

FRE7241 Algorithmic Portfolio Management

Lecture#1, Spring 2025

Jerzy Pawlowski jp3900@nyu.edu

NYU Tandon School of Engineering

March 18, 2025



NYU

**TANDON SCHOOL
OF ENGINEERING**

Welcome Students!

My name is Jerzy Pawlowski jp3900@nyu.edu

I'm an adjunct professor at NYU Tandon because I love teaching and I want to share my professional knowledge with young, enthusiastic students.

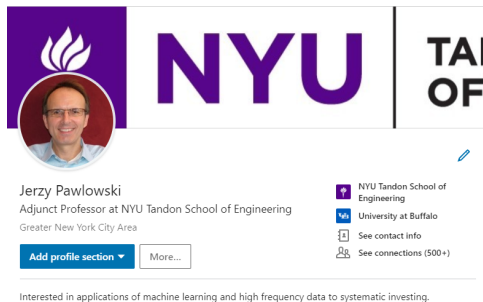
I'm interested in applications of *machine learning* to *systematic investing*.

I'm an advocate of *open-source software*, and I share it on GitHub:

[My GitHub account](#)





In my finance career, I have worked as a hedge fund *portfolio manager*, *CLO banker* (structurer), and *quant risk analyst*.

[My LinkedIn profile](#)

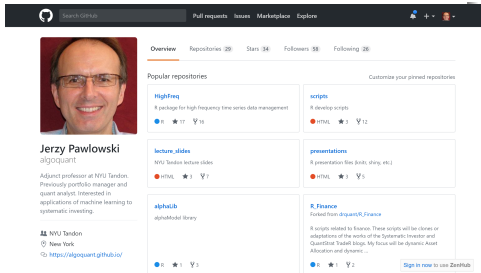


Jerzy Pawlowski
Adjunct Professor at NYU Tandon School of Engineering
Greater New York City Area


[Add profile section](#) [More...](#)

-  NYU Tandon School of Engineering
-  University at Buffalo
-  See contact info
-  See connections (500+)

Interested in applications of machine learning and high frequency data to systematic investing.




[Search GitHub](#) [Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



Jerzy Pawlowski
algoquant

Adjunct professor at NYU Tandon. Previously portfolio manager and quant analyst. Interested in applications of machine learning to systematic investing.

 NYU Tandon
New York
<https://algoquant.github.io/>

Overview [Repositories](#) 29 [Stars](#) 34 [Followers](#) 58 [Following](#) 26

Popular repositories

Repository	Stars	Language
HighFreq A package for high frequency time series data management	17	Python
lecture_slides NYU Tandon lecture slides	7	HTML
alphanlib alphanlib library	3	Python
scripts A develop scripts	12	HTML
presentations A presentation files (pdfs, shpg, etc.)	5	HTML
R_Finance R scripts related to Finance. These scripts will be clones or adaptations of the works of the Systematic Investor and QuantGest Trade bings. My focus will be dynamic Asset Allocation and dynamic ...	2	Python

[Sign in now to use ZenHub](#)

FRE7241 Course Description and Objectives

Course Description

The course will apply the R programming language to *trend following*, *momentum trading*, *statistical arbitrage* (pairs trading), and other active portfolio management strategies. The course will implement volatility and price *forecasting models*, asset pricing and *factor models*, and *portfolio optimization*. The course will apply *machine learning* techniques, such as *parameter regularization* (shrinkage), *bagging* and *backtesting* (cross-validation). **This course is challenging, so it requires devoting a significant amount of time!**

FRE7241 Course Description and Objectives

Course Description

The course will apply the R programming language to *trend following*, *momentum trading*, *statistical arbitrage* (pairs trading), and other active portfolio management strategies. The course will implement volatility and price *forecasting models*, asset pricing and *factor models*, and *portfolio optimization*. The course will apply *machine learning* techniques, such as *parameter regularization* (shrinkage), *bagging* and *backtesting* (cross-validation). **This course is challenging, so it requires devoting a significant amount of time!**

Course Objectives

Students will learn through R coding exercises how to:

- download data from external sources, and to scrub and format it.
- estimate time series parameters, and fit models such as *ARIMA*, *GARCH*, and factor models.
- optimize portfolios under different constraints and risk-return objectives.
- backtest active portfolio management strategies and evaluate their performance.

FRE7241 Course Description and Objectives

Course Description

The course will apply the R programming language to *trend following*, *momentum trading*, *statistical arbitrage* (pairs trading), and other active portfolio management strategies. The course will implement volatility and price forecasting models, asset pricing and *factor models*, and *portfolio optimization*. The course will apply *machine learning* techniques, such as *parameter regularization* (shrinkage), *bagging* and *backtesting* (cross-validation). **This course is challenging, so it requires devoting a significant amount of time!**

Course Objectives

Students will learn through R coding exercises how to:

- download data from external sources, and to scrub and format it.
- estimate time series parameters, and fit models such as *ARIMA*, *GARCH*, and factor models.
- optimize portfolios under different constraints and risk-return objectives.
- backtest active portfolio management strategies and evaluate their performance.

Course Recommendations

It's recommended that you take *FRE6123 Financial Risk Management and Asset Pricing*. The R language is considered to be challenging, so this course requires programming experience with other languages such as C++ or Python. Students with less programming experience are encouraged to first take *FRE6871 R in Finance*, and also *FRE6883 Financial Computing* by prof. Song Tang. Students should also have knowledge of basic statistics (random variables, estimators, hypothesis testing, regression, etc.)

Homeworks and Tests

Homeworks and Tests

Grading will be based on homeworks and tests. There will be no final exam.

The tests will be announced several days in advance.

The homeworks and tests will require writing code in R, which should run directly when loaded into an R session, and should produce the required output, **without any modifications**.

The tests will be closely based on code contained in the lecture slides, so students are encouraged to become very familiar with those slides.

Students must submit their homework and test files only through *Brightspace* (not emails).

Students will be allowed to copy code from the lecture slides, and to copy from books or any online sources, but they will be required to provide references to those external sources (such as links or titles and page numbers).

Students are encouraged to use AI applications, such as ChatGPT, *GitHub Copilot*, *Copilot for RStudio*, etc. But you must include the name of the AI application in your solution.

Students will be required to bring their laptop computers to class and run the R Interpreter, and the RStudio Integrated Development Environment (*IDE*), during the lecture.

Homeworks will also include reading assignments designed to help prepare for tests.

Homeworks and Tests

Homeworks and Tests

Grading will be based on homeworks and tests. There will be no final exam.

The tests will be announced several days in advance.

The homeworks and tests will require writing code in R, which should run directly when loaded into an R session, and should produce the required output, **without any modifications**.

The tests will be closely based on code contained in the lecture slides, so students are encouraged to become very familiar with those slides.

Students must submit their homework and test files only through *Brightspace* (not emails).

Students will be allowed to copy code from the lecture slides, and to copy from books or any online sources, but they will be required to provide references to those external sources (such as links or titles and page numbers).

Students are encouraged to use AI applications, such as ChatGPT, *GitHub Copilot*, *Copilot for RStudio*, etc. But you must include the name of the AI application in your solution.

Students will be required to bring their laptop computers to class and run the R Interpreter, and the RStudio Integrated Development Environment (*IDE*), during the lecture.

Homeworks will also include reading assignments designed to help prepare for tests.

Graduate Assistant

The graduate assistant (GA) will be Lakshay Dua ld3074@nyu.edu.

The GA will answer questions during office hours, or via *Brightspace* forums, not via emails. Please send emails regarding lecture matters from *Brightspace* (not personal emails).

Tips for Solving Homeworks and Tests

Tips for Solving Homeworks and Tests

The assignments will mostly require copying code samples from the lecture slides, making some modifications to them, and combining them with other code samples.

Partial credit will be given even for code that doesn't produce the correct output, but that has elements of code that can be useful for producing the right answer.

So don't leave test assignments unanswered, and instead copy any code samples from the lecture slides that are related to the solution and make sense.

Contact the GA during office hours via text or phone, and submit questions to the GA or to me via *Brightspace*.

Tips for Solving Homeworks and Tests

Tips for Solving Homeworks and Tests

The assignments will mostly require copying code samples from the lecture slides, making some modifications to them, and combining them with other code samples.

Partial credit will be given even for code that doesn't produce the correct output, but that has elements of code that can be useful for producing the right answer.

So don't leave test assignments unanswered, and instead copy any code samples from the lecture slides that are related to the solution and make sense.

Contact the GA during office hours via text or phone, and submit questions to the GA or to me via *Brightspace*.

Please Submit *Minimal Working Examples* With Your Questions

When submitting questions, please provide a *minimal working example* that produces the error in R, with the following items:

- The *complete* R code that produces the error, including the seed value for random numbers,
- The version of R (output the of command: `sessionInfo()`), and the versions of R packages,
- The type and version of your operating system (Windows or OSX),
- The dataset file used by the R code,
- The text or screenshots of error messages,

You can read more about producing *minimal working examples* here: <http://stackoverflow.com/help/mcve>
<http://www.jaredknowles.com/journal/2013/5/27/writing-a-minimal-working-example-mwe-in-r>

Course Grading Policies

Numerical Scores

Homeworks and tests will be graded and assigned numerical scores. Each part of homeworks and tests will be graded separately and assigned a numerical score.

Maximum scores will be given only for complete code, that produces the correct output when it's pasted into an R session, without any modifications. As long as the R code uses the required functions and produces the correct output, it will be given full credit.

Partial credit will be given even for code that doesn't produce the correct output, but that has elements of code that can be useful for producing the right answer.

Course Grading Policies

Numerical Scores

Homeworks and tests will be graded and assigned numerical scores. Each part of homeworks and tests will be graded separately and assigned a numerical score.

