FRE7241 Algorithmic Portfolio Management
Lecture#2, Fall 2021

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

September 14, 2021

Risk-adjusted Return Measures

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

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

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

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

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

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

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

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

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

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

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

Risk and Return of Aggregated Returns

When stock returns are aggregated over a longer holding period, then their skewness, kurtosis, and tail risks decrease significantly.

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

```
> # Calculate VTI percentage returns
> re_turns <- na.omit(rutils::etf_env$re_turns$VTI)
> re_turns <- drop(zoo::coredata(re_turns))
> n_rows <- NROW(re_turns)
> # Calculate aggregated VTI returns
> n_agg <- 252
> agg_rets <- sapply(1:n_rows, function(x) {
+     sum(re_turns[sample.int(n_rows, size=n_agg, replace=TRUE)])
+ })  # end sapply
> mean(re_turns)
> mean(agg_rets)/n_agg
> # Calculate standard deviation, skewness, and kurtosis
> da_ta <- cbind(re_turns, agg_rets)
> colnames(da_ta) <- c("VTI", "Agg")
> apply(da_ta, MARGIN=2, function(x) {
+   # Calculate standard deviation
+   stddev <- sd(x)
+   # Standardize the returns
+   x <- (x - mean(x))/stddev
+   c(stddev=stddev, skew=mean(x^3), kurt=mean(x^4))
+ })  # end sapply
> # Calculate the Sharpe ratios
> conf_level <- 0.05
> std_dev <- sd(re_turns)
> va_r <- unname(quantile(re_turns, probs=conf_level))
> c_var <- mean(re_turns[re_turns < va_r])
> sqrt(252)*mean(re_turns)/c(Sharpe=std_dev, Dowd=-va_r, DowdC=-c_va
> # Calculate the Sharpe ratios of aggregated returns
> std_dev <- sd(agg_rets)
> va_r <- unname(quantile(agg_rets, probs=conf_level))
> c_var <- mean(agg_rets[agg_rets < va_r])
> sqrt(252/n_agg)*mean(agg_rets)/c(Sharpe=std_dev, Dowd=-va_r, Dowd(
```

Tests for Market Timing Skill

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

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

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

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

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

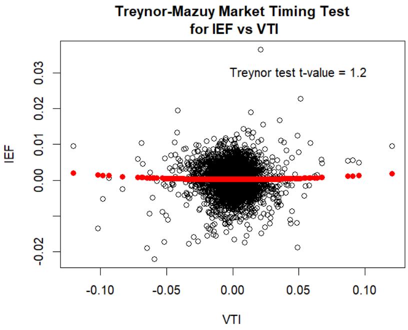
If the coefficient γ is statistically significant, then it's very likely due to *market timing* skill.

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

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

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

```
> # Test if IEF can time VTI
> re_turns <- na.omit(rutils::etf_env$re_turns[, c("IEF", "VTI")])
> vt_i <- re_turns$VTI
> de_sign <- cbind(re_turns, 0.5*(vt_i+abs(vt_i)), vt_i^2)
> colnames(de_sign)[3:4] <- c("merton", "treynor")
```




```
> # Merton-Henriksson test
> mod_el <- lm(IEF ~ VTI + merton, data=de_sign); summary(mod_el)
> # Treynor-Mazuy test
> mod_el <- lm(IEF ~ VTI + treynor, data=de_sign); summary(mod_el)
> # Plot residual scatterplot
> x11(width=6, height=5)
> residual_s <- (de_sign$IEF - mod_el$coeff[2]*de_sign$VTI)
> plot.default(x=de_sign$VTI, y=residual_s, xlab="VTI", ylab="IEF")
> title(main="Treynor-Mazuy Market Timing Test\n for IEF vs VTI", li
> # Plot fitted (predicted) response values
> fit_ted <- (mod_el$coeff["(Intercept)"] +
+         mod_el$coeff["treynor"]*vt_i^2)
> points.default(x=de_sign$VTI, y=fit_ted, pch=16, col="red")
> text(x=0.05, y=0.03, paste("Treynor test t-value =", round(summary
```

