Portfolio Creation | Mimicking 10-Year S&P 500 Returns via DCC Multivariate EGARCH Processes

Nathan Matare

[a] *The University of Chicago Booth School of Business*

[b] *nmatare@chicagobooth.edu*

Abstract

Create a portfolio that best mimics the daily returns of the S&P 500 Index over the past 10 years and does not hold more than 10 securities at any one time. Securities may consist of individual company stocks and not funds nor any other type of investment vehicle.

Methodology

I will mimic the daily returns of the S&P 500 by isolating and holding ten securities whose returns are most similar to the S&P 500 during each period. This is a non-trivial process, thus I employ Dynamic Conditional Correlation (DCC) Multivariate Exponential Generalized Autoregressive Conditional Heteroscedasticity (EGARCH) models to identify and isolate similar return-yielding securities for each period throughout the 10-year sample.

The process outline is such:

1. Collect daily price levels on S&P 500 secruities;
2. Clean and pre-process data;
3. Estimate time varying conditional correlation via eGARCH and DCC models;
4. Isolate the top-ten most correlated securities and form a daily portfolio; and
5. Compare portfolio returns to S&P 500 returns.

Project Summary:

First, a targeted universe of securities is generated. Next, raw price levels are cleaned, logged, and first-order integrated. Several GARCH models are considered [1] in order to best capture the variance dynamics in each time series. After completion, an appropriate VAR DCC model is fit to the security and market time-series data. Next, the top-ten most correlated securities are selected and inputed into the final portfolio. Finally, the portfolio returns are compared against the daily S&P 500 returns.

Collect Daily Price Levels

I begin by compiling a list of all stocks currently listed on the S&P 500. These stocks constitute the 'universe of securities' and form the basis of candidate stocks. In order to expedite the collection of granular price series data, I program the function "SecurityScraper" [2] to integrate with Quandl.com's API and scrape daily price level data for the past 10 years.[3]

[1] SGARCH, APARCH, GJR-GARCH, EGARCH
[2] See appendix for additional information
[3] See Quandl datasets

```r
enddate <- Sys.Date()
startdate <- enddate - 365*10
source("secruityscraper.R")
SecruityScraper(name="SP500",
                startdate=startdate,
                enddate=enddate, type="WIKI",
                key="X", sleep=0)
```

Collect Daily Price Levels

The collected 10MB raw dataset contains weekends, holidays, and missing observations. It is necessary to clean and pre-process the data in order to facilitate smooth analysis. I deploy another function, "SecruityCleaner" [4] to clean the data. [5] Securities with too few observations are removed from the candidate pool; the universe of potential securities totals 474 stocks with 2514 daily observations.

```r
source("secruitycleaner.R")
SecruityCleaner(name="SP500", days=144)
```

Additionally, I append daily S&P 500 price levels to the dataset:

```r
sp <- Quandl("YAHOO/INDEX_GSPC", start_date=startdate, end_date=enddate)
rows <- match(as.character(sp[,1]), as.character(raw[,1])) #match SP levels to clean dataset
raw$SP500 <- sp[,2] #append S&P 500 to vector
```

Now that the dataset contains the entire universe of cleaned price levels, I further transform the data via a natural log and first-order integration. Such transformation grants an advantageous property of approximating continuously compounding returns:

$$r = ln(p_t) - ln(p_{t-1})$$

```r
#Convert data to log level and take difference
data <- data.frame(apply(raw[,-1], MARGIN=2, log))
data$Date <- as.character(dates)

#Difference log prices to find returns
rtrn <- data.frame(apply(data[,-dim(data)[2]], MARGIN=2, diff))
rtrn$Date <- as.character(dates[-1])
rownames(rtrn) <- sp$Date[-1]
```

Analysis: Univariate Variance Modeling

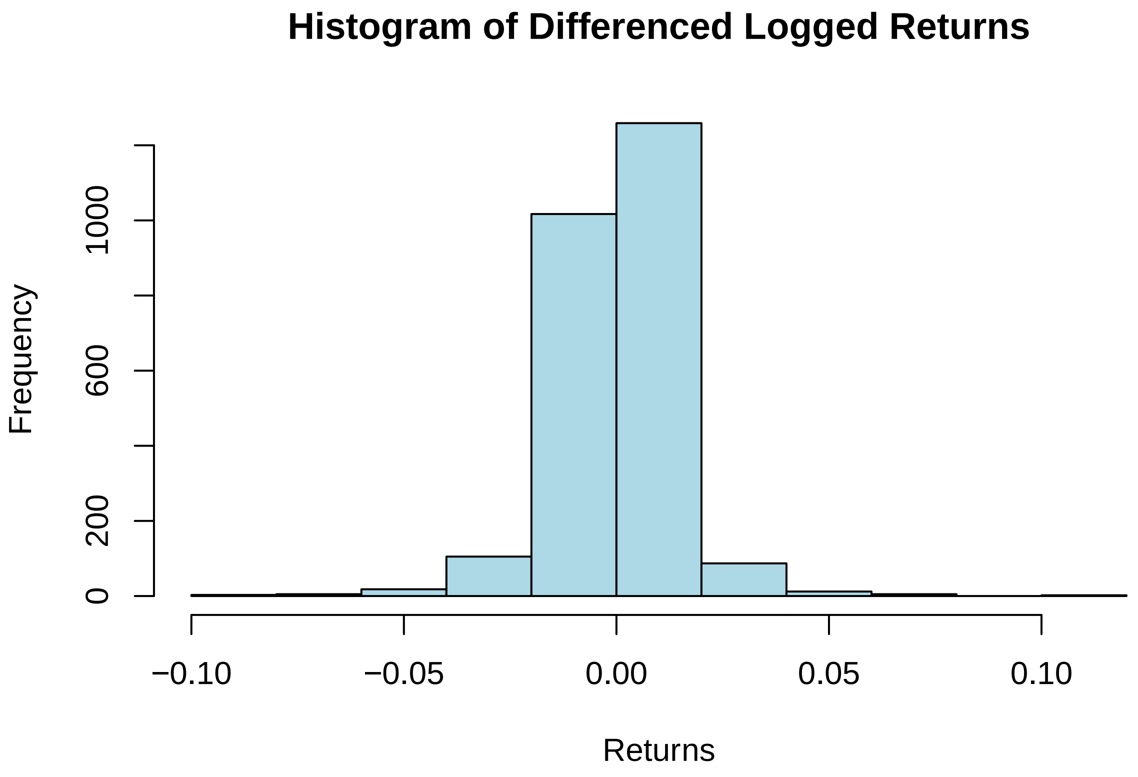
The following analysis will become computationally expensive, I instantiate a parallel environment to increase the speed of computation.

```r
library(parallel)
cl <- makeCluster(min(detectCores(),5), type=ifelse(.Platform$OS.type=="unix","FORK","PSOCK"))
```

