Attachment

1. References

[1] Ruppert, D. (2011). Statistics and data analysis for financial engineering (Vol. 13). New York: Springer.

[2] Semenov, M., & Smagulov, D. (2017). Portfolio risk assessment using copula models. arXiv preprint arXiv:1707.03516.

[3] Kole, E., Koedijk, K., & Verbeek, M. (2007). Selecting copulas for risk management. Journal of Banking & Finance, 31(8), 2405-2423.

[4] Sklar, A. (1959). Fonctions dé repartition à n dimension et leurs marges. Université Paris, 8(3.2), 1-3.

[5] Brockwell, P. J., Davis, R. A., & Calder, M. V. (2002). Introduction to time series and forecasting (Vol. 2). New York: springer.

[6] Grégoire, V., Genest, C., & Gendron, M. (2008). Using copulas to model price dependence in energy markets. Energy risk, 5(5), 58-64.

2. A detailed description of Historical and Monte-Carlo risk calculation

- Nonparametric Approach

In nonparametric approach, we calculate VaR without make any assumptions on the distribution of assets or portfolio's returns. Historical method of calculating Value-at-Risk is the most straightforward calculation method that we calculate Value-at-Risk directly from past returns. It treated all historical observations equally and re-organize actual historical returns, putting them in order from worst to best. For example, 95% VaR can be calculated by read the 5% off the empirical distribution from the historical returns and 95% CVaR can be calculated by calculating the average of loss exceeds 95% VaR.

One of the drawback of historical method is that it is easy to compute but need large number of historical and is feasible for large α, but not for small α. Another disadvantage of it is that it assumes the returns are independent and no unconditional covariance structure.

- Parametric Approach - Monte Carlo VaR

Monte Carlo VaR is fitting the model to risk factors and calculate VaR by simulation. It uses a computer program to generate a series of random numbers to predict scenarios (or market conditions). Based on these predicted values we can see what would happen to the assets within a portfolio under certain conditions. The value of the portfolio being assessed is then calculated for each set of market conditions generated.

We assume each underlying stocks follows Geometric Brownian Motion (GBM):

$$dS_t = rS_t dt + \sigma S_t dW_t$$

$$S_t = S_0 \exp\left(\left(r - \sigma^2/2\right)t + \sigma W_t\right)$$

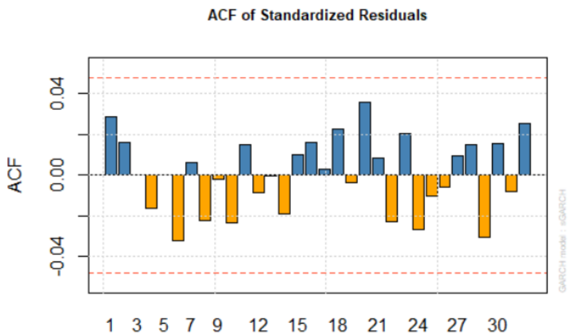
In general, we generate Monte-Carlo samples in which each sample consists of N correlated stock prices driven by N correlated Brownian motions. The approach we used is as follows:

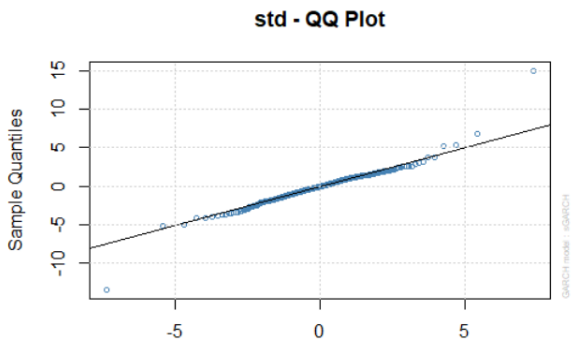
1. Estimate the portfolio's current value and construct the empirical covariance matrix D using the historical data from stocks.
2. Create the Cholesky decomposition of the covariance matrix which $D = AA^T$
3. Generate a vector of n normal variables with zero mean and unit variance $X = < x_1, \ldots, x_n >^T$.
4. Multiply the matrix resulting from the Cholesky decomposition with the vector of standard normal variables to get a vector of multivariate normal samples with zero mean and covariance matrix D. $X' = A^T X$
5. Use Geometric Brownian Motion formula to calculate the stock price at time T
6. Re-evaluate the portfolio's value at time T and get the portfolio return
7. Sort the returns from lowest value to highest value, read (1-VaRp) percentile off scenario to get VaR and calculate the mean of the first (1-ESp)th values to get Expected Shortfall.

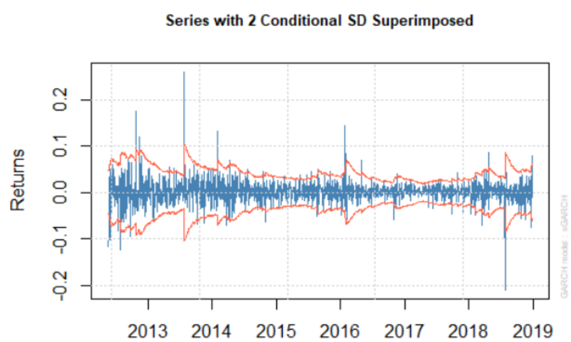
Monte Carlo simulation also has some weaknesses. First of all, it requires users to make assumptions about the stochastic process. This is subject to model risk. Besides, it also involves large amount of computations compared to historical VaR.

