

A Statistical Arbitrage Strategy in R

Jacques Joubert

For those of you who have been following my blog posts for the last 6 months will know that I have taken part in the [Executive Program in Algorithmic Trading](http://www.quantinsti.com/courses/epat/) offered by QuantInsti.

It’s been a journey and this article serves as a report on my final project focusing on statistical arbitrage, coded in R. This article is a combination of my class notes and my source code.

I uploaded everything to [GitHub](https://github.com/Jackal08/QuantInsti-Final-Project-Statistical-Arbitrage) in order to welcome readers to contribute, improve, use, or work on this project. It will also form part of my [Open Source Hedge Fund project](http://www.quantsportal.com/home/collection/open-source-hedge-fund/) on my blog QuantsPortal

**History of Statistical Arbitrage:**

* First developed and used in the mid 1980s by Nunzio Tartaglia’s quantitative group at Morgan Stanly
* Pair Trading is a “contrarian strategy” designed to harness mean-reverting behavior of the pair ratio
* David Shaw, founder of D.E Shaw & Co, left Morgan Stanley and started his own “Quant” trading firm in the late 1980s dealing mainly in pair trading

**What is Pair Trading:**

Statistical arbitrage trading or pairs trading as it is commonly known is defined as trading one financial instrument or a basket of financial instruments – in most cases to create a value neutral basket.

It is the idea that a co-integrated pair is mean reverting in nature. There is a spread between the instruments and the further it deviates from its mean, the greater the probability of a reversal.

Note however that statistical arbitrage is not a risk free strategy. Say for example that you have entered positions for a pair and then the spread picks up a trend rather than mean reverting.

**The Concept:**

**Step 1**: Find 2 related securities

Find two securities that are in the same sector / industry, they should have similar market capitalization and average volume traded.

An example of this is Anglo Gold and Harmony Gold.

**Step 2**: Calculate the spread

In the code to follow I used the pair ratio to indicate the spread. It is simply the price of asset A / price asset B.

**Step 3**: Calculate the mean, standard deviation, and z-score of the pair ratio / spread.

**Step 4**: Test for co-integration

In the code to follow I use the Augmented Dicky Fuller Test (ADF Test) to test for co-integration. I set up three tests, each with a different number of observations (120, 90, 60), all three tests have to reject the null hypothesis that the pair is not co-integrated.

**Step 5**: Generate trading signals

Trading signals are based on the z-score, given they pass the test for co-integration. In my project I used a z-score of 1 as I noticed that other algorithms that I was competing with were using very low parameters. (I would have preferred a z-score of 2, as it better matches the literature, however it is less profitable)

**Step 6**: Process transactions based on signals

**Step 7**: Reporting

# R markdown for my project

## Import packages and set directory

The first step is always to import the packages needed.

#Imports  
require(tseries)  
require(urca) #Used for the ADF Test  
require(PerformanceAnalytics)

This strategy will be run on shares listed on the Johannesburg Stock Exchange (JSE). Because of this I won’t be using the quantmod package to pull data from Yahoo Finance, instead I have already gotten and cleaned the data that I stored in a SQL database and then moved to csv files on the Desktop.

I added all the pairs used in the strategy to a folder which I now set to be the working directory. (If you are connecting from GitHub then you will want to set the working directory to where you stored the file)

##Change this to match where you stored the csv files folder name FullList  
setwd("C:\\Users\\Administrator\\Documents\\GitHub\\QuantInsti-Final-Project-Statistical-Arbitrage\\database\\FullList")

## Functions that will be called from within other functions (No user interaction)

Next: I create all the functions that will be needed to run the strategy. The functions below will be called from within other functions so you dont need to worry about their arguments.

### AddColumns

The AddColumns function is used to add columns to the dataframe that will be needed to store variables.

#Add Columns to csvDataframe  
AddColumns <- function(csvData){  
 csvData$spread <- 0  
 csvData$adfTest <- 0  
 csvData$mean <- 0  
 csvData$stdev <- 0  
 csvData$zScore <- 0  
 csvData$signal <- 0  
 csvData$BuyPrice <- 0  
 csvData$SellPrice <- 0  
 csvData$LongReturn <- 0  
 csvData$ShortReturn <- 0  
 csvData$Slippage <- 0  
 csvData$TotalReturn <- 0  
 return(csvData)  
}

### PrepareData

The PrepareData function calculates the pair ratio and the log10 prices of the pair. It also calls the AddColumns funtion within it.

PrepareData <- function(csvData){  
 #Calculate the Pair Ratio  
 csvData$pairRatio <- csvData[,2] / csvData[,3]  
   
 #Calculate the log prices of the two time series  
 csvData$LogA <- log10(csvData[,2])  
 csvData$LogB <- log10(csvData[,3])  
   
 #Add columns to the DF  
 csvData <- AddColumns(csvData)  
   
 #Make sure that the date column is not read in as a vector of characters  
 csvData$Date <- as.Date(csvData$Date)  
   
 return(csvData)  
}

### GenerateRowValue

The GenerateRowValue function Calculates the mean, standard deviation and the z-score for a given row in the dataframe.