[4] See appendix for additional information

[5] Missing observations are imputed with either the nearest column value or column mean, dependent on the situation, see appendix for code logic

Before fitting GARCH models, I inspect several return series. For the S&P 500 series, I observe non-normality, heavy kurtosis, and slight negative skew. Given that standard GARCH models assume ~N(0,1) errors, I caution against the standard model—it may be necessary to adjust the distribution parameters to account for the skewed non-normality. Similar analysis and conclusions follow for individual stock variance distributions.



```
kurtosis(moddata[,1])
```

```
## [1] 12.93363
```

```
skewness(moddata[,1])
```

```
## [1] -0.3189841
```

Thus, to account for non-normality and left skew, I vary model parameters in an attempt to capture the underlying dynamics. I begin by adjusting distribution parameters: normal, normal-skewed, t-distribution, t-skewed; before settling on a normal distribution. It appears that while the data may not be perfectly normal—the normal distribution, in-fact, captures the underlying process more sufficiently than any other available distribution.

Further, standard GARCH models assume symmetry between negative and positive returns. That is, previous period positive returns affect current period variance at the same magnitude as previous period negative returns. However, empirical research suggests otherwise; indeed, previous period negative returns affect current period variance at a greater magnitude than previous period positive returns. Thus, I introduce several asymmetric models, namely: EGARCH, GJR-GARCH, APARCH[6], before determining through model diagnostics that an exponential GARCH is the best candidate.[7]

[6] http://public.econ.duke.edu/~boller/Papers/glossary_arch.pdf

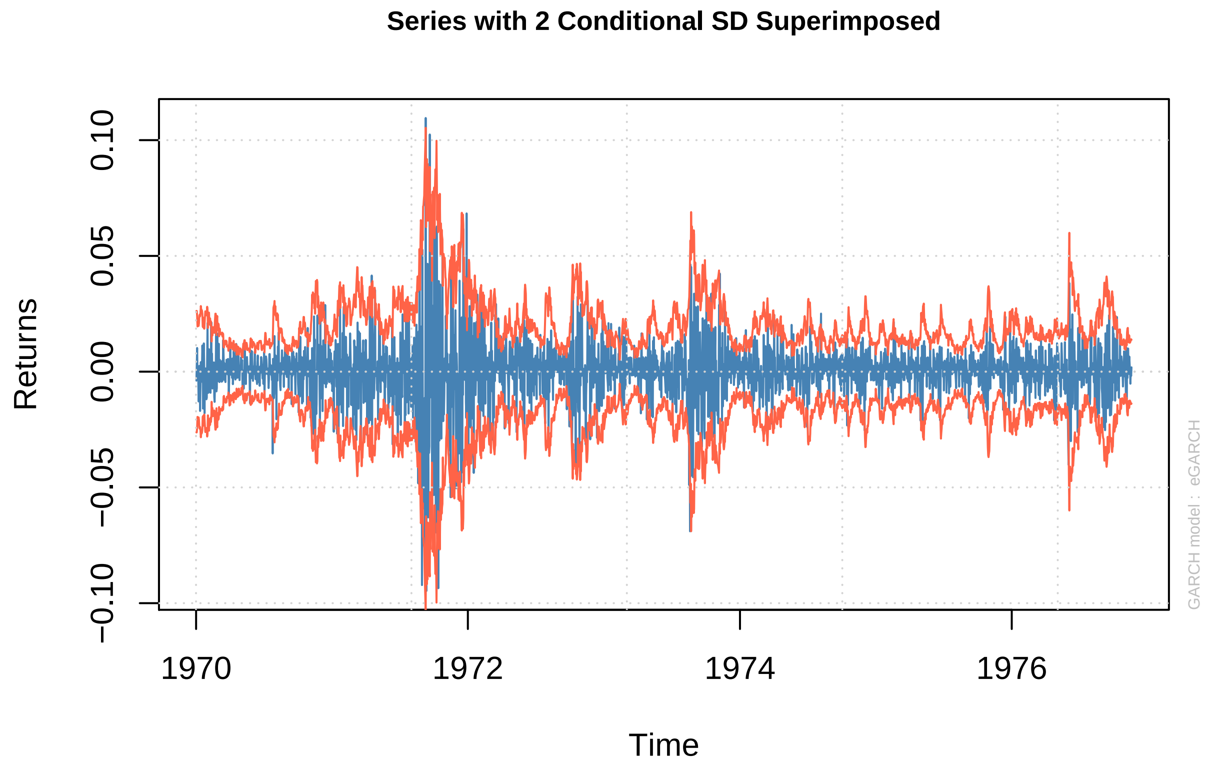
[7] While an GJR-GARCH process fit the series better—matrix singularities when the model is piped into the VAR DCC model force me to select an EGARCH process

$$\log \sigma_t^2 = \omega + \sum_{k=1}^{q} \beta_k g(Z_{t-k}) + \sum_{k=1}^{p} \alpha_k \log \sigma_{t-k}^2$$