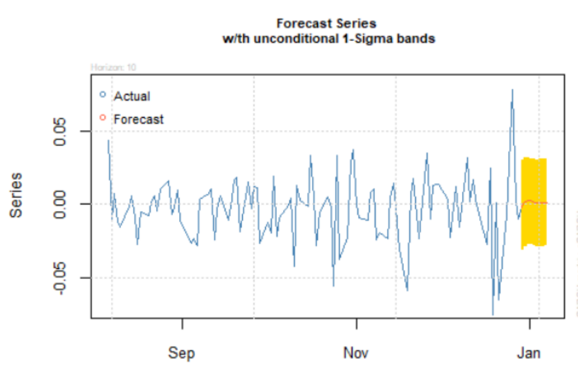
3. Important plots

Plots for FB:

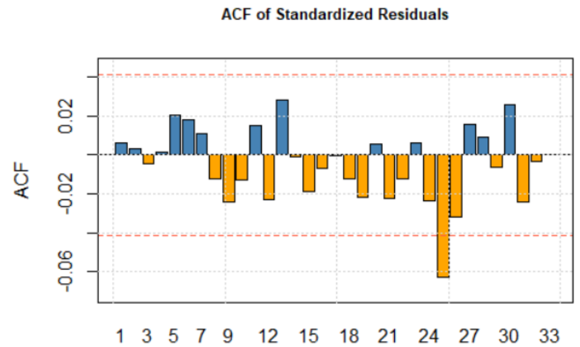


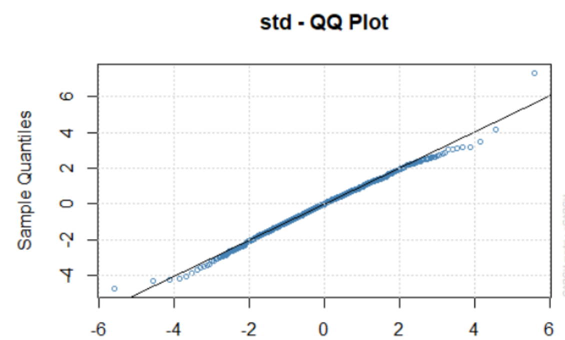


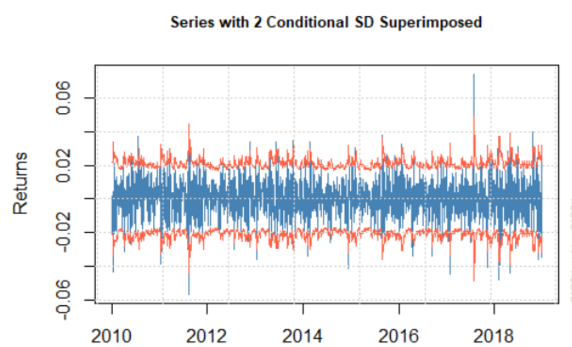


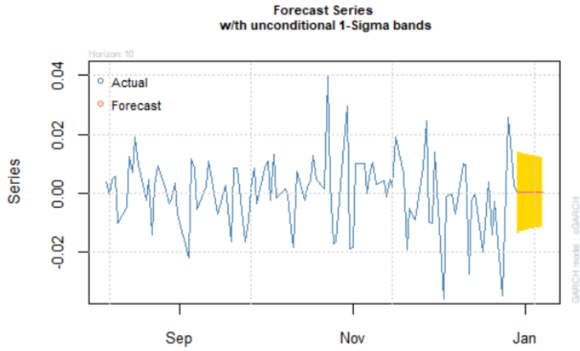


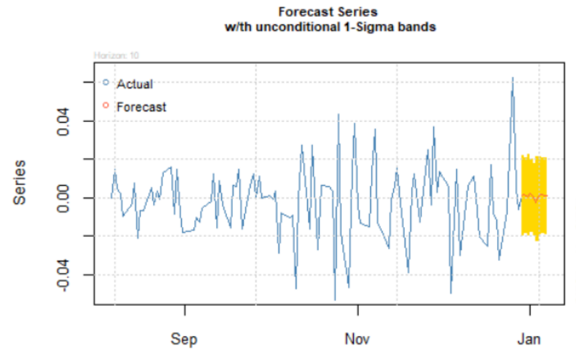
Plots for VZ:



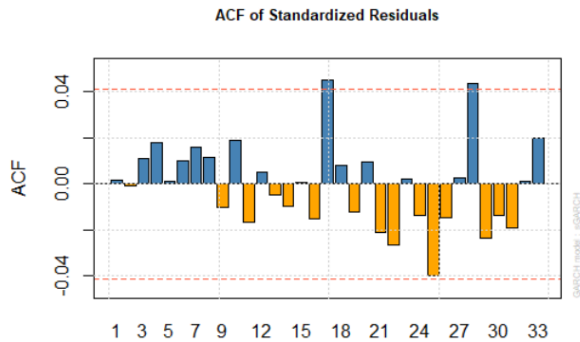


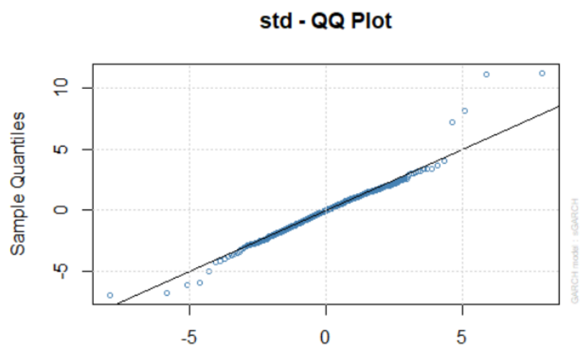


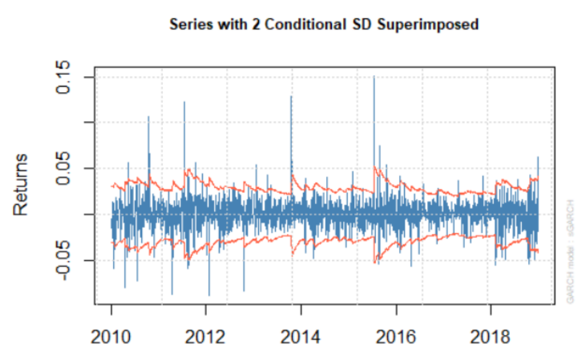




Plots for GOOGL:







