnalytics
> library(PerformanceAnalytics)
> # Get documentation for package PerformanceAnalytics
> # Get short description
> packageDescription("PerformanceAnalytics")
> # Load help page
> help(package="PerformanceAnalytics")
> # List all objects in PerformanceAnalytics
> ls("package:PerformanceAnalytics")
> # List all datasets in PerformanceAnalytics
> data(package="PerformanceAnalytics")
> # Remove PerformanceAnalytics from search path
> detach("package:PerformanceAnalytics")
```

```
> perf_data <- unclass(data(
+     package="PerformanceAnalytics"))$results[, -(1:2)]
> apply(perf_data, 1, paste, collapse=" - ")
> # Load "managers" data set
> data(managers)
> class(managers)
> dim(managers)
> head(managers, 3)
```

Plots of Cumulative Returns

The function `chart.CumReturns()` from package *PerformanceAnalytics* plots the cumulative returns of a time series of returns.

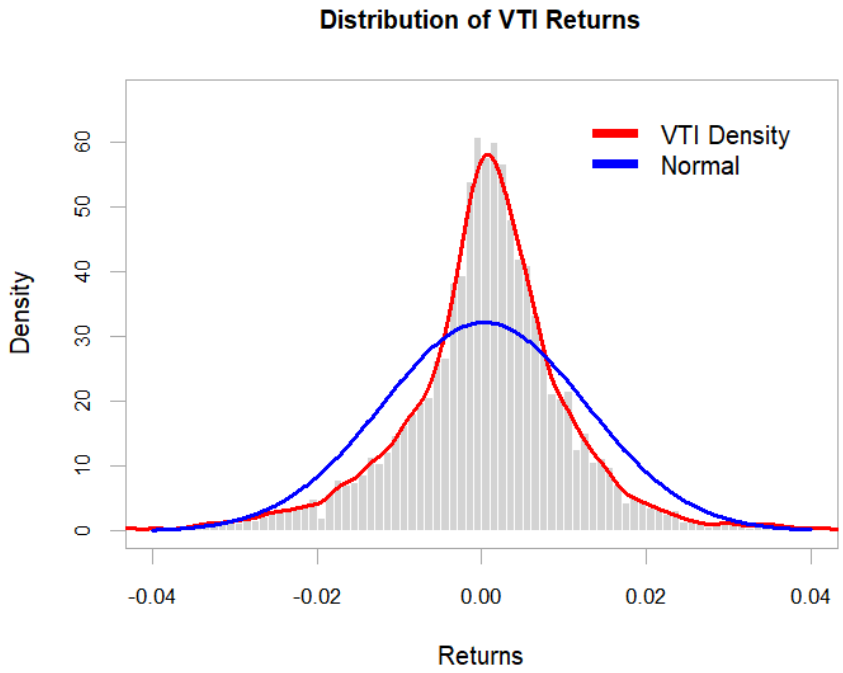
```
> # Load package "PerformanceAnalytics"
> library(PerformanceAnalytics)
> # Calculate ETF returns
> retsp <- rutils::etfenv$returns[, c("VTI", "DBC", "IEF")]
> retsp <- na.omit(retsp)
> # Plot cumulative ETF returns
> x11(width=6, height=5)
> chart.CumReturns(retsp, lwd=2, ylab="",
+   legend.loc="topleft", main="ETF Cumulative Returns")
```



The Distribution of Asset Returns

The function `chart.Histogram()` from package *PerformanceAnalytics* plots the histogram (frequency distribution) and the density of returns.

```
> retsp <- rutils::etfenv$returns$VTI
> retsp <- na.omit(retsp)
> x11(width=6, height=5)
> chart.Histogram(retsp, xlim=c(-0.04, 0.04),
+   colorset = c("lightgray", "red", "blue"), lwd=3,
+   main=paste("Distribution of", colnames(retsp), "Returns"),
+   methods = c("add.density", "add.normal"))
> legend("topright", inset=0.05, bty="n",
+  leg=c("VTI Density", "Normal"),
+  lwd=6, lty=1, col=c("red", "blue"))
```



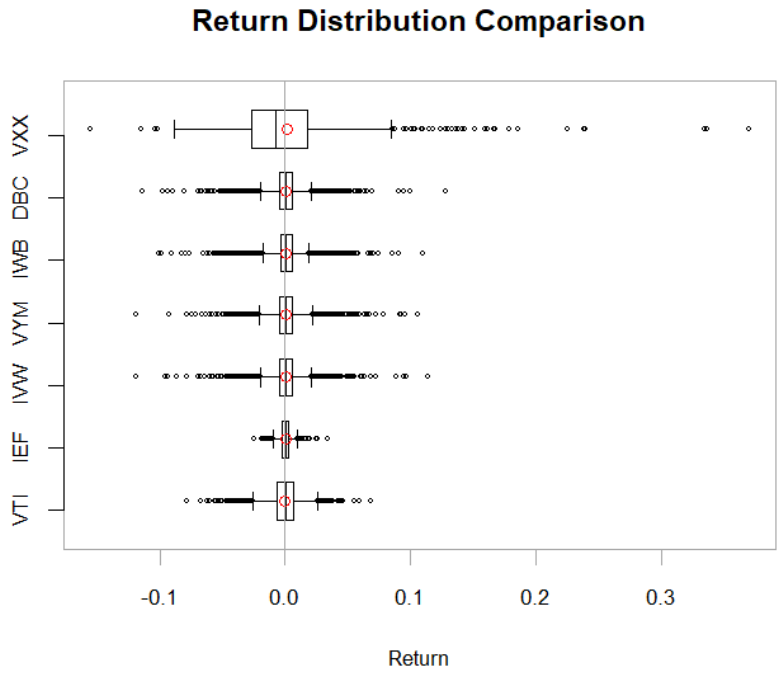
Boxplots of Returns

The function `chart.Boxplot()` from package *PerformanceAnalytics* plots a box-and-whisker plot for a distribution of returns.

The function `chart.Boxplot()` is a wrapper and calls the function `graphics::boxplot()` to plot the box plots.

A *box plot* (box-and-whisker plot) is a graphical display of a distribution of data:
The *box* represents the upper and lower quartiles,
The vertical lines (whiskers) represent values beyond the quartiles,
Open circles represent values beyond the nominal range (outliers).

```
> retsp <- rutils::etfenv$returns[,
+   c("VTI", "IEF", "IVW", "VYM", "IWB", "DBC", "VXX")]
> x11(width=6, height=5)
> chart.Boxplot(names=FALSE, retsp)
> par(cex.lab=0.8, cex.axis=0.8)
> axis(side=2, at=(1:NCOL(retsp))/7.5-0.05,labels=colnames(retsp))
```



The Median Absolute Deviation Estimator of Dispersion

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

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

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
> cluster <- makeCluster(ncores)  # Initialize compute cluster
> bootd <- parLapply(cluster, 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(cluster)  # 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
> retsp <- rutils::etfenv$returns$VTI
> retsp <- na.omit(retsp)
> nrows <- NROW(retsp)
> sd(retsp)
> mad(retsp)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:10000, function(x) {
+   samplev <- retsp[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
> cluster <- makeCluster(ncores)  # Initialize compute cluster
> clusterExport(cluster, c("nrows", "returns"))
> bootd <- parLapply(cluster, 1:10000,
+   function(x) {
+     samplev <- retsp[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 <- retsp[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> stopCluster(cluster)  # 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
> returns_sub <- (retsp - targetr)
> returns_sub <- ifelse(returns_sub < 0, returns_sub, 0)
> nrows <- NROW(returns_sub)
> # Calculate the downside deviation
> all.equal(sqrt(sum(returns_sub^2)/nrows),
+   drop(DownsideDeviation(retsp, MAR=targetr, method="full")))
> # Calculate the subset downside returns
> returns_sub <- (retsp - targetr)
> returns_sub <- returns_sub[returns_sub < 0]
> nrows <- NROW(returns_sub)
> # Calculate the downside deviation
> all.equal(sqrt(sum(returns_sub^2)/nrows),
+   drop(DownsideDeviation(retsp, MAR=targetr, method="subset")))
```

Drawdown Risk

The *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))
> draw_downs <- (closep - cummax(closep))
> # Extract the date index from the time series closep
> dates <- zoo::index(closep)
> # Calculate the maximum drawdown date and depth
> index_min <- which.min(draw_downs)
> date_min <- dates[index_min]
> max_drawdown <- draw_downs[date_min]
> # Calculate the drawdown start and end dates
> startd <- max(dates[(dates < date_min) & (draw_downs == 0)])
> endd <- min(dates[(dates > date_min) & (draw_downs == 0)])
> # dygraph plot of VTI drawdowns
> datav <- cbind(closep, draw_downs)
> 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(draw_downs)+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(date_min, "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, dates)
> yval <- max(closep)
> abline(v=xval, col="blue")
> text(x=xval, y=0.95*yval, "start drawdown", col="blue", cex=0.9)
> xval <- match(date_min, dates)
> abline(v=xval, col="red")
> text(x=xval, y=0.9*yval, "max drawdown", col="red", cex=0.9)
> xval <- match(endd, dates)
> 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.