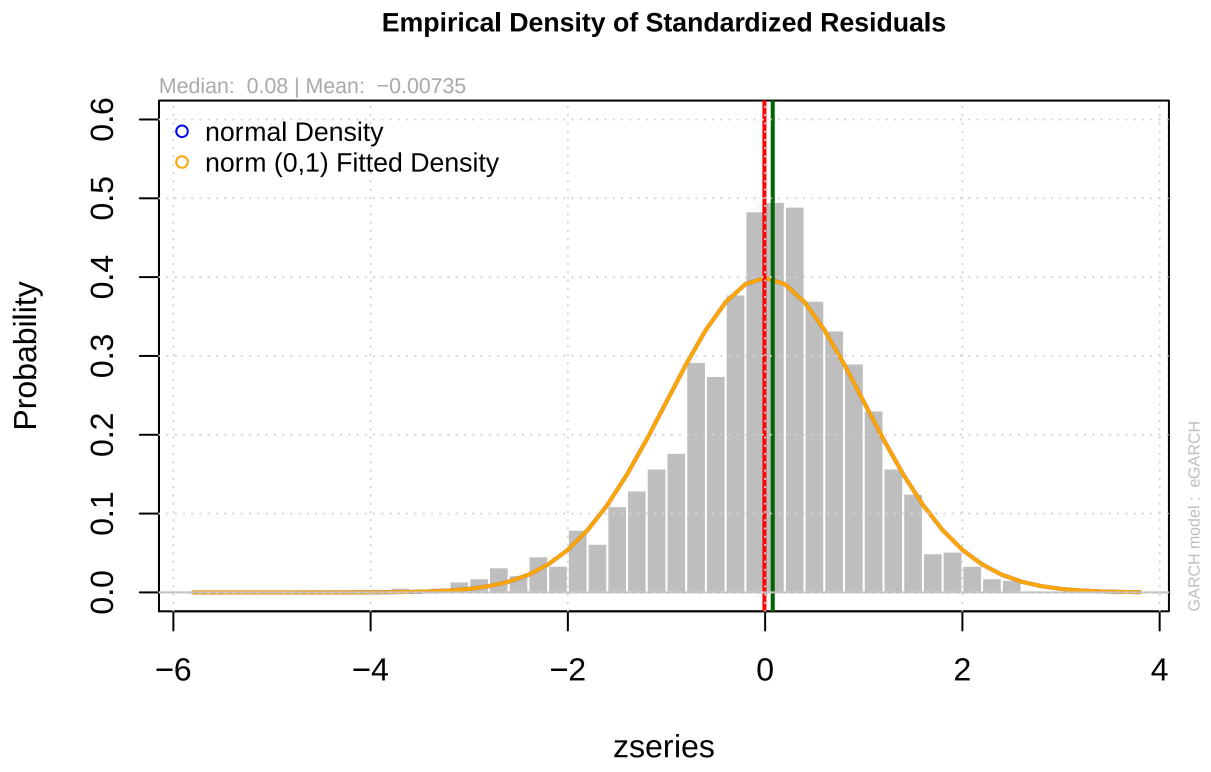
Although I am able to manually determine that an EGARCH(2,2) process passes all diagnostic tests and fits the S&P 500 series sufficiently, it is time prohibitive to replicate this manual process for all individual secruities. I could automate this process and select the most advantageous EGARCH(p,q) model based upon residual, autocorrelation function (ACF), partial autocorrelation function (PACF), and information criteria; but given time constraints, I settle on an EGARCH(1,1) process for all individual securities.

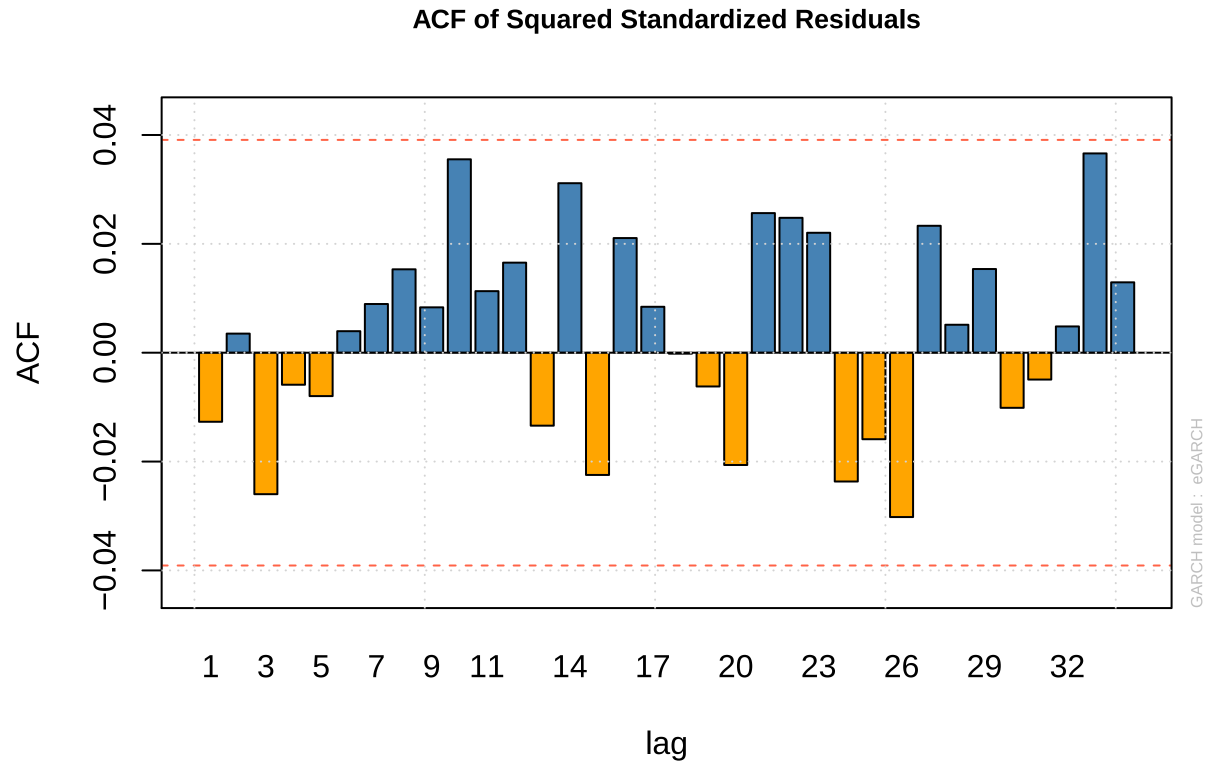
```
library(rmgarch)
require(rugarch)

#Build parameters for market GARCH
        spec1 <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(2,2)),
                       distribution.model= "norm")
#Build parameters for secruity GARCH
        spec2 <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1,1)),
                       distribution.model= "norm")
#Fit GARCH models
        garch1 <- ugarchfit(spec=spec1, data=moddata[,1], solver.control = list(trace=0),
                     cluster=cl)
        garch2 <- try(ugarchfit(spec=spec2, data=moddata[,2], solver.control = list(trace=0),
                     cluster=cl), silent=TRUE)
```

Graphic displays of diagnostic checks and assumptions follow:[8]



[8]The dates in the below plots are incorrect; the rmgarch package has an error that defaults to this time period