5261prpject

05/08/2019

```
################################ Hist & MC VaR Calculating File ################################
# This file is designed for calculating VaR and ES
# This part is contributed by Xiaoyun Qin
###############################################################################################
######### functions for historical & MC calculation ####################
stock1 <- function (data,investment,date) {
  # calculat stock's position
  stockposition <- investment/data[Date == date,"Adj Close"]
  # Caculate stocks' return seperately
  n <- nrow(data)
  stockrtn <- as.numeric(unlist(data[-n,"Adj Close"]))/as.numeric(unlist(data[-1,"Adj Close"]))
  stocklogrtn <- log(stockrtn)
  stock <- data[Date != min(Date) ,.(Date
                                     ,price = get("Adj Close")
                                     , position = stockposition)]
  return(stock)
}

StockValue1<-function(StockData){
  num.stock<-(ncol(dat)-1)/2
  stock_value<-matrix(NA,nrow(dat),num.stock)
  for(index in 1:num.stock){
    stock_value[,index] <- dat[,2*index]*dat[,(2*index+1)]
  }
  StockData$Stock_Value<-rowSums(stock_value)

  return(StockData)
}

# Calculate the Portfolio Return Vector
returnVector<-function(data,t){
  len<-nrow(data)-t
  vector<-rep(NA,len)
  for(i in 1:len){
    vector[i]<-log(data$Stock_Value[i]/data$Stock_Value[i+t])
  }
  return(vector)
}

# Calculate the historical VaR on a single date
historical_VaR<-function(data,return.vector,current_date,p=0.99,S0){

  # data: A dataframe contains the Date, Portfolio_Value, log_return and square_log_return
  # return.vector: A vector contains the log returns over a given time horizon
  # current_date: The specific date when the Historical VaR is calculated
  # p: Confidence Level, Default = 99%
  # S0: Assume a fixed position in the portfolio
```

```r
  cutoff<-quantile(return.vector,1-p,na.rm=T)

  ans<-S0-S0*exp(cutoff)

  return(ans)
}
# Calculate the historical ES on a single date
historical_ES<-function(data,return.vector,current_date,p=0.99,S0)
{

  # data: A dataframe contains the Date, Portfolio_Value, log_return and square_log_return
  # return.vector: A vector contains the log returns over a given time horizon
  # current_date: The specific date when the Historical ES is calculated
  # p: Confidence Level, Default = 97.5%
  # S0: Assume a fixed position in the portfolio

  v<-quantile(return.vector,1-p,na.rm=T)
  cutoff<-mean(return.vector[return.vector<=v])
  ans<-S0-S0*exp(cutoff)

  return(ans)
}

MC <- function(investment,current_date,
               stock, stockposition, rtn,
               p.VaR , p.ES , t , l , npaths ) {

  # Inverstment: Assume a fixed position in the portfolio, Default = 10000
  # reducepct: The percentage of Inverstment invest in options
  # current_date: The specific date when the Monte Carlo VaR is calculated
  # stock: A dataframe contains the stocks' prices
  # stockposition: A vectory contains the position of stocks
  # logrtn: A dataframe contains the logreturn of stocks
  # iv: Implied volitility of options
  # mat: Time to maturity
  # Type: Put or Call
  # r: risk free rate
  # p.VaR: Confidence Level, Default = 99%
  # p.ES: Confidence level, Default = 97%
  # t: VaR Time Horizon, Default = 5 days (5/252 years)
  # l: Window Length, Default = 5 years
  # npaths: The number of generated random values using Monte Carlo methods, Deafult = 10000

  logrtn <- log(rtn)
  dt <- t

  # portfolio contains multiple stocks

  # Initialize matrix
  covriance <- matrix(nrow = ncol(rtn), ncol = ncol(rtn))
  simga <- matrix(nrow = ncol(rtn), ncol = ncol(rtn))
  rho <- matrix(nrow = ncol(rtn), ncol = ncol(rtn))
```

```r
  # get the volitility and drift
  vol <- unlist(winEst2(rtn, current_date, l)[2])
  mu <- unlist(winEst2(rtn, current_date, l)[1])

  # Caculate stocks' covariance
  endtemp <- min(length(rtn[,1]),l*252+current_date)
  covmatrix <- cov(logrtn[current_date:endtemp,])

  # caculate correlated matrix
  volB <-  matrix(rep(vol,length(vol)),nrow = length(vol), byrow = TRUE)
  volA <- matrix(rep(vol,length(vol)), nrow = length(vol))
  volmatrix <- volA*volB
  rho <- 252*(covmatrix/volmatrix)
  diag(rho) <- rep(1,nrow(rho))

  # generate random numbers
  randomsample<-mvrnorm(npaths,rep(0,ncol(stock)),rho)

  # caculate the value of portfolio
  volm <- matrix(rep(vol,npaths),ncol = ncol(stock), byrow = TRUE)
  mum <- matrix(rep(mu,npaths),ncol = ncol(stock), byrow = TRUE)

  S0 <- sum(stock[current_date,]*stockposition)
  portfoliotemp <- (exp(randomsample*volm*sqrt(dt)+ (mum - volm^2/2)*dt)) * matrix(rep(stock[current_da
  St <- rowSums(portfoliotemp)


  stockshare <- investment/S0
  Vt.stock <- stockshare*St
  V0.stock <- S0*stockshare


  # portfoio does not have option
  V0 <- V0.stock
  portfolio <- Vt.stock
  portsort <- sort(portfolio,decreasing = F)

  # long Portfolio
  VaR <- V0-portsort[ceiling((1-p.VaR)*npaths)]
  ES <- V0-mean(portsort[1:ceiling((1-p.ES)*npaths)])

  # get results
  return(c(VaR, ES))
}
winEst2<-function(rtn, current_date, l){

  # data: A dataframe contains the log_return
  # current_date: The specific date when the drift and volatility terms are calculated
  # l: Window Length, Default = 1 years

  endtemp <- min(length(rtn[,1]),l*252+current_date)
  logrtn <- log(rtn)
  logrtn2 <- logrtn*logrtn
```

```
  # Initial matrix
  mean_log_return <- matrix(NA,nrow = 1, ncol = ncol(logrtn))
  mean_square_log_return <- matrix(NA,nrow = 1, ncol = ncol(logrtn))
  sd_log_return <- matrix(NA,nrow = 1, ncol = ncol(logrtn))
  volatility <- matrix(NA,nrow = 1, ncol = ncol(logrtn))
  drift <- matrix(NA,nrow = 1, ncol = ncol(logrtn))

  mean_log_return[1,]<-unlist(apply(logrtn[current_date:endtemp,], 2, mean))
  mean_square_log_return[1,]<-unlist(apply(logrtn2[current_date:endtemp,], 2, mean))
  sd_log_return[1,]<-sqrt(mean_square_log_return-(mean_log_return)^2)

  volatility[1,]<-sd_log_return*sqrt(252)
  drift[1,]<-252*mean_log_return+volatility^2/2

  return(list(drift,volatility))
}
blackScholes<-function(s0,x,sigma,r,t,q){
  d1=(log(s0/x)+t*(r-q+sigma^2/2))/(sigma*sqrt(t))
  d2=d1-sigma*sqrt(t)
  call=s0*exp(-q*t)*pnorm(d1)-x*exp(-r*t)*pnorm(d2)
  put=x*exp(-r*t)*pnorm(-d2)-s0*exp(-q*t)*pnorm(-d1)
  results<-cbind(call,put)
  return(results)
}

stockm <- function (data) {
  # Identify how may stocks we input and the length of the data
  stocksnum <- (ncol(data)-1)/2
  n <- nrow(data)

  # Seperate Stocks prices and their positions
  stock <- matrix(nrow = n, ncol = stocksnum)
  stockposition <- matrix(nrow = 1, ncol = stocksnum)
  for (i in 1:stocksnum){
    temp <- 2*i
    stock[,i] <- data[,temp]
    temp <- temp+1
    stockposition[1,i] <- data[1,temp]
  }

  # Caculate stocks' return seperately
  stockrtn <- stock[-n,]/stock[-1,]
  stocklogrtn <- log(stockrtn)

  return(list(stock[-1,],stockposition,stocklogrtn,stockrtn))
}

VaR_calculate <- function(data,S0,h,p.VaR,p.ES,t,l,portfolio){
  hist.t<-t*252
  num.return<-min(252*l,nrow(portfolio))
  portfolio.return<-returnVector(portfolio,hist.t)

  # Calculate the Historical VaR and ES for the past 1 year
```

```
  VaR_length <- 252*h

  HistVaR<-rep(NA,VaR_length)
  HistES<-rep(NA,VaR_length)


  for(i in 1:VaR_length){
    HistVaR[i]<-historical_VaR(portfolio,portfolio.return[i:(i+num.return-1)],i,p.VaR,S0)
  }
  for(i in 1:VaR_length){
    HistES[i]<-historical_ES(portfolio,portfolio.return[i:(i+num.return-1)],i,p.ES,S0)
  }

  # Rearrange the Historical VaR and ES from oldest to newest
  HistVaR<-rev(HistVaR)
  HistES<-rev(HistES)

  ########## MC VaR ################################
  library(data.table)
  data_stock <- data.frame(data.table(data)[Date <= as.Date(investment_date)+365,])

  stock <- matrix(unlist(stockm(data_stock)[1]), ncol = (ncol(data_stock)-1)/2, byrow = FALSE)
  stockposition <- unlist(stockm(data_stock)[2])
  stockrtn <- matrix(unlist(stockm(data_stock)[4]), ncol = (ncol(data_stock)-1)/2, byrow = FALSE)


  library(MASS)