#Calculate mean, stdDev, and z-score for the given Row [end]  
GenerateRowValue <- function(begin, end, csvData){  
 average <- mean(csvData$spread[begin:end])  
 stdev <- sd(csvData$spread[begin:end])  
   
 csvData$mean[end] <- average  
 csvData$stdev[end] <- stdev  
 csvData$zScore[end] <- (csvData$spread[end]-average)/stdev  
   
 return(csvData)   
}

### GenerateSignal

The GenerateSignal function creates a long, short, or close signal based on the z-score. You can manually change the z-score. I have set it to 1 and -1 for entry signals and any z-score between 0.5 and -0.5 will create a close/exit signal.

#Generate trading signals based on a z-score of 1 and -1   
GenerateSignal <- function(counter, csvData){  
 #Trigger and close represent the entry and exit zones (value refers to the z-score value)  
 trigger <- 1  
 close <- 0.5  
   
 currentSignal <- csvData$signal[counter]  
 prevSignal <- csvData$signal[counter-1]  
   
 #Set trading signal for the given [counter] row  
 if(csvData$adfTest[counter] == 1)  
 {  
 #If there is a change in signal from long to short then you must allow for the current trade to first be closed  
 if(currentSignal == -1 && prevSignal == 1)  
 csvData$signal[counter] <- 0  
 else if(currentSignal == 1 && prevSignal == -1)  
 csvData$signal[counter] <- 0  
   
 #Create a long / short signal if the current z-score is larger / smaller than the trigger value (respectively)  
 else if(csvData$zScore[counter] > trigger)  
 csvData$signal[counter] <- -1  
 else if (csvData$zScore[counter] < -trigger)  
 csvData$signal[counter] <- 1  
   
 #Close the position if z-score is beteween the two "close" values  
 else if (csvData$zScore[counter] < close && csvData$zScore[counter] > -close)  
 csvData$signal[counter] <- 0  
 else   
 csvData$signal[counter] <- prevSignal  
 }  
 else   
 csvData$signal[counter] <- 0  
   
 return(csvData)  
}

### GenerateTransactions

The GenerateTransactions function is responsible for setting the entry and exit prices for the respective long and short positions needed to create a pair.

Note: QuantInsti taught us a very specific way of backtesting a trading strategy. They used excel to teach strategies and when I coded this strategy I used a large part of the excel methodology.

Going forward however I would explore other ways of storing variabels. One of the great things about this method is that you can pull the entire dataframe and analyze why a trade was made and all the details pertaining to it.

#Transactions based on trade signal  
#Following the framework set out initially by QuantInsti (Note: this can be coded better)   
GenerateTransactions <- function(currentSignal, prevSignal, end, csvData){  
 #In a pair trading strategy you need to go long one share and short the other and then reverse the transaction when you close  
   
 ##First Leg of the trade (Set Long position)  
 #If there is no change in signal  
 if(currentSignal == 0 && prevSignal == 0)  
 csvData$BuyPrice[end] <- 0   
 else if(currentSignal == prevSignal)  
 csvData$BuyPrice[end] <- csvData$BuyPrice[end-1]   
   
 #If the signals point to a new trade  
 #Short B and Long A  
 else if(currentSignal == 1 && currentSignal != prevSignal)  
 csvData$BuyPrice[end] <- csvData[end, 2]   
 #Short A and Long B  
 else if(currentSignal == -1 && currentSignal != prevSignal){  
 csvData$BuyPrice[end] <- csvData[end, 3] \* csvData$pairRatio[end]  
 transactionPairRatio <<- csvData$pairRatio[end]  
 }  
   
 #Close trade  
 else if(currentSignal == 0 && prevSignal == 1)  
 csvData$BuyPrice[end] <- csvData[end, 2]   
 else if(currentSignal == 0 && prevSignal == -1)  
 csvData$BuyPrice[end] <- csvData[end, 3] \* transactionPairRatio   
   
   
   
 ##Second Leg of the trade (Set Short position)  
 ##Set Short Prices if there is no change in signal  
 if(currentSignal == 0 && prevSignal == 0)  
 csvData$SellPrice[end] <- 0   
 else if(currentSignal == prevSignal)  
 csvData$SellPrice[end] <- csvData$SellPrice[end-1]   
   
 #If the signals point to a new trade  
 else if(currentSignal == 1 && currentSignal != prevSignal){  
 csvData$SellPrice[end] <- csvData[end, 3] \* csvData$pairRatio[end]  
 transactionPairRatio <<- csvData$pairRatio[end]  
 }  
 else if(currentSignal == -1 && currentSignal != prevSignal)  
 csvData$SellPrice[end] <- csvData[end, 2]   
   
 #Close trade  
 else if(currentSignal == 0 && prevSignal == 1){  
 csvData$SellPrice[end] <- csvData[end, 3] \* transactionPairRatio  
 }  
 else if(currentSignal == 0 && prevSignal == -1)  
 csvData$SellPrice[end] <- csvData[end, 2]   
   
 return(csvData)  
}

### GetReturns

GetReturns calculates the returns on each position after it has been closed and then calculates the total returns and adds slippage.