Calculating Asset Returns

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

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

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

```
> # Extract ETF prices from rutils::etf_env$price_s
> price_s <- rutils::etf_env$price_s
> price_s <- zoo::na.locf(price_s, na.rm=FALSE)
> price_s <- zoo::na.locf(price_s, fromLast=TRUE)
> # Calculate simple dollar returns
> rets_dollar <- rutils::diff_it(price_s)
> # Or
> # rets_dollar <- lapply(price_s, rutils::diff_it)
> # rets_dollar <- rutils::do_call(cbind, rets_dollar)
> # Calculate log returns
> rets_log <- rutils::diff_it(log(price_s))
> # Calculate percentage returns
> rets_percent <- rets_dollar/
+   rutils::lag_it(price_s, lagg=1, pad_zeros=FALSE)
```

Compounding Asset Returns

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

The compounded percentage returns: $\prod_{i=1}^{n} r_i^p$ also represent the wealth from owning a *fixed number of shares*.

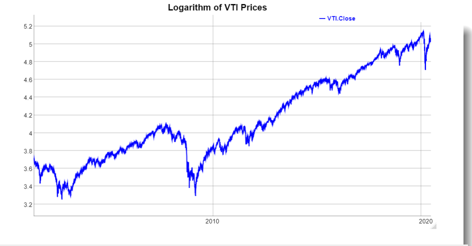
The sum of the percentage returns (without compounding): $\sum_{i=1}^{n} r_i^p$ represents the wealth from owning a *fixed dollar amount* of stock.

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

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

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

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




```
> # Calculate prices from simple dollar returns
> rets_dollar[1, ] <- price_s[1, ]
> new_prices <- cumsum(rets_dollar)
> all.equal(new_prices, price_s)
> # Compound the percentage returns
> new_prices <- cumprod(1+ rets_percent)
> new_prices <- lapply(1:NCOL(new_prices), function (i)
+     init_prices[i]*new_prices[, i])
> new_prices <- rutils::do_call(cbind, new_prices)
> all.equal(new_prices, price_s)
> # Sum the percentage returns
> new_prices <- cumsum(rets_percent)
> methods(cumsum)
> new_prices <- lapply(1:NCOL(new_prices), function (i)
+     new_prices[, i] + log(init_prices[i]))
> new_prices <- rutils::do_call(cbind, new_prices)
> # Only approximately equal
> all.equal(new_prices, log(price_s))
> # Plot log VTI prices
> dygraphs::dygraph(log(quantmod::Cl(rutils::etf_env$VTI)),
+   main="Logarithm of VTI Prices") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Funding Costs of Single Asset Rebalancing

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

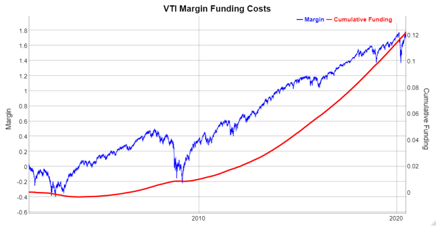
The cash in the *margin account* can be positive (accumulated profits) or negative (losses).

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

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

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

```
> # Calculate percentage VTI returns
> price_s <- rutils::etf_env$price_s$VTI
> price_s <- na.omit(price_s)
> re_turns <- rutils::diff_it(price_s)/
+   rutils::lag_it(price_s, lagg=1, pad_zeros=FALSE)
```




```
> # Funding rate per day
> f_rate <- 0.01/252
> # Margin account
> mar_gin <- cumsum(re_turns)
> # Cumulative funding costs
> f_costs <- cumsum(f_rate*mar_gin)
> # Add funding costs to margin account
> mar_gin <- (mar_gin + f_costs)
> # dygraph plot of margin and funding costs
> da_ta <- cbind(mar_gin, f_costs)
> col_names <- c("Margin", "Cumulative Funding")
> colnames(da_ta) <- col_names
> dygraphs::dygraph(da_ta, main="VTI Margin Funding Costs") %>%
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", col="blue") %>%
+   dySeries(name=col_names[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Transaction Costs of Trading