  MC.results <- matrix(NA,ncol = 2, nrow = VaR_length)

  # Calculate the Monte VaR and ES

  for (i in 1:VaR_length) {
    MC.results[i,] <- MC(investment = 10000,current_date=i,
                       stock=stock , stockposition=stockposition, rtn=stockrtn,
                       p.VaR , p.ES, t , l, npaths = 10000)
  }


  # Get VaR and ES
  MC.VaR <- rev(MC.results[,1])
  MC.ES <- rev(MC.results[,2])


  length <- nrow(portfolio)
  realloss <- (1-portfolio[1:(length-hist.t),10]/portfolio[(hist.t+1):length,10])*S0

  realloss<-realloss[VaR_length:1]
  names(realloss) <- c("realloss")
  data.temp <-data.frame(Date=as.Date(portfolio$Date[VaR_length:1]),HistVaR,HistES,MC.VaR,MC.ES,realloss

}
```

```
############################### Copula Fitting & VaR Calculation ##############################
# This file is designed for calculating VaR and ES
```

```
# This part, together with all forthcoming chunks, is contributed by Zhenhuan Xie
##############################################################################################


#####Function to calculate return #####

stock <- function (data,investment,date) {
  # calculat stock's position
  stockposition <- investment/data[Date == date,"Adj Close"]
  # Caculate stocks' return seperately
  n <- nrow(data)
  stockrtn <- as.numeric(unlist(data[-1,"Adj Close"]))/as.numeric(unlist(data[-n,"Adj Close"]))
  stocklogrtn <- log(stockrtn)
  stock <- data[Date != min(Date) ,.(Date
                                     ,price = get("Adj Close")
                                     ,stocklogrtn
                                     , position = stockposition)]
  return(stock)
}

StockValue<-function(StockData){
  num.stock<-(ncol(dat)-1)/3
  stock_value<-matrix(NA,nrow(dat),num.stock)
  j=2
  for(index in 1:num.stock){
    stock_value[,index] <- dat[,j]*dat[,(j+2)]
    j = j+3
  }
  StockData$Stock_Value<-rowSums(stock_value)

  return(StockData)
}
```

```
######### Historical & MC Calculation###################
library(data.table)
vz <- fread("./data/VZ.csv")
fb <- fread("./data/FB.csv")
googl <- fread("./data/GOOGL.csv")
msft <- fread("./data/MSFT.csv")
vz <- vz[order(Date,decreasing = TRUE)]
fb <- fb[order(Date,decreasing = TRUE)]
googl <- googl[order(Date,decreasing = TRUE)]
msft <- msft[order(Date,decreasing = TRUE)]
S0<-10000 # the total amount invested in stocks
investment_date <- c("2018-01-02") # date invested in stocks

#weight=wts.gmv.etl
weight=c(0.25,0.25,0.25,0.25)
#weight=wts.tan.etl

# Calculate the Portfolio Return Vector
dat <- data.frame(stock1(vz,S0*weight[1],investment_date)
                  ,stock1(fb,S0*weight[2],investment_date)[,c(2,3)]
                  ,stock1(googl,S0*weight[3],investment_date)[,c(2,3)]
```

```r
                  ,stock1(msft,S0*weight[4],investment_date)[,c(2,3)])
names(dat) <- c("Date","vz","vz position"
                ,"fb","fb position"
                ,"googl","googl position"
                ,"msft","msft position")
portfolio <- data.table(StockValue1(dat))
portfolio <- portfolio[order(Date,decreasing = TRUE)]

# Calculate VaR and CVaR
dat_VaR <- VaR_calculate(data = dat,S0,h=1,p.VaR=0.99,p.ES=0.99,t=1/252,l=1,portfolio[Date <= as.Date(in

write.csv(dat_VaR,"vardata.csv",row.names = FALSE)
```

```r
#####Data Preparation for copula method#####
library(data.table)
vz <- fread("./data/VZ.csv")
fb <- fread("./data/FB.csv")
googl <- fread("./data/GOOGL.csv")
msft <- fread("./data/MSFT.csv")
S0<-10000 # the total amount invested in stocks
investment_date <- c("2018-01-02") # date invested in stocks

#weight=wts.gmv.etl
weight=c(0.25,0.25,0.25,0.25)
#weight=wts.tan.etl

dat <- data.frame(stock(vz,S0*weight[1],investment_date)
                  ,stock(fb,S0*weight[2],investment_date)[,c(2,3,4)]
                  ,stock(googl,S0*weight[3],investment_date)[,c(2,3,4)]
                  ,stock(msft,S0*weight[4],investment_date)[,c(2,3,4)])

names(dat) <- c("Date","vz","vz_logrtn","vz position"
                ,"fb","fb_logrtn","fb position"
                ,"googl","googl_logrtn","googl position"
                ,"msft","msft_logrtn","msft position")
data <- dat[1008:1509,c(1,3,6,9,12)]
names(data) <- c("Date","VZ","FB","GOOGL","MSFT")
```

```r
#####Investigating marginal distribution#####
library(MASS)
par(mfrow=c(2,4))
normfit_FB<-fitdistr(data[252:502,]$FB,densfun = "normal")
hist(data[252:502,]$FB, pch=20, breaks=25, prob=TRUE, main="")
curve(dnorm(x, normfit_FB$estimate[1], normfit_FB$estimate[2]), col="red", lwd=2, add=T)

normfit_VZ<-fitdistr(data[252:502,]$VZ,densfun = "normal")
hist(data[252:502,]$VZ, pch=20, breaks=25, prob=TRUE, main="")
curve(dnorm(x, normfit_VZ$estimate[1], normfit_VZ$estimate[2]), col="red", lwd=2, add=T)

normfit_GOOGL<-fitdistr(data[252:502,]$GOOGL,densfun = "normal")
hist(data[252:502,]$GOOGL, pch=20, breaks=25, prob=TRUE, main="")
curve(dnorm(x, normfit_GOOGL$estimate[1], normfit_GOOGL$estimate[2]), col="red", lwd=2, add=T)

normfit_MSFT<-fitdistr(data[252:502,]$MSFT,densfun = "normal")
```

```r
hist(data[252:502,]$MSFT, pch=20, breaks=25, prob=TRUE, main="")
curve(dnorm(x, normfit_MSFT$estimate[1], normfit_MSFT$estimate[2]), col="red", lwd=2, add=T)

library(QRM) #for fit.st function
library(metRology) #for scaled t distribution
library(goftest) #for ad.test, cvm.test function

tfit_FB<-fit.st(data$FB)
hist(data[252:502,]$FB, pch=20, breaks=25, prob=TRUE, main="")
curve(dt.scaled(x, tfit_FB$par.ests[1],tfit_FB$par.ests[2],tfit_FB$par.ests[3]),col="red", lwd=2,add=T)
ks.test(data[252:502,]$FB,"pt.scaled",df=tfit_FB$par.ests[1],mean=tfit_FB$par.ests[2],sd=tfit_FB$par.es
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  data[252:502, ]$FB
## D = 0.098505, p-value = 0.01533
## alternative hypothesis: two-sided
```

```r
ad.test(data[252:502,]$FB,null="pt.scaled",df=tfit_FB$par.ests[1],mean=tfit_FB$par.ests[2],sd=tfit_FB$pa
```

```
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.70478345538432, mean =
##  0.00151495524235622, sd = 0.0102700599282677
##
## data:  data[252:502, ]$FB
## An = 5.531, p-value = 0.001608
```

```r
cvm.test(data[252:502,]$FB,null="pt.scaled",df=tfit_FB$par.ests[1],mean=tfit_FB$par.ests[2],sd=tfit_FB$p
```

```
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.70478345538432, mean =
##  0.00151495524235622, sd = 0.0102700599282677
##
## data:  data[252:502, ]$FB
## omega2 = 0.68795, p-value = 0.01356
```

```r
tfit_VZ<-fit.st(data$VZ)
hist(data[252:502,]$VZ, pch=20, breaks=25, prob=TRUE, main="")
curve(dt.scaled(x, tfit_VZ$par.ests[1],tfit_VZ$par.ests[2],tfit_VZ$par.ests[3]),col="red", lwd=2,add=T)
ks.test(data[252:502,]$VZ,"pt.scaled",df=tfit_VZ$par.ests[1],mean=tfit_VZ$par.ests[2],sd=tfit_VZ$par.es
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  data[252:502, ]$VZ
## D = 0.062764, p-value = 0.2761
## alternative hypothesis: two-sided
```

```r
ad.test(data[252:502,]$VZ,"pt.scaled",df=tfit_VZ$par.ests[1],mean=tfit_VZ$par.ests[2],sd=tfit_VZ$par.es
```

```
##
##  Anderson-Darling test of goodness-of-fit
```

```
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 4.06365537342911, mean =
##  0.000428542539831312, sd = 0.00852920184136147
##
## data:  data[252:502, ]$VZ
## An = 1.4859, p-value = 0.1799
```

```
cvm.test(data[252:502,]$VZ,"pt.scaled",df=tfit_VZ$par.ests[1],mean=tfit_VZ$par.ests[2],sd=tfit_VZ$par.e
```

```
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 4.06365537342911, mean =
##  0.000428542539831312, sd = 0.00852920184136147
##
## data:  data[252:502, ]$VZ
## omega2 = 0.22795, p-value = 0.2193
```

```
tfit_GOOGL<-fit.st(data$GOOGL)
hist(data[252:502,]$GOOGL, pch=20, breaks=25, prob=TRUE, main="")
curve(dt.scaled(x, tfit_GOOGL$par.ests[1],tfit_GOOGL$par.ests[2],tfit_GOOGL$par.ests[3]),col="red", lwd
ks.test(data[252:502,]$GOOGL,"pt.scaled",df=tfit_GOOGL$par.ests[1],mean=tfit_GOOGL$par.ests[2],sd=tfit_
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  data[252:502, ]$GOOGL
## D = 0.098433, p-value = 0.01544
## alternative hypothesis: two-sided
```