#Calculate the returns generated after each transaction  
#Add implementation shortfall / slippage  
GetReturns <- function(end, csvData, slippage){  
 #Calculate the returns generated on each leg of the deal (the long and the short position)  
 #Long leg of the trade  
 if(csvData$signal[end] == 0 && csvData$signal[end-1] != 0 )  
 csvData$LongReturn[end] <- (csvData$BuyPrice[end] / csvData$BuyPrice[end-1]) - 1  
 #Short Leg of the trade  
 if(csvData$signal[end] == 0 && csvData$signal[end-1] != 0 )  
 csvData$ShortReturn[end] <- (csvData$SellPrice[end-1] / csvData$SellPrice[end]) - 1  
   
 #Add slippage  
 if(csvData$ShortReturn[end] != 0)  
 csvData$Slippage[end] <- slippage  
   
 #If a trade was closed then calculate the total return  
 if(csvData$ShortReturn[end] != 0 && csvData$LongReturn[end] != 0)  
 csvData$TotalReturn[end] <- ((csvData$ShortReturn[end] + csvData$LongReturn[end]) / 2) + csvData$Slippage[end]  
   
 return(csvData)  
}

### GenerateReports

The next two arguments are used to generate reports. A report includes the following: Charting:

1. An Equity curve
2. Drawdown curve
3. Daily returns bar chart

Statistics:

1. Annual Retruns
2. Annualized Sharpe Ratio
3. Maximum Drawdown
4. Total trades
5. Success ratio
6. PnL ratio

Table:

1. Top 5 Drawdowns and their duration

Note: If you have some extra time then you can further break this function down into smaller functions in order to reduce the lines of code and improve userbility. Less code = Less Bugs

#Returns an equity curve, annualized return, annualized sharpe ratio, and max drawdown  
GenerateReport <- function(pairData, startDate, endDate){  
 #Subset the dates   
 returns <- xts(pairData$TotalReturn, as.Date(pairData$Date))  
 returns <- returns[paste(startDate,endDate,sep="::")]  
   
 #Plot  
 charts.PerformanceSummary(returns)  
   
 #Metrics  
 print(paste("Annual Returns: ",Return.annualized(returns)))  
 print(paste("Annualized Sharpe: " ,SharpeRatio.annualized(returns)))  
 print(paste("Max Drawdown: ",maxDrawdown(returns)))  
   
 #var returns = xts object  
 totalTrades <- 0  
 positiveTrades <- 0  
 profitsVector <- c()  
 lossesVector <- c()  
   
 #loop through the data to find the + & - trades and total trades  
 for(i in returns){  
 if(i != 0){  
 totalTrades <- totalTrades + 1  
 if(i > 0){  
 positiveTrades <- positiveTrades + 1  
 profitsVector <- c(profitsVector, i)  
 }  
 else if (i < 0){  
 lossesVector <- c(lossesVector, i)  
 }  
 }  
 }  
   
 #Print the results to the console  
 print(paste("Total Trades: ", totalTrades))  
 print(paste("Success Rate: ", positiveTrades/totalTrades))  
 print(paste("PnL Ratio: ", mean(profitsVector)/mean(lossesVector\*-1)))  
 print(table.Drawdowns(returns))  
   
}  
  
#Use this one if you have the returns in xts format and want to generate a report  
GenerateReport.xts <- function(returns, startDate = '2005-01-01', endDate = '2015-11-23'){  
 returns <- returns[paste(startDate,endDate,sep="::")]  
   
 #Plot  
 charts.PerformanceSummary(returns)  
   
 #Metrics  
 print(paste("Annual Returns: ",Return.annualized(returns)))  
 print(paste("Annualized Sharpe: " ,SharpeRatio.annualized(returns)))  
 print(paste("Max Drawdown: ",maxDrawdown(returns)))  
   
 #var returns = xts object  
 totalTrades <- 0  
 positiveTrades <- 0  
 profitsVector <- c()  
 lossesVector <- c()  
   
 #Itterate through data to get the + & - trades  
 for(i in returns){  
 if(i != 0){  
 totalTrades <- totalTrades + 1  
 if(i > 0){  
 positiveTrades <- positiveTrades + 1  
 profitsVector <- c(profitsVector, i)  
 }  
 else if (i < 0){  
 lossesVector <- c(lossesVector, i)  
 }  
 }  
 }  
   
 #Print results to Console  
 print(paste("Total Trades: ", totalTrades))  
 print(paste("Success Rate: ", positiveTrades/totalTrades))  
 print(paste("PnL Ratio: ", mean(profitsVector)/mean(lossesVector\*-1)))  
 print(table.Drawdowns(returns))  
   
}

## Functions that the user will pass parameters to

The next two functions are the only functions that the user should fiddle with.

