Opportunity Trading of Short Run Reversals - Evidence from the JSE

Marc Labe(1445149) and Shane Serebro(1430275)

Introduction

Firms that experience large negative price shocks are often deemed to be bad investments in the short run owing to investor sentiment and increased volatility in the period after such price shock. However, this may not necessarily be the case as seen in Bremer and Sweeney (1991). Moreover, shares that experience positive price shocks are often considered good investments as many believe this trend will continue for the foreseeable future. In this study, an assessment is carried out to determine whether these firms experience share price reversals and therefore can be traded opportunistically and profitably.

Furthermore, this study will add to the existing literature with regards to whether the EMH holds in the Johannesburg Stock Exchange (JSE) or does the overreaction hypothesis postulated by De Bondt and Thaler (1985) serve as a better explanation of share price movements. If the EMH does indeed hold true on the JSE, abnormal profits cannot be obtained from trading on significant price movements whereas these profits are obtainable if evidence of the overreaction hypothesis can be found on the JSE. In essence, this paper sets out to reveal whether shares that experience a sudden and significant change in price undergo a reversal in the short-term. Furthermore, it aims to determine whether investing in a company that experiences a large share price movement is a profitable investment strategy for the active trader.

Methodology

Step 1: Importing Packages

First we import the libraries. The ‘googledrive’ package allows us to fetch documents directly from Google Drive. This is where we have stored our data. The second package ‘readxl’ allows us to firstly import and secondly read Microsoft Excel files. This is important as our original data came from an excel spreadsheet. Finally, the ‘qrmtools’ package provides a range of quantitative finance functions that will be used later on.

library(googledrive)  
library(readxl)  
library(qrmtools)

Step 2: Data Claning and Manipulation

As a part of the Google Drive package, drive\_download finds and downloads a file on Google Drive based on an unique identifier (id) into the working directory. If the file already exists, this cfunction is ignored. Using read\_excel from the readxl package, the data is imported into the PriceData dataframe.

id <- "1SGigXMnzubpP15Y1W18GqnlVQFiv95jN"  
#try(drive\_download(as\_id(id), overwrite = FALSE), silent = TRUE)  
PriceData <- read\_excel("Price-Volume-MarketCap.xlsx",   
 sheet = "Price (D)")   
  
knitr::kable(PriceData[1:10, 1:8])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Date | ABG SJ Equity | ABI SJ Equity | ABL SJ Equity | ABT SJ Equity | ACL SJ Equity | ACP SJ Equity | ACT SJ Equity |
| 2000-01-04 | 2785 | 5240 | NA | NA | 401.63 | NA | 19.25 |
| 2000-01-05 | 2720 | 5100 | NA | NA | 396.35 | NA | 19.25 |
| 2000-01-06 | 2695 | 5100 | NA | NA | 410.44 | NA | 19.25 |
| 2000-01-07 | 2790 | 5070 | NA | NA | 475.62 | NA | 19.25 |
| 2000-01-10 | 2950 | 5150 | NA | NA | 496.76 | NA | 19.25 |
| 2000-01-11 | 3115 | 5150 | NA | NA | 453.60 | NA | 19.25 |
| 2000-01-12 | 3125 | 5120 | NA | NA | 458.00 | NA | 18.33 |
| 2000-01-13 | 3050 | 5180 | NA | NA | 450.95 | NA | 18.33 |
| 2000-01-14 | 3080 | 5170 | NA | NA | 458.00 | NA | 17.42 |
| 2000-01-17 | 3150 | 5180 | NA | NA | 472.97 | NA | 16.50 |

We now declare 2 functions that deals with basic data cleaning. NaFunction creates a data frame that consists of booleans of whether each variable in the PriceData dataframe contains only NA or 0 with false meaning that the column contains only NA or 0 and vice versa.

colClean1’s purpose is to remove the “SJ Equity” from the share names in the dataframe.