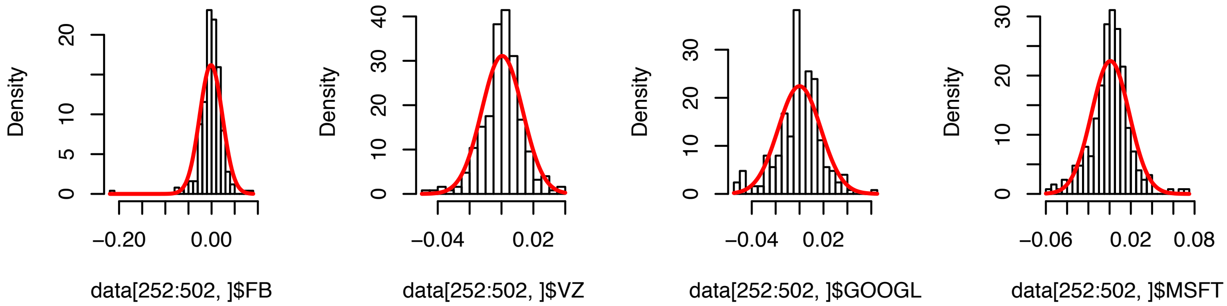
```
ad.test(data[252:502,]$GOOGL,"pt.scaled",df=tfit_GOOGL$par.ests[1],mean=tfit_GOOGL$par.ests[2],sd=tfit_
```