### BacktestPair

BacktestPair is used when you want to run a backtest on a trading pair (the pair is passed in via the csv file)

Function’s arguments:

* pairData = the csv file date
* mean = the number of observations used to calculate the mean of the spread.
* slippage = the amount of basis points that act as brokerage as well as slippage
* adfTest = a boolean value - if the backtest should test for co-integration
* criticalValue = Critical Value used in the ADF Test to test for co-integration
* generateReport = a boolean value - if a report must be generated

#The function that will be called by the user to backtest a pair  
BacktestPair <- function(pairData, mean = 35, slippage = -0.0020, adfTest = TRUE, criticalValue = -2.58, startDate = '2005-01-01', endDate = '2014-11-23', generateReport = TRUE){  
 # At 150 data points  
 # Critical value at 1% : -3.46  
 # Critical value at 5% : -2.88  
 # Critical value at 10% : -2.57  
   
 #Prepare the initial dataframe by adding columns and pre calculations  
 pairData <- PrepareData(pairData)  
   
 #Itterate through each day in the time series  
 for(i in 1:length(pairData[,2])){  
   
 #For each day after the amount of days needed to run the ADF test  
 if(i > 130){  
 begin <- i - mean + 1  
 end <- i  
   
 #Calculate Spread  
 spread <- pairData$pairRatio[end]  
 pairData$spread[end] <- spread  
   
 #ADF Test for co-integration  
 #120 - 90 - 60   
 if(adfTest == FALSE){  
 pairData$adfTest[end] <- 1   
 }  
 else {  
 if(adf.test(pairData$spread[(i-120):end], k = 1)[1] <= criticalValue){  
 if(adf.test(pairData$spread[(i-90):end], k = 1)[1] <= criticalValue){  
 if(adf.test(pairData$spread[(i-60):end], k = 1)[1] <= criticalValue){  
 #If co-integrated then set the ADFTest value to true / 1  
 pairData$adfTest[end] <- 1   
 }  
 }  
 }  
 }  
   
 #Calculate the remainder variables needed  
 if(i >= mean){  
 #Generate Row values  
 pairData <- GenerateRowValue(begin, end, pairData)  
   
 #Generate the Signals  
 pairData <- GenerateSignal(i, pairData)  
   
 currentSignal <- pairData$signal[i]  
 prevSignal <- pairData$signal[i-1]  
   
 #Generate Transactions  
 pairData <- GenerateTransactions(currentSignal, prevSignal, i, pairData)  
   
 #Get the returns with added slippage  
 pairData <- GetReturns(i, pairData, slippage)  
 }  
 }  
 }  
   
 if(generateReport == TRUE)  
 GenerateReport(pairData, startDate, endDate)  
   
 return(pairData)  
}

### BacktestPortfolio

BacktestPortfolio accepts a vector of csv files and then generates an equaly weighted portfolio.

Function’s arguments:

* names = an atomic vector of csv file names, example: c('DsyLib.csv', 'OldSanlam.csv')
* mean = the number of observations used to calculate the mean of the spread.
* leverage = how much leverage you want to apply to the portfolio

#An equally weighted portfolio of shares  
BacktestPortfolio <- function(names, mean = 35,leverage = 1, startDate = '2005-01-01', endDate = '2015-11-23'){  
 ##Iterates through all the pairs and backtests each one  
 ##stores the data in a list of numerical vectors  
 returns.list <- list()  
 counter <- F  
 ticker <- 1  
 for (name in names){  
 #A notification to let you know how far it is (Uncomment this if you want notifications  
 #print(paste(ticker, " of ", length(names)))  
 #ticker <- ticker + 1  
   
 #Run the backtest on the pair  
 data <- read.csv(name)   
 BackTest.df <- BacktestPair(data, mean, generateReport = FALSE)  
   
 #Store the dates in a seperate vector  
 if (counter == F){  
 dates <<- as.Date(BackTest.df$Date)  
 counter <- T  
 }  
   
 #Append to list  
 returns.list <- c(returns.list, list(BackTest.df[,18]))  
 }  
   
 ##Aggregates the returns for each day and then calculates the average for each day  
 total.returns <- c()  
 for (i in 1:length(returns.list)){  
 if(i == 1)  
 total.returns = returns.list[[i]]  
 else  
 total.returns = total.returns + returns.list[[i]]  
 }  
   
 total.returns <- total.returns / length(returns.list)  
   
 ##Generate a report for the portfolio  
 returns <- xts(total.returns \* leverage, dates)  
 GenerateReport.xts(returns, startDate, endDate)  
   
 return(returns)  
}

# Running Backtests

Now we can start testing strategies using our code.

## Pure arbitrage on the JSE

When starting this project the main focus was on using statistical arbitrage to find pairs that were co-integrated and then to trade those, however I very quickly realised that the same code could be used to trade shares that had both its primary listing as well as access to its secondary listing on the same exchange.

If both listings are found on the same exchange, it opens the door for a pure arbitrage strategy due to both listings refering to the same asset. Therefore you dont need to test for co-integration.