NaFunction <- function(column){  
 !all(is.na(column) | column == 0)  
}  
  
  
colClean1 <- function(x){  
 gsub(" SJ Equity", "", names(x), fixed = TRUE)  
}

exclP stores the result of running the NaFunction on PriceData. Since the NaFunction computes columns containing only NA or 0 to false, we subset PriceData where exclP is true.

To ensure correct interpretation, we coerce the date column to the date data type.

Finally, we run colClean1 to remove “SJ Equity” from all the variable names.

exclP <- sapply(PriceData, NaFunction)  
PriceData <- PriceData[exclP]  
PriceData$Date <- as.Date(PriceData$Date)  
names(PriceData) <- colClean1(PriceData)

Returns

Returns are calculated for each share.

First we extract the share prices only(no dates) in order to convert the PriceData dataframe to a numeric matrix.

The returns are the then calculated using the “returns” function of the qrmtools package and are then returned as a dataframe. As the returns function ignores the first row, it is inserted manually using a combination of rbind and seq.

PriceDataNew <- PriceData[, -1]  
  
PriceDataNew <- sapply(PriceDataNew, function(x){  
 as.numeric(x)  
})

## Warning in FUN(X[[i]], ...): NAs introduced by coercion

returnsDf <- as.data.frame(returns(PriceDataNew, method = "simple"))  
returnsDf <- rbind(seq(from = 0, to = 0, length.out = 448), returnsDf)  
  
knitr::kable(returnsDf[(1:10), (1:8)])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ABG | ABI | ABT | ACL | ACP | ACT | ADH | ADI |
| 0.0000000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| -0.0233393 | -0.0267176 | NA | -0.0131464 | NA | 0.0000000 | -0.0790041 | 0 |
| -0.0091912 | 0.0000000 | NA | 0.0355494 | NA | 0.0000000 | 0.0000000 | 0 |
| 0.0352505 | -0.0058824 | NA | 0.1588052 | NA | 0.0000000 | 0.0000000 | 0 |
| 0.0573477 | 0.0157791 | NA | 0.0444472 | NA | 0.0000000 | 0.0952474 | 0 |
| 0.0559322 | 0.0000000 | NA | -0.0868830 | NA | 0.0000000 | 0.0522142 | 0 |
| 0.0032103 | -0.0058252 | NA | 0.0097002 | NA | -0.0477922 | -0.0082988 | 0 |
| -0.0240000 | 0.0117188 | NA | -0.0153930 | NA | 0.0000000 | -0.0749722 | 0 |
| 0.0098361 | -0.0019305 | NA | 0.0156337 | NA | -0.0496454 | 0.0360011 | 0 |
| 0.0227273 | 0.0019342 | NA | 0.0326856 | NA | -0.0528129 | 0.0434821 | 0 |

Step 3: Calculating Maximum Drawdowns and Minimum Drawups

Settings

This block sets parameters for the following functions

lookback <- 5   
window <- 5  
triggerDD <- -0.15  
triggerDU <- 0.15

Function Declarations

We now declare the functions that will calculate the maximum drawdown and minimum drawups:

1. Maximum drawdowns are calculated according to the following formula:
2. with
3. = the share price at time t and
4. = the lookback period (defined earlier as 5 days)

We create a for loop which calculates the drawdowns according to the formula, with the result being written to a new vector that will be a component of the new data frame. However, if the calculations result in a NA, it is replaced with a 0 in the new vector

maxDD <- function(column, lb){  
 dd <- vector(mode = "double", length = (length(column) - lb))  
 for (i in (lb+1):length(column)){  
 if(is.na(max(column[(i-lb):i]))){  
 dd[i] <- 0  
 } else{  
 dd[i] <- (column[i] - max(column[(i-lb):i], na.rm = T))/max(column[(i-lb):i], na.rm = T)  
 }  
 }  
 return(dd)  
}

1. Similarly with minimum drawdowns, an identical function is created, however with the following formula:

minDU <- function(column, lb){  
 du <- vector(mode = "double", length = (length(column) - lb))  
 for (i in (lb+1):length(column)){  
 if(is.na(min(column[(i-lb):i]))){  
 du[i] <- 0  
 } else{  
 du[i] <- (column[i] - min(column[(i-lb):i], na.rm = T))/min(column[(i-lb):i], na.rm = T)  
 }  
 }  
 return(du)  
}

Running the function (with lb set to lookback as defined above), we created a new dataframe that stores the maximum drawdowns (DDdf) and the minimum drawups (DUdf).