As discussed previously, residual diagnostics reveal that the errors are not ~N(0,1) I.D. Adjusting otherwise worsen the errors.

```
kurtosis(garch1@fit$residuals)
```

```
## [1] 12.63651
```

```
skewness(garch1@fit$residuals)
```

```
## [1] -0.4385969
```

	Estimate	Std. Error	t value	Pr(>|t|)
mu	0.0002682	0.0001182	2.269900	0.0232137
ar1	0.3271671	0.0554327	5.902059	0.0000000
ma1	-0.3909436	0.0537169	-7.277854	0.0000000
omega	-0.2859863	0.0179554	-15.927569	0.0000000
alpha1	-0.2968012	0.0303461	-9.780529	0.0000000
alpha2	0.1269753	0.0272309	4.662917	0.0000031
beta1	0.9999999	0.0000088	113420.181733	0.0000000
beta2	-0.0312999	0.0019570	-15.993674	0.0000000
gamma1	-0.1449825	0.0391676	-3.701592	0.0002143
gamma2	0.2973042	0.0425145	6.993006	0.0000000

Upon inspection of the EGARCH(2,2) S&P 500 process, I observe highly significant covariates. Notably, the asymmetric terms(gamma(1), gamma(2)) are highly significant. It appears that the variance may be non-stationary. The beta(1) coefficient is nearly 1 with an extremely high t-stat—indicative of a random walk process. Intuitively this appears spurious—variance, one would expect, should not arbitrary depart and wander from a mean-level. All other coefficients appear proper.

	Estimate	Std. Error	t value	Pr(>|t|)
mu	0.0000000	0.0000006	0.0000047	0.9999962
ar1	0.3685570	0.0319029	11.5524440	0.0000000
ma1	-0.6756093	0.0083962	-80.4663500	0.0000000
omega	-0.4826973	0.0193245	-24.9785618	0.0000000
alpha1	0.1133147	0.0264116	4.2903325	0.0000178
beta1	0.9002138	0.0012210	737.2798943	0.0000000
gamma1	1.4878254	0.0442731	33.6056502	0.0000000

Upon inspection of the EGARCH(1,1) individual security process, I observe mostly significant covariates. Again, the asymmetric term gamma(1) is highly significant. While the spurious beta coefficient is no longer observed, the mean-level error term (mu) is highly insignificant. Indicative of the aforementioned non-normal issues. Unlike the static S&P 500 EGARCH(2,2) model, this EGARCH(1,1) model is fit to all individual securities and may not be the most appropriate process. However, given resource constraints I continue otherwise. [9]

Analysis: Multivariate Dynamic Conditional Correlations

With the estimated univariate EGARCH(2,2) and EGARCH(1,1) models, I am ready to incorporate the asymmetric conditional variance into a multivariate dynamically conditioned vector auto-regression (VAR). Intuitively, when the S&P 500 index and individual security move in the same direction, as measured by variance and returns, the correlation increases slightly. The opposite holds if both the index and security move in opposite directions. As with variance estimations, correlations are assumed to only temporarily deviate from a long run mean—as indicated

[9] After continually running the entire analysis, a EGARCH(2,2) S&P 500 process in-conjunction with a GJR-GARCH(1,1) individual security process produces the lowest error rate; I remain with the EGARCH process however, given the robust diagnostic checks

by the omega parameter. Further, given asymmetry in the correlation dynamics, these increased(decreased) correlations should be exacerbated in economic up(down) turns.

Thus, I consider both a DCC and an ADCC (asymmetric) VAR:

$$\rho_t = \bar{\rho}(1 - \alpha - \beta) + \alpha z_{1,t-1} z_{2,t-1} + \beta \rho_{t-1}$$

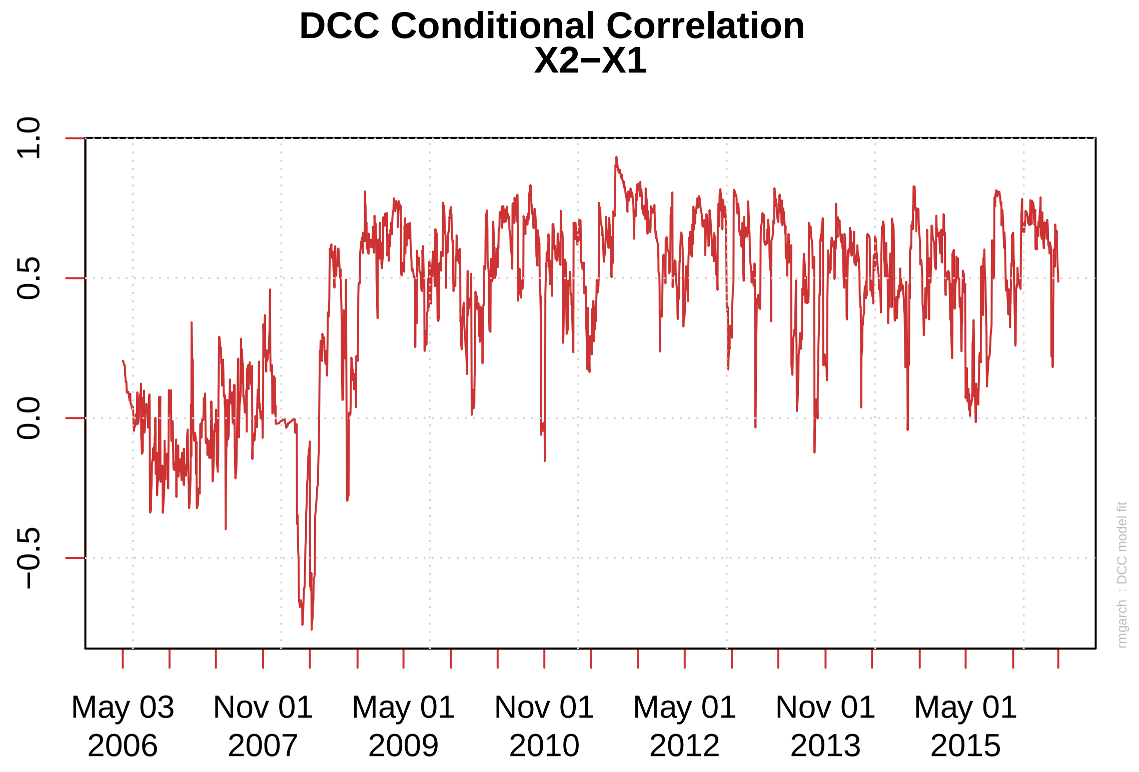
$$\rho_t = \omega + \alpha z_{1,t-1} z_{2,t-1} + \gamma z_{1,t-1} z_{2,t-1} + (I_{z_{1t}<0})(I_{z_{2t}<0}) + \beta \rho_{t-1}$$