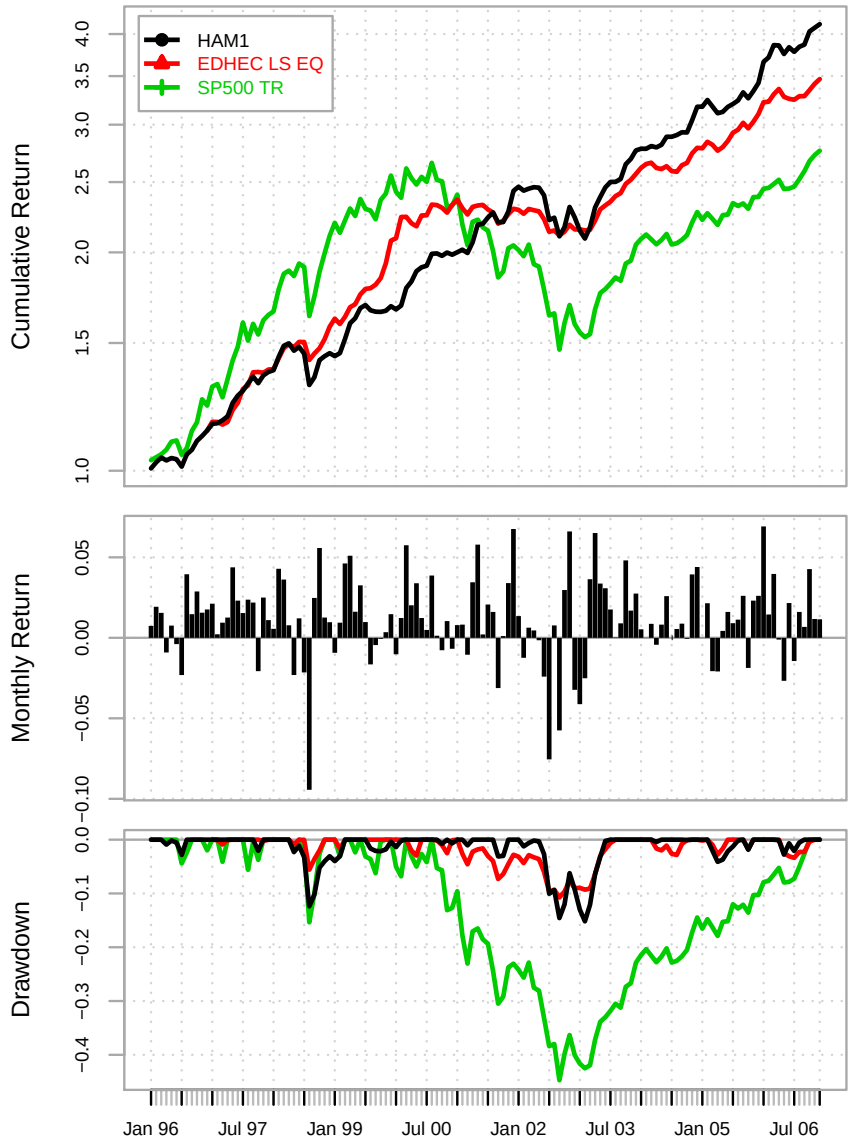
```
> library(xtable)
> library(PerformanceAnalytics)
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> retsp <- rutils::diffit(closep)
> # Calculate table of VTI drawdowns
> tablev <- PerformanceAnalytics::table.Drawdowns(retsp, geometric=FALSE)
> # Convert dates to strings
> tablev <- cbind(sapply(tablev[, 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	2022-06-16		-0.10	149.00	114.00	
2018-09-21	2018-12-24	2019-04-23	-0.10	146.00	65.00	81.00

PerformanceSummary Plots

The function `charts.PerformanceSummary()` from package *PerformanceAnalytics* plots three charts: cumulative returns, return bars, and drawdowns, for time series of returns.

```
> data(managers)
> charts.PerformanceSummary(ham1,
+   main="", lwd=2, ylog=TRUE)
```



The Loss Distribution of Asset Returns

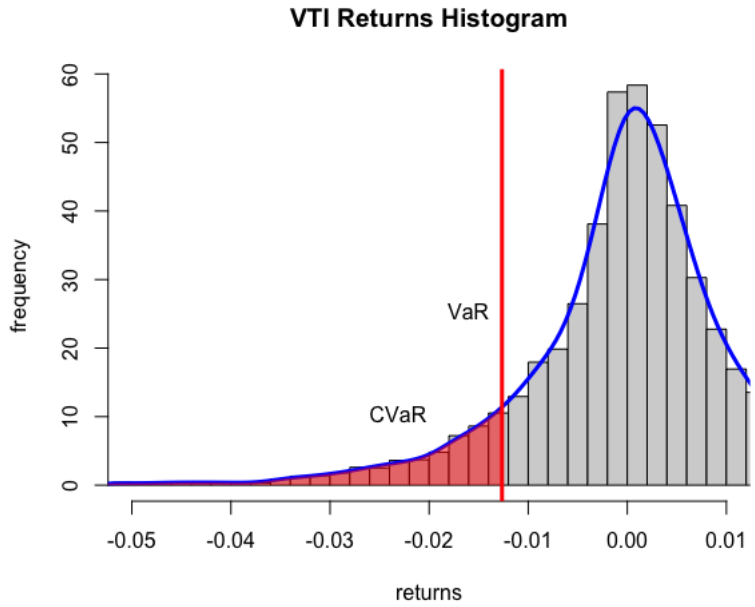
The distribution of returns has a long left tail of negative returns representing the risk of loss.

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
> retsp <- na.omit(rutils::etfenv$returns$VTI)
> confl <- 0.1
> varisk <- quantile(retsp, confl)
> cvar <- mean(retsp[retsp < 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(retsp, col="lightgrey",
+   xlab="returns", ylab="frequency", breaks=100,
+   xlim=c(-0.05, 0.01), freq=FALSE, main="VTI Returns Histogram")
> # Calculate density
> densv <- density(retsp, 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:

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

Where $\mathrm{f}(r)$ 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`.

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

```
> # Calculate VTI percentage returns
> retsp <- na.omit(rutils::etfenv$returns$VTI)
> confl <- 0.05
> # Calculate VaR as quantile
> varisk <- quantile(retsp, probs=confl)
> # Or by sorting
> sortv <- sort(as.numeric(retsp))
> indeks <- round(confl*NROW(retsp))
> varisk <- sortv[indeks]
> # PerformanceAnalytics VaR
> PerformanceAnalytics::VaR(retsp,
+   p=(1-confl), method="historical")
> all.equal(unname(varisk),
+   as.numeric(PerformanceAnalytics::VaR(retsp,
+   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(retsp, confl)
> # Calculate CVaR as expected loss
> cvar <- mean(retsp[retsp < varisk])
> # Or by sorting
> sortv <- sort(as.numeric(retsp))
> indeks <- round(confl*NROW(retsp))
> varisk <- sortv[indeks]
> cvar <- mean(sortv[1:indeks])
> # PerformanceAnalytics VaR
> PerformanceAnalytics::ETL(retsp,
+   p=(1-confl), method="historical")
> all.equal(cvar,
+   as.numeric(PerformanceAnalytics::ETL(retsp,
+   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
> risk_ret <-
+   PerformanceAnalytics::table.Stats(rutils::etfenv$returns)
> class(risk_ret)
> # Transpose the data frame
> risk_ret <- as.data.frame(t(risk_ret))
> # Add Name column
> risk_ret$Name <- rownames(risk_ret)
> # Add Sharpe ratio column
> risk_ret$Sharpe <- risk_ret$"Arithmetic Mean"/risk_ret$Stdev
> # Sort on Sharpe ratio
> risk_ret <- risk_ret[order(risk_ret$Sharpe, decreasing=TRUE), ]
```

	Sharpe	Skewness	Kurtosis
USMV	0.056	-0.962	23.16
IEF	0.048	-0.013	2.81
QUAL	0.045	-0.642	14.56
MTUM	0.040	-0.713	12.13
VLUE	0.033	-1.076	18.83
XLP	0.030	-0.120	8.82
XLY	0.028	-0.385	6.92
GLD	0.027	-0.332	6.20
XLV	0.026	0.070	10.10
VTI	0.024	-0.401	11.01
IWB	0.024	-0.411	10.21
VYM	0.024	-0.700	14.65
VTV	0.024	-0.680	13.75
XLU	0.024	0.006	12.45
IWD	0.024	-0.502	12.74
IVW	0.024	-0.329	8.58
TLT	0.022	-0.034	4.13
XLI	0.022	-0.386	7.59
IWF	0.022	-0.691	31.73
XLB	0.020	-0.387	5.39
XLK	0.018	0.064	6.77
EEM	0.017	0.019	15.26
XLE	0.016	-0.542	12.82
VNQ	0.016	-0.561	17.98
IVE	0.016	-0.491	10.14
XLF	0.011	-0.122	13.87
VEU	0.007	-0.524	11.54
SVXY	0.006	-17.520	603.90
DBC	0.000	-0.513	3.44
USO	-0.021	-1.182	14.69
VXX	-0.070	1.126	5.14

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*.

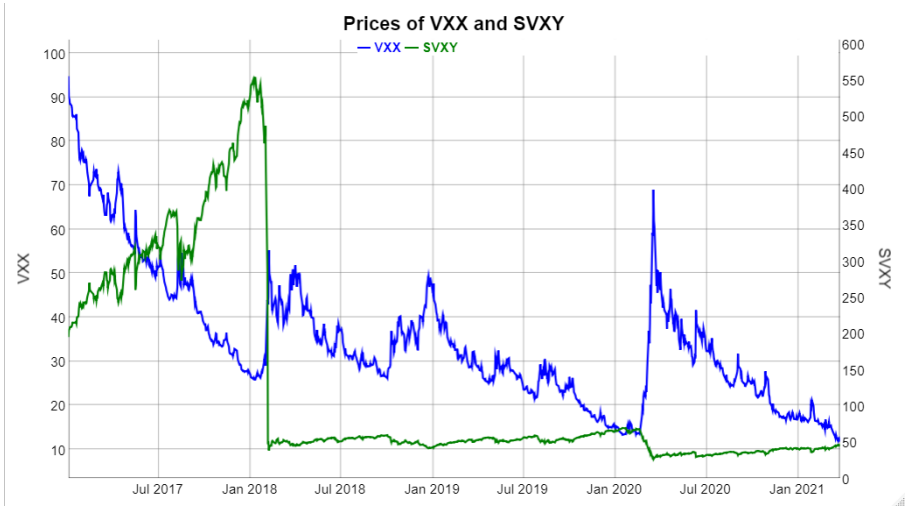
The *VXX* ETF is long the *VIX* index (effectively long an option), so it has positive skewness and small kurtosis, but negative returns (it's short market risk).

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	-0.070	1.13	5.14
SVXY	0.006	-17.52	603.90