DDdf <- as.data.frame(sapply(PriceData[-1], maxDD, lb = lookback))  
DUdf <- as.data.frame(sapply(PriceData[-1], minDU, lb = lookback))  
  
knitr::kable(DDdf[(1:10), (1:8)])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ABG | ABI | ABT | ACL | ACP | ACT | ADH | ADI |
| 0.0000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | 0.0000000 | 0 | 0.0000000 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | -0.0171756 | 0 | -0.0868830 | 0 | 0.0000000 | 0.0000000 | 0 |
| 0.0000 | -0.0058252 | 0 | -0.0780256 | 0 | -0.0477922 | -0.0082988 | 0 |
| -0.0240 | 0.0000000 | 0 | -0.0922176 | 0 | -0.0477922 | -0.0826488 | 0 |
| -0.0144 | -0.0019305 | 0 | -0.0780256 | 0 | -0.0950649 | -0.0496232 | 0 |
| 0.0000 | 0.0000000 | 0 | -0.0478903 | 0 | -0.1428571 | -0.0082988 | 0 |

Cleaning and formatting

DDdf <- cbind(PriceData$Date, DDdf)  
names(DDdf)[names(DDdf) == "PriceData$Date"] <- "Date"  
DUdf <- cbind(PriceData$Date, DUdf)  
names(DUdf)[names(DUdf) == "PriceData$Date"] <- "Date"

Trigger Indices

The following 2 functions create one list of lists respectively that captures when the drawdowns and drawups exceed the trigger value defined above.

The function consists of a while loop that returns the index (i.e. row number) of all the values within the column that exceed the trigger (-0.15 for drawdowns and 0.15 for drawups). An if statement if used to allow the while loop to continue ignoring any NA values

trigIndexDD <- lapply(DDdf[-1], function(i){  
 trigDD <- vector(mode = "integer", length = 1)  
 l = length(i)  
 s = 0  
 pos = 1  
 while(s <= l){  
 x = which(i[(s+1):l] <= triggerDD)[1]+s  
 if(is.na(x)){  
 break  
 } else{  
 trigDD[pos] = x  
 }  
 s = trigDD[pos] + window  
 pos = pos + 1  
 }  
 return(trigDD)  
})  
  
trigIndexDU <- lapply(DUdf[-1], function(i){  
 trigDU <- vector(mode = "integer", length = 1)  
 l = length(i)  
 s = 0  
 pos = 1  
 while(s <= l){  
 x = which(i[(s+1):l] >= triggerDU)[1]+s  
 if(is.na(x)){  
 break  
 } else{  
 trigDU[pos] = x  
 }  
 s = trigDU[pos] + window  
 pos = pos + 1  
 }  
 return(trigDU)  
})  
  
  
print(trigIndexDD[1:5])

## $ABG  
## [1] 19 508 1607 1618 2202 3987  
##   
## $ABI  
## [1] 0  
##   
## $ABT  
## [1] 0  
##   
## $ACL  
## [1] 40 49 74 206 221 512 534 645 685 736 2188 2194 2200 2213  
## [15] 2219 2257 2325 2366 2542 3729 3811 3836 3873 3890 3903 3914 3922 3946  
## [29] 3952 3963 3969 3987 3993 4009 4027 4050 4103 4118 4325 4356 4363 4465  
## [43] 4484 4490 4540 4579 4594 4607 4620 4652 4665 4698 4729 4838  
##   
## $ACP  
## [1] 1620