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

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

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

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

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

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

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

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

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

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

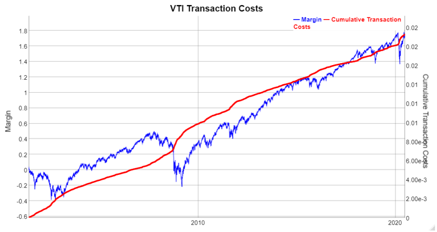
Transaction Costs of Single Asset Rebalancing

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

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

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

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




```
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Cumulative transaction costs
> cost_s <- bid_offer*cumsum(abs(re_turns))/2
> # Subtract transaction costs from margin account
> mar_gin <- cumsum(re_turns)
> mar_gin <- (mar_gin - cost_s)
> # dygraph plot of margin and transaction costs
> da_ta <- cbind(mar_gin, cost_s)
> col_names <- c("Margin", "Cumulative Transaction Costs")
> colnames(da_ta) <- col_names
> dygraphs::dygraph(da_ta, main="VTI Transaction Costs") %>%
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", col="blue") %>%
+   dySeries(name=col_names[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Combining the Returns of Multiple Assets

There are several ways of combining the returns of multiple assets.

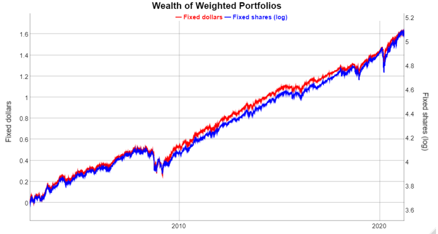
Adding the weighted simple (dollar) returns is equivalent to buying a *fixed number of shares* proportional to the weights.

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

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

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

```
> # Calculate VTI and IEF dollar returns
> price_s <- rutils::etf_env$price_s[, c("VTI", "IEF")]
> price_s <- na.omit(price_s)
> date_s <- index(price_s)
> rets_dollar <- rutils::diff_it(price_s)
> # Calculate VTI and IEF percentage returns
> rets_percent <- rets_dollar/
+   rutils::lag_it(price_s, lagg=1, pad_zeros=FALSE)
```




```
> # Wealth of fixed shares (without rebalancing)
> weight_s <- c(0.5, 0.5)
> rets_dollar[1, ] <- price_s[1, ]
> wealth_fixed_shares <- cumsum(rets_dollar %*% weight_s)
> # Wealth of fixed dollars (with rebalancing)
> wealth_fixed_dollars <- cumsum(rets_percent %*% weight_s)
> # Plot log wealth
> weal_th <- cbind(wealth_fixed_dollars, log(wealth_fixed_shares))
> weal_th <- xts::xts(weal_th, index(price_s))
> colnames(weal_th) <- c("Fixed dollars", "Fixed shares (log)")
> col_names <- colnames(weal_th)
> dygraphs::dygraph(weal_th, main="Wealth of Weighted Portfolios") %
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", col="red", strokeWidth=2)
+   dySeries(name=col_names[2], axis="y2", col="blue", strokeWidth=
+   dyLegend(show="always", width=500)
```

Transaction Costs of Weighted Portfolio Rebalancing

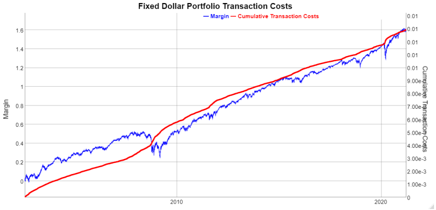
Maintaining a *fixed dollar amount* of stock requires periodic *rebalancing*, selling shares when their price goes up, and vice versa.

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

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

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

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