There are two very obvious examples on the JSE.

### First Example Investec:

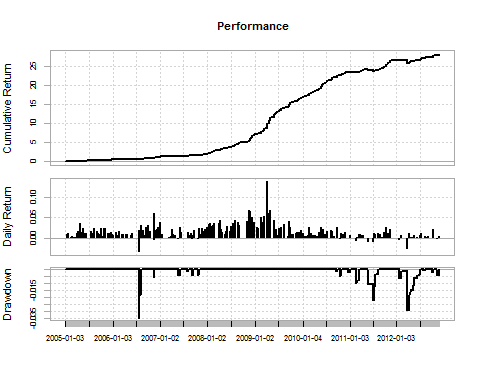
Primary = Investec Ltd : Secondary = Investec PLC

#### Investec In-Sample Test (2005-01-01 - 2012-11-23)

Test the following parameters

* The Investec ltd / plc pair
* mean = 35
* Set adfTest = F (Dont test for co-integration)
* Leverage of x3

#Investec  
leverage <- 3  
data <- read.csv('investec.csv')   
investec <- BacktestPair(data, 35, generateReport = F, adfTest = F)   
  
#Format to an xts object and pass to GenerateReport.xts()  
investec.returns <- xts(investec[,18] \* leverage, investec$Date)  
GenerateReport.xts(investec.returns, startDate = '2005-01-01', endDate = '2012-11-23')



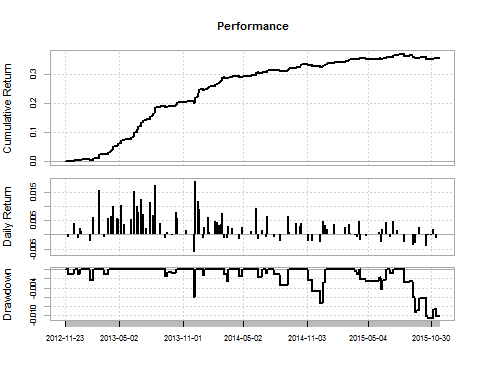
## [1] "Annual Returns: 0.538"  
## [1] "Annualized Sharpe: 4.55"  
## [1] "Max Drawdown: 0.035"  
## [1] "Total Trades: 233"  
## [1] "Success Rate: 0.91"  
## [1] "PnL Ratio: 2.47"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2006-07-21 2006-07-21 2006-08-04 -0.0351 11 1 10  
## 2 2012-01-20 2012-03-22 2012-06-28 -0.0290 110 44 66  
## 3 2011-05-17 2011-07-01 2011-08-04 -0.0221 56 32 24  
## 4 2011-02-11 2011-02-11 2011-03-10 -0.0096 20 1 19  
## 5 2006-11-14 2006-11-14 2006-11-16 -0.0055 3 1 2

#### Investec Out-of-Sample Test (2012-11-23 - 2015-11-23)

Note: if you increase the slippage, you will very quickly kiss profits good bye.

GenerateReport.xts(investec.returns, startDate = '2012-11-23', endDate = '2015-11-23')



## [1] "Annual Returns: 0.107"  
## [1] "Annualized Sharpe: 3.34"  
## [1] "Max Drawdown: 0.01"  
## [1] "Total Trades: 102"  
## [1] "Success Rate: 0.69"  
## [1] "PnL Ratio: 2.91"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2015-08-14 2015-10-30 <NA> -0.0107 72 55 NA  
## 2 2014-11-04 2014-12-09 2014-12-24 -0.0073 36 26 10  
## 3 2013-12-03 2013-12-03 2013-12-05 -0.0061 3 1 2  
## 4 2015-04-08 2015-06-08 2015-06-23 -0.0042 52 42 10  
## 5 2014-07-30 2014-08-15 2014-09-08 -0.0034 29 13 16

### Second Example Mondi:

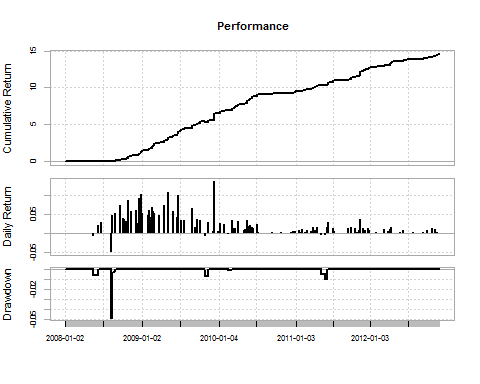
Primary = Mondi Ltd : Secondary = Mondi PLC

#### Mondi In-Sample Test (2008-01-01 - 2012-11-23)

Test the following parameters

* The Mondi ltd / plc pair
* mean = 35
* Set adfTest = F (Dont test for co-integration)
* Leverage of x3

data <- read.csv('mondi.csv')   
mondi <- BacktestPair(data, 35, generateReport = F, adfTest = F)  
  
mondi.returns <- xts(mondi[,18] \* leverage, mondi$Date)  
GenerateReport.xts(mondi.returns, startDate = '2008-01-01', endDate = '2012-11-23')