```
> # dygraph plot of VXX versus SVXY
> prices <- na.omit(rutils::etfenv$prices[, c("VXX", "SVXY")])
> prices <- prices["2017/"]
> colnamev <- c("VXX", "SVXY")
> colnames(prices) <- colnamev
> dygraphs::dygraph(prices, 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=500)
```

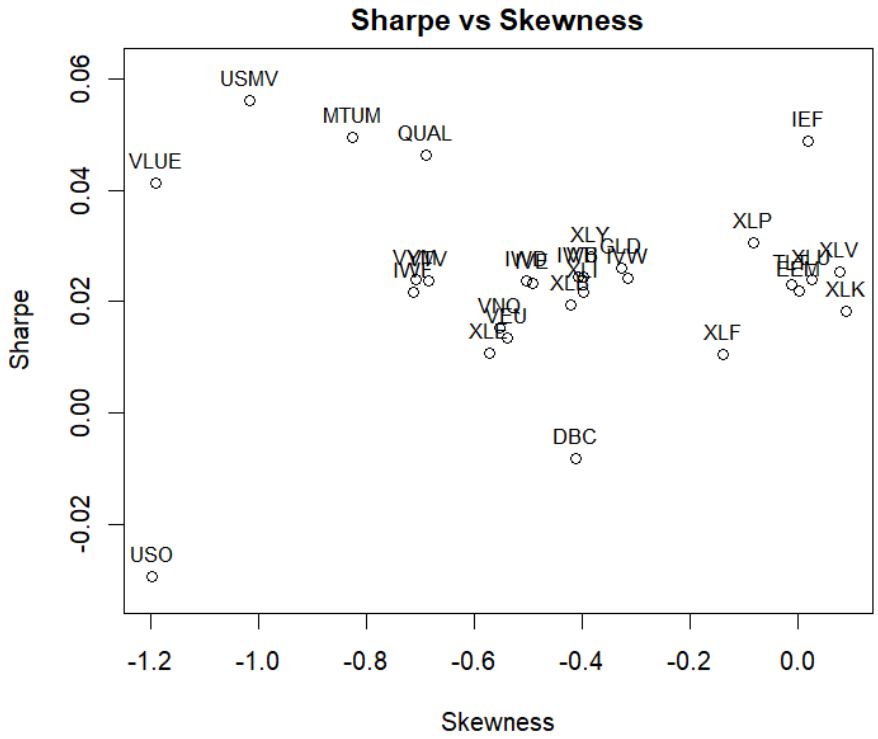
Skewness and Return Tradeoff

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



Risk-adjusted Return Measures

The *Sharpe ratio* $\mathrm{S_r}$ is equal to the excess returns (in excess of the risk-free return 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:

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

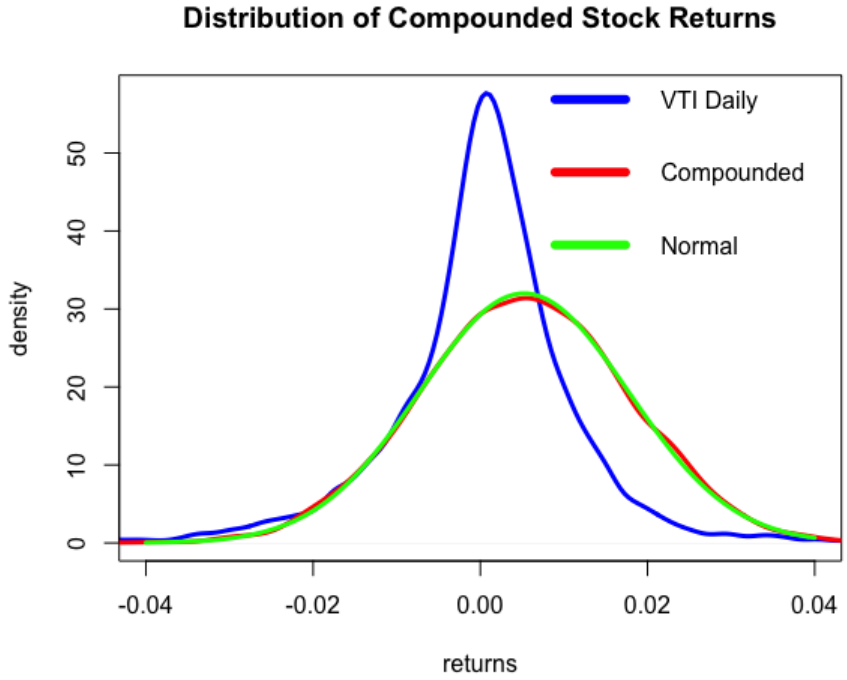
```
> library(PerformanceAnalytics)
> retsp <- rutils::etfenv$returns[, c("VTI", "IEF")]
> retsp <- na.omit(retsp)
> # Calculate the Sharpe ratio
> confl <- 0.05
> PerformanceAnalytics::SharpeRatio(retsp, p=(1-confl),
+   method="historical")
> # Calculate the Sortino ratio
> PerformanceAnalytics::SortinoRatio(retsp)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

Risk and Return of Compounded Stock Returns

Compounded stock returns become closer to normally distributed, and their skewness, kurtosis, and tail risks decrease significantly compared to daily returns.

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

```
> # Calculate VTI percentage returns
> retsp <- na.omit(rutils::etfenv$returns$VTI)
> retsp <- drop(zoo::coredata(retsp))
> nrows <- NROW(retsp)
> # Calculate compounded VTI returns
> holdp <- 252
> retc <- sqrt(holdp)*sapply(1:nrows, function(x) {
+     mean(retsp[sample.int(nrows, size=holdp, replace=TRUE)])
+ })  # end sapply
> # Calculate mean, standard deviation, skewness, and kurtosis
> datav <- cbind(retsp, retc)
> colnames(datav) <- c("VTI", "Agg")
> apply(datav, MARGIN=2, function(x) {
+   # Standardize the returns
+   meanval <- mean(x); stddev <- sd(x); x <- (x - meanval)/stddev
+   c(mean=meanval, stddev=stddev, skew=mean(x^3), kurt=mean(x^4))
+ })  # end sapply
> # Calculate the Sharpe and Dowd ratios
> confl <- 0.05
> sapply(colnames(datav), function(name) {
+   x <- datav[, name]; stddev <- sd(x)
+   varisk <- unname(quantile(x, probs=confl))
+   cvar <- mean(x[x < varisk])
+   ratio <- 1
+   if (name == colnames(datav)[2]) {ratio <- holdp}
+   sqrt(252/ratio)*mean(x)/c(Sharpe=stddev, Dowd=-varisk, DowdC=-cvar)
+ })  # end sapply
```




```
> # Plot the densities of returns
> x11(width=6, height=5)
> par(mar=c(4, 4, 3, 1), oma=c(0, 0, 0, 0))
> plot(density(retsp), t="l", lwd=3, col="blue",
+      xlab="returns", ylab="density", xlim=c(-0.04, 0.04),
+      main="Distribution of Compounded Stock Returns")
> lines(density(retc), t="l", col="red", lwd=3)
> curve(expr=dnorm(x, mean=mean(retc), sd=sd(retc)), col="green", lw
> legend("topright", legend=c("VTI Daily", "Compounded", "Normal"),
+  inset=-0.1, bg="white", lty=1, lwd=6, col=c("blue", "red", "green
```

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 ETF prices from rutils::etfenv$prices
> pricets <- rutils::etfenv$prices
> pricets <- zoo::na.locf(pricets, na.rm=FALSE)
> pricets <- zoo::na.locf(pricets, fromLast=TRUE)
> dates <- zoo::index(pricets)
> # Calculate simple dollar returns
> retsd <- rutils::diffit(pricets)
> # Or
> # retsd <- lapply(pricets, rutils::diffit)
> # retsd <- rutils::do_call(cbind, retsd)
> # Calculate percentage returns
> retsp <- retsd/rutils::lagit(pricets, lagg=1, pad_zeros=FALSE)
> # Calculate log returns
> retsl <- rutils::diffit(log(pricets))
```

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.

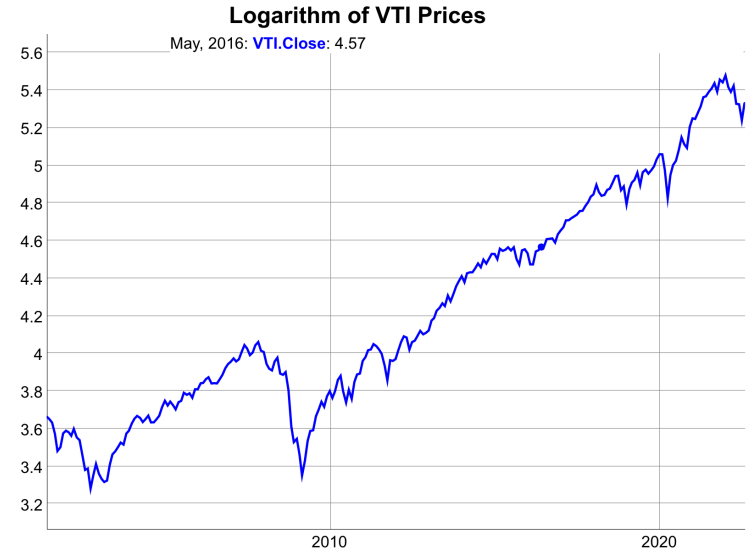
The sum of the percentage returns (without compounding): $\sum_{i=1}^{n} r_i^p$ represents the wealth path from owning a *fixed dollar amount* of stock.

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.

Rebalancing requires borrowing from a *margin account*, and it also incurs trading costs.

The logarithm of the wealth of a *fixed number of shares* is often used to compare investments, and it's approximately equal to the sum of the percentage returns.