Maximum scores will be given only for complete code, that produces the correct output when it's pasted into an R session, without any modifications. As long as the R code uses the required functions and produces the correct output, it will be given full credit.

Partial credit will be given even for code that doesn't produce the correct output, but that has elements of code that can be useful for producing the right answer.

Letter Grades

Letter grades for the course will be derived from the percentage scores obtained for all the homeworks and tests. The percentage scores will be calculated by adding together the scores of all the homeworks and tests, and dividing them by the sum of the maximum scores. The percentage scores are usually very high - above 90%. So a very high percentage score will not guarantee an A letter grade, since grading will also depend on the difficulty of the assignments.

Course Grading Policies

Numerical Scores

Homeworks and tests will be graded and assigned numerical scores. Each part of homeworks and tests will be graded separately and assigned a numerical score.

Maximum scores will be given only for complete code, that produces the correct output when it's pasted into an R session, without any modifications. As long as the R code uses the required functions and produces the correct output, it will be given full credit.

Partial credit will be given even for code that doesn't produce the correct output, but that has elements of code that can be useful for producing the right answer.

Letter Grades

Letter grades for the course will be derived from the percentage scores obtained for all the homeworks and tests. The percentage scores will be calculated by adding together the scores of all the homeworks and tests, and dividing them by the sum of the maximum scores. The percentage scores are usually very high - above 90%. So a very high percentage score will not guarantee an A letter grade, since grading will also depend on the difficulty of the assignments.

Plagiarism

Plagiarism (copying from other students) and cheating will be punished.

But copying code from lecture slides, books, or any online sources is allowed and encouraged.

Students must provide references to any external sources from which they copy code (such as links or titles and page numbers).

FRE7241 Course Materials

Lecture Slides

The course will be mostly self-contained, using detailed lecture slides containing extensive, working R code examples.

The course will also utilize data and tutorials which are freely available on the internet.

FRE7241 Course Materials

Lecture Slides

The course will be mostly self-contained, using detailed lecture slides containing extensive, working R code examples.

The course will also utilize data and tutorials which are freely available on the internet.

FRE7241 Recommended Textbooks

- *Advances in Financial Machine Learning* by Marcos Lopez de Prado - Machine learning techniques applied to trading and portfolio management.
- *Systematic Trading* by Robert Carver - Practical trading knowledge by an experienced portfolio manager.
- *Systematic Investing* by Robert Carver - Practical investment knowledge by a successful investor.
- *Quantitative Trading* by Xin Guo, Tze Leung Lai, Howard Shek, Samuel Po-Shing Wong - Advanced topics in quantitative trading by academic experts.
- *Financial Data and Models Using R* by Clifford Ang - Good introduction to time series, portfolio optimization, and performance measures.
- *Automated Trading* by Chris Conlan - How to implement practical computer trading systems.
- *Statistics and Data Analysis for Financial Engineering* by David Ruppert - Introduces regression, cointegration, multivariate time series analysis, *ARIMA*, *GARCH*, *CAPM*, and factor models, with examples in R.
- *Financial Risk Modelling and Portfolio Optimization with R* by Bernhard Pfaff - Introduces volatility models, portfolio optimization, and tactical asset allocation, with a great review of R packages and examples in R.

Many textbooks can be downloaded in electronic format from the [NYU Library](#).

FRE7241 Supplementary Textbooks

Supplementary Textbooks

- *Introduction to Statistical Learning* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, introduces machine learning techniques using R, but without deep learning.
- *Quantitative Risk Management* by Alexander J. McNeil, Rudiger Frey, and Paul Embrechts: review of Value at Risk, factor models, ARMA and GARCH, extreme value theory, and credit risk models.
- *Applied Econometrics with R* by Christian Kleiber and Achim Zeileis, introduces advanced statistical models and econometrics.
- *The Art of R Programming* by Norman Matloff, contains a good introduction to R and to statistical models.
- *Advanced R* by Hadley Wickham, is the best book for learning the advanced features of R.
- *Numerical Recipes in C++* by William Press, Saul Teukolsky, William Vetterling, and Brian Flannery, is a great reference for linear algebra and numerical methods, implemented in working C++ code.
- The books *R in Action* by Robert Kabacoff and *R for Everyone* by Jared Lander, are good introductions to R and to statistical models.
- *Quant Finance books* by Jerzy Pawlowski.
- *Quant Trading books* by Jerzy Pawlowski.

FRE7241 Supplementary Materials

Robert Carver's trading blog

Great blog about practical systematic trading and investments, with Python code: <http://qoppac.blogspot.com/>

Introduction to Computational Finance with R

Good course by prof. Eric Zivot, with lots of R examples:

<https://www.datacamp.com/community/open-courses/computational-finance-and-financial-econometrics-with-r>

Notepad++ is a free source code editor for MS Windows, that supports several programming languages, including R.

Notepad++ has a very convenient and fast *search and replace* function, that allows *search and replace* in multiple files.

<http://notepad-plus-plus.org/>



What is R?

- An open-source software environment for statistical computing and graphics.
- An interpreted language, that allows interactive code development.
- A functional language where every operator is an R function.
- A very expressive language that can perform complex operations with very few lines of code.
- A language with metaprogramming facilities that allow programming on the language.
- A language written in C/C++, which can easily call other C/C++ programs.
- Can be easily extended with *packages* (function libraries), providing the latest developments like *Machine Learning*.
- Supports object-oriented programming with *classes* and *methods*.
- Vectorized functions written in C/C++, allow very fast execution of loops over vector elements.



Differences Between R and Python

R was designed for statistics and data science, while Python was designed as a general-purpose programming language.

Why R is Better Than Python

- R was designed for statistics and data science - Python wasn't.
- R has native date and time objects built in - Python doesn't.
- R has native dataframe objects built in - Python doesn't.
- R has native vector and matrix objects built in - Python doesn't.
- R is designed to be easily extended with C++ code - Python isn't.

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("SPY", "VTI", "QQQ", "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", "AIE")
> # Read etf database into data frame
> etflist <- read.csv(file="/Users/jerzy/Develop/lecture_slides/data/etf_list.csv")
> rownames(etflist) <- etflist$Symbol
> # Select from etflist only those ETF's in symbolv
> etflist <- etflist[symbolv, ]
> # Shorten names
> etfnames <- sapply(etflist$Name, function(name) {
+   namesplit <- strsplit(name, split=" ")[[1]]
+   namesplit <- namesplit[c(-1, -NROW(namesplit))]
+   name_match <- match("Select", namesplit)
+   if (!is.na(name_match))
+     namesplit <- namesplit[-name_match]
+   paste(namesplit, 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 Database for Investment Strategies

The database 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.

The *SPY ETF* owns the *S&P500* index constituents. *SPY* is the biggest, the most liquid, and the oldest ETF. *SPY* has over \$400 billion of shares outstanding, and trades over \$20 billion per day, at a bid-ask spread of only one tick (cent=\$0.01, or about 0.0022%).

The *QQQ ETF* owns the *Nasdaq-100* index constituents.

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.

Symbol	Name	Fund.Type
SPY	S&P 500	US Equity ETF
VTI	Total Stock Market	US Equity ETF
QQQ	QQQ Trust	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 ETF
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
AIEQ	AI Powered Equity	US Asset Allocation 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.

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*.

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("SPY", "VTI", "QQQ", "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", "AIE")
> library(rutils) # Load package rutils
> etfenv <- new.env() # New environment for data
> # Boolean vector of symbols already downloaded
> isdown <- 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
> n attempts <- 0 # number of download attempts
> while ((sum(!isdown) > 0) & (n attempts < 10)) {
+ # Download data and copy it into environment
+ n attempts <- n attempts + 1
+ cat("Download attempt = ", n attempts, "\n")
+ for (symboln in na.omit(symbolv[!isdown][1:5])) {
+ cat("Processing: ", symboln, "\n")
+ tryCatch( # With error handler
+ quantmod::getSymbols.av(symboln, adjust=TRUE, env=etfenv, auto.assign=FALSE)
+ # Error handler captures error condition
+ error=function(msg) {
+ print(paste0("Error handler: ", msg))
+ }, # end error handler
+ finally=print(paste0("Symbol = ", symboln))
+ ) # end tryCatch
+ } # end for
+ # Update vector of symbols already downloaded
+ isdown <- 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 error
> # 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(namev) class(get(namev)))
> # 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 if 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 (symboln in ls(etfenv)) {
+   assign(symboln,
+     adjustOHLC(get(symboln, 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*: `C1()`, `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
> pricev <- mget(symbolv, envir=rutils::etfenv)
> # pricev is a list of xts series
> class(pricev)
> class(pricev[[1]])
> tail(pricev[[1]])
> # Extract close prices
> pricev <- lapply(pricev, quantmod::C1)
> # Collapse list into time series the hard way
> prices2 <- cbind(pricev[[1]], pricev[[2]], pricev[[3]], pricev[[4]])
> class(prices2)
> dim(prices2)
> # Collapse list into time series using do.call()
> pricev <- do.call(cbind, pricev)
> all.equal(price2, pricev)
> class(pricev)
> dim(pricev)
> # Or extract and cbind in single step
> pricev <- do.call(cbind, lapply(
+   mget(symbolv, envir=rutils::etfenv), quantmod::C1))
> # Or extract and bind all data, subset by symbolv
> pricev <- lapply(symbolv, function(symboln) {
+   quantmod::C1(get(symboln, envir=rutils::etfenv))
+ }) # end lapply
> # Or loop over etfenv without anonymous function
> pricev <- do.call(cbind,
+   lapply(as.list(rutils::etfenv)[symbolv], quantmod::C1))
> # Same, but works only for OHLC series - produces error
> pricev <- do.call(cbind,
+   eapply(rutils::etfenv, quantmod::C1)[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(pricev)
> strsplit(colnames(pricev), split=".")
> do.call(rbind, strsplit(colnames(pricev), split="."))
> do.call(rbind, strsplit(colnames(pricev), split="."))[, 1]
> # Drop ".Close" from colnames
> colnames(pricev) <- rutils::get_name(colnames(pricev))
> # Or
> # colnames(pricev) <- do.call(rbind,
> #   strsplit(colnames(pricev), split="."))[, 1]
> tail(pricev, 3)
> # Which objects in global environment are class xts?
> unlist(eapply(globalenv(), is.xts))
> # Save xts to csv file
> write.zoo(pricev,
+   file="/Users/jerzy/Develop/lecture_slides/data/etf_series.csv",
> # Copy prices into etfenv
> etfenv$pricev <- pricev
> # Or
> assign("pricev", pricev, 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
> pricev <- etfenv$prices[,"VTI"]
> pricev <- na.omit(pricev)
> # Calculate percentage returns "by hand"
> pricel <- as.numeric(pricev)
> pricel <- c(pricel[1], pricel[-NROW(pricel)])
> pricel <- xts(pricel, zoo::index(pricev))
> retp <- (pricev-pricel)/pricel
> # Calculate percentage returns using dailyReturn()
> retld <- quantmod::dailyReturn(pricev)
> head(cbind(retld, retp))
> all.equal(retld, retp, check.attributes=FALSE)
> # Calculate returns for all prices in etfenv$prices
> retp <- lapply(etfenv$prices, function(xtsv) {
+   retld <- quantmod::dailyReturn(na.omit(xtsv))
+   colnames(retld) <- names(xtsv)
+   retld
+ }) # end lapply
> # "retp" is a list of xts
> class(retp)
> class(retp[[1]])
> # Flatten list of xts into a single xts
> retp <- do.call(cbind, retp)
> class(retp)
> dim(retp)
> # Copy retp into etfenv and save to .RData file
> # assign("retp", retp, envir=etfenv)
> etfenv$retp <- retp
> save(etfenv, file="/Users/jerzy/Develop/lecture_slides/data/etf_d
```

Managing Data Inside Environments

The function `as.environment()` coerces objects (list) 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
> newenv <- as.environment(eapply(etfenv, "[",
+   paste(startd, endd, sep="/")))
> # Select only symbolv in environment and return as environment
> newenv <- 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(symboln) {
+     xtsv <- quantmod::Cl(get(symboln, etfenv))
+     colnames(xtsv) <- symboln
+     xtsv
+   })), envir=newenv)
> # 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=newenv)
```

