Mimicing S&P 500 Returns via eGARCH and DCC models

Nathan Matare

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

[b] *nmatare@chicagobooth.edu*

Abstract

Create a portfolio that best mimics the monthly 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 or any other type of investment vehicle.

Methodology

In order to appropiately mimic the daily returns of the S&P 500, one must isolate secruities whose returns are "most like" the S&P 500 at any given point in time. By selecting ten secruites who most mirror the index, one can form a comparable(in returns) profolio.

Whily various metrics such as cointegration and PCA could be considered as a similarity criterion, I employ dynamic time varying conditional correlation (DCC) models in order to find and select the most benefical secruties. This is a non-trivial process; I outline five steps:

1. Collect daily stock level prices 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 secruites and form a daily profolio
5. Compare profolio returns to S&P 500 returns

First, a targeted universe of secruities is generated. Secondly, the raw price levels are cleaned, pre-processed, and transformed in order to faciliate smooth analysis. I next consider several GARCH models[1] in order to accurately capture the variance dynamics of each time series; after compleletion, an appropiate VAR DCC model is fit to the secruity and market timeseries. Next, the top-ten most correlated secruties are selected and used as the basis for a daily profolio. Finally, the profolio returns are compared against daily S&P 500 returns.

Collect Daily Price Levels

I begin by compiling all stocks currrently listed on the S&P 500. These stocks constitute a 'universe' of secruities, and form the basis of candidate secruites. In order to expediate the data collection process, I write a function "SecruityScraper"[2] to scrape daily price level data for the past 10 years.[3]

[1] GARCH, APARCH, GJG-GARCH, EGARCH are considered
[2] See appendix for additional information
[3] See Quandl datasets

```
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

Because the raw data contains weekends, holidays, and missing observations it is necessary to clean and preprocess the data in order to faciliate smooth analysis. I write another function, "SecruityCleaner"[4] to clean the data. [5] Secruites that have too few observations are removed from the candidate pool; the universe of secruties totals 474.

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

During this step, I append daily S&P 500 price levels in order to compare the returns of the formed profolio against the index.

```
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
```

The dataset now contains the entire universe of cleaned secruties and S&P 500 price levels. I further manipulate the data by taking the natural log of the price levels; after differencing the data, I am able to take advantage of a natural property of logs. I now have a continously compounded time series.

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

```
#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: Conditional Correlations

This analysis will become computationally expensive. I create a parallel environment in order to increase the speed of computation.

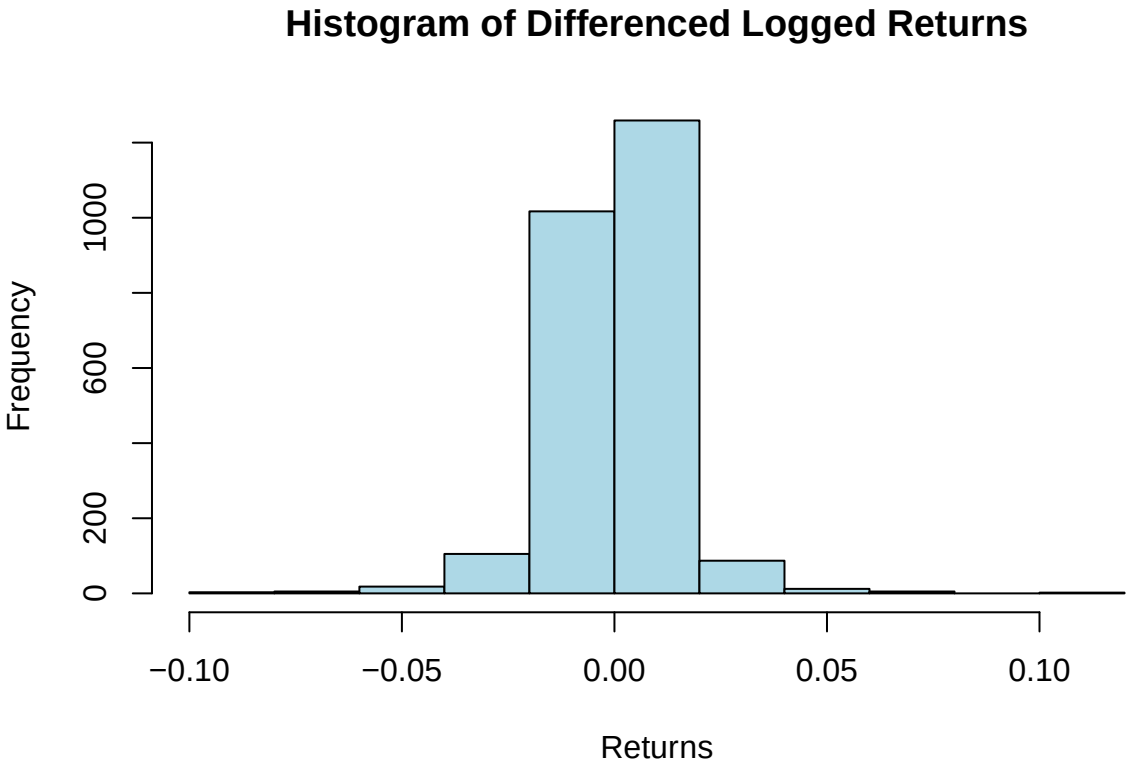
[4] See appendix for additional information