```
> # Set the initial dollar returns
> retsd[1, ] <- pricets[1, ]
> # Calculate prices from dollar returns
> pricesn <- cumsum(retsd)
> all.equal(pricesn, pricets)
> # Compound the percentage returns
> pricesn <- cumprod(1+retsp)
> # Set the initial prices
> pricesi <- as.numeric(pricets[1, ])
> pricesn <- lapply(1:NCOL(pricesn), function (i)
+     pricesi[i]*pricesn[, i])
> pricesn <- rutils::do_call(cbind, pricesn)
> # Or
> # pricesn <- t(t(pricesn)*pricesi)
> all.equal(pricesn, pricets, check.attributes=FALSE)
> # Plot log VTI prices
> endp <- rutils::calc_endpoints(rutils::etfenv$VTI, interval="month
> dygraphs::dygraph(log(quantmod::Cl(rutils::etfenv$VTI)[endp]),
+   main="Logarithm of VTI Prices") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
```

Funding Costs of Single Asset Rebalancing

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

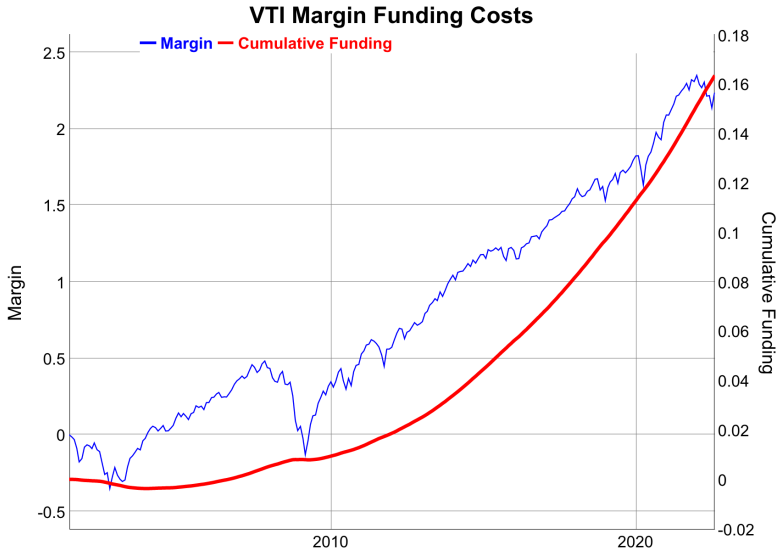
The cash in the *margin account* can be positive (accumulated profits) or negative (losses).

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

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

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

```
> # Calculate percentage VTI returns
> pricets <- rutils::etfenv$prices$VTI
> pricets <- na.omit(pricets)
> retsp <- rutils::diffit(pricets)/
+   rutils::lagit(pricets, lagg=1, pad_zeros=FALSE)
```




```
> # Funding rate per day
> frate <- 0.01/252
> # Margin account
> margin <- cumsum(retsp)
> # Cumulative funding costs
> fcosts <- cumsum(frate*margin)
> # Add funding costs to margin account
> margin <- (margin + fcosts)
> # dygraph plot of margin and funding costs
> datav <- cbind(margin, fcosts)
> colnamev <- c("Margin", "Cumulative Funding")
> colnames(datav) <- colnamev
> endp <- rutils::calc_endpoints(datav, interval="months")
> dygraphs::dygraph(datav[endp], 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)
+   dyLegend(show="always", width=500)
```

Transaction Costs of Trading

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

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

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

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

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

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

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

The *bid-offer spread* for liquid stocks can be assumed to be about 10 basis points (bps).

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

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

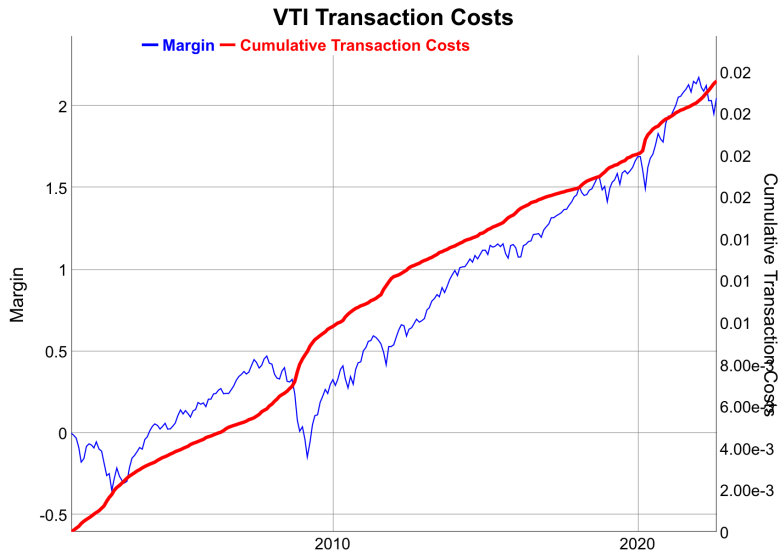
Transaction Costs of Single Asset Rebalancing

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

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

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

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




```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(abs(retsp))/2
> # Subtract transaction costs from margin account
> margin <- cumsum(retsp)
> margin <- (margin - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(margin, costs)
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endp], 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=500)
```

Combining the Returns of Multiple Assets

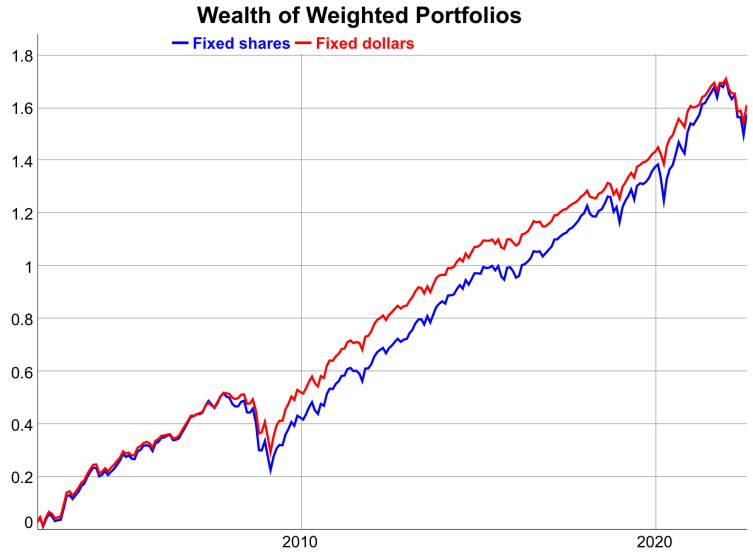
Adding the weighted dollar returns is equivalent to buying a *fixed number of shares* (aka *Fixed Share Allocation* or FSA) proportional to the weights.

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

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




```
> # Calculate VTI and IEF dollar returns
> pricets <- rutils::etfenv$prices[, c("VTI", "IEF")]
> pricets <- na.omit(pricets)
> retsd <- rutils::diffit(pricets)
> # Calculate VTI and IEF percentage returns
> retsp <- retsd/rutils::lagit(pricets, lagg=1, pad_zeros=FALSE)
> # Set the initial dollar returns
> retsd[1, ] <- pricets[1, ]
> # Wealth of fixed shares equal to $0.5 each (without rebalancing)
> weightv <- c(0.5, 0.5)  # dollar weights
> # Scale the dollar returns using the dollar weights
> pricesi <- as.numeric(pricets[1, ])
> wealth_fsa <- cumsum(retsd %*% (weightv/pricesi))
> # Or using percentage returns
> wealth_fsa2 <- cumprod(1+retsp) %*% weightv
> all.equal(wealth_fsa, drop(wealth_fsa2))
```

```
> # Wealth of fixed dollars (with rebalancing)
> wealth_fda <- cumsum(retsp %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(log(wealth_fsa), wealth_fda)
> wealthv <- xts::xts(wealthv, zoo::index(pricets))
> 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)
> endp <- rutils::calc_endpoints(retsp, interval="months")
> dygraphs::dygraph(wealthv[endp], 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=500)
```

Transaction Costs of Weighted Portfolio Rebalancing

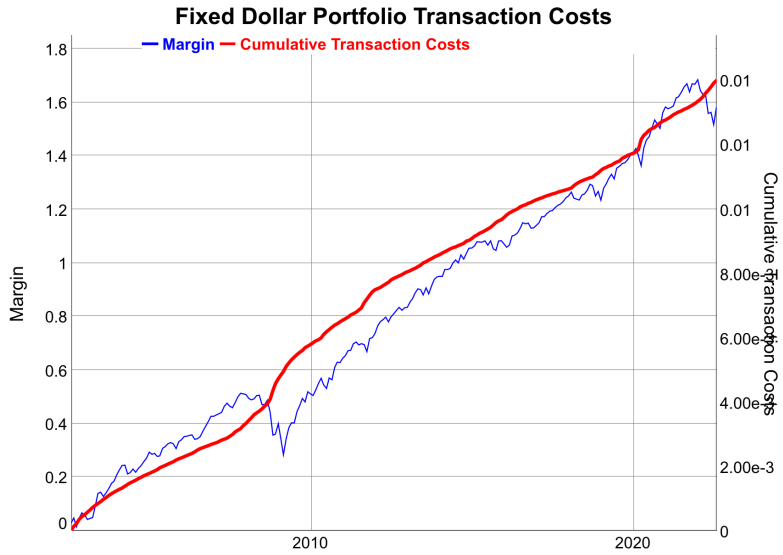
Maintaining a *fixed dollar amount* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

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-offer 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* $\sum_{i=1}^{t} c_i^r$ must be subtracted from the *margin account* m_t: $m_t - \sum_{i=1}^{t} c_i^r$.