Stock Databases And Survivorship Bias

The file `sp500_constituents.csv` contains a *data frame* of over 700 present (and also some past) *S&P500* index constituents.

The file `sp500_constituents.csv` is updated with stocks recently added to the *S&P500* index by downloading the *SPY ETF Holdings*.

But the file `sp500_constituents.csv` doesn't include companies that have gone bankrupt. For example, it doesn't include Enron, which was in the *S&P500* index before it went bankrupt in 2001.

Most databases of stock prices don't include companies that have gone bankrupt or have been liquidated.

This introduces a *survivorship bias* to the data, which can skew portfolio simulations and strategy backtests.

Accurate strategy simulations require starting with a portfolio of companies at a "point in time" in the past, and tracking them over time.

Research databases like the *WRDS* database provide stock prices of companies that are no longer traded.

The stock tickers are stored in the column "Ticker" of the `sp500 data frame`.

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/sp500.csv")
> # 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)
> duplicatv <- tablev[tablev > 1]
> duplicatv <- names(duplicatv)
> # Get duplicate records (rows) of sp500
> sp500[symbolv %in% duplicatv, ]
> # 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"
```

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.

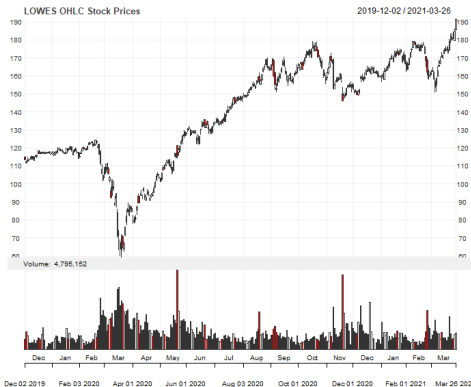
```
> # The date-time index of AAPL is POSIXct
> class(zoo::index(sp500env$AAPL))
> # Coerce the date-time index of AAPL to Date
> zoo::index(sp500env$AAPL) <- as.Date(zoo::index(sp500env$AAPL))
> # Coerce all the date-time indices to Date
> for (symboln in ls(sp500env)) {
+   ohlc <- get(symboln, envir=sp500env)
+   zoo::index(ohlc) <- as.Date(zoo::index(ohlc))
+   assign(symboln, 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
> namev <- rutils::get_name(colnames(sp500env$LOW), field=2)
> # Or
> # namev <- do.call(rbind, strsplit(colnames(sp500env$LOW),
> #                                   split="."))[, 2]
> # Rename "LOW" colnames to "LOWES"
> colnames(sp500env$LOW) <- paste("LOWES", namev, sep=".")
> sp500env$LOWES <- sp500env$LOW
> rm(LOW, envir=sp500env)
> # Rename BF-B colnames to "BFB"
> colnames(sp500env$BF-B) <- paste("BFB", namev, 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$BRKB) <- gsub("BRK-B", "BRKB", colnames(sp500env$BRKB))
> # 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", namev, 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 OHLC Stock Prices",
+   dyCandlestick())
```

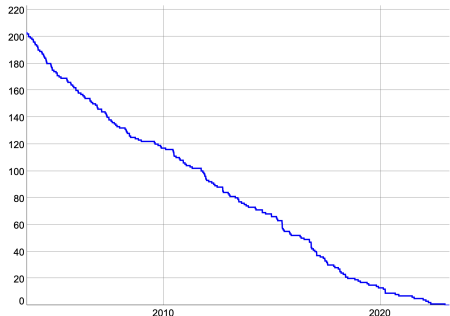
S&P500 Stock Index Constituent Prices

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

Number of S&P500 Constituents Without Prices

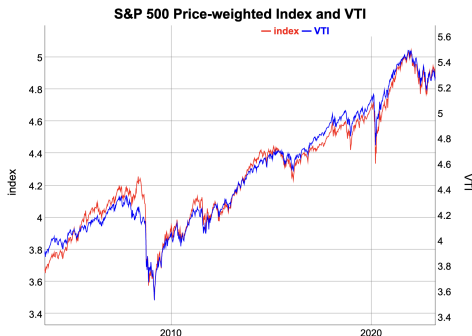


```
> # Calculate number of constituents without prices
> datav <- rowSums(is.na(pricev))
> datav <- xts::xts(datav, order.by=zoo::index(pricev))
> dygraphs::dygraph(datav, main="Number of S&P500 Constituents Without Prices")
+   dyOptions(colors="blue", strokeWidth=2)
```


S&P500 Stock Portfolio Index

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

```
> # Calculate price weighted index of constituent
> ncol <- NCOL(pricev)
> pricev <- zoo::na.locf(pricev, fromLast=TRUE)
> indeks <- xts(rowSums(pricev)/ncol, zoo::index(pricev))
> colnames(indeks) <- "index"
> # Combine index with VTI
> datav <- cbind(indeks[zoo::index(etfenv$VTI)], etfenv$VTI[, 4])
> colv <- c("index", "VTI")
> colnames(datav) <- colv
> # Plot index with VTI
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(log(datav)[endd],
+   main="S&P 500 Price-weighted Index and VTI") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="red") %>%
+   dySeries(name=colv[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
> dirn <- "/Users/jerzy/Develop/lecture_slides/data/"
> save(sp500env, file=paste0(dirn, "sp500.RData"))
> # Save the ETF prices into CSV files
> dirn <- "/Users/jerzy/Develop/lecture_slides/data/SP500/"
> for (symboln in ls(sp500env)) {
+   zoo::write.zoo(sp500env$symbol, file=paste0(dirn, symboln, ".csv"))
+ } # end for
> # Or using lapply()
> filev <- lapply(ls(sp500env), function(symboln) {
+   xtsv <- get(symboln, envir=sp500env)
+   zoo::write.zoo(xtsv, file=paste0(dirn, symboln, ".csv"))
+   symboln
+ }) # end lapply
> unlist(filev)
> # Or using eapply() and data.table::fwrite()
> filev <- eapply(sp500env, function(xtsv) {
+   filen <- rutils::get_name(colnames(xtsv)[1])
+   data.table::fwrite(data.table::as.data.table(xtsv), file=paste0(
+     dirn,
+     filen
+   )) # end eapply
+ }) # end eapply
> unlist(filev)
```

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

Calculating Prices and Returns From *OHLC* Data

The function `na.locf()` from package *zoo* replaces NA values with the most recent non-NA values prior to it.

The function `na.locf()` with argument `fromLast=TRUE` replaces NA values with non-NA values in reverse order, starting from the end.

The function `rutils::get_name()` extracts symbol names (tickers) from a vector of character strings.