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

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

In order to fit DCC models to each respective secruity and S&P 500 index, I first estimate two univariate GARCH models in order to capture the variance dynamics of each series.

I observe non-normality upon inspection of the raw logged returns. That is, a fat left-tail. Given this non-normality, it may be that I must relax normality assumption on my GARCH episolon errors and choose a more appropiate distribution.




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

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

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

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

I consider asymettric EGARCH, GJG-GARCH, APARCH models and the GARCH model while varying lagged orders and error distributions: (normal, normal-skewed, t-distribution, t-skewed) before settling on a EGARCH(2,2) for the S&P 500 index and a EGARCH(1,1) for all individual secruities.

put in latex for EGARCH model

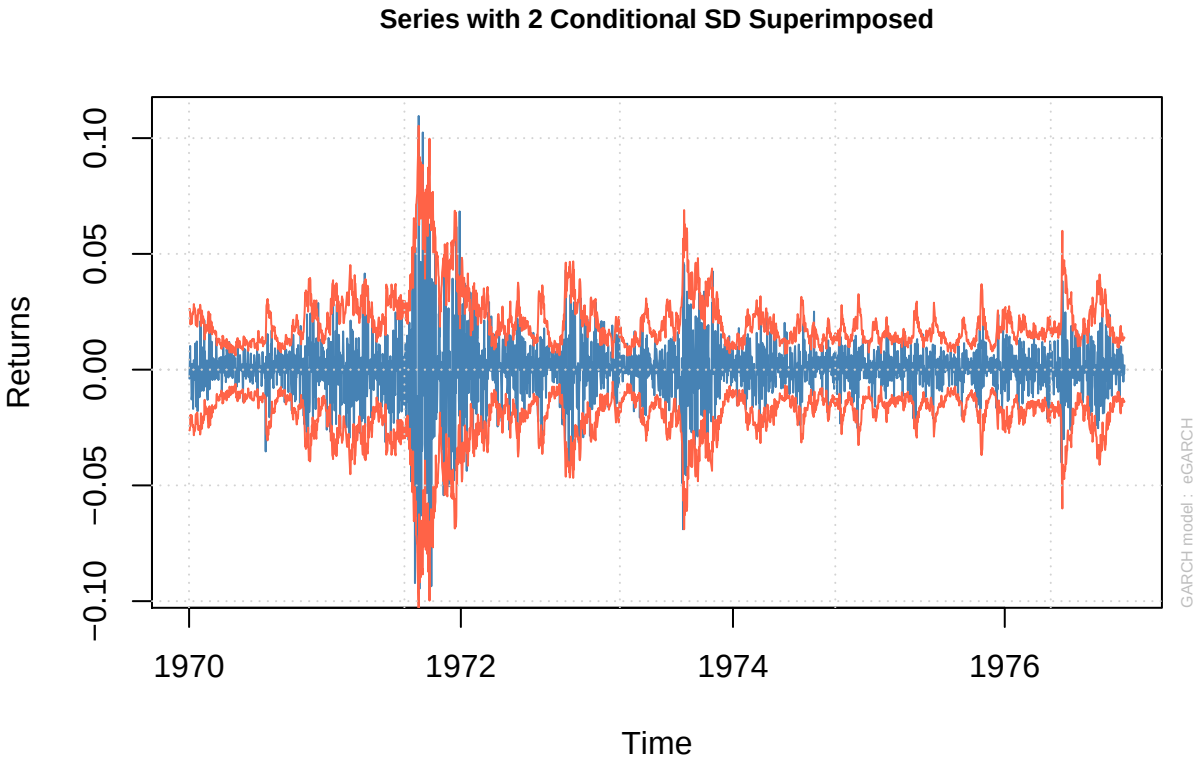
It is unsurprising that an assymetric model works well here; times of high variance correlation etc... talk about what the model is doing.