```r
> # Margin account for fixed dollars (with rebalancing)
> mar_gin <- cumsum(rets_percent %*% weight_s)
> # Cumulative transaction costs
> cost_s <- bid_offer*cumsum(abs(rets_percent) %*% weight_s)/2
> # Subtract transaction costs from margin account
> mar_gin <- (mar_gin - cost_s)
> # dygraph plot of margin and transaction costs
> da_ta <- cbind(mar_gin, cost_s)
> da_ta <- xts::xts(da_ta, index(price_s))
> col_names <- c("Margin", "Cumulative Transaction Costs")
> colnames(da_ta) <- col_names
> dygraphs::dygraph(da_ta, main="Fixed Dollar Portfolio Transaction
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", col="blue") %>%
+   dySeries(name=col_names[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Portfolio With Fixed Ratios of Dollar Amounts

Consider a portfolio with *fixed ratios of dollar amounts*, not fixed dollar amounts.

The total wealth is equal to the portfolio market value, so there's no margin account.

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

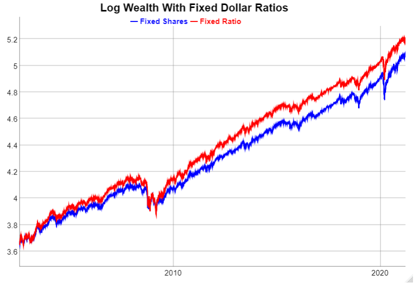
The total portfolio wealth at time t is equal to the wealth at time $t-1$ multiplied by the weighted returns: $w_t = w_{t-1}(1 + \bar{r}_t)$.

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

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

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

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




```
> # Wealth of fixed shares (without rebalancing)
> wealth_fixed_shares <- cumsum(rets_dollar %*% weight_s)
> # Calculate weighted percentage returns
> rets_weighted <- rets_percent %*% weight_s
> # Wealth of fixed ratio of dollar amounts (with rebalancing)
> wealth_fixed_ratio <- cumprod(1 + rets_weighted)
> wealth_fixed_ratio <- wealth_fixed_shares[1]*wealth_fixed_ratio
> # Plot log wealth
> weal_th <- log(cbind(wealth_fixed_shares, wealth_fixed_ratio))
> weal_th <- xts::xts(weal_th, index(price_s))
> colnames(weal_th) <- c("Fixed Shares", "Fixed Ratio")
> dygraphs::dygraph(weal_th, main="Log Wealth of Fixed Dollar Ratios
+   dyOptions(colors=c("blue","red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Transaction Costs With Fixed Dollar Ratios

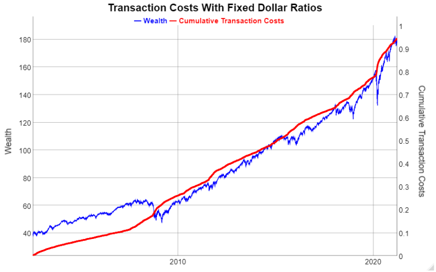
In each period the stocks must be rebalanced to maintain the fixed ratio of dollar amounts.

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

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

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

```
> # Returns in excess of weighted returns
> ex_cess <- lapply(rets_percent, function(x) (rets_weighted - x))
> ex_cess <- do.call(cbind, ex_cess)
> sum(ex_cess %*% weight_s)
> # Calculate weighted sum of absolute excess returns
> ex_cess <- abs(ex_cess) %*% weight_s
> # Total dollar amount of stocks that need to be traded
> ex_cess <- ex_cess*rutils::lag_it(wealth_fixed_ratio)
> # Cumulative transaction costs
> cost_s <- bid_offer*cumsum(ex_cess)/2
> # Subtract transaction costs from wealth
> wealth_fixed_ratio <- (wealth_fixed_ratio - cost_s)
```