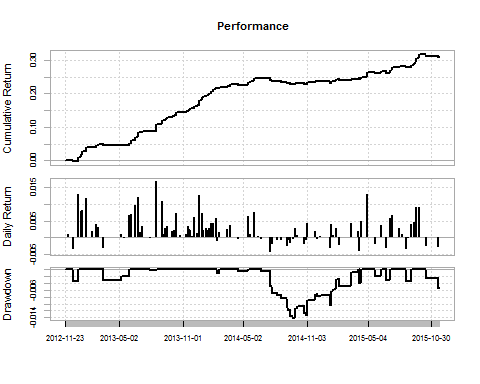
## [1] "Annual Returns: 0.75"  
## [1] "Annualized Sharpe: 4.21"  
## [1] "Max Drawdown: 0.049"  
## [1] "Total Trades: 115"  
## [1] "Success Rate: 0.93"  
## [1] "PnL Ratio: 2.59"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2008-08-07 2008-08-07 2008-08-22 -0.0492 12 1 11  
## 2 2011-05-04 2011-05-24 2011-06-03 -0.0093 22 14 8  
## 3 2009-10-27 2009-10-27 2009-11-11 -0.0068 12 1 11  
## 4 2008-05-15 2008-05-15 2008-06-06 -0.0062 17 1 16  
## 5 2010-02-15 2010-02-15 2010-03-01 -0.0008 11 1 10

#### Mondi Out-of-Sample Test (2012-11-23 - 2015-11-23)

Note: In all of my testing I found that the further down the timeline my data was, the harder it was to make profits on end of day data. I tested this same strategy on intraday data and it has a higher return profile.

GenerateReport.xts(mondi.returns, startDate = '2012-11-23', endDate = '2015-11-23')



## [1] "Annual Returns: 0.094"  
## [1] "Annualized Sharpe: 3.31"  
## [1] "Max Drawdown: 0.014"  
## [1] "Total Trades: 100"  
## [1] "Success Rate: 0.73"  
## [1] "PnL Ratio: 2.41"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2014-06-25 2014-09-22 2015-03-30 -0.0142 194 64 130  
## 2 2015-10-15 2015-11-19 <NA> -0.0053 29 26 NA  
## 3 2015-04-02 2015-04-02 2015-04-09 -0.0040 4 1 3  
## 4 2012-12-12 2012-12-12 2013-01-02 -0.0035 12 1 11  
## 5 2015-08-19 2015-08-26 2015-09-03 -0.0034 12 6 6

## 

## Statistical Arbitrage on the JSE

Next we will look at a pair trading strategy.

Typically a pair consits of 2 shares that:

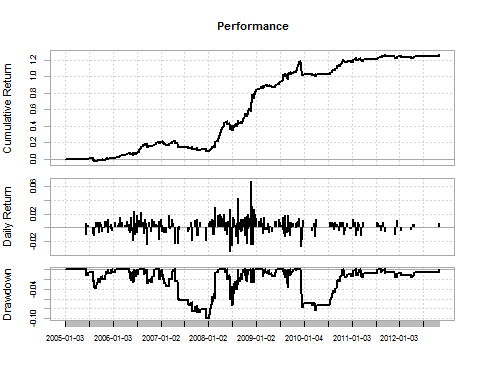
* Share a market sector
* Have a similar market cap
* Similar business model and clients
* Are co-integrated

In all of the portfolios below I use 3x leverage

## Contruction Portfolio

### In-sample test (2005-01-01 - 2012-11-01)

names <- c('groupmr.csv', 'groupppc.csv', 'groupavenge.csv', 'groupwhbo.csv', 'mrppc.csv', 'mravenge.csv')  
  
ReturnSeries <- BacktestPortfolio(names, startDate = '2005-01-01', endDate = '2012-11-01', leverage = 3)

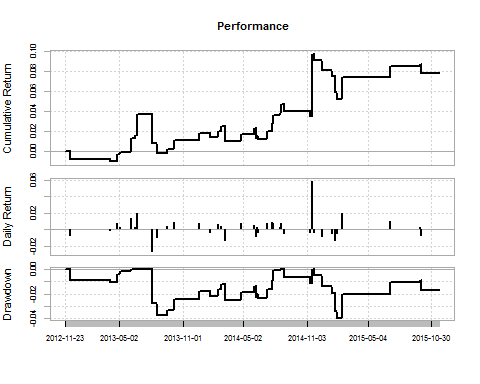


## [1] "Annual Returns: 0.11"  
## [1] "Annualized Sharpe: 1.49"  
## [1] "Max Drawdown: 0.10"  
## [1] "Total Trades: 316"  
## [1] "Success Rate: 0.64"  
## [1] "PnL Ratio: 0.98"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2007-04-23 2007-12-12 2008-03-11 -0.1005 224 164 60  
## 2 2009-12-02 2010-03-19 2010-10-22 -0.0837 223 75 148  
## 3 2008-05-23 2008-07-02 2008-08-11 -0.0729 56 28 28  
## 4 2005-06-14 2005-08-22 2005-11-22 -0.0383 114 48 66  
## 5 2009-08-13 2009-08-27 2009-09-02 -0.0357 15 11 4