Diagnostic checks indicate that the asymmetric correlation term is insignificant for most pairs; I discard the ADCC model. Through AIC/BIC selection criteria, I determine that a DCC(1,1) model is most appropriate for the data.[10]

After fitting the model, I aggregate the correlation list into a matrix and terminate the parallel cluster.

```r
#Build parameters for DCC model
dccspec <- dccspec(VAR=TRUE, uspec = multispec(c(spec1, spec2)), dccOrder = c(1,1),
                   distribution = "mvnorm")

#Fit DCC model
dcc <- try(dccfit(dccspec, data = moddata, fit.control=list(scale=TRUE), cluster=cl),
           silent=TRUE)

corrmatrix <- rcor(dcc, type="R")
corrmatrix <- zoo(corrmatrix[1,2,], order.by=as.Date(rownames(moddata)))
timevarcor[,n] <- corrmatrix

stopCluster(cl)
```

[10]While higher order DCC models have comparable AIC/BIC scores and decreased residual errors, the higher order lagged coefficients are deemed insignificant. With caution towards over-fitting, I hold a DCC(1,1) process.



Upon inspection, the multivariate DCC(1,1) VAR formed from a EGARCH(1,1) and EGARCH(2,2) process yields mostly significant coefficients. As previously discussed, several coefficients appear insignificant. Otherwise, the model appears to capture the dynamically conditioned correlation well.

	Estimate	Std. Error	t value	Pr(>|t|)
[X1].omega	-0.2812474	0.0478390	-5.8790349	0.0000000
[X1].alpha1	-0.2931646	0.0282848	-10.3647370	0.0000000
[X1].alpha2	0.1245218	0.0278806	4.4662556	0.0000080
[X1].beta1	1.0000000	0.0000232	43143.5149533	0.0000000
[X1].beta2	-0.0309803	0.0052412	-5.9109612	0.0000000
[X1].gamma1	-0.1388295	0.0202280	-6.8632455	0.0000000
[X1].gamma2	0.2874979	0.0294032	9.7777600	0.0000000
[X2].omega	-0.0115805	0.0208363	-0.5557843	0.5783584
[X2].alpha1	-0.0448929	0.1792462	-0.2504539	0.8022364
[X2].beta1	0.9977422	0.0016093	620.0005213	0.0000000
[X2].gamma1	0.1383119	0.1087630	1.2716814	0.2034863
[Joint]dcca1	0.1147132	0.0295329	3.8842557	0.0001026
[Joint]dccb1	0.8813116	0.0311983	28.2486921	0.0000000

Analysis: Portfolio Evaluation I

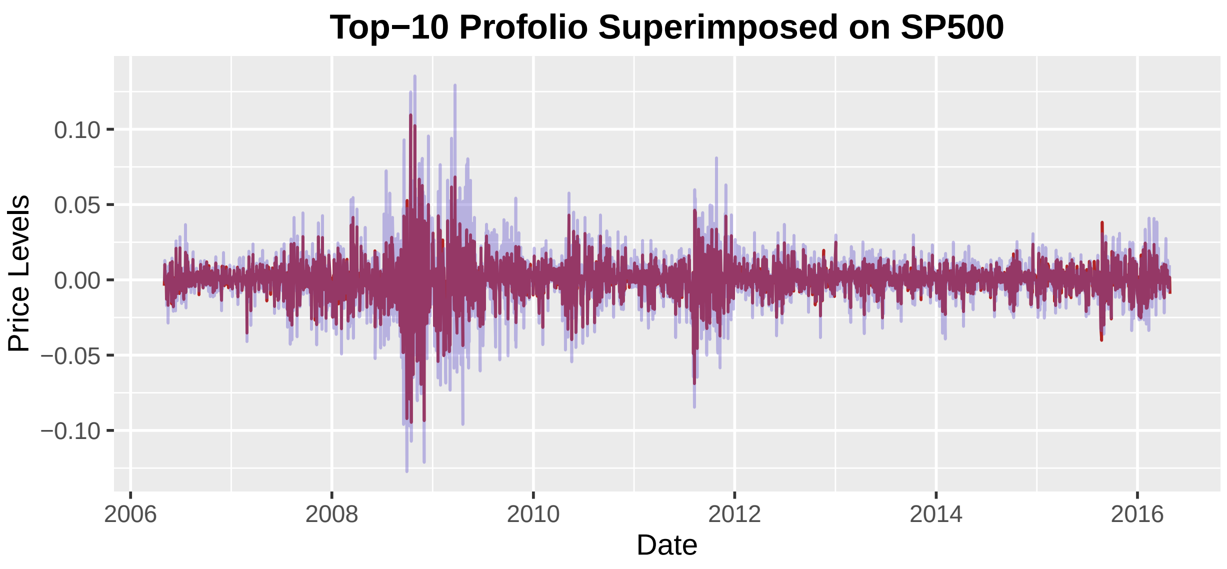
From the 20MB csv file containing the dynamically conditioned time varying correlation for all 474 securities, I select the ten-most correlated securities (to the S&P 500) for each period. Put another way, for each period in the sample, I select the ten-most correlated securities; the ten-most correlated securities then form the mimicking portfolio. I lookup the returns for each security per period and equally weight each individual security's return against the entire portfolio. This is a rather naive approach; I am essentially equally weighting each security under the assumption that the overall portfolio will mirror the S&P 500's returns well. A more pragmatic approach would be to optimally weight each individual security's return based upon some intrinsic property—say regime or security characteristic.