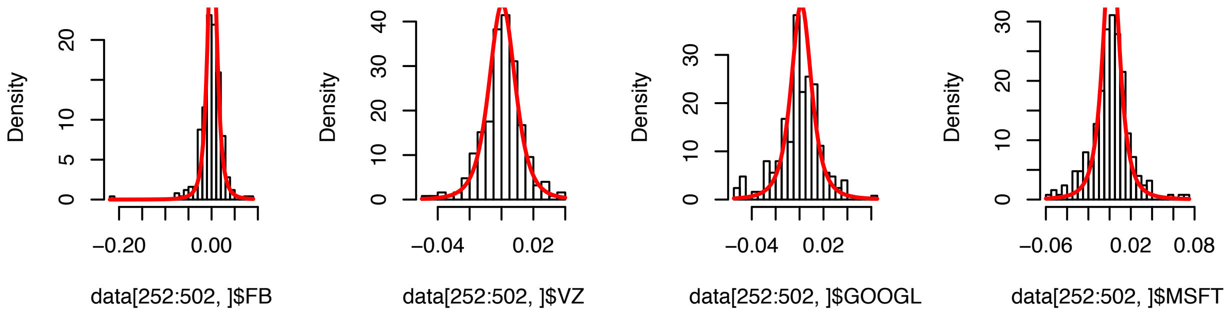
```
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.80190092080924, mean =
##  0.00143463008865657, sd = 0.00900786211232015
##
## data:  data[252:502, ]$GOOGL
## An = 5.0682, p-value = 0.002666
```

```
cvm.test(data[252:502,]$GOOGL,"pt.scaled",df=tfit_GOOGL$par.ests[1],mean=tfit_GOOGL$par.ests[2],sd=tfit
```

```
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.80190092080924, mean =
##  0.00143463008865657, sd = 0.00900786211232015
##
## data:  data[252:502, ]$GOOGL
## omega2 = 0.58116, p-value = 0.02484
```

```
tfit_MSFT<-fit.st(data$MSFT)
hist(data[252:502,]$MSFT, pch=20, breaks=25, prob=TRUE, main="")
curve(dt.scaled(x, tfit_MSFT$par.ests[1],tfit_MSFT$par.ests[2],tfit_MSFT$par.ests[3]),col="red", lwd=2,
```






```r
ks.test(data[252:502,]$MSFT,"pt.scaled",df=tfit_MSFT$par.ests[1],mean=tfit_MSFT$par.ests[2],sd=tfit_MSF
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  data[252:502, ]$MSFT
## D = 0.085032, p-value = 0.05305
## alternative hypothesis: two-sided
```