### Out-of-sample test (2012-11-23 - 2015-11-23)

GenerateReport.xts(ReturnSeries, startDate = '2012-11-23', endDate = '2015-11-23')



## [1] "Annual Returns: 0.025"  
## [1] "Annualized Sharpe: 0.52"  
## [1] "Max Drawdown: 0.04"  
## [1] "Total Trades: 41"  
## [1] "Success Rate: 0.609"  
## [1] "PnL Ratio: 0.994"

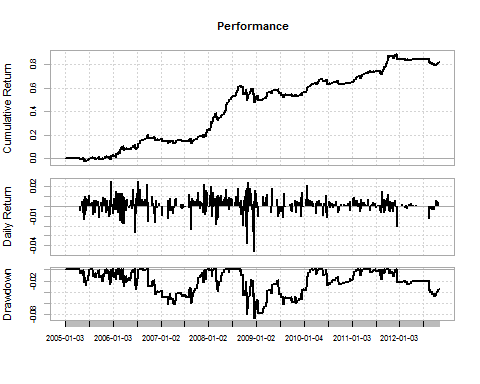
Drawdowns Table

## From Trough To Depth Length To Trough Recovery  
## 1 2014-11-21 2015-01-29 <NA> -0.0403 252 46 NA  
## 2 2013-08-05 2013-08-19 2014-08-15 -0.0378 257 10 247  
## 3 2014-08-28 2014-11-11 2014-11-17 -0.0109 57 53 4  
## 4 2012-12-05 2013-04-05 2013-06-04 -0.0104 122 81 41

## Insurance Portfolio

### In-sample test (2005-01-01 - 2012-11-01)

names <- c('disclib.csv', 'discmmi.csv', 'discsanlam.csv', 'libmmi.csv', 'mmiold.csv', 'mmisanlam.csv', 'oldsanlam.csv')  
  
ReturnSeries <- BacktestPortfolio(names, startDate = '2005-01-01', endDate = '2012-11-01', leverage = 3)



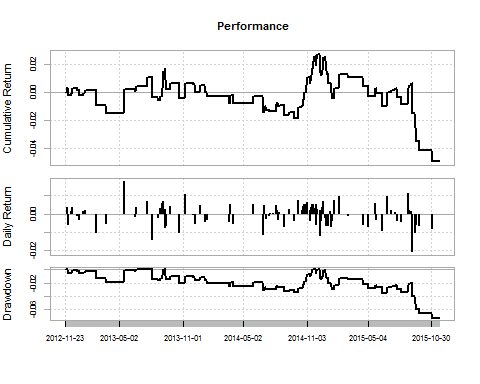
## [1] "Annual Returns: 0.08"  
## [1] "Annualized Sharpe: 1.25"  
## [1] "Max Drawdown: 0.087"  
## [1] "Total Trades: 429"  
## [1] "Success Rate: 0.63"  
## [1] "PnL Ratio: 0.88"

Drawdowns Table

## From Trough To Depth Length To Trough Recovery  
## 1 2008-09-03 2008-12-11 2010-02-11 -0.0873 361 71 290  
## 2 2006-09-28 2007-04-16 2007-12-03 -0.0619 297 136 161  
## 3 2011-12-05 2012-09-27 <NA> -0.0477 229 203 NA  
## 4 2008-02-26 2008-03-11 2008-05-05 -0.0384 45 11 34  
## 5 2006-06-23 2006-06-26 2006-07-13 -0.0355 15 2 13

### Out-of-sample test (2012-11-23 - 2015-11-23)

GenerateReport.xts(ReturnSeries, startDate = '2012-11-23', endDate = '2015-11-23')



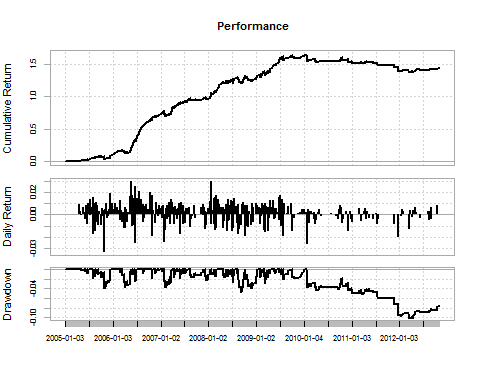
## [1] "Annual Returns: -0.016"  
## [1] "Annualized Sharpe: -0.46"  
## [1] "Max Drawdown: 0.074"  
## [1] "Total Trades: 99"  
## [1] "Success Rate: 0.52"  
## [1] "PnL Ratio: 0.74"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2014-12-08 2015-11-02 <NA> -0.0741 241 225 NA  
## 2 2013-09-11 2014-09-26 2014-11-19 -0.0349 298 260 38  
## 3 2012-11-28 2013-03-20 2013-06-19 -0.0186 137 77 60  
## 4 2013-08-05 2013-08-21 2013-09-05 -0.0166 23 12 11  
## 5 2014-11-26 2014-11-26 2014-12-01 -0.0061 4 1 3