```
> pricev <- eapply(sp500env, quantmod::C1)
> pricev <- rutils::do_call(cbind, pricev)
> # Carry forward non-NA prices
> pricev <- zoo::na.locf(pricev, na.rm=FALSE)
> # Get first column name
> colnames(pricev[, 1])
> rutils::get_name(colnames(pricev[, 1]))
> # Modify column names
> colnames(pricev) <- rutils::get_name(colnames(pricev))
> # Or
> # colnames(pricev) <- do.call(rbind,
> #   strsplit(colnames(pricev), split=".[.]"))[, 1]
> # Calculate percentage returns
> retp <- xts::diff.xts(pricev)/rutils::lagit(pricev, pad_zeros=FALSE)
> # Select a random sample of 100 prices and returns
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> samplev <- sample(NCOL(retp), s=100, replace=FALSE)
> prices100 <- pricev[, samplev]
> returns100 <- retp[, samplev]
> # Save the data into binary files
> save(pricev, prices100,
+       file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.R")
> save(retp, returns100,
+       file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.R")
```

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

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

Downloading VIX Futures Files from CBOE

The CFE (CBOE Futures Exchange) provides daily **CBOE Historical Data for Volatility Futures**, including the *VIX* futures.

The CBOE data includes *OHLC* prices and also the *settlement* price (in column "Settle").

The *settlement* price is usually defined as the weighted average price (*WAP*) or the midpoint price, and is different from the *Close* price.

The *settlement* price is used for calculating the daily *mark to market* (value) of the futures contract.

Futures exchanges require that counterparties exchange (settle) the *mark to market* value of the futures contract daily, to reduce counterparty default risk.

The function `download.file()` downloads files from the internet.

The function `tryCatch()` executes functions and expressions, and handles any *exception conditions* produced when they are evaluated.

```
> # Read CBOE futures expiration dates
> datev <- read.csv(file="/Users/jerzy/Develop/lecture_slides/data/1
+   row.names=1)
> # Create directory for data
> dirn <- "/Users/jerzy/Develop/data/vix_data"
> dir.create(dirn)
> namev <- rownames(datev)
> filev <- file.path(dirn, paste0(namev, ".csv"))
> filelog <- file.path(dirn, "log_file.txt")
> urlcboe <- "https://markets.cboe.com/us/futures/market_statistics
> urlv <- paste0(urlcboe, datev[, 1])
> # Download files in loop
> for (it in seq_along(urlv)) {
+   tryCatch( # Warning and error handler
+     download.file(urlv[it], destfile=filev[it], quiet=TRUE),
+   # Warning handler captures warning condition
+   warning=function(msg) {
+     cat(paste0("Warning handler: ", msg, "\n"), file=filelog, append
+   }, # end warning handler
+   # Error handler captures error condition
+   error=function(msg) {
+     cat(paste0("Error handler: ", msg, "\n"), append=TRUE)
+   }, # end error handler
+   finally=cat(paste0("Processing file name = ", filev[it], "\n"), ap
+   ) # end tryCatch
+ } # end for
```

Downloading VIX Futures Data Into an Environment

The function `quantmod::getSymbols()` with the parameter `src="cfe"` downloads CFE data into the specified *environment*. (But this requires first loading the package *qmao*.)

Currently `quantmod::getSymbols()` doesn't download the most recent data.

```
> # Create new environment for data
> vixenv <- new.env()
> # Download VIX data for the months 6, 7, and 8 in 2018
> library(qmao)
> quantmod::getSymbols("VX", Months=1:12,
+   Years=2018, src="cfe", auto.assign=TRUE, env=vixenv)
> # Or
> qmao::getSymbols.cfe(Symbols="VX",
+   Months=6:8, Years=2018, env=vixenv,
+   verbose=FALSE, auto.assign=TRUE)
> # Calculate the classes of all the objects
> # In the environment vixenv
> unlist(eapply(vixenv, function(x) {class(x)[1]}))
> class(vixenv$VX_M18)
> colnames(vixenv$VX_M18)
> # Save the data to a binary file called "vix_cboe.RData".
> save(vixenv,
+   file="/Users/jerzy/Develop/data/vix_data/vix_cboe.RData")
```

The Median Absolute Deviation Estimator of Dispersion

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

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

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

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

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

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

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


The Median Absolute Deviation of Asset Returns

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

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

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

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

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

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

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

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

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

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

The Downside Deviation of Asset Returns

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

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

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

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

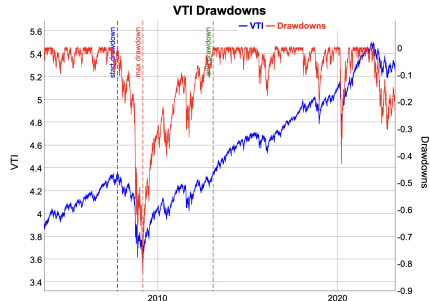
```
> library(PerformanceAnalytics)
> # Define target rate of return of 50 bps
> targetr <- 0.005
> # Calculate the full downside returns
> retsub <- (retp - targetr)
> retsub <- ifelse(retsub < 0, retsub, 0)
> nrow <- NROW(retsub)
> # Calculate the downside deviation
> all.equal(sqrt(sum(retsub^2)/nrow),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

Drawdown Risk

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

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

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



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

Drawdown Risk Using PerformanceAnalytics::table.Drawdowns()

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

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

From	Trough	To	Depth	Length	To Trough	Recovery
2007-10-10	2009-03-09	2012-03-13	-0.57	1115.00	355.00	760.00
2001-06-06	2002-10-09	2004-11-04	-0.45	858.00	336.00	522.00
2020-02-20	2020-03-23	2020-08-12	-0.18	122.00	23.00	99.00
2022-01-04	2022-10-12	2023-12-18	-0.10	492.00	195.00	297.00
2018-09-21	2018-12-24	2019-04-23	-0.10	146.00	65.00	81.00

The Loss Distribution of Asset Returns

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

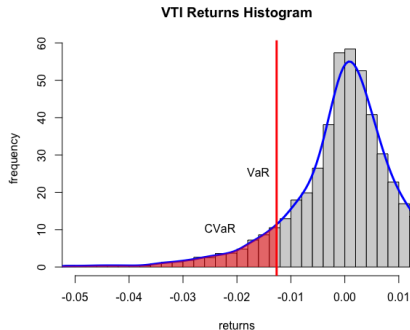
The *Value at Risk* (VaR) is equal to the quantile of returns corresponding to a given confidence level α .

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

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

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

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



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

Value at Risk (VaR)

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

$$\alpha = \int_{-\infty}^{\text{VaR}(\alpha)} f(r) dr$$

Where $f(r)$ is the probability density (distribution) of returns.

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

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

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

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

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

Conditional Value at Risk (CVaR)

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

$$\text{CVaR} = \frac{1}{\alpha} \int_0^{\alpha} \text{VaR}(p) dp$$

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

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

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

Risk and Return Statistics

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

```
> # Calculate the risk-return statistics
> riskstats <-
+   PerformanceAnalytics::table.Stats(rutils::etfenv$returns)
> class(riskstats)
> # Transpose the data frame
> riskstats <- as.data.frame(t(riskstats))
> # Add Name column
> riskstats$Name <- rownames(riskstats)
> # Add Sharpe ratio column
> riskstats$"Arithmetic Mean" <-
+   sapply(rutils::etfenv$returns, mean, na.rm=TRUE)
> riskstats$Sharpe <-
+   sqrt(252)*riskstats$"Arithmetic Mean"/riskstats$Stdev
> # Sort on Sharpe ratio
> riskstats <- riskstats[order(riskstats$Sharpe, decreasing=TRUE), 1]
```

	Sharpe	Skewness	Kurtosis
USMV	0.838	-0.864	21.73
AIEQ	0.823	-0.482	1.35
QUAL	0.732	-0.514	12.83
MTUM	0.689	-0.641	11.03
SPY	0.536	-0.295	11.06
VLUE	0.485	-0.941	17.19
GLD	0.484	-0.317	6.04
IEF	0.475	0.050	2.50
VTI	0.452	-0.385	10.79
VTV	0.449	-0.668	14.10
XLV	0.443	0.067	10.25
VYM	0.440	-0.681	14.91
XLP	0.423	-0.124	8.77
XLY	0.419	-0.363	6.53
IWB	0.400	-0.395	10.05
XLI	0.399	-0.376	7.61
IWW	0.395	-0.305	8.20
IWD	0.369	-0.488	12.83
XLU	0.365	-0.004	11.71
IVE	0.357	-0.483	10.31
IWF	0.356	-0.655	30.25
QQQ	0.353	-0.033	6.49
XLK	0.340	0.060	6.54
XLB	0.323	-0.369	5.43
EEM	0.292	0.025	15.91
XLE	0.263	-0.531	12.61
TLT	0.252	-0.011	3.42
VNQ	0.247	-0.538	18.38
XLF	0.195	-0.125	14.47
SVXY	0.163	-18.273	680.40
VEU	0.157	-0.509	12.00
DBC	0.021	-0.487	3.32
USO	-0.285	-1.127	14.12
VXX	-1.143	1.170	6.05

Investor Risk and Return Preferences

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

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

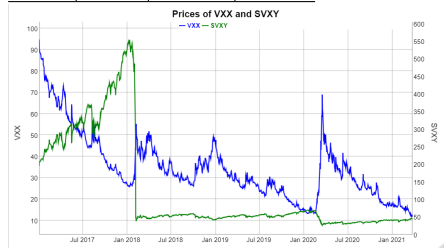
The *VXX* ETF is long the *VIX* index (effectively long an option), so it has positive skewness and small kurtosis, but negative returns (it's short market risk).