```
> # Margin account for fixed dollars (with rebalancing)
> margin <- cumsum(retsp %*% weightv)
> # Cumulative transaction costs
> costs <- bid_offer*cumsum(abs(retsp) %*% weightv)/2
> # Subtract transaction costs from margin account
> margin <- (margin - costs)
> # dygraph plot of margin and transaction costs
> datav <- cbind(margin, costs)
> datav <- xts::xts(datav, zoo::index(pricets))
> colnamev <- c("Margin", "Cumulative Transaction Costs")
> colnames(datav) <- colnamev
> dygraphs::dygraph(datav[endp], 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=500)
```

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

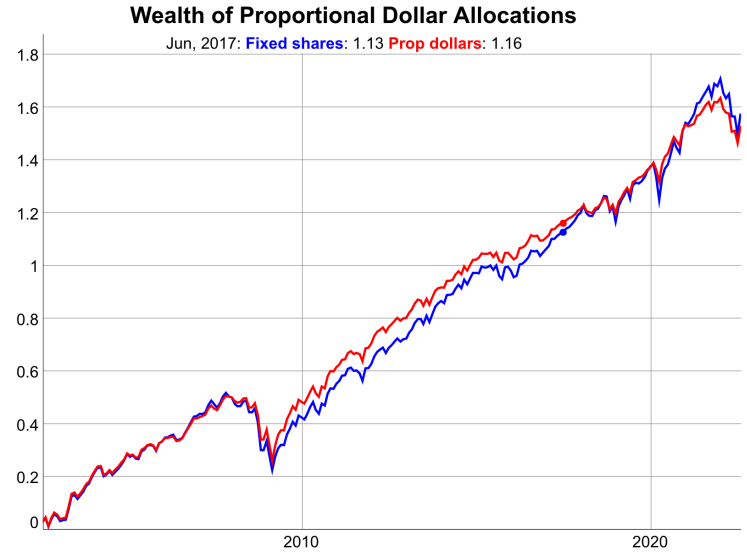
The total portfolio wealth at time t is equal to the wealth at time $t-1$ multiplied by the weighted returns: $w_t = w_{t-1}(1 + \bar{r}_t)$.

The dollar amount of stock i at time t increases by $\omega_i r_i$ so it's equal to $\omega_i w_{t-1}(1 + r_i)$, 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:

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

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




```
> # Wealth of fixed shares (without rebalancing)
> wealth_fsa <- cumsum(retsd %*% (weightv/pricesi))
> # Or compound the percentage returns
> wealth_fsa <- cumprod(1+retsp) %*% weightv
> # Wealth of proportional allocations (with rebalancing)
> wealth_pda <- cumprod(1 + retsp %*% weightv)
> wealthv <- cbind(wealth_fsa, wealth_pda)
> wealthv <- xts::xts(wealthv, zoo::index(pricets))
> 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[endp],
+   main="Wealth of Proportional Dollar Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

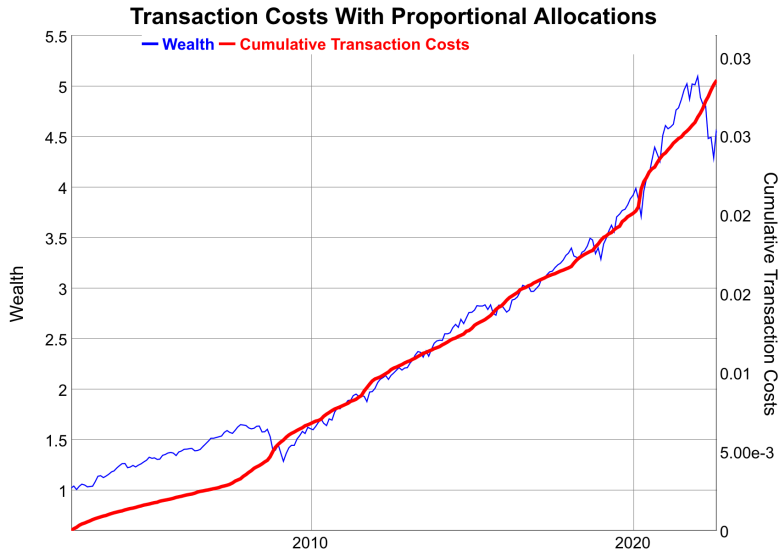
Transaction Costs With Proportional Dollar Allocations

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

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

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

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




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

```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealth_pda, costs)
> wealthv <- xts::xts(wealthv, zoo::index(pricets))
> colnamev <- c("Wealth", "Cumulative Transaction Costs")
> colnames(wealthv) <- colnamev
> dygraphs::dygraph(wealthv[endp],
+   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=500)
```

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)
> wealth_fsa <- drop(apply(retsp, 2, function(x) cumprod(1+x)) %*% w
> # Wealth of proportional dollar allocations (with rebalancing)
> wealth_pda <- cumprod(1 + retsp %*% weightv) - 1
> # Wealth of proportional target allocation (with rebalancing)
> retsp <- zoo::coredata(retsp)
> threshold <- 0.05
> wealthv <- matrix(nrow=NROW(retsp), ncol=2)
> colnames(wealthv) <- colnames(retsp)
> wealthv[1, ] <- weightv
> for (it in 2:NROW(retsp)) {
+   # Accrue wealth without rebalancing
+   wealthv[it, ] <- wealthv[it-1, ]*(1 + retsp[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(wealth_pda, wealthv)
> wealthv <- xts::xts(wealthv, zoo::index(pricets))
> 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=500)
```

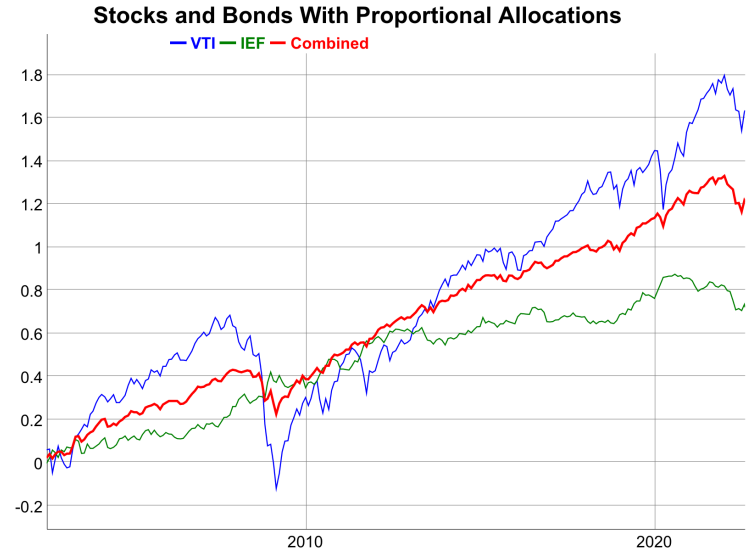
Stock and Bond Portfolio With Proportional Dollar Allocations

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

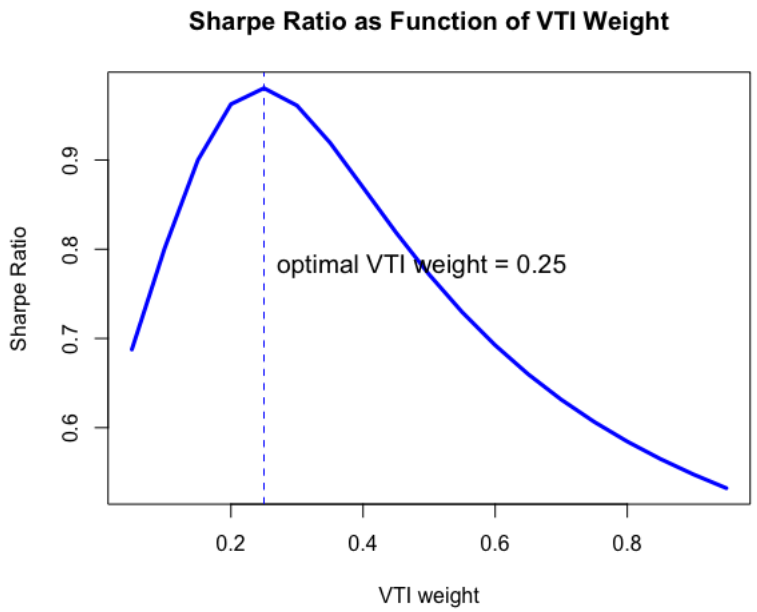
```
> # Wealth of proportional allocations
> wealthv <- cumprod(1 + retsp)
> # Calculate a vector of monthly end points
> endp <- rutils::calc_endpoints(retsp, interval="months")
> # Plot cumulative log wealth
> dygraphs::dygraph(log(wealthv[endp]),
+   main="Stocks and Bonds With Proportional Allocations") %>%
+   dyOptions(colors=c("blue", "green", "blue", "red")) %>%
+   dySeries("Combined", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

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 0.25.

```
> # Calculate the Sharpe ratios
> sqrt(252)*sapply(retsp, 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)
+   retsp <- (retsp[, 1:2] %*% weightv)
+   mean(retsp)/sd(retsp)
+ })  # 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)
+   retsp <- (retsp[, 1:2] %*% weightv)
+   -mean(retsp)/sd(retsp)
+ }  # end calc_sharpe
> optv <- optimize(calc_sharpe, interval=c(0, 1))
> weightm <- optv$minimum
```