## General Retail Portfolio

### In-sample test (2005-01-01 - 2012-11-01)

names <- c('wooltru.csv', 'woolmr.csv', 'wooltfg.csv', 'trumr.csv', 'trutfg.csv', 'MRTFG.csv')  
  
ReturnSeries <- BacktestPortfolio(names, startDate = '2005-01-01', endDate = '2012-11-01', leverage = 3)

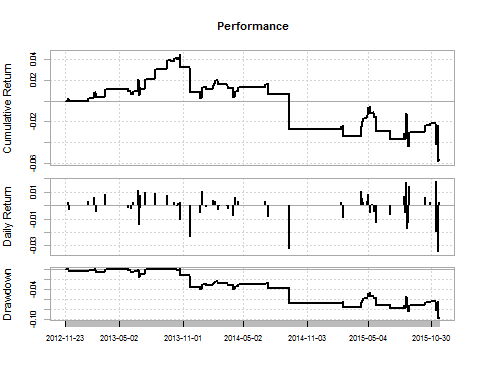


## [1] "Annual Returns: 0.12"  
## [1] "Annualized Sharpe: 1.93"  
## [1] "Max Drawdown: 0.10"  
## [1] "Total Trades: 411"  
## [1] "Success Rate: 0.67"  
## [1] "PnL Ratio: 0.91"

Drawdown Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2010-01-20 2012-03-13 <NA> -0.1024 694 533 NA  
## 2 2008-08-13 2008-09-05 2008-10-02 -0.0467 36 18 18  
## 3 2008-10-08 2008-11-18 2009-01-21 -0.0401 72 30 42  
## 4 2005-09-07 2005-10-25 2005-12-07 -0.0391 66 35 31  
## 5 2007-01-25 2007-02-05 2007-03-26 -0.0377 42 8 34

### Out-of-sample test (2012-11-23 - 2015-11-23)

GenerateReport.xts(ReturnSeries, startDate = '2012-11-23', endDate = '2015-11-23')



## [1] "Annual Returns: -0.019"  
## [1] "Annualized Sharpe: -0.40"  
## [1] "Max Drawdown: 0.097"  
## [1] "Total Trades: 66"  
## [1] "Success Rate: 0.54"  
## [1] "PnL Ratio: 0.66"

Drawdowns Table  
## From Trough To Depth Length To Trough Recovery  
## 1 2013-10-24 2015-11-19 <NA> -0.0979 521 518 NA  
## 2 2013-06-27 2013-06-27 2013-07-15 -0.0147 13 1 12  
## 3 2013-02-20 2013-02-20 2013-03-18 -0.0053 19 1 18  
## 4 2013-05-28 2013-06-04 2013-06-25 -0.0047 20 6 14  
## 5 2012-11-30 2012-11-30 2013-02-14 -0.0033 51 1 50

## Conclusion:

At the end of all my testing, and trust me – there is a lot more testing I did than what this report, I came to the conclusion that the Pure Arbitrage Strategy has great hope in being used as a retail strategy using real money, but the Pair Trading Strategy is strained and not likely to be used in production in its current form.

### More on the Pure Arbitrage Trading Strategy:

I have only found two shares that have duel listings on the same exchange; this means that we can’t allocate large sums of money to the strategy as it will have a high market impact. However we could use multiple exchanges and increase the number of shares used.

### More on the Pair Trading Strategy:

1. The number of observations used in the ADF Tests are largely to blame. The problems is that a test for co-integration has to be done in order to make a claim for statistical arbitrage, however by using 120, 90, and 60 as parameters to the three tests, it is very difficult to find pairs that match the criteria and that will continue in this form for the near future. (Kalman filtering may be useful here)
2. I haven’t spent a lot of time changing the different parameters like the number of observations in the mean calculation. (This requires further exploration)
3. From the above sector portfolios we can see that the early years are very profitable but the further down the timeline we go, the lower returns get.
4. The same is noticed with other portfolios that I didn’t include in this report but is in the R Code file.
5. I believe that this is due to large institutions using the same bread and butter strategy. You will note (if you spend enough time testing all the strategies) that in 2009 there seems to be a sudden shift in the data to lower returns.
6. I feel that the end of day data I am using is limiting me and if I were to test the strategy on intraday data then profits would be higher. (I ran one test on intraday data on Mondi and the results were much higher, but I am still to test it on sector portfolios)
7. This is one of the simpler statistical arbitrage strategies and I believe that if we were to improve on the way we calculate the spread and change some of the entry and exit rules, the strategy would become a more profitable.

If you made it to the end of this article, I thank you and hope that it added some value. This is the first time that I am using Github, so I am looking forward to see if there are any new contributors to the project.

Github repository: <https://github.com/Jackal08/QuantInsti-Final-Project-Statistical-Arbitrage>