```
> # dygraph plot of wealth and transaction costs
> weal_th <- cbind(wealth_fixed_ratio, cost_s)
> weal_th <- xts::xts(weal_th, index(price_s))
> col_names <- c("Wealth", "Cumulative Transaction Costs")
> colnames(weal_th) <- col_names
> dygraphs::dygraph(weal_th, main="Transaction Costs With Fixed Doll
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", col="blue") %>%
+   dySeries(name=col_names[2], axis="y2", col="red", strokeWidth=3)
+   dyLegend(show="always", width=500)
```

Stock and Bond Portfolio With Fixed Dollar Ratio

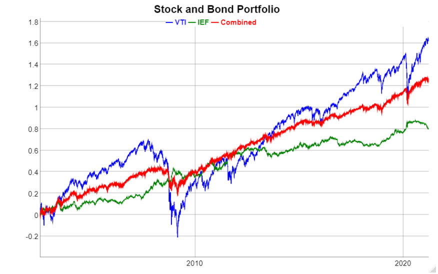
Portfolios combining stocks and bonds can provide a much better risk versus return tradeoff than either of the assets separately, because the returns of stocks and bonds are usually negatively correlated, so they are natural hedges of each other.

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

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

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

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




```
> # Wealth of fixed ratio of dollar amounts
> weal_th <- cumprod(1 + re_turns)
> # Plot cumulative wealth
> dygraphs::dygraph(log(weal_th), main="Stock and Bond Portfolio") %
+   dyOptions(colors=c("blue","green","blue","red")) %>%
+   dySeries("Combined", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

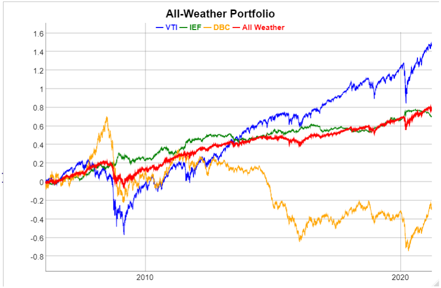
The *All-Weather* Portfolio

The *All-Weather* portfolio is a static portfolio of stocks (30%), bonds (55%), and commodities and precious metals (15%) (approximately), and was designed by Bridgewater Associates, the largest hedge fund in the world:
https://www.bridgewater.com/research-library/the-all-weather-strategy/
http://www.nasdaq.com/article/remember-the-allweather-portfolio-its-having-a-killer-year-cm6855

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

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

```
> # Extract ETF returns
> sym_bols <- c("VTI", "IEF", "DBC")
> re_turns <- na.omit(rutils::etf_env$re_turns[, sym_bols])
> # Calculate all-weather portfolio wealth
> weights_aw <- c(0.30, 0.55, 0.15)
> re_turns <- cbind(re_turns, re_turns %*% weights_aw)
> colnames(re_turns)[4] <- "All Weather"
```




```
> # Calculate cumulative wealth from returns
> weal_th <- cumsum(re_turns)
> # dygraph all-weather wealth
> dygraphs::dygraph(weal_th, main="All-Weather Portfolio") %>%
+   dyOptions(colors=c("blue", "green", "orange", "red")) %>%
+   dySeries("All Weather", color="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot all-weather wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("orange", "blue", "green", "red")
> quantmod::chart_Series(weal_th, theme=plot_theme, lwd=c(2, 2, 2,
+        name="All-Weather Portfolio")
> legend("topleft", legend=colnames(weal_th),
+   inset=0.1, bg="white", lty=1, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```

Risk Parity Strategy

In a risk parity strategy the dollar portfolio allocations are rebalanced daily so that their dollar volatilities remain the same.

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

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

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

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

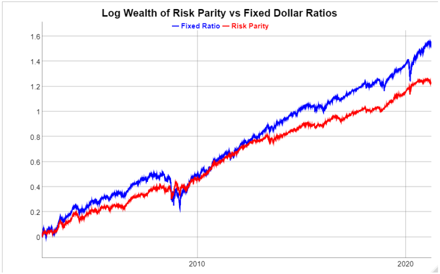
The risk parity strategy is also called the equal risk contributions (ERC) strategy.