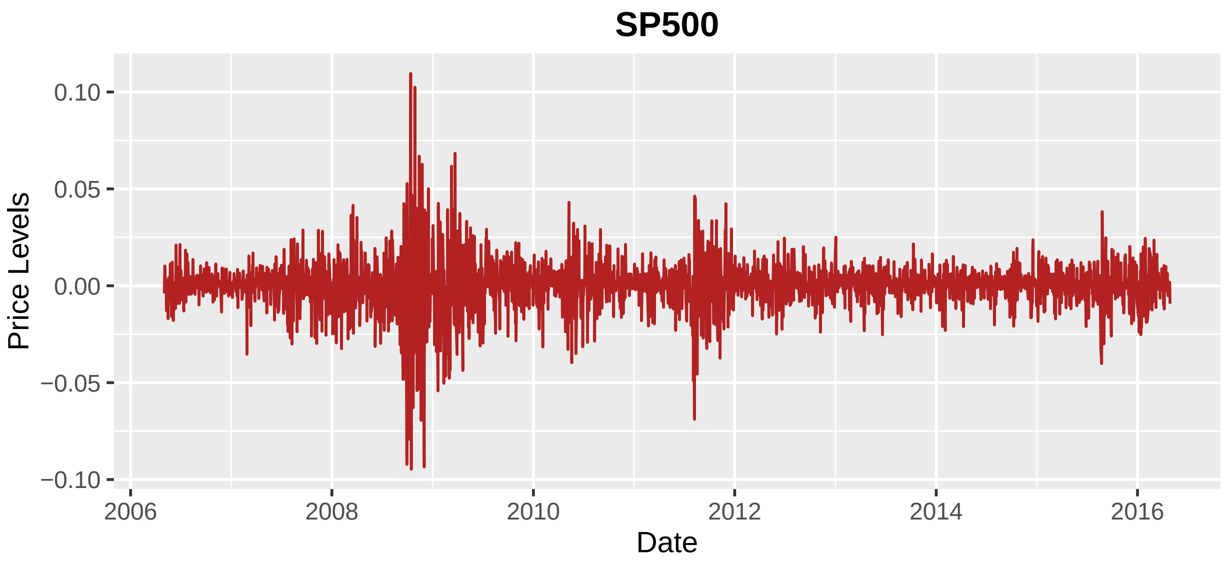
```r
top10names <- apply(timevarcor[,-1], MARGIN=1, FUN=function(x)
                   names(head(sort(x, decreasing = decreasing=TRUE),10)))

#Compare returns against SP&500 Returns
L <- dim(top10names)[2]
eval <- data.frame(Date=sp$Date[-1])
eval$SP500 <- rtrn[,474]
eval$benchmark <- NA

for(n in 1:L){
    date <- names(top10names[1,n])
    topstocks <- as.character(top10names[,n])
    lowstocks <- as.character(low10names[,n])

    eval$benchmark[n] <- mean(as.numeric(rtrn[date,topstocks]))
}
eval$error <- eval$SP500 - eval$benchmark
```

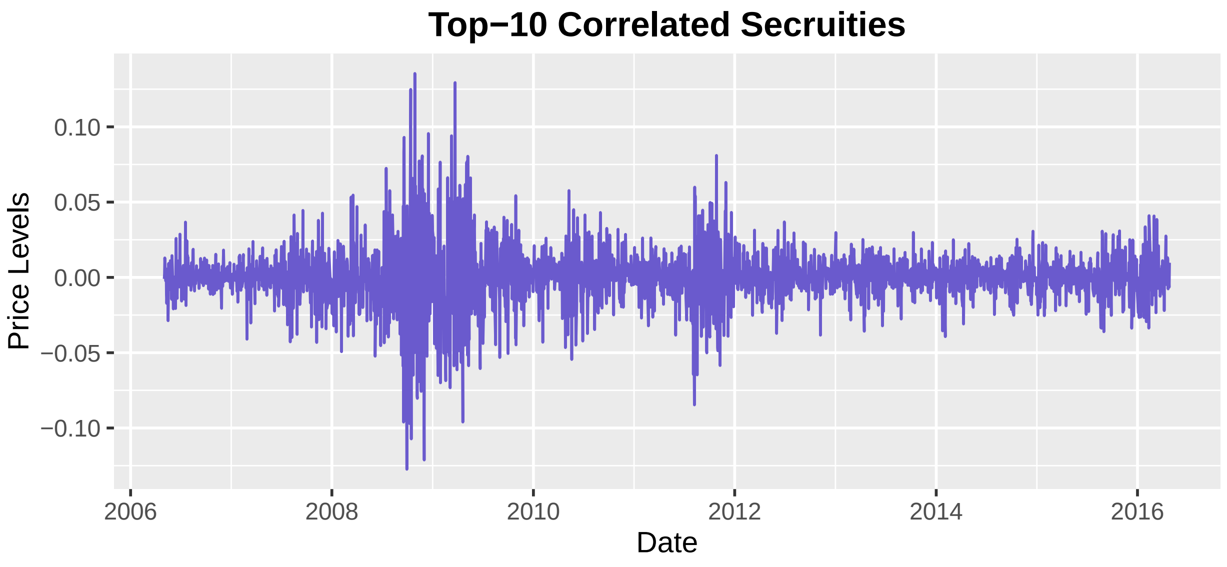
I now have a vector of portfolio returns for each period in the 10-year sample. I directly compare the performance and error[11] of this portfolio to the S&P 500 index.

[11] The eval$error vector stores the difference between the returns from the portfolio and the S&P 500, IE the 'residual' error

Analysis: Portfolio Evaluation II







The constructed portfolio adequately mimics the returns of the S&P 500; the bottommost graph displays the constructed returns. The middle graph displays the S&P 500 returns. And the topmost graph displays the portfolio returns superimposed against the S&P 500 returns. The EGARCH DCC VAR selected portfolio adequately mirrors the S&P 500 benchmark; however, during regimes of high volatility the residual errors are far greater than during periods of relatively tranquility. This is rather intuitive, correlation and variance are tough to estimate during highly unpredictably periods.

I now quantify the accuracy of the tracking portfolio via a Sharpe Ratio:

$$S_a = \frac{E[R_a - R_b]}{\sqrt{\mathrm{var}[R_a - R_b]}} * \sqrt{N}$$

Where N = 221 (number of trading days), Ra = average return, and Rb = risk free rate

The daily annualized Sharpe Ratio of the S&P 500 Index over a 10-year sample is 0.2052 while the daily annualized Sharpe Ratio of the tracking portfolio is 0.2017. Given that the objective is to mimic the daily returns of the S&P 500 (without considering volatility), it is unsurprising that the tracking portfolio performed worse. The benefits of diversification are lesser in a ten security portfolio.