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

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

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

	Sharpe	Skewness	Kurtosis
VXX	-1.143	1.17	6.05
SVXY	0.163	-18.27	680.40



```
> # dygraph plot of VXX versus SVXY
> pricev <- na.omit(rutils::etfenv$prices[, c("VXX", "SVXY")])
> pricev <- pricev["2017/"]
> colv <- c("VXX", "SVXY")
> colnames(pricev) <- colv
> dygraphs::dygraph(pricev, main="Prices of VXX and SVXY") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name=colv[2], axis="y2", strokeWidth=2, col="green") %>%
+   dyLegend(show="always", width=300) %>% dyLegend(show="always", width=300)
```

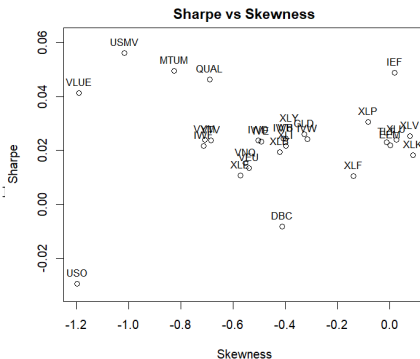
Skewness and Return Tradeoff

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

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

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

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



Risk-adjusted Return Measures

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

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

The *Sortino ratio* S_{Or} 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):

$$S_{Or} = \frac{E[r - r_t]}{\sigma_d}$$

The *Calmar ratio* C_r is equal to the excess returns divided by the *maximum drawdown* DD of the returns:

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

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

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

The *Conditional Dowd ratio* D_{c_r} is equal to the excess returns divided by the *Conditional Value at Risk* (CVaR) of the returns:

$$D_{c_r} = \frac{E[r - r_f]}{CVaR}$$

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

Risk and Return of Stocks Over Longer Holding Periods

Stocks held over longer holding periods often have higher risk-adjusted returns than over shorter holding periods - provided the long-term stock returns are positive.

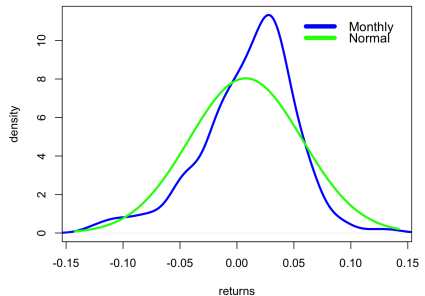
This is because returns are proportional to the holding period, while risk is proportional to the square root of the holding period.

Therefore investors with longer holding periods may choose to own a higher percentage of stocks than bonds.

The skewness of monthly returns is higher than for daily returns, but their kurtosis and tail risks are lower.

```
> # Calculate VTI daily log returns
> pricev <- log(drop(coredata(na.omit(rutils::etfenv$prices$VTI))))
> retp <- rutils::diffit(pricev)
> nrow <- NROW(retp)
> # Calculate VTI monthly log returns
> holdp <- 22 # Holding period in days
> pricem <- pricev[rutils::calc_endpoints(pricev, holdp)]
> retm <- rutils::diffit(pricem)
> retm <- retm[-1] # Drop the first zero return
> # Calculate the mean, standard deviation, skewness, and kurtosis
> datav <- list(retp, retm)
> names(datav) <- c("Daily", "Monthly")
> do.call(cbind, lapply(datav, function(x) {
+   # Standardize the returns
+   meanv <- mean(x); stdev <- sd(x); x <- (x - meanv)/stdev
+   c(mean=meanv, stdev=stdev, skew=mean(x^3), kurt=mean(x^4))
+ }))) # end lapply
```

Distribution of Monthly VTI Returns



```
> # Calculate the Sharpe and Dowd ratios
> do.call(cbind, lapply(datav, function(x) {
+   meanv <- mean(x); stdev <- sd(x)
+   varisk <- unname(quantile(x, probs=0.02))
+   cvar <- mean(x[x < varisk])
+   # Annualize the ratios
+   sqrt(252*NROW(x)/nrow)*mean(x)/c(Sharpe=stdev, Dowd=-varisk,
+   }))) # end lapply
> # Plot the density of monthly returns
> plot(density(retm), t="l", lwd=3, col="blue",
+   xlab="returns", ylab="density", xlim=c(-4*mad(retm), 4*mad(retm)),
+   main="Distribution of Monthly VTI Returns")
> curve(expr=dnorm(x, mean=mean(retm), sd=sd(retm)), col="green", lwd=3,
+   legend("topright", legend=c("Monthly", "Normal"), y.intersp=0.4,
+   inset=0.0, bg="white", lty=1, lwd=6, col=c("blue", "green"), bty="n")
```

Tests for Market Timing Skill

Market timing skill a trading strategy is the ability to switch market positions, from long risk to short and vice versa, in anticipation of future market returns.

If a trading strategy has timing skill, then its returns have a positive convexity with respect to the market returns. The beta-adjusted strategy returns are positive, both when the market returns are positive and when they are negative.

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

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

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

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

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

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

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

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

```
> # Create a design matrix of IEF and VTI returns
> desm <- na.omit(rutils::etfenv$returns[, c("IEF", "VTI")])
> retvti <- desm$VTI
> # Add returns with perfect timing skill
> desm <- cbind(desm, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[3:4] <- c("merton", "treynor")
> # Perform Merton-Henriksson test regression
> regmod <- lm(IEF ~ VTI + merton, data=desm); summary(regmod)
> # Perform Treynor-Mazuy test regression
> regmod <- lm(IEF ~ VTI + treynor, data=desm); summary(regmod)
```

Market Timing Skill of Bonds

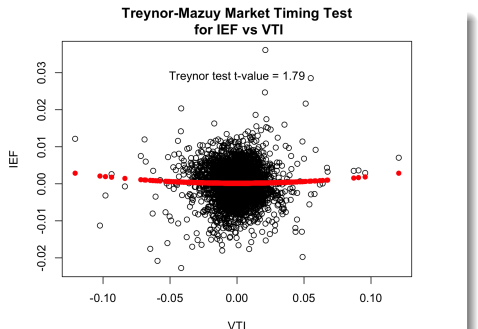
Even if a trading strategy has timing skill, it doesn't necessarily mean that its returns can be used to forecast the market returns.

The *IEF* 10-year Treasury bond ETF has a small market timing skill, because it has a slightly positive convexity with respect to the *VTI* stock ETF.

The slight market timing ability of Treasury bonds is significant, because it contributes to their superior risk-adjusted returns.

As a general rule, trend-following and momentum strategies have positive timing skill. Because they buy stocks when their prices are rising and sell them when the prices are dropping, expecting that the trend will continue.

Contrarian strategies have negative timing skill. Because they buy stocks when their prices are dropping and sell them when the prices are rising, expecting that the trend will reverse.



```
> # Plot residual scatterplot
> resid <- (desm$IEF - regmod$coeff["VTI"]*retvti)
> plot.default(x=retvti, y=resid, xlab="VTI", ylab="IEF")
> title(main="Treynor-Mazuy Market Timing Test\n for IEF vs VTI", 1)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(resid), paste("Treynor test t-value =", tvalue))
```

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:

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

The initial returns are all equal to zero.

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

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

Compounding Asset Returns

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

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

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

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

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

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

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

Logarithm of VTI Prices



```
> # Plot log VTI prices
> endd <- rutils::calc_endpoints(rutils::etfenv$VTI, interval="week")
> dygraphs::dygraph(log(quantmod::CI(rutils::etfenv$VTI)[endd]),
+   main="Logarithm of VTI Prices") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```


Funding Costs of Single Asset Rebalancing

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

The wealth accumulated from owning a *fixed dollar amount* of stock is equal to the cash earned from rebalancing, which is proportional to the sum of the percentage returns, and it's kept in a *margin account*:

$$m_t = \sum_{i=1}^t r_i^P.$$

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

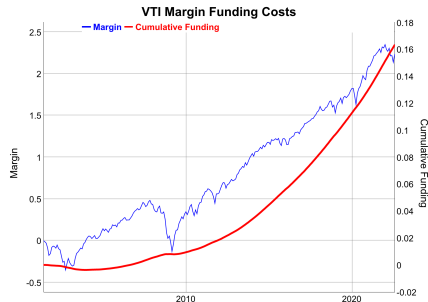
The *funding costs* c_t^f are approximately equal to the *margin account* m_t times the *funding rate* f :

$$c_t^f = f m_t = f \sum_{i=1}^t r_i^P.$$

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

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

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



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

Transaction Costs of Trading

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

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

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

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

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

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

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

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

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

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

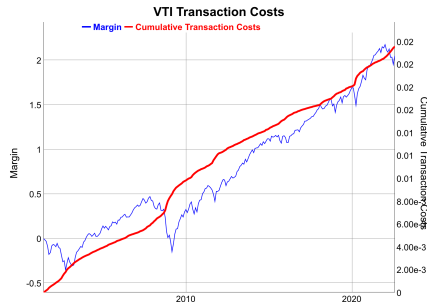
Transaction Costs of Single Asset Rebalancing

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

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

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

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



```
> # The bid-ask spread is equal to 1 bp for liquid ETFs
> bidask <- 0.001
> # Cumulative transaction costs
> costv <- bidask*cumsum(abs(retp))/2
> # Subtract transaction costs from margin account
> marginv <- cumsum(retp)
> marginv <- (marginv - costv)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, costv)
> colv <- c("Margin", "Transaction Costs")
> colnames(datav) <- colv
> dygraphs::dygraph(datav[ends], main="VTI Transaction Costs") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue") %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Combining the Returns of Multiple Assets

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

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

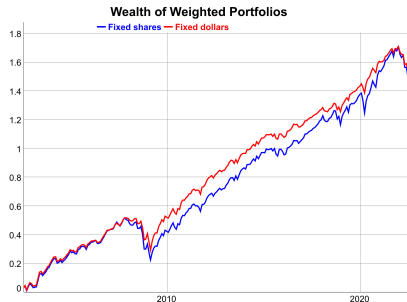
The portfolio allocations must be periodically rebalanced to keep the dollar amounts of the stocks proportional to the weights.

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

The *FDA* portfolio has a slightly higher Sharpe ratio than the *FSA* portfolio.