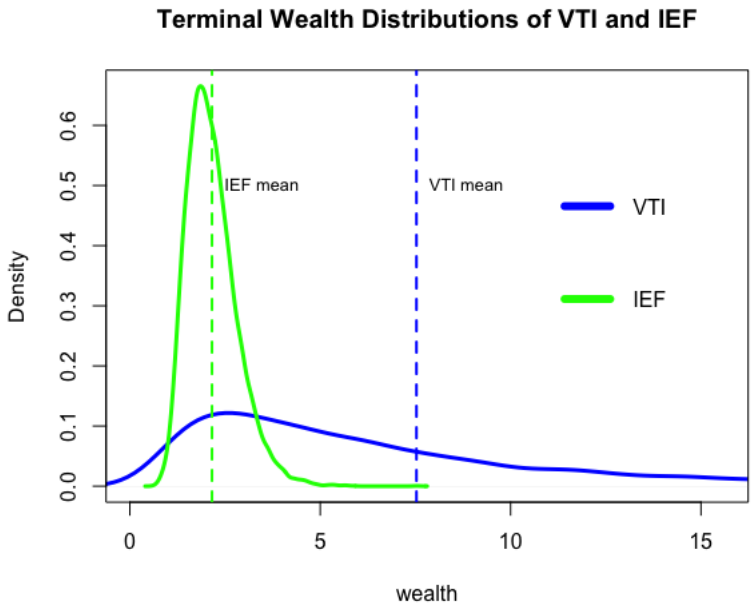

```
> # Plot Sharpe ratios
> plot(x=weightv, y=sharpev,
+      main="Sharpe Ratio as Function of VTI Weight",
+      xlab="VTI weight", ylab="Sharpe Ratio",
+      t="l", lwd=3, col="blue")
> abline(v=weightm, lty="dashed", lwd=1, col="blue")
> text(x=weightm, y=0.7*max(sharpev), pos=4, cex=1.2,
+  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

The Distributions of Terminal Wealth From 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.

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




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

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

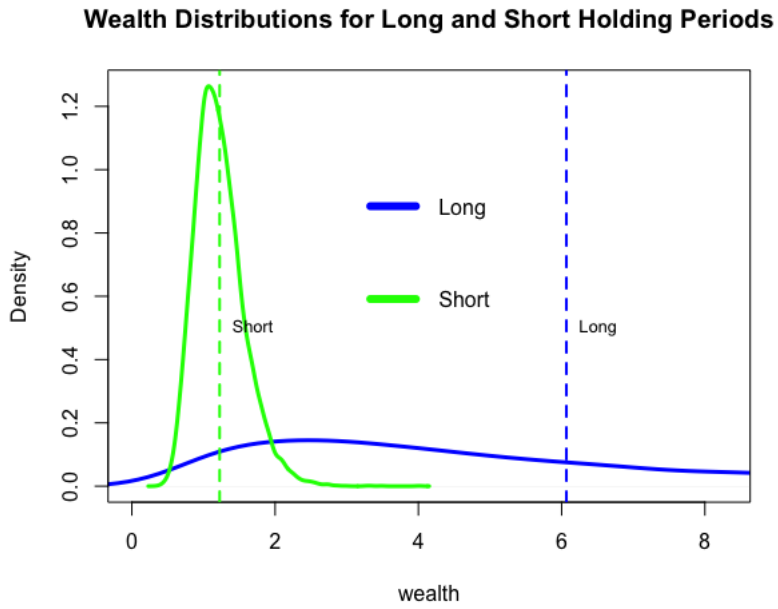
The Distribution of Stock Wealth and Holding Period

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

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

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

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




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

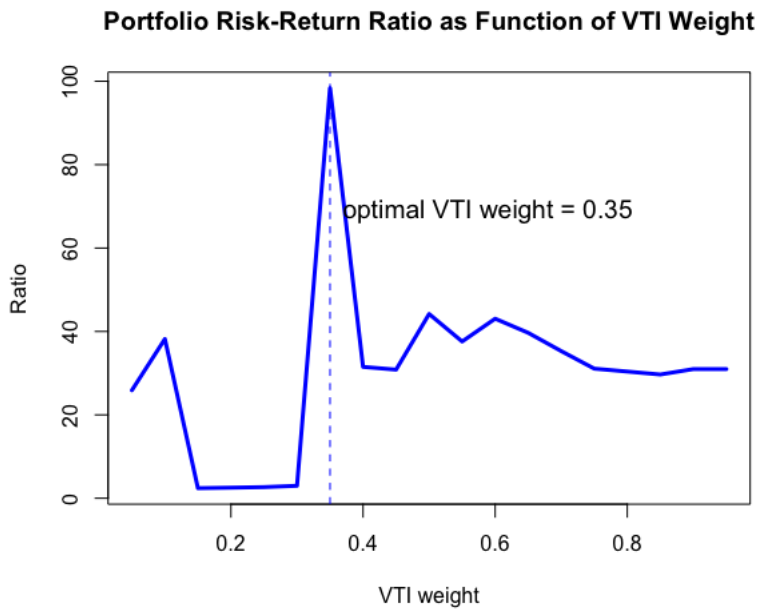
```
> # Plot the stock wealth for long and short holding periods
> wealth1 <- wealthm[, 9]
> wealth2 <- wealthm[, 1]
> mean1 <- mean(wealth1); mean2 <- mean(wealth2)
> dens1 <- density(wealth1); dens2 <- density(wealth2)
> plot(dens1, col="blue", lwd=3, xlab="wealth",
+      xlim=c(0, 2*max(dens2$x)), ylim=c(0, max(dens2$y)),
+      main="Wealth Distributions for Long and Short Holding Periods
> lines(dens2, col="green", lwd=3)
> abline(v=mean1, col="blue", lwd=2, lty="dashed")
> text(x=mean1, y=0.5, labels="Long", pos=4, cex=0.8)
> abline(v=mean2, col="green", lwd=2, lty="dashed")
> text(x=mean2, y=0.5, labels="Short", pos=4, cex=0.8)
> legend(x="top", legend=c("Long", "Short"),
+  inset=0.1, cex=1.0, bg="white", bty="n",
+  lwd=6, lty=1, col=c("blue", "green"))
```

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 0.35.

```
> # Calculate the distributions of portfolio wealth
> weightv <- seq(0.05, 0.95, 0.05)
> wealthm <- sapply(bootd, function(retsp) {
+   sapply(weightv, function(weight) {
+     prod(1 + retsp %*% c(weight, 1-weight))
+   })  # end sapply
+ })  # end sapply
> wealthm <- t(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)))
```

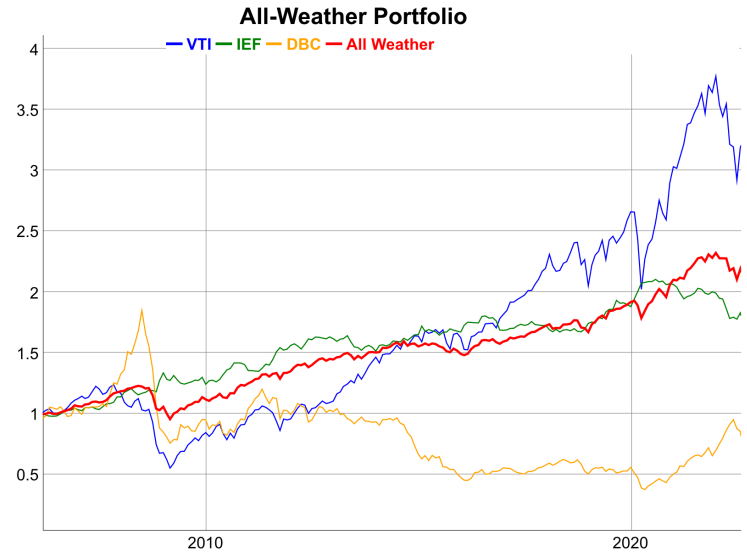
The *All-Weather* Portfolio

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

```
> # Calculate cumulative wealth from returns
> wealthv <- cumprod(1+retsp)
> # Calculate a vector of monthly end points
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> # dygraph all-weather wealth
> dygraphs::dygraph(wealthv[endp], main="All-Weather Portfolio") %>%
+   dyOptions(colors=c("blue", "green", "orange", "red")) %>%
+   dySeries("All Weather", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # 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,
+   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 VTI returns
> retsp <- na.omit(rutils::etfenv$returns$VTI["2008/2009"])
> dates <- zoo::index(retsp)
> nrows <- NROW(retsp)
> retsp <- drop(zoo::coredata(retsp))
> # 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

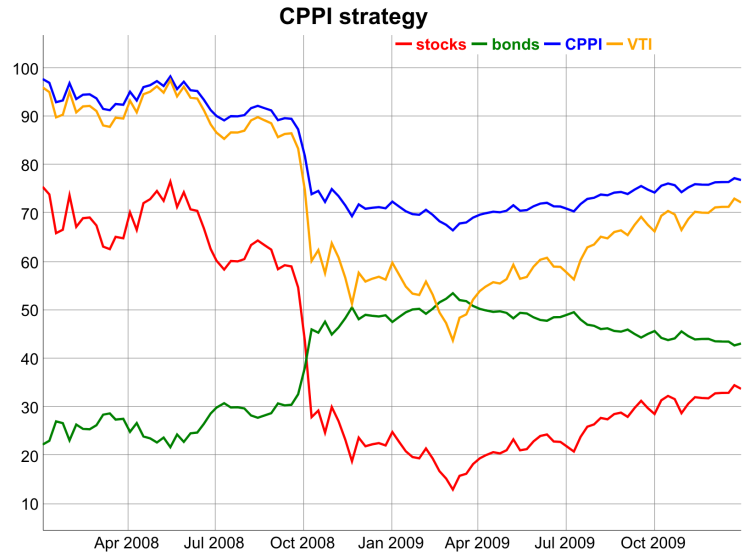
If the stock price changes and the portfolio value becomes P_t, then the dollar amount invested in stocks must be adjusted to: $C * (P_t - F)$. The amount invested in stocks changes both because the stock price changes and because of rebalancing with the zero-coupon bonds.

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]*retsp[t]
+   stockv[t] <- min(coeff*(portfv[t] - bfloor), portfv[t])
+   bondv[t] <- (portfv[t] - stockv[t])
+ }  # end for
> # dygraph plot of CPPI strategy
> pricevti <- 100*cumprod(1+retsp)
> datav <- xts::xts(cbind(stockv, bondv, portfv, pricevti), dates)
> colnames(datav) <- c("stocks", "bonds", "CPPI", "VTI")
> endp <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endp], main="CPPI strategy") %>%
+   dyOptions(colors=c("red", "green", "blue", "orange"), strokeWidt
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy

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

This means that the allocations a_i are proportional to the *standardized prices* ($\frac{p_i}{\sigma_i^d}$ - the dollar amounts of stocks with unit dollar volatilities): $a_i \propto \frac{p_i}{\sigma_i^d}$, where σ_i^d is the dollar volatility.

But the *standardized prices* are equal to the inverse of the percentage volatilities σ_i: $\frac{p_i}{\sigma_i^d} = \frac{1}{\sigma_i}$, so the allocations a_i are proportional to the inverse of the percentage volatilities $a_i \propto \frac{1}{\sigma_i}$.