```
> # Calculate dollar and percentage returns for VTI and IEF.
> price_s <- rutils::etf_env$price_s[, c("VTI", "IEF")]
> price_s <- na.omit(price_s)
> rets_dollar <- rutils::diff_it(price_s)
> rets_percent <- rets_dollar/rutils::lag_it(price_s, lagg=1, pad_ze
> # Calculate wealth of fixed ratio of dollar amounts.
> weight_s <- c(0.5, 0.5)
> rets_weighted <- rets_percent %*% weight_s
> wealth_fixed_ratio <- cumprod(1 + rets_weighted)
> # Calculate rolling percentage volatility.
> look_back <- 21
> vo_l <- roll::roll_sd(rets_percent, width=look_back)
> vo_l <- zoo::na.locf(vo_l, na.rm=FALSE)
> vo_l <- zoo::na.locf(vo_l, fromLast=TRUE)
> # Calculate the risk parity portfolio allocations.
> allocation_s <- lapply(1:NCOL(price_s),
+   function(x) weight_s[x]/vo_l[, x])
> allocation_s <- do.call(cbind, allocation_s)
> # Scale allocations to 1 dollar total.
> allocation_s <- allocation_s/rowSums(allocation_s)
> # Lag the allocations
> allocation_s <- rutils::lag_it(allocation_s)
> # Calculate wealth of risk parity.
> rets_weighted <- rowSums(rets_percent*allocation_s)
> wealth_risk_parity <- cumprod(1 + rets_weighted)
```

Risk Parity Strategy Performance

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

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



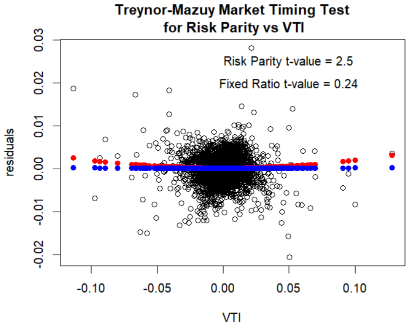

```
> # Calculate the log wealths.
> weal_th <- log(cbind(wealth_fixed_ratio, wealth_risk_parity))
> weal_th <- xts::xts(weal_th, index(price_s))
> colnames(weal_th) <- c("Fixed Ratio", "Risk Parity")
> # Calculate the Sharpe ratios.
> sqrt(252)*sapply(rutils::diff_it(weal_th), function (x) mean(x)/sc
> # Plot a dygraph of the log wealths.
> dygraphs::dygraph(weal_th, main="Log Wealth of Risk Parity vs Fixe
+   dyOptions(colors=c("blue","red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Risk Parity Strategy Market Timing Skill

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

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

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



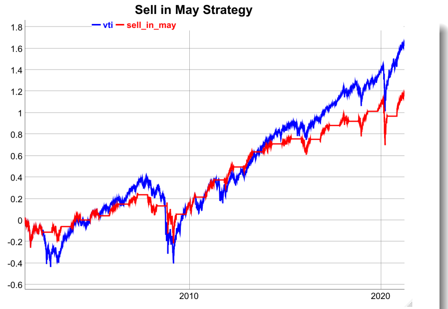

```
> # Test risk parity market timing of VTI using Treynor-Mazuy test
> re_turns <- rutils::diff_it(weal_th)
> vt_i <- rets_percent$VTI
> de_sign <- cbind(re_turns, vt_i, vt_i^2)
> de_sign <- na.omit(de_sign)
> colnames(de_sign)[1:2] <- c("fixed","risk_parity")
> colnames(de_sign)[4] <- "treynor"
> mod_el <- lm(risk_parity ~ VTI + treynor, data=de_sign)
> summary(mod_el)
> # Plot residual scatterplot
> residual_s <- (de_sign$risk_parity - mod_el$coeff[2]*de_sign$VTI)
> residual_s <- mod_el$residuals
> x11(width=6, height=5)
> plot.default(x=de_sign$VTI, y=residual_s, xlab="VTI", ylab="resi
> title(main="Treynor-Mazuy Market Timing Test\n for Risk Parity vs
> # Plot fitted (predicted) response values
> fit_ted <- (mod_el$coeff["(Intercept)"] +
+         mod_el$coeff["treynor"]*vt_i^2)
> points.default(x=de_sign$VTI, y=fit_ted, pch=16, col="red")
> text(x=0.05, y=0.025, paste("Risk Parity t-value =", round(summary(mod_el)$coeff["treynor", "t value"], 2)))
```

```
> # Test for fixed ratio market timing of VTI using Treynor-Mazuy te
> mod_el <- lm(fixed ~ VTI + treynor, data=de_sign)
> summary(mod_el)
> # Plot fitted (predicted) response values
> fit_ted <- (mod_el$coeff["(Intercept)"] + mod_el$coeff["treynor"]*
> points.default(x=de_sign$VTI, y=fit_ted, pch=16, col="blue")
> text(x=0.05, y=0.02, paste("Fixed Ratio t-value =", round(summary(
```