I now consider the performance of the tracking portfolio as compared to the S&P 500, namely:

$$0.05958 = \frac{E[0.0002585 - 0.0001821]}{\sqrt{\mathrm{var}[0.0002585 - 0.0001821]}} * \sqrt{221}$$

Where, Ra= average return of formed portfolio, and Rb = average return of S&P 500 benchmark

The tracking portfolio, after considering returns and volatility, inadvertently outperforms the S&P 500 index. Finally, I construct a loosely defined Sum of Square Residuals (SSR) metric. While this metric is not valuable in isolation—one must re-run the entire analysis and vary model parameters to achieve insight —one can use it to compare across models. Given a multivariate DCC(1,1) ~N(0,1) with univariate EGARCH(2,2) ~N(0,1) and EGARCH(1,1) ~N(0,1) process I achieve an 'SSR' of 0.1836. [12]

```r
sum(((eval$error) ** 2), na.rm=TRUE)
```

```
## [1] 0.1835966
```

Appendix

```r
SecruityScraper <- function(name, startdate, enddate, type, key, sleep){

# Description
# This script scraps Quandl.com for data given a date range and symbol list

# Arguments

#'name' is the name of a csv file containing stock symbols to download #do not include .csv
#'startdate' is the start date of data
#'enddate' is the end data of data
#'type' is the Quandl database header; this should be input as a character  IE 'WIKI'
```