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

Jerzy Pawlowski (NYU Tandon)



```
> # Wealth of fixed dollars equal to $0.5 each (with rebalancing)
> wealthfd <- cumsum(retp %*% weightv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(log(wealthfs), wealthfd)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Fixed dollars")
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))))
> # Plot the log wealth
> colv <- colnames(wealthv)
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(wealthv[endd], main="Wealth of Weighted Portfolios")
+   dySeries(name=colv[1], col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Transaction Costs of Weighted Portfolio Rebalancing

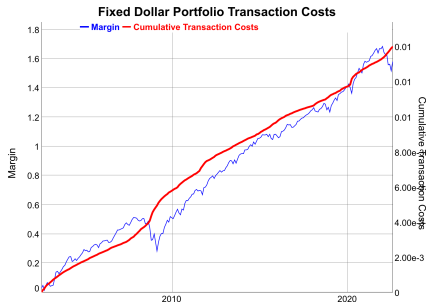
Maintaining a *fixed dollar allocation* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

Adding the weighted percentage returns is equivalent to investing in *fixed dollar amounts of stock* proportional to the weights w_1, w_2 .

The dollar amount of stock that must be traded in a given period is equal to the weighted sum of the absolute percentage returns: $w_1 |r_t^1| + w_2 |r_t^2|$.

The *transaction costs* c_t^r due to rebalancing are equal to half the *bid-ask spread* δ times the dollar amount of the traded stock: $c_t^r = \frac{\delta}{2} (w_1 |r_t^1| + w_2 |r_t^2|)$.

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



```
> # Margin account for fixed dollars (with rebalancing)
> marginv <- cumsum(retp %>% weightv)
> # Cumulative transaction costs
> costv <- bidask*cumsum(abs(retp) %>% weightv)/2
> # Subtract transaction costs from margin account
> marginv <- (marginv - costv)
> # dygraph plot of margin and transaction costs
> datav <- cbind(marginv, costv)
> datav <- xts::xts(datav, datev)
> colv <- c("Margin", "Transaction Costs")
> colnames(datav) <- colv
> dygraphs::dygraph(datav[end], main="Fixed Dollar Portfolio Trans
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue") %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Proportional Wealth Allocations

In the *proportional wealth allocation* strategy (PWA), the total wealth Π_t is allocated to the assets Π_i proportional to the portfolio weights w_i : $\Pi_i = w_i \Pi_t$.

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

The portfolio is rebalanced daily to maintain the dollar allocations Π_i equal to the total wealth $\Pi_t = \sum \Pi_i$ times the portfolio weights: w_i : $\Pi_i = w_i \Pi_t$.

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

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

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

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

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

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

Wealth of Proportional Wealth Allocations



```
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- cumsum(retd %*% (weightv/prici))
> # Or compound the percentage returns
> wealthfs <- drop(cumprod(1 + retp) %*% weightv)
> # Wealth of proportional wealth strategy (with rebalancing)
> wealthpr <- cumprod(1 + retp %*% weightv)
> wealthv <- cbind(wealthfs, wealthpr)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Prop wealth")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))))
> # Plot the log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Proportional Wealth Allocations") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Transaction Costs With Proportional Wealth Allocations

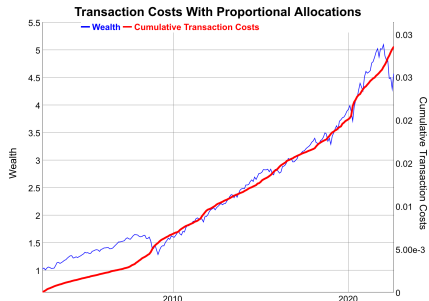
In each period the stocks must be rebalanced to maintain the proportional wealth allocations.

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

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

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

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



```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealthpr, costv)
> wealthv <- xts::xts(wealthv, datev)
> colv <- c("Wealth", "Transaction Costs")
> colnames(wealthv) <- colv
> dygraphs::dygraph(wealthv[ennd],
+   main="Transaction Costs With Equal Wealths") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue") %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Proportional Target Allocation Strategy

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

In the *proportional wealth allocation* strategy (*PWA*), the portfolio is rebalanced daily to maintain the dollar allocations Π_i equal to the total wealth $\Pi_t = \sum \Pi_i$ times the portfolio weights: $w_i: \Pi_i = w_i \Pi_t$.

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

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 *PWA*.

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

```
> # Wealth of fixed shares (without rebalancing)
> wealthfs <- drop(cumprod(1 + retp) %*% weightv)-1
> # Wealth of proportional wealth (with rebalancing)
> wealthpr <- cumprod(1 + retp %*% weightv) - 1
> # Wealth of proportional target allocation (with rebalancing)
> retp <- zoo::coredata(retp)
> threshv <- 0.05
> wealthv <- matrix(nrow=NROW(retp), ncol=2)
> colnames(wealthv) <- colnames(retp)
> wealthv[1, ] <- weightv
> for (it in 2:NROW(retp)) {
+   # Accrue wealth without rebalancing
+   wealthv[it, ] <- wealthv[it-1, ]*(1 + retp[it, ])
+   # Rebalance if wealth allocations differ from weights
+   if (sum(abs(wealthv[it, ] - sum(wealthv[it, ])*weightv))/sum(wea
+     # cat("Rebalance at:", it, "\n")
+     wealthv[it, ] <- sum(wealthv[it, ])*weightv
+   } # end if
+ } # end for
> wealthv <- rowSums(wealthv) - 1
> wealthv <- cbind(wealthpr, wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Equal Wealths", "Proportional Target")
> dygraphs::dygraph(wealthv, main="Wealth of Proportional Target All
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

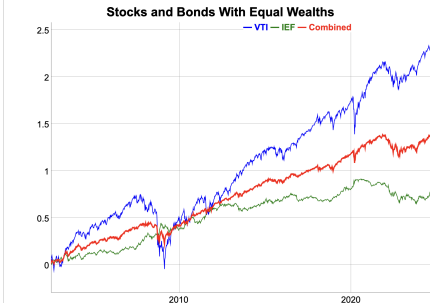

Stock and Bond Portfolio With Proportional Wealth Allocations

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

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

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

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



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

```
> # Wealth of equal wealth strategy
> wealthv <- cumsum(retp)
> # Calculate the a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> # Plot cumulative log wealth
> dygraphs::dygraph(wealthv[endd],
+   main="Stocks and Bonds With Equal Wealths") %>%
+   dyOptions(colors=c("blue", "green", "blue", "red")) %>%
+   dySeries("Combined", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Optimal Stock and Bond Portfolio Allocations

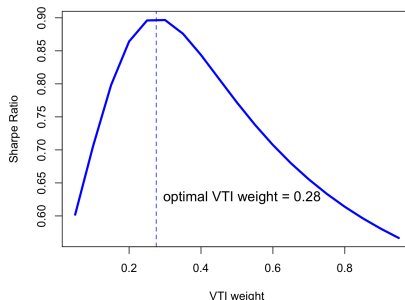
The optimal stock and bond weights can be calculated using optimization.

Using the past 20 years of data, the optimal *VTI* weight is about 0.28.

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

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

Sharpe Ratio as Function of VTI Weight



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

Simulating Wealth Scenarios Using Block Bootstrap

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

In block bootstrap, multiple rows of the time series data are sampled to generate new data.

The bootstrap data represent the possible cumulative *VTI* and *IEF* returns over the given holding period, based on past history.

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

```
> # Coerce the log prices from xts time series to matrix
> pricev <- na.omit(rutils::etfenv$prices[, c("VTI", "IEF")])
> pricev <- log(zoo::coredata(pricev))
> nrow <- NROW(pricev)
> holdp <- 10*252 # Holding period of 10 years in days
> # Sample the start dates for the bootstrap
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> startd <- sample.int(nrow-holdp, 1e3, replace=TRUE)
> # Bootstrap the wealth
> wealthv <- sapply(startd, function(x) {
+   pricev[x+holdp-1, ] - pricev[x, ]
+ }) # end sapply
> dim(wealthv)
```

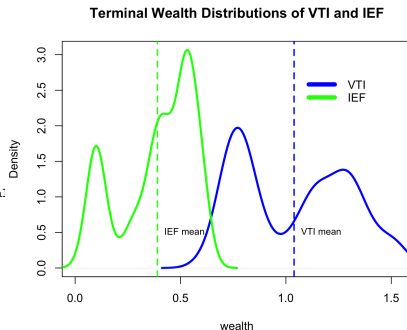
The Distributions of Terminal Wealth From Bootstrap

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

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

The distributions of wealth are bimodal (have two maxima) because there have been two historical market regimes: bull and bear markets.

```
> # Calculate the means and standard deviations of the terminal wea
> apply(wealthhv, 1, mean)
> apply(wealthhv, 1, sd)
> # Extract the terminal wealths of VTI and IEF
> vtiw <- wealthhv["VTI", ]
> iefw <- wealthhv["IEF", ]
```



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

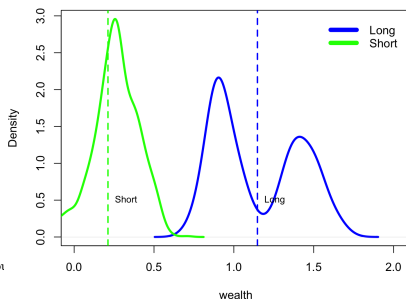
The Distribution of Stock Wealth and Holding Period

The distribution of stock wealth for short holding periods is close to symmetric.