Sell in May Calendar Strategy

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

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

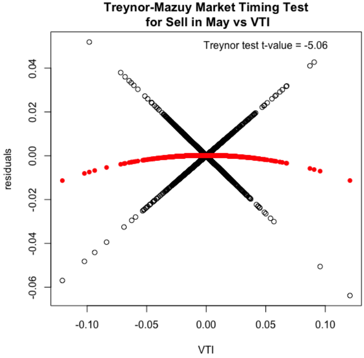



```
> # Plot wealth of Sell in May strategy
> dygraphs::dygraph(cumsum(weal_th), main="Sell in May Strategy") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # OR: Open x11 for plotting
> x11(width=6, height=5)
> par(mar=c(4, 4, 3, 1), oma=c(0, 0, 0, 0))
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("blue", "red")
> quantmod::chart_Series(weal_th, theme=plot_theme, name="Sell in Ma
> legend("topleft", legend=colnames(weal_th),
+   inset=0.1, bg="white", lty=1, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```

Sell in May Strategy Market Timing

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

```
> # Test if Sell in May strategy can time VTI
> de_sign <- cbind(vt_i, 0.5*(vt_i+abs(vt_i)), vt_i^2)
> colnames(de_sign) <- c("VTI", "merton", "treynor")
> # Perform Merton-Henriksson test
> mod_el <- lm(sell_inmay ~ VTI + merton, data=de_sign)
> summary(mod_el)
> # Perform Treynor-Mazuy test
> mod_el <- lm(sell_inmay ~ VTI + treynor, data=de_sign)
> summary(mod_el)
> # Plot Treynor-Mazuy residual scatterplot
> residual_s <- (sell_inmay - mod_el$coeff[2]*vt_i)
> plot.default(x=vt_i, y=residual_s, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for Sell in May vs
> # Plot fitted (predicted) response values
> fit_ted <- (mod_el$coeff["(Intercept)"] +
+         mod_el$coeff["treynor"]*vt_i^2)
> points.default(x=vt_i, y=fit_ted, pch=16, col="red")
> text(x=0.05, y=0.05, paste("Treynor test t-value =", round(summary
```



Seasonal Overnight Market Anomaly

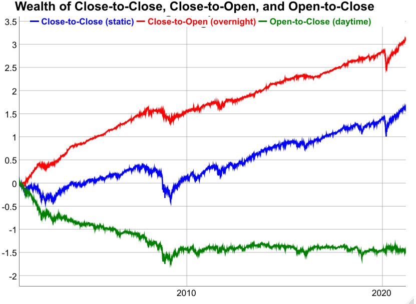
The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the daytime returns.

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

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

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

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