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

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

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

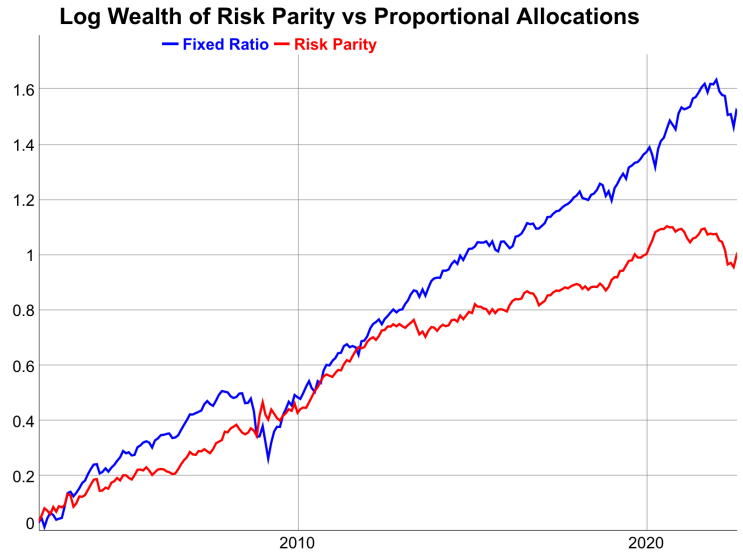
```
> # Calculate dollar and percentage returns for VTI and IEF.
> pricets <- rutils::etfenv$prices[, c("VTI", "IEF")]
> pricets <- na.omit(pricets)
> retsd <- rutils::diffit(pricets)
> retsp <- retsd/rutils::lagit(pricets, lagg=1, pad_zeros=FALSE)
> # Calculate wealth of proportional allocations.
> weightv <- c(0.5, 0.5)
> retsw <- retsp %*% weightv
> wealth_pda <- cumprod(1 + retsw)
> # Calculate rolling percentage volatility.
> look_back <- 21
> volat <- HighFreq::roll_var(retsp, look_back=look_back)
> iszero <- (rowSums(volat) == 0)
> volat[iszero, ] <- 1
> # Calculate the risk parity portfolio allocations.
> alloc <- lapply(1:NCOL(pricets),
+   function(x) weightv[x]/volat[, x])
> alloc <- do.call(cbind, alloc)
> # Scale allocations to 1 dollar total.
> alloc <- alloc/rowSums(alloc)
> # Lag the allocations
> alloc <- rutils::lagit(alloc)
> # Calculate wealth of risk parity.
> retsw <- rowSums(retsp*alloc)
> wealth_risk_parity <- cumprod(1 + retsw)
```

Risk Parity Strategy Performance

The risk parity strategy for *VTI* and *IEF* has a higher *Sharpe ratio* than the fixed ratio strategy because it's more overweight bonds, which is also why it has lower absolute returns.

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

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



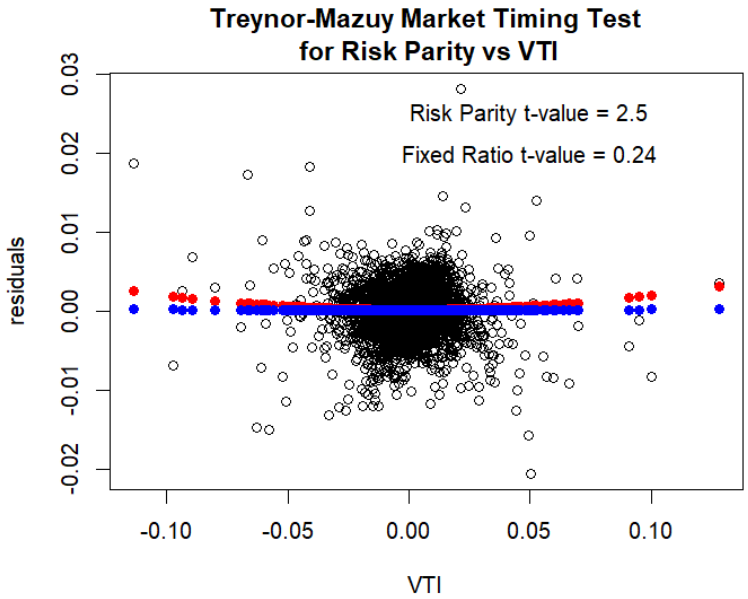

```
> # Calculate the log wealths.
> wealthv <- log(cbind(wealth_pda, wealth_risk_parity))
> wealthv <- xts::xts(wealthv, zoo::index(pricets))
> colnames(wealthv) <- c("Fixed Ratio", "Risk Parity")
> # Calculate the Sharpe ratios.
> sqrt(252)*sapply(rutils::diffit(wealthv), function (x) mean(x)/sd
> # Plot a dygraph of the log wealths.
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(wealthv[endp],
+   main="Log Wealth of Risk Parity vs Proportional Allocations") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Risk Parity Strategy Market Timing Skill

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

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

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



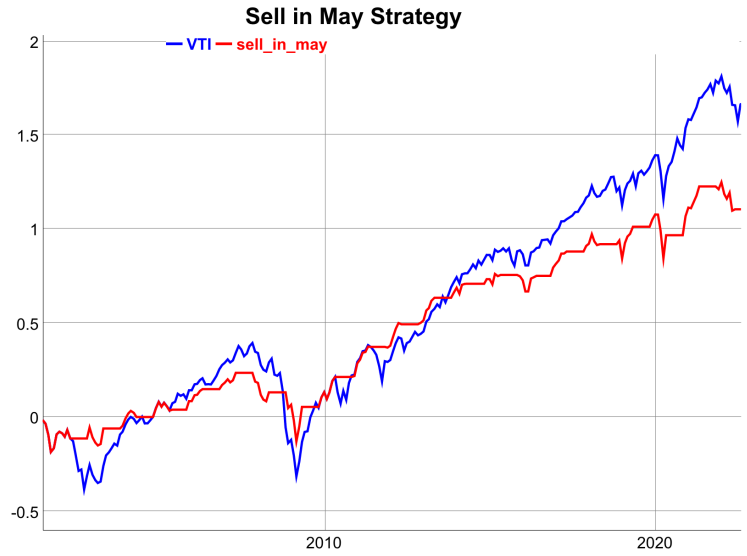

```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> retsrp <- rutils::diffit(wealthv)
> vti <- retsp$VTI
> design <- cbind(retsrp, vti, vti^2)
> design <- na.omit(design)
> colnames(design)[1:2] <- c("fixed", "risk_parity")
> colnames(design)[4] <- "treynor"
> model <- lm(risk_parity ~ VTI + treynor, data=design)
> summary(model)
> # Plot residual scatterplot
> residuals <- (design$risk_parity - model$coeff[2]*vti)
> residuals <- model$residuals
> x11(width=6, height=5)
> plot.default(x=vti, y=residuals, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Risk Parity vs VTI", line=0.5)
> # Plot fitted (predicted) response values
> fittedv <- (model$coeff["(Intercept)"] + model$coeff["treynor"]*vti^2)
> points.default(x=vti, y=fittedv, pch=16, col="red")
> text(x=0.05, y=0.8*max(residuals), paste("Risk Parity t-value =", round(summary(model)$coeff["treynor", "t value"], 2)))
```

```
> # Test for fixed ratio market timing of VTI using Treynor-Mazuy te
> model <- lm(fixed ~ VTI + treynor, data=design)
> summary(model)
> # Plot fitted (predicted) response values
> fittedv <- (model$coeff["(Intercept)"] + model$coeff["treynor"]*vt
> points.default(x=vti, y=fittedv, pch=16, col="blue")
> text(x=0.05, y=0.6*max(residuals), paste("Fixed Ratio t-value =",
```

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

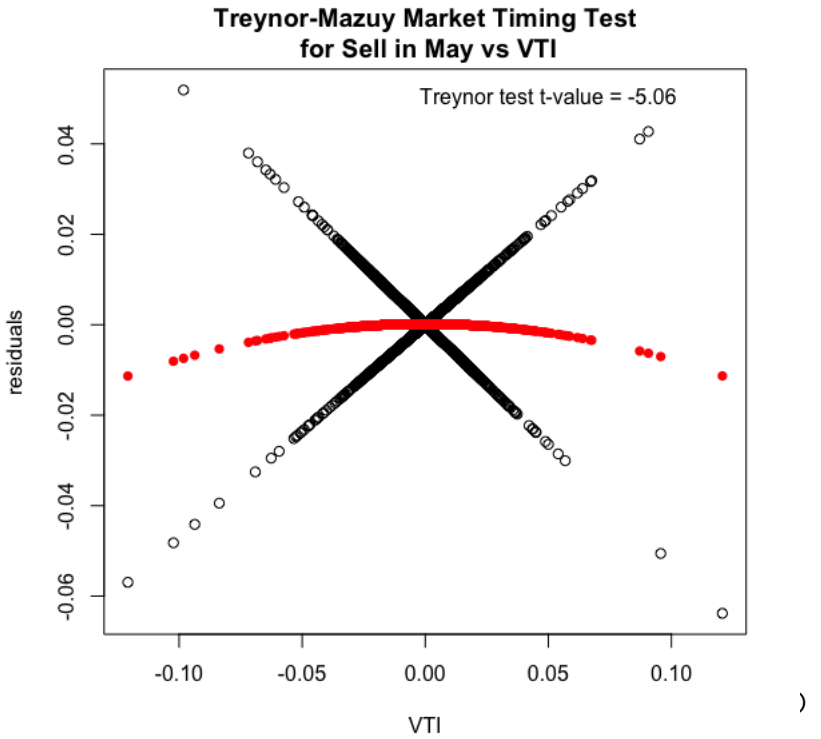