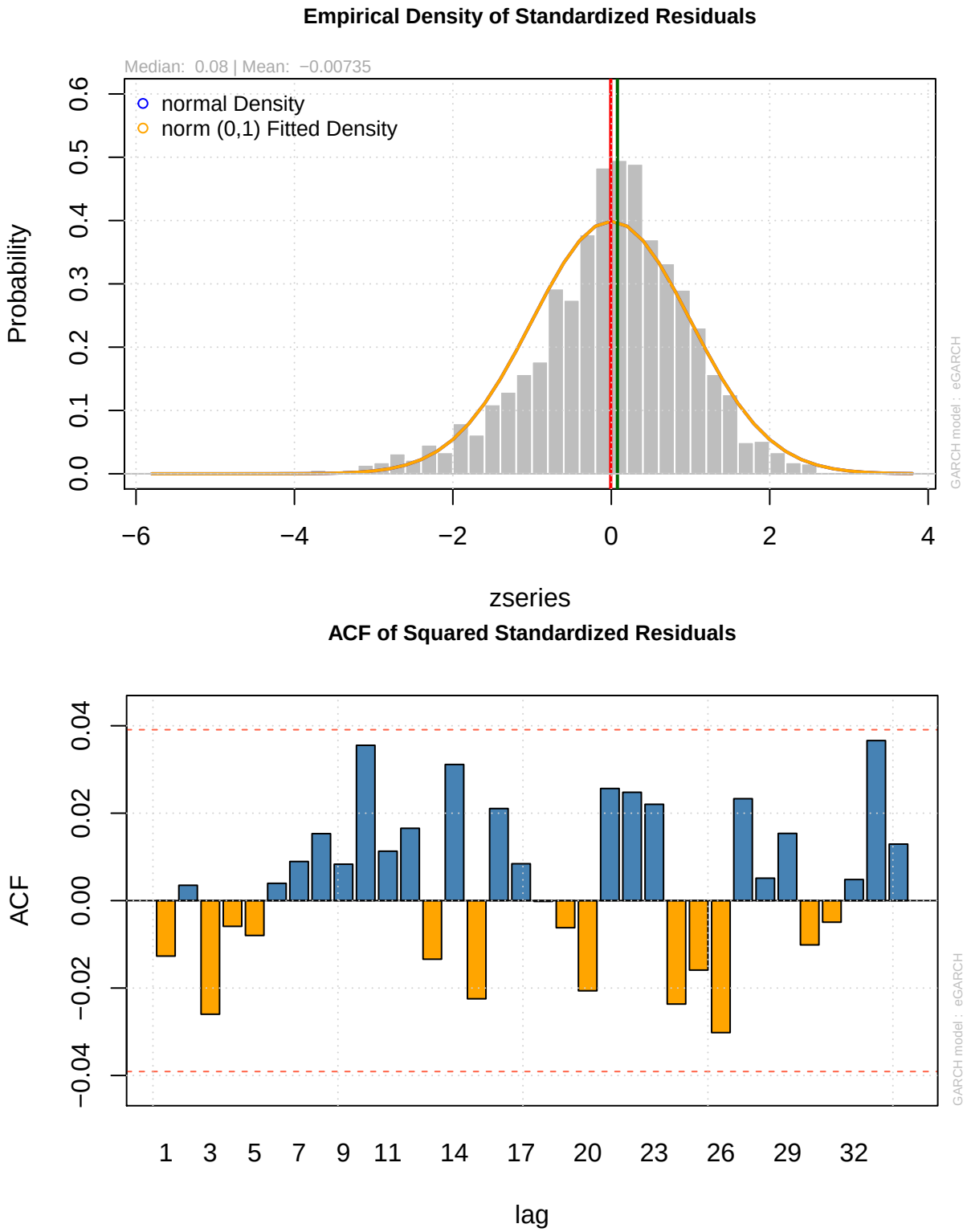
Due to computational and time limitations, I blanket an EGARCH(1,1) to all individual secruites; while most GARCH processes are of either order (1,1) or (2,2), I have drawn a blanket statement and assummed that all secruites will follow this process. This is highly uncertain.

In fact, through dianoistic checking I determine that an GRG-GARCH(1,1) fits mosts series better than any other GARCH process; however when piping the assymetric process into a DCC model, inverse singular matrix errors force adoptation of a slightly less complicated model.

```
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)
```

Before continuing, I conduct several quick diagnostic checks. My S&P 500 EGARCH(2,2) process captures all serial correlation in the time series. While I cannot confirm that each individaul secruity follows a EGARCH(1,1) process, I accept this assumption given time and computational constrains and proceed with the analysis.