The distributions for longer holding periods have larger means and standard deviations, and are more positively skewed, with a smaller kurtosis.

```
> # Calculate the distributions of stock wealth
> holdv <- nrows*seq(0.1, 0.5, 0.1)
> wealthm <- sapply(holdv, function(holdp) {
+   startd <- sample.int(nrows-holdp, 1e3, replace=TRUE)
+   wealthv <- sapply(startd, function(x) {
+     pricev[x+holdp-1, "VTI"] - pricev[x, "VTI"]
+   }) # end sapply
+ }) # end sapply
> dim(wealthm)
> colnames(wealthm) <- paste0(round(holdv/252), "years")
> # Calculate the skewness and kurtosis of the stock wealth distrib
> apply(wealthm, 2, function(x) {
+   # Standardize the returns
+   x <- (x - mean(x))/sd(x)
+   c(skew=mean(x^3), kurt=mean(x^4))
+ }) # end apply
```

Wealth Distributions for Long and Short Holding Periods



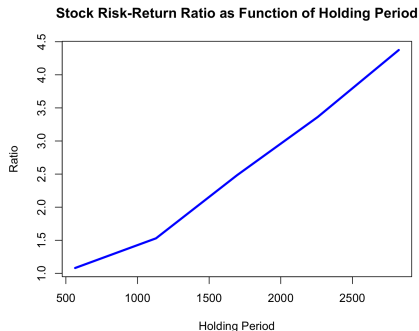
```
> # Plot the stock wealth for long and short holding periods
> wealth1 <- wealthm[, 5]
> wealth2 <- wealthm[, 1]
> mean1 <- mean(wealth1); mean2 <- mean(wealth2)
> dens1 <- density(wealth1); dens2 <- density(wealth2)
> plot(dens1, col="blue", lwd=3, xlab="wealth",
+   xlim=c(0, 2.5*max(dens2$x)), ylim=c(0, max(dens2$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="topright", legend=c("Long", "Short"),
+   inset=0.0, cex=1.0, bg="white", bty="n", v.intersp=0.5,
```

Risk-adjusted Stock Wealth and Holding Period

The risk-adjusted wealth measure is equal to the mean wealth divided by its standard deviation.

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

```
> # Define the risk-adjusted wealth measure
> riskretfun <- function(wealthv) {
+   mean(wealthv)/sd(wealthv)
+ } # end riskretfun
> # Calculate the stock wealth risk-return ratios
> riskrets <- apply(wealthm, 2, riskretfun)
```



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

Optimal Stock and Bond Portfolio Allocations From Bootstrap

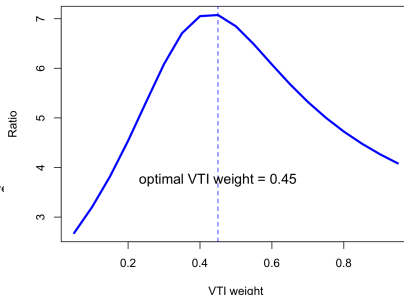
The optimal stock and bond weights can be calculated using block bootstrap simulation.

Bootstrapping the stock and bond returns over a 10 year holding period, the optimal *VTI* weight is about 0.45.

This weight is higher than the 0.28 calculated from the past 20 years of data, because the risk is measured as the standard deviation of the terminal wealth, not the standard deviation of the daily returns.

```
> # Calculate the distributions of portfolio wealth for different w
> weightv <- seq(0.05, 0.95, 0.05)
> wealthm <- sapply(weightv, function(weight) {
+   wealthv <- sapply(startd, function(x) {
+     (pricev[x+holdp-1, ] - pricev[x, ]) %*% c(weight, 1-weight)
+   }) # end sapply
+ }) # end sapply
> dim(wealthm)
> # Calculate the portfolio risk-return ratios
> riskrets <- apply(wealthm, 2, riskretfun)
> # Calculate the optimal VTI weight
> weightm <- weightv[which.max(riskrets)]
```

Portfolio Risk-Return Ratio as Function of VTI Weight



```
> # 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=0.5*max(riskrets), pos=3, cex=1.2,
+   labels=paste("optimal VTI weight =", round(weightm, 2)))
```

The All-Weather Portfolio

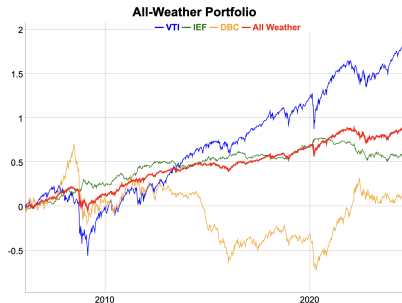
The *All-Weather* portfolio is a portfolio with proportional allocations of stocks (30%), bonds (55%), and commodities and precious metals (15%) (approximately).

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

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

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

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



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


Constant Proportion Portfolio Insurance Strategy

The *Constant Proportion Portfolio Insurance* (CPPI) strategy rebalances its portfolio between stocks and zero-coupon bonds.

The CPPI strategy has a fixed maturity date, and it's designed to pay back the initial principal investment at maturity.

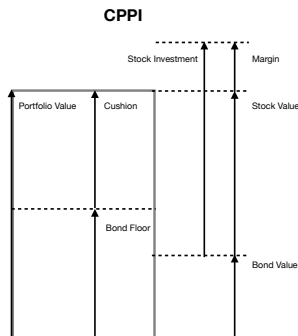
When stock prices are declining, the CPPI strategy sells its stocks and buys bonds, to protect against the loss of principal at maturity.

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.

The bond floor F is set so that its value at maturity is equal to par - the initial principal investment.

The value of the CPPI portfolio P , is equal to the sum of the stock SV and bond B values: $P = SV + B$.

The stock investment is levered by the *CPPI multiplier* C through borrowing. The amount invested in stocks SI is equal to the *cushion* ($P - F$) times the multiplier C : $SI = \max(C * (P - F), 0)$. The remaining amount of the portfolio is invested in zero-coupon bonds and is equal to: $B = \max(P - C * (P - F), 0)$.



Additional stock purchases can be funded by borrowing from a margin account.

The stock investment SI is the dollar amount of stocks owned by the CPPI strategy. The net value of the stocks SV is equal to the stock investment SI minus the borrowed amount M : $SV = SI - M$. If there's no borrowing, then the stock value is equal to the stock investment $SV = SI$.

CPPI Strategy Scenarios

Assume a *CPPI* strategy with a bond floor of $F = \$60$, and a multiplier of $C = 2$. The initial stock investment is: $SI = 2 * (\$100 - \$60) = \$80$, and the amount invested in bonds is: $B = \$100 - \$80 = \$20$.

Assume that stocks have a positive return of 20%, so that the stock value increases to $SV = \$80 * 1.2 = \96 , while the bond value remains: $B = \$20$. So that the portfolio value (wealth) increases to: $P = SV + B = \$96 + \$20 = \$116$.

The stock investment SI must increase to: $SI = 2 * (\$116 - \$60) = \$112$, while the bond investment must decrease to: $B = \$116 - \$112 = \$4$. The additional \$16 of stocks is purchased from the sale of the bonds.

Assume next that stocks have another positive return of 20%, so that the stock value increases to $SV = \$112 * 1.2 = \134.4 , while the bond value remains: $B = \$4$. So that the wealth increases to $P = SV + B = \$134.4 + \$4 = \$138.4$.

The stock investment SI must increase to: $SI = 2 * (\$134.4 - \$60) = \$148.8$, while the bond investment drops to zero: $B = \max(\$138.4 - \$148.8, 0) = \$0$. The additional \$14.4 of stocks is purchased from the sale of \$4 of the bonds plus \$10.4 of borrowed money.

If stock prices keep rising, then the stock investment and the CPPI portfolio value both increase, and the bond investment decreases.

But if stock prices decline, then the stock investment and the CPPI portfolio value both decrease, and the CPPI strategy sells its stocks and buys bonds.

Assume that stocks have a negative return of -20% , so that the stock value decreases to $SV = \$80 * 0.8 = \64 . Then the portfolio value (wealth) decreases to: $P = SV + B = \$64 + \$20 = \$84$.

The stock investment SI must decrease to: $SI = 2 * (\$84 - \$60) = \$48$, while the bond investment must increase to: $B = \$84 - \$48 = \$36$. The additional \$16 of bonds is purchased from the sale of the stocks.

If stock prices keep declining and the portfolio value P drops to the bond floor F , then all the stocks are sold, and only the zero-coupon bond remains, so that at maturity the investor is paid back at least the full initial principal $P = \$100$.

CPPI Strategy Simulation

Simulating the CPPI strategy requires performing a `for()` loop over time, since the strategy is path-dependent.

At each time step, the stock price and portfolio value are updated based on the stock return.

Then the CPPI leverage is applied to the stock investment based on the updated portfolio value.

Applying the CPPI leverage may require borrowing money to buy more stocks (we assume zero borrowing costs).

The amount invested in stocks changes both because the stock price changes and because of additional CPPI leverage.

The stock purchases are funded from the sale of bonds and with borrowing from a margin account.

```
> # Calculate the VTI returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrow <- NROW(retp)
> # Bond floor
> bfloor <- 60
> # CPPI multiplier
> coeff <- 2
> # Portfolio market values
> portfv <- numeric(nrow)
> # Initial principal
> portfv[1] <- 100
> # Stock investment
> stocki <- numeric(nrow)
> stocki[1] <- max(coeff*(portfv[1] - bfloor), 0)
> # Stock value
> stockv <- numeric(nrow)
> stockv[1] <- stocki[1]
> # Bond value
> bondv <- numeric(nrow)
> bondv[1] <- max(portfv[1] - stocki[1], 0)
> # Margin account
> margv <- numeric(nrow)
> # Simulate the CPPI strategy
> for (t in 2:nrow) {
+   # Update the portfolio value
+   stocki[t] <- (1 + retp[t])*stocki[t-1]
+   stockv[t] <- stocki[t] - margv[t-1]
+   portfv[t] <- stockv[t] + bondv[t-1]
+   # Update the CPPI leverage
+   stockt <- max(coeff*(portfv[t] - bfloor), 0)
+   bondv[t] <- max(portfv[t] - stockt, 0)
+   margv[t] <- (bondv[t] - bondv[t-1]) + (stockt - stocki[t]) + margv[t-1]
+   stocki[t] <- stockt
+ } # end for
```

CPPI Strategy Dynamics

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.