```
> # Plot wealth of Sell in May strategy
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Sell in May Strateg
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # OR: Open x11 for plotting
> 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,
+   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
> design <- cbind(wealth$sell_in_may, 0.5*(retsp+abs(retsp)), retsp
> colnames(design) <- c("VTI", "merton", "treynor")
> # Perform Merton-Henriksson test
> model <- lm(sell_inmay ~ VTI + merton, data=design)
> summary(model)
> # Perform Treynor-Mazuy test
> model <- lm(sell_inmay ~ VTI + treynor, data=design)
> summary(model)
> # Plot Treynor-Mazuy residual scatterplot
> residuals <- (sell_inmay - model$coeff[2]*retsp)
> plot.default(x=retsp, y=residuals, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Sell in May vs
> # Plot fitted (predicted) response values
> fittedv <- (model$coeff["(Intercept)"] +
+         model$coeff["treynor"]*retsp^2)
> points.default(x=retsp, y=fittedv, pch=16, col="red")
> text(x=0.05, y=0.8*max(residuals), paste("Treynor test t-value ="
```



Seasonal Overnight Market Anomaly

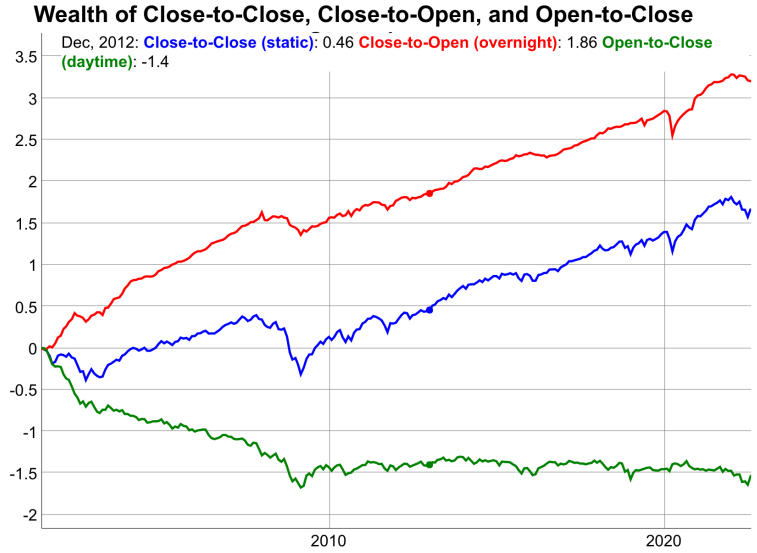
The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the daytime returns.

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

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

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

The *Overnight Market Anomaly* has mostly disappeared after the 2008–2009 financial crisis.




```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> # Calculate the close-to-close log returns, the intraday
> # open-to-close returns and the overnight close-to-open returns.
> close_close <- rutils::diffit(closep)
> colnames(close_close) <- "close_close"
> open_close <- (closep - openp)
> colnames(open_close) <- "open_close"
> close_open <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(close_open) <- "close_open"
```

```
> # Calculate Sharpe and Sortino ratios
> wealthv <- cbind(close_close, close_open, open_close)
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot log wealth
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp],
+   main="Wealth of Close-to-Close, Close-to-Open, and Open-to-Close
+   dySeries(name="close_close", label="Close-to-Close (static)", st
+   dySeries(name="close_open", label="Close-to-Open (overnight)", s
+   dySeries(name="open_close", label="Open-to-Close (daytime)", str
+   dyLegend(width=600)
```

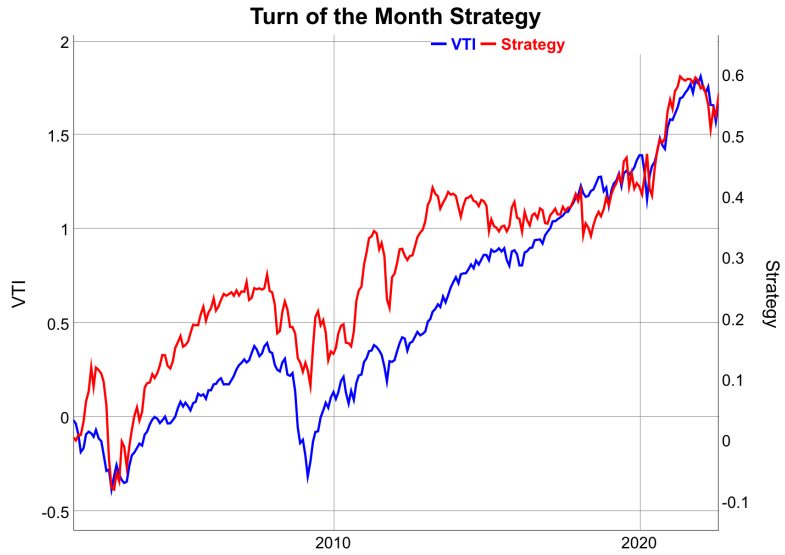
Turn of the Month Effect

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

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

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

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




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

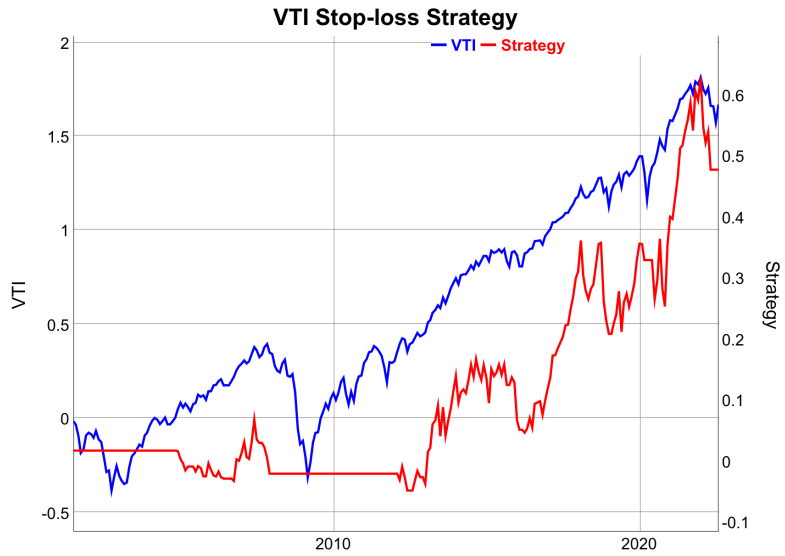
```
> # Combine data
> wealthv <- cbind(retsp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI Turn of the Month strategy
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp],
+   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")
```

Stop-loss Rules

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

For example, a simple stop-loss rule is to sell the stock if its price drops by 5% below the recent maximum price, and buy it back when the price recovers.

```
> # Calculate the VTI returns
> retsp <- na.omit(rutils::etfenv$returns$VTI)
> dates <- zoo::index(retsp)
> retsp <- drop(coredata(retsp))
> nrows <- NROW(retsp)
> # Simulate stop-loss strategy
> stopl <- 0.05
> maxp <- 0.0
> retc <- 0.0
> pnls <- retsp
> for (i in 1:(nrows-1)) {
+ # Calculate drawdown
+   retc <- retc + retsp[i]
+   maxp <- max(maxp, retc)
+   dd <- (retc - maxp)
+ # Check for stop-loss
+   if (dd < -stopl*maxp)
+     pnls[i+1] <- 0
+ }  # end for
> # Same but without using explicit loops
> cumsumv <- cumsum(retsp)
> cummaxv <- cummax(cumsumv)
> dd <- (cumsumv - cummaxv)
> pnls2 <- retsp
> isdd <- rutils::lagit(dd < -stopl*cummaxv)
> pnls2 <- ifelse(isdd, 0, pnls2)
> all.equal(pnls, pnls2)
```

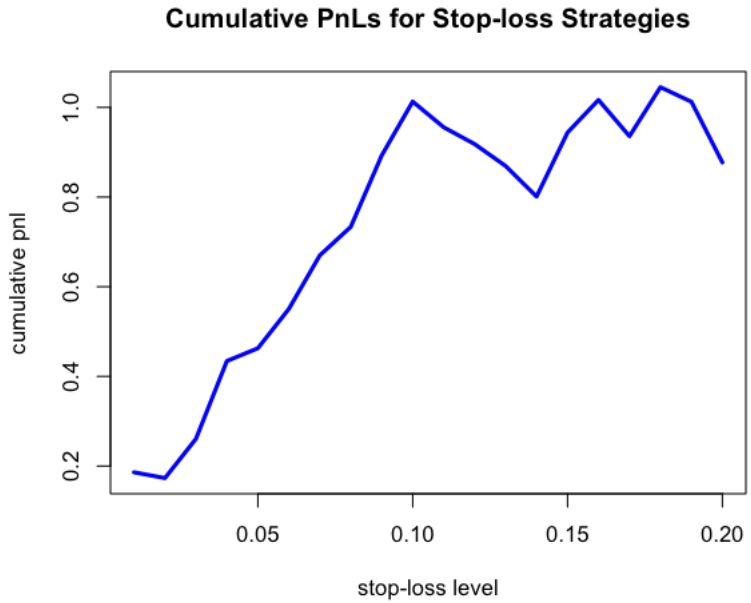



```
> # Combine data
> wealthv <- xts::xts(cbind(retsp, pnls), dates)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI stop-loss strategy
> endp <- rutils::calc_endpoints(wealthv, interval="months")
> dygraphs::dygraph(cumsum(wealthv)[endp],
+   main="VTI Stop-loss 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")
```

Optimal Stop-loss Rules

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

```
> # Simulate multiple stop-loss strategies
> cumsumv <- cumsum(retsp)
> cummaxv <- cummax(cumsumv)
> dd <- (cumsumv - cummaxv)
> cum_pnls <- sapply(0.01*(1:20), function(stopl) {
+   pnls <- retsp
+   isdd <- rutils::lagit(dd < -stopl*cummaxv)
+   pnls <- ifelse(isdd, 0, pnls)
+   sum(pnls)
+ })  # end sapply
```




```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=0.01*(1:20), y=cum_pnls,
+      main="Cumulative PnLs for Stop-loss Strategies",
+      xlab="stop-loss level", ylab="cumulative pnl",
+      t="l", lwd=3, col="blue")
```

Homework Assignment

Required

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

Recommended

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

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