Although heavy left skew remains in my residuals–indicative of a less-than-perfect fit ARCH model, these are the best results I am able to get.

While the residual errors are not perfectly normally distributed, relaxing the assumption and estimating student-t or skewed-normal distrubtion worsens the kurtosis and skewness. I proceed under the normal assumption.

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

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

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

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

Inspecting the EGARCH(2,2) S&P 500 coefficients, I observe significant covariates. Notably the assymetric terms(gamma, and gamma2) are highly signifcant.

Inspecting, the EGARCH(1,1) individaul secruity coefficients, I observe highly significant covariates with the expection of the long-run mean (mu). It would appear to indicate that there is no long-run mean-variance for this secruity–at least one estimated by statistically sound parameters.

	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

	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

Now that i have finished estimating univariate variances, I am ready to incorporate this information into a multivariate DCC model. Although I had considered asymmetric variance dynamics in the previous EGARCH estimations, I discard an aDCC model due to its insignificant coefficients and non-intuitive understanding.

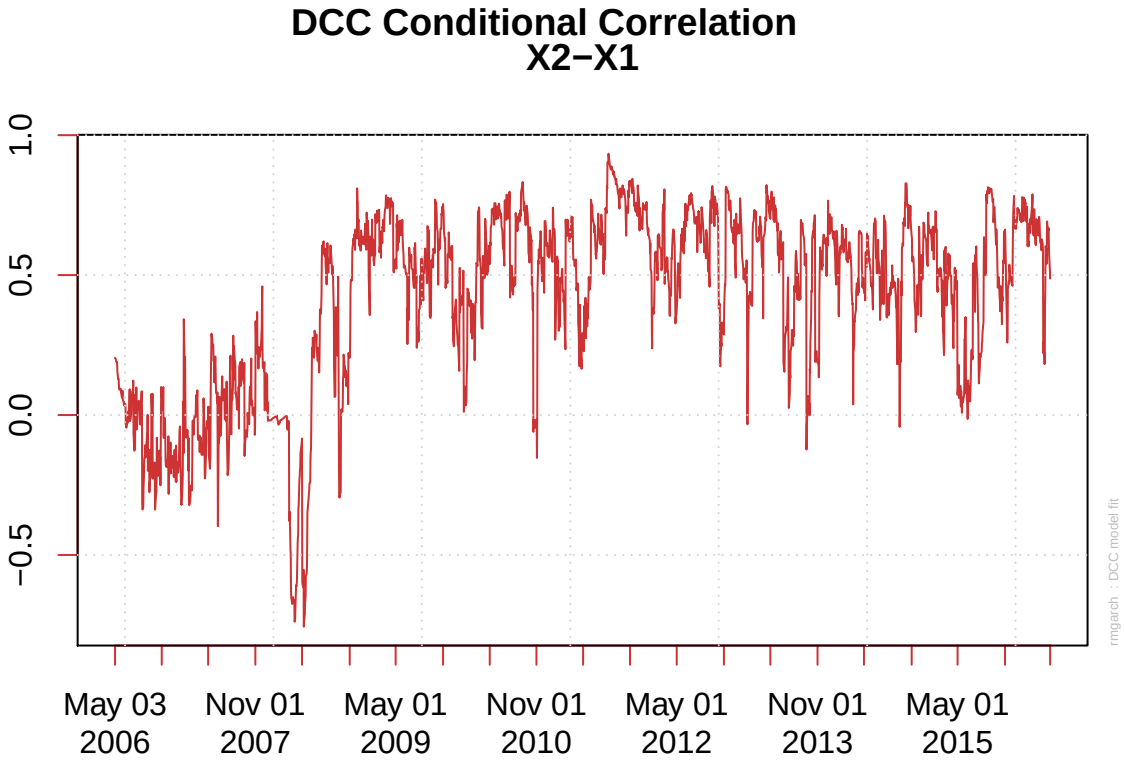
“Include Equation description here”"

Diagnostic checking here indicates that a mulitvariate DCC(1,1) model is the most suitable for this process, while higher orders models appear to have comproable BIC/AIC scores, and better fit to the skewed, non-normal data–the higher order coefficients are insignicant and I fear that I risk overfitting at this point.

After fitting the model, I construct the correlation matrix and terminate the parralel cluster.

```
#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)
```



A mulivariate DCC(1,1) indicates that most coefficients are significant, although I believe I have spurious estimations for the X1/X2 Beta1 coefficient. Individaul secruity coefficients appear to be weaker (X2 variables) than the coefficients fit for the S&P 500.