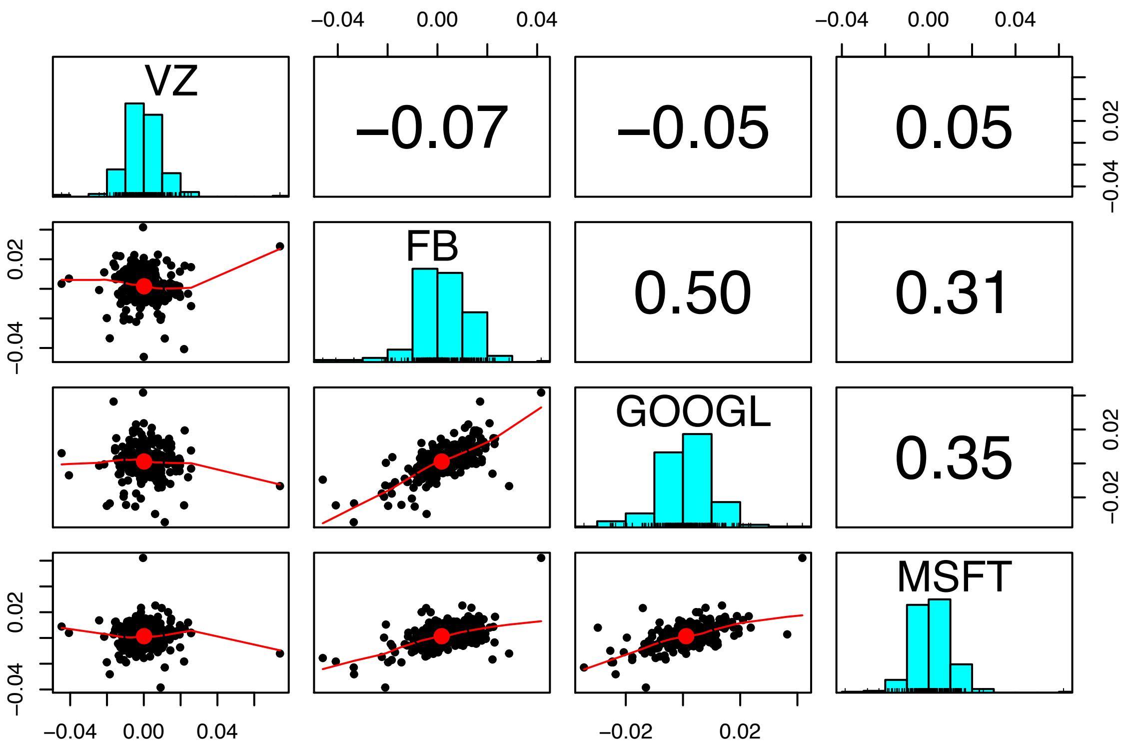
```r
ad.test(data[252:502,]$MSFT,"pt.scaled",df=tfit_MSFT$par.ests[1],mean=tfit_MSFT$par.ests[2],sd=tfit_MSF
```

```
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.54193468955741, mean =
##  0.00150798395887005, sd = 0.00823376575345946
##
## data:  data[252:502, ]$MSFT
## An = 4.9128, p-value = 0.003163
```