[12] Given multivariate DCC(1,1) ~N(0,1) with either univariate GJR-GARCH(2,2) ~STD(0,1) and GJR-GARCH(1,1) ~STD(0,1) or GARCH(2,2) ~N(0,1) and GARCH(1,1) ~N(0,1) this 'SSR' reveals 0.1654 and 0.2043, respectively.

```r
#'key' is the Quandl key
#'sleep' is the number of seconds to wait before querying Quandl server

# Example
# SecruityScraper("NASDAQ", "2001-09-11", "2015-01-01", "WIKI","40F..U&E")

# Requirements

# Requires the Quandl library
# Requires loaded csv file to have a column header of "ticker" for all secruity symbols
# Startdate and enddate must not be same date

#setup environment
name <- name
raw <- read.csv(paste(name,".csv", sep=''))
tickers <- as.character(raw$ticker)

enddate <- enddate
startdate <- startdate
dates <- seq.Date(as.Date(startdate), as.Date(enddate), by='day') #create daily dates

#allocate memory for data
L <- length(tickers)
D <- length(dates)
dataset <- matrix(ncol=L, nrow=D)
dimnames(dataset) <- list(rownames(dataset,
                                   do.NULL = FALSE,
                                   prefix = "row"),
                                   colnames(dataset,
                                   do.NULL = FALSE,
                                   prefix = "col"))
colnames(dataset) <- tickers
rownames(dataset) <- as.character(dates)

#specify date range
enddate <- dates[length(dates)]
startdate <- dates[1]
header <- paste(type, "/", sep="")

#retrive stock data
require(Quandl)
Quandl.api_key(key)

for(i in 1:L){

tryCatch({
sym <- paste(header, tickers[i], sep="")
info <- Quandl(sym, start_date=startdate, end_date=enddate)
tempdate <- info$Date

info <- data.frame(info$Close)
rownames(info) <- tempdate
put <- merge(info, dataset[,i], by=0, all=TRUE)
dataset[,i] <- put$info.Close
message("Scraping data for stock: ", tickers[i], " | Number: ", i, "/", L)

Sys.sleep(sleep) #API speed limit
}, error=function(e)    {
cat("ERROR :",conditionMessage(e), "\n")
#Last value throws error
})

}

#export data
write.csv(dataset, file = paste(name,"-output.csv", sep=''))

}


# Utility script for Secruity Cleaner
remove_outliers <- function(x, na.rm = TRUE, ...) {
  qnt <- quantile(x, probs=c(.25, .75), na.rm = na.rm, ...)
  H <- 1.5 * IQR(x, na.rm = na.rm)
  y <- x
  y[x < (qnt[1] - H)] <- NA
  y[x > (qnt[2] + H)] <- NA
  y
}
```

```r
SecruityCleaner <- function(name, days){

# Description
# This utility script removes non-trading days from SecruityScraper and imputes
  #missing values with the nearest value

# Arguments
#'name' is the name of a csv file containing stock symbols to download #do not include .csv
#'days' is the number of expected non-trading days per average year, default is 144

# Example
# SecruityCleaner("NASDAQ", 144)

# Requirements

# Requires the zoo library

# Other

# Error "ERROR(expected) : undefined columns selected" is excpected as columns are removed and
  #length of for-loop does not dynamically change

#setup environment
raw <- read.csv(paste(name,"-output.csv", sep=''))
colnames(raw)[colnames(raw)=="X"] <- "date"
L <- length(raw)
# take out the non trading days aka weekends and holidays
narows <- as.numeric(rownames(raw[rowSums(is.na(raw))>=dim(raw)[2]-10,]))
raw <- raw[-narows,]
U <- dim(raw)[1]/365*(days-3) #number of years worth of data * number of non-trading days =
#max expected NAs #where 3 is a buffer
L <- length(raw)

#Remove secruites with missing observations
for(n in 2:(L-1)){

  tryCatch({
        if (sum(is.na(raw[,n])) < U) {
                        message("Pass: ", n)
        } else {
                      message("Remove (not enough observations): ", n)
                        raw <- raw[,-c(n)]
        }
    }, error=function(e)    {
      #as columns are strunk, n becomes larger than initial column number set
      #and produces errors at the end
    cat("ERROR(expected) :",conditionMessage(e), "\n")
    })

}

require(zoo)
L <- length(raw)
end <- dim(raw)[1]

#Impute missing price levels with nearest price level
for(n in 2:L){

#place value in first
raw[1,n] <- ifelse(is.na(raw[1,n])==TRUE, na.locf(raw[,n])[1], raw[1,n])
#place value in last
raw[end,n] <- ifelse(is.na(raw[end,n])==TRUE, na.locf(raw[,n])[end], raw[end,n])
raw[,n] <- na.locf(raw[,n] ,fromlast=TRUE) # scrub NA from backwards
message("Imputing missing values with nearest value: ", n)

}

#Remove outliers and replace them with mean
L <- length(raw)
for(n in 2:L){
            removed <- remove_outliers(raw[,n])
            removed[is.na(removed)] = mean(removed, na.rm=TRUE)
            raw[,n] <- removed
            message("Removing outliers and imputing with mean: ", n)
            }

write.csv(raw, file = paste(name,"-output-clean.csv", sep=''))

}
```

```r
rm(list=ls(all=TRUE))
options("width"=200)
options(scipen=999)
options(digits=4)
wkdir <- "/home/johnconnor/projecthedge/"
lbdir <- "/home/johnconnor/projecthedge/lib/"
datdir <- "/home/johnconnor/projecthedge/data/"

##Set Parameters
enddate <- Sys.Date()
startdate <- enddate - 365*10 #appox 10 years of data, not counting trading days

#Load required libraries & utility scripts
setwd(lbdir)
library(Quandl)
library(zoo)
library(ggplot2)
library(rmgarch)
library(parallel)
require(rugarch)
require(gridExtra)
library(moments)
source("secruityscraper.R")
source("secruitycleaner.R")
setwd(datdir)

raw <- read.csv("SP500-output-clean.csv")
dates <- raw$date
raw <- raw[,-c(1)]
#append S&P 500 for comparision
sp <- Quandl("YAHOO/INDEX_GSPC", start_date=startdate, end_date=enddate)
sp <- sp[-c(2,3,4,6,7)] #keep only dates and close date
sp <- sp[order(sp$Date),] #reorder data
rows <- match(as.character(sp[,1]),as.character(raw[,1])) #match SP levels to clean dataset
raw <- raw[rows,]
dates <- dates[rows]
raw$SP500 <- sp[,2] #append S&P 500 to vector list

#Convert data to log level and take difference
data <- data.frame(apply(raw[,-1], MARGIN=2, log)) #convert to log prices
data$Date <- as.character(dates)

#Difference log prices to find returns
rtrn <- data.frame(apply(data[,-dim(data)[2]], MARGIN=2, diff)) #take difference of log prices
rtrn$Date <- as.character(dates[-1])
rownames(rtrn) <- sp$Date[-1]

#Method 1 Compute Correlation Matrix
L <- dim(rtrn)[2]-2 #all secruities minus SP500 and dates
 #Create data.frame for time conditional variance
timevarcor <- matrix(ncol=L, nrow=dim(rtrn)[1]-2)
colnames(timevarcor) <- head(colnames(rtrn), -2) #all secruites minus SP500 and dates
rownames(timevarcor) <- head(rtrn$Date,-2)
```

```r
#Create parallel environment for each iteration
cl <- makeCluster(min(detectCores(),5),type=ifelse(.Platform$OS.type=="unix","FORK","PSOCK"))

for(n in 1:L){
    #n<-1
        ###Create data.frame for models #subtract 1 because of differencing
        moddata <- matrix(ncol=2, nrow=dim(rtrn)[1]-1)
        moddata[,1] <- head(rtrn[,"SP500"],-1)
        moddata[,2] <- head(rtrn[,n],-1)
        rownames(moddata) <- head(rtrn$Date,-1)
        moddata <- moddata[-nrow(moddata),] #remove last row
        moddata <- data.frame(moddata)

        #Build parameters for market GARCH
        spec1 <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(2,2)),
                                distribution.model= "norm")
        #Build parameters for secruity GARCH
        spec2 <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1,1)),
                                distribution.model= "norm")

        #Fit GARCH models
        garch1 <- ugarchfit(spec=spec1, data=moddata[,1], solver.control = list(trace=0),
                            cluster=cl)
        garch2 <- try(ugarchfit(spec=spec2, data=moddata[,2], solver.control = list(trace=0),
                                cluster=cl), silent=TRUE)

        #Build parameters for DCC model
        dccspec <- dccspec(VAR=TRUE, uspec = multispec(c(spec1, spec2)), dccOrder = c(1,1),
                           distribution = "mvnorm")

        #Fit DCC model
        dcc <- try(dccfit(dccspec, data = moddata, fit.control=list(scale=TRUE),
                          cluster=cl), silent=TRUE)

        tryCatch({

            plot(dcc, which=4)
            corrmatrix <- rcor(dcc, type="R")
            corrmatrix <- zoo(corrmatrix[1,2,], order.by=as.Date(rownames(moddata)))
            timevarcor[,n] <- corrmatrix

        }, error=function(e) {cat("ERROR :",conditionMessage(e), "\n")
        })

        message("Calculating Time Varying Correlation for Secruity: ", n, "/", L)

}

stopCluster(cl)

top10names <- apply(timevarcor[,-1], MARGIN=1,
                    FUN=function(x) names(head(sort(x, decreasing=TRUE),10)))
low10names <- apply(timevarcor[,-1], MARGIN=1,
                    FUN=function(x) names(head(sort(x, decreasing=FALSE),10)))

#Compare returns against SP&500 Returns
L <- dim(top10names)[2]
eval <- data.frame(Date=sp$Date[-1])
eval$SP500 <- rtrn[,474]
eval$benchmark <- NA
for(n in 1:L){
    date <- names(top10names[1,n])
    topstocks <- as.character(top10names[,n])
    lowstocks <- as.character(low10names[,n])

    eval$benchmark[n] <- mean(as.numeric(rtrn[date,topstocks]))
}
eval$error <- eval$SP500 - eval$benchmark

#Visual Representation
plot1 <-    ggplot() +
            geom_line(data = eval, aes(x = Date, y = SP500, color = 'SP500'),
                      color='firebrick') +
            geom_line(data = eval, aes(x = Date, y = benchmark, color = 'Benchmark'),
                      alpha=0.4, color='slateblue') +
                ggtitle("Benchmark Superimposed on SP500") +
                theme(plot.title = element_text(lineheight=.8, face="bold")) +
                labs(color = "Legend") +
                xlab('Date') +
                ylab('Price Levels')

plot2 <- ggplot() + geom_line(data = eval, aes(x = Date, y = SP500, color = 'SP500'),
                              color='firebrick') +
                ggtitle("SP500") + theme(plot.title = element_text(lineheight=.4, face="bold")) +
                labs(color = "Legend") +
                xlab('Date') +
                ylab('Price Levels')

plot3 <- ggplot() + geom_line(data = eval, aes(x = Date, y = benchmark,
                                    color = 'Benchmark'), color='slateblue') +
                ggtitle("Top 10 Correlated Secruities") +
                            theme(plot.title = element_text(lineheight=.4, face="bold")) +
                labs(color = "Legend") +
                xlab('Date') +
                ylab('Price Levels')
```