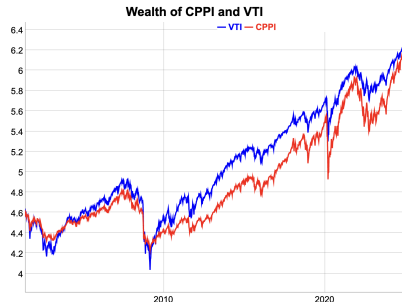


```
> pricev <- 100*cumprod(1 + retp)
> datav <- cbind(stockv, bondv, portfv, pricev)["2008/2009"]
> colnames(datav) <- c("stocks", "bonds", "CPPI", "VTI")
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endd], main="CPPI Strategy") %>%
+   dyOptions(colors=c("red", "green", "blue", "black"), strokeWidth=2)
+   dyLegend(show="always", width=300)
```

CPPI Strategy Performance

The performance of the CPPI strategy depends on the level of the bond floor F and the multiplier C , with larger returns for higher values of C , but also higher risk.

When stocks rally, the margin account grows and the bond allocation decreases.



```
> # Plot the CPPI margin account
> margv <- cbind(pricev, margv)
> colnames(margv)[2] <- "margin"
> endd <- rutils::calc_endpoints(margv, interval="weeks")
> dygraphs::dygraph(margv[endd], main="CPPI Margin and VTI") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Calculate the Sharpe of CPPI wealth
> wealthv <- cbind(pricev, portfv)
> colnames(wealthv)[2] <- "CPPI"
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))))
> # Plot the CPPI wealth
> dygraphs::dygraph(log(wealthv[endd]), main="Wealth of CPPI and VTI")
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy For Stocks and Bonds

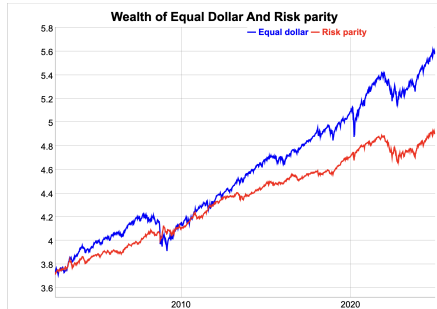
The stock dollar amounts of the risk parity strategy are such that they have *equal dollar volatilities*.

The stock dollar amounts must be inversely proportional to the dollar volatilities: $w_i \propto \frac{1}{\sigma_i}$.

The risk parity strategy has a higher Sharpe ratio than the fixed share strategy with equal initial dollar amounts because it's overweight bonds. But it also has lower absolute returns.

The risk parity strategy was developed in the early 1990s by *Bridgewater Associates*.

```
> # Calculate the dollar and percentage returns of VTI and IEF
> pricev <- na.omit(rutils::etfenv$prices[, c("VTI", "IEF")])
> datev <- zoo::index(pricev)
> retv <- rutils::diffit(pricev)
> retv[1, ] <- retv[2, ]
> retp <- retv/rutils::lagit(pricev, lag=1, pad_zeros=FALSE)
> # Calculate the risk parity weights
> weightv <- 1/apply(retv, sd)
> weightv <- weightv/sum(weightv)
> # Wealth of risk parity (fixed shares)
> wealthrp <- drop(pricev %*% weightv)
> # Wealth of equal dollar (fixed shares)
> weightv <- 1/as.numeric(pricev[1, ])
> weightv <- weightv/sum(weightv)
> wealthv <- (pricev %*% weightv)
> # Scale the wealth to start equal to wealthrp
> wealthv <- wealthrp[1]*wealthv/wealthrp[1]
```



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

The Rolling Risk Parity Allocations

In practice, the dollar volatility σ_t changes over time so it must be recalculated, and the weights must be updated: $w_t \propto \frac{1}{\sigma_t}$.

The stock allocations (*dollar amounts*) increase when the volatility is low, and vice versa.

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

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

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

Where σ_t^2 is the trailing variance at time t .

The decay factor λ determines how quickly the variance estimates are updated, with smaller values of λ producing faster updating, giving more weight to recent returns, and vice versa.



```
> # Calculate the trailing dollar volatilities
> lambdav <- 0.99
> vold <- HighFreq::run_var(retd, lambda=lambdav)
> vold <- sqrt(vold[, 3:4])
> # Calculate the rolling risk parity weights
> weightv <- ifelse(vold > 0, 1/vold, 1)
> weightv <- weightv/rowSums(weightv)
> # Calculate the risk parity allocations
> pricerp <- pricev*weightv
> # Plot the risk parity allocations
> colnames(pricerp) <- c("Stocks", "Bonds")
> dygraph(log(pricerp[end]), main="Risk Parity Allocations") %>%
+   dyOptions(colors=c("red", "blue"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Risk Parity Strategy

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

The dollar returns of risk parity r_d are equal to the percentage returns of the original prices r_t times the stock allocations p_a : $r_d = r_t p_a$.

The risk parity strategy for stocks and bonds has a higher Sharpe ratio than the proportional wealth strategy. But it also has lower absolute returns.

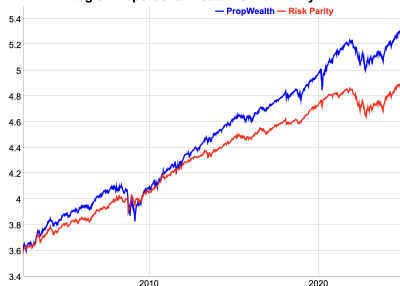
The absolute returns can be increased by increasing the leverage - buying more stocks and bonds with borrowed money.

Risk parity performs better for assets with negative or low correlations of returns, like stocks and bonds.

Risk parity performed poorly during the Covid crisis because of the *simultaneous losses of both stocks and bonds* (positive correlations).

```
> # Calculate the dollar returns of risk parity
> retrp <- retrp:rutils::lagit(pricerp)
> retrp[1, ] <- pricerp[1, ]
> # Calculate the wealth of risk parity
> wealthrp <- cumsum(rowSums(retrp))
> # Wealth of proportional wealth (with rebalancing)
> wealthpr <- cumprod(1 + rowMeans(retrp))
> wealthpr <- wealthpr*wealthrp[1]
```

Log of Proportional Wealth vs Risk Parity



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(wealthpr, wealthrp)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("PropWealth", "Risk Parity")
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))))
> # Plot a dygraph of the log wealths
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(log(wealthv[endd]),
+   main="Log of Proportional Wealth vs Risk Parity") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```


Risk Parity Strategy Market Timing Skill

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

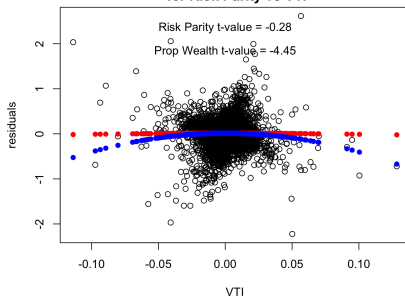
This allows the risk parity strategy to time the markets better - selling when prices are dropping and buying when prices are rising. But only when the changes of the volatility are significant.

But the t-value of the risk parity strategy is negative, because it's contrarian when the volatility is stable.

The t-value of the proportional wealth allocation strategy is negative, because it's contrarian - it buys stocks when their prices drop.

```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> retp <- rutils::diffit(wealthv)
> retvti <- retp$VTI
> desm <- cbind(retp, retvti, retvti^2)
> colnames(desm)[1:2] <- c("prop", "riskp")
> colnames(desm)[4] <- "Treynor"
> regmod <- lm(riskp ~ VTI + Treynor, data=desm)
> summary(regmod)
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Risk Parity v
```

**Treynor-Mazuy Market Timing Test
for Risk Parity vs VTI**



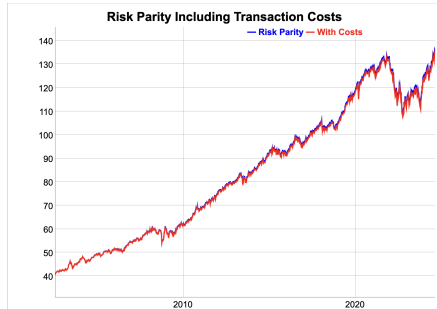
```
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["Treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.9*max(resids), paste("Risk Parity t-value =", tvalue))
> # Test for equal wealth strategy market timing of VTI using Treynor-Mazuy test
> regmod <- lm(prop ~ VTI + Treynor, data=desm)
> summary(regmod)
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> points.default(x=retvti, y=fitv, pch=16, col="blue")
> text(x=0.0, y=0.7*max(resids), paste("Prop Wealth t-value =", tvalue))
```

Transaction Costs of Risk Parity Strategy

The risk parity strategy has higher transaction costs because it trades more shares than the proportional wealth strategy, and it may also use leverage to buy more shares.

But the higher transaction costs are not large enough to completely cancel the higher returns of the strategy.

```
> # Total dollar amount of stocks that need to be traded
> notx <- rutils::diffit(pricerp)
> # The bid-ask spread is equal to 1 bp for liquid ETFs
> bidask <- 0.001
> # Calculate the cumulative transaction costs
> costv <- 0.5*bidask*cumsum(rowSums(abs(notx)))
```



```
> # dygraph plot of wealth and transaction costs
> wealthv <- cbind(wealthrp, wealthrp-costv)
> wealthv <- xts::xts(wealthv, datev)
> colv <- c("Risk Parity", "With Costs")
> colnames(wealthv) <- colv
> dygraphs::dygraph(wealthv[end], main="Risk Parity Including Transaction Costs")
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Homework Assignment

Required

- Study all the lecture slides in `FRE7241_Lecture_1.pdf`, and run all the code in `FRE7241_Lecture_1.R`,

Recommended

- Read the documentation for packages `rutils.pdf` and `HighFreq.pdf`,