```
> # Calculate the log of OHLC VTI prices
> oh_lc <- log(rutils::etf_env$VTI)
> op_en <- quantmod::Op(oh_lc)
> hi_gh <- quantmod::Hi(oh_lc)
> lo_w <- quantmod::Lo(oh_lc)
> clos_e <- quantmod::Cl(oh_lc)
> # Calculate the close-to-close log returns, the intraday
> # open-to-close returns and the overnight close-to-open returns.
> close_close <- rutils::diff_it(clos_e)
> colnames(close_close) <- "close_close"
> open_close <- (clos_e - op_en)
> colnames(open_close) <- "open_close"
> close_open <- (op_en - rutils::lag_it(clos_e, lagg=1, pad_zeros=FALSE))
> colnames(close_open) <- "close_open"
```

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

Turn of the Month Effect

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

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

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

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


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

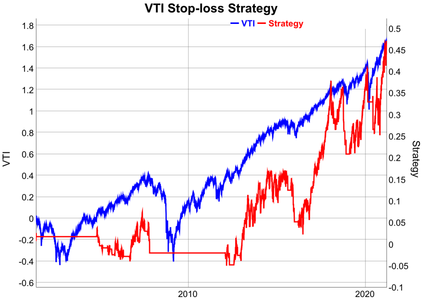
```
> # Combine data
> weal_th <- cbind(vt_i, pnl_s)
> col_names <- c("VTI", "Strategy")
> colnames(weal_th) <- col_names
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(weal_th,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI Turn of the Month strategy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

Stop-loss Rules

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

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

```
> # Calculate the VTI returns
> vt_i <- na.omit(rutils::etf_env$re_turns$VTI)
> date_s <- zoo::index(vt_i)
> vt_i <- drop(coredata(vt_i))
> n_rows <- NROW(vt_i)
> # Simulate stop-loss strategy
> sto_p <- 0.05
> ma_x <- 0.0
> cum_ret <- 0.0
> pnl_s <- vt_i
> for (i in 1:n_rows) {
+ # Calculate drawdown
+   cum_ret <- cum_ret + vt_i[i]
+   ma_x <- max(ma_x, cum_ret)
+   dd <- (cum_ret - ma_x)
+ # Check for stop-loss
+   if (dd < -sto_p*ma_x)
+     pnl_s[i+1] <- 0
+ }  # end for
> # Same but without using explicit loops
> cum_sum <- cumsum(vt_i)
> cum_max <- cummax(cumsum(vt_i))
> dd <- (cum_sum - cum_max)
> pnls2 <- vt_i
> is_dd <- rutils::lag_it(dd < -sto_p*cum_max)
> pnls2 <- ifelse(is_dd, 0, pnls2)
> all.equal(pnl_s, pnls2)
```

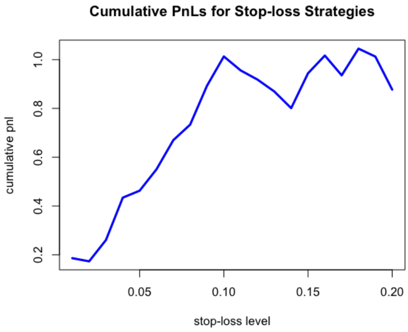



```
> # Combine data
> weal_th <- xts::xts(cbind(vt_i, pnl_s), date_s)
> col_names <- c("VTI", "Strategy")
> colnames(weal_th) <- col_names
> # Calculate Sharpe and Sortino ratios
> sqrt(252)*sapply(weal_th,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot VTI stop-loss strategy
> dygraphs::dygraph(cumsum(weal_th), main="VTI Stop-loss Strategy")
+   dyAxis("y", label=col_names[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=col_names[2], independentTicks=TRUE) %>%
+   dySeries(name=col_names[1], axis="y", strokeWidth=2, col="blue")
+   dySeries(name=col_names[2], axis="y2", strokeWidth=2, col="red")
```

Optimal Stop-loss Rules

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

```
> # Simulate multiple stop-loss strategies
> cum_sum <- cumsum(vt_i)
> cum_max <- cummax(cumsum(vt_i))
> dd <- (cum_sum - cum_max)
> cum_pnls <- sapply(0.01*(1:20), function(sto_p) {
+   pnl_s <- vt_i
+   is_dd <- rutils::lag_it(dd < -sto_p*cum_max)
+   pnl_s <- ifelse(is_dd, 0, pnl_s)
+   sum(pnl_s)
+ })  # end sapply
```




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

Homework Assignment

Required

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

Recommended

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

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