	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

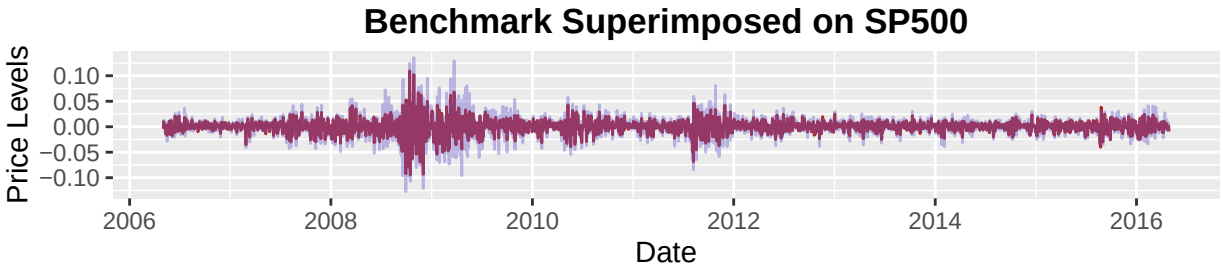
	Estimate	Std. Error	t value	Pr(>|t|)
[Joint]dccb1	0.8813116	0.0311983	28.2486921	0.0000000

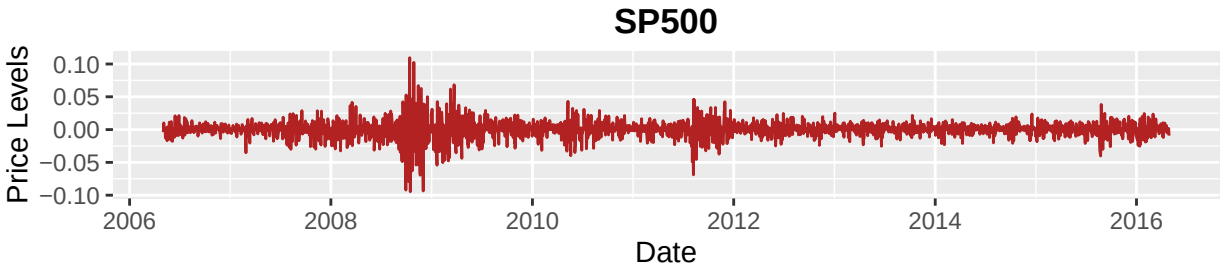
Talka bouthuge matrix of correlationsh now and speak about finding top 10 biggest

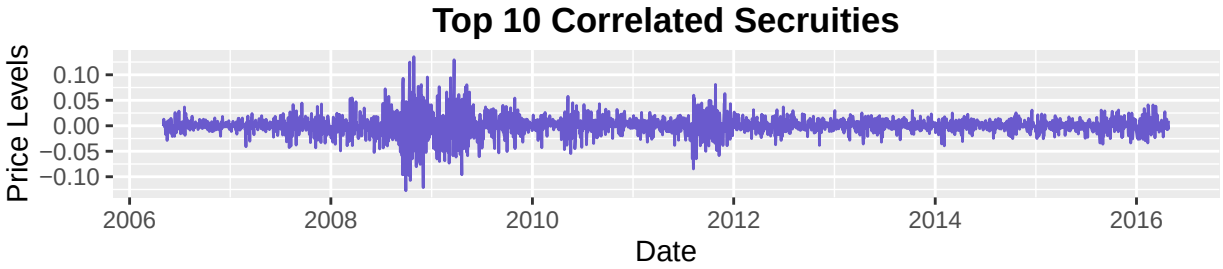
```
top10names <- apply(timevarcor[,-1], MARGIN=1, FUN=function(x)
                    names(head(sort(x, decreasing = decreasing=TRUE),10)))

#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
```

Talk about errors and include big graph

```
## Warning: Removed 2 rows containing missing values (geom_path).

## Warning: Removed 2 rows containing missing values (geom_path).
```







talk about getting the top 10 correlations per day

Conclusion

Talk about naive weighting, introduce variable weighting, better model estimatoin, automatiion of GARCH selection (1,1) assumption, higher order DCC model?

Appendix

```
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'
#'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","4OF..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
}
```

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

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

# 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

#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
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
    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){

raw[1,n] <- ifelse(is.na(raw[1,n])==TRUE, na.locf(raw[,n])[1], raw[1,n]) #place value in first
raw[end,n] <- ifelse(is.na(raw[end,n])==TRUE, na.locf(raw[,n])[end], raw[end,n]) #place value in last
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=''))

}
```