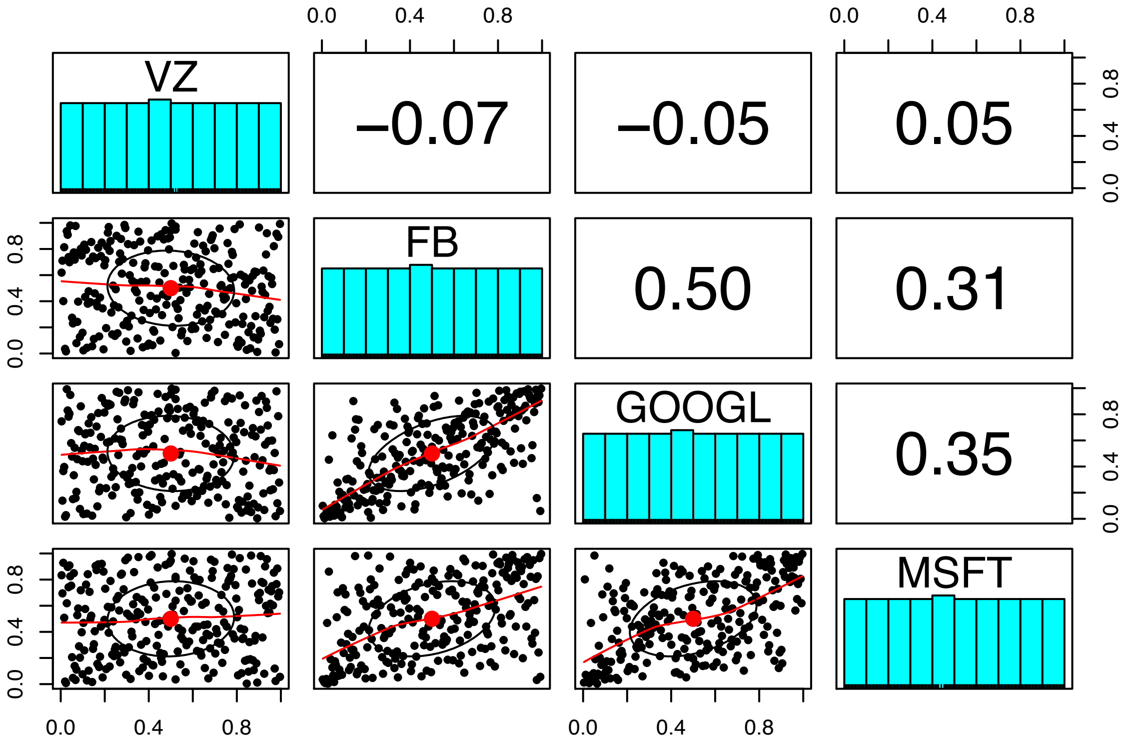
```r
cvm.test(data[252:502,]$MSFT,"pt.scaled",df=tfit_MSFT$par.ests[1],mean=tfit_MSFT$par.ests[2],sd=tfit_MS
```

```
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: distribution 'pt.scaled'
##  with parameters df = 2.54193468955741, mean =
##  0.00150798395887005, sd = 0.00823376575345946
##
## data:  data[252:502, ]$MSFT
## omega2 = 0.5704, p-value = 0.02642
```

```r
#####Fitting copula to data#####
library(copula)
t=length(data[1:251,]$FB)
data_pseudo=cbind(rank(data[1:251,]$VZ)/(t+1),rank(data[1:251,]$FB)/(t+1),rank(data[1:251,]$GOOGL)/(t+1)
data_pseudo=as.data.frame(data_pseudo)
names(data_pseudo)=c("VZ","FB","GOOGL","MSFT")
#using "pobs" function will give the same result

library(psych)#for the pairs.panels function
pairs.panels(data[1:251,2:5],method = "kendall",density=F)#original data overview
```




```r
pairs.panels(data_pseudo,method="kendall",density=F)#pseudo observation overview
```




```
cor_tau=cor(data_pseudo,method="kendall")
tcop=tCopula(dim=4,dispstr="un",df=5,df.fixed=TRUE)
#tcop=tCopula(dim=4,dispstr="un")
ft1=fitCopula(tcop,data_pseudo,method="mpl",start=c(cor_tau[1,2],cor_tau[1,3],cor_tau[1,4],cor_tau[2,3]
tcop_coef=summary(ft1)$coefficients
tcop_est=tCopula(dim=4,param=tcop_coef[1:6,1],dispstr="un",df=5,df.fixed = TRUE)
gofCopula(tcop_est,data_pseudo) #This line takes several minutes.
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of t-copula, dim.
##  d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.03107, parameter.rho.1 = -0.097430, parameter.rho.2
## = -0.073258, parameter.rho.3 = 0.068216, parameter.rho.4 =
## 0.726390, parameter.rho.5 = 0.502890, parameter.rho.6 = 0.567860,
## p-value = 0.2812
```

```
ft2=fitCopula(copula = claytonCopula(dim=4),data=data_pseudo,method="mpl",start=1)
claytoncop_est=claytonCopula(param=summary(ft2)$coefficients[1],dim=4)
gofCopula(claytoncop_est,data_pseudo)
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of Clayton
##  copula, dim. d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.14297, parameter = 0.39588, p-value = 0.0004995
```

```
ft3=fitCopula(copula = frankCopula(dim=4),data=data_pseudo,method="mpl",start=1)
frankcop_est=frankCopula(param=summary(ft3)$coefficients[1],dim=4)
gofCopula(frankcop_est,data_pseudo)
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of Frank copula,
##  dim. d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.18157, parameter = 1.4351, p-value = 0.0004995
```

```
ft4=fitCopula(copula = gumbelCopula(dim=4),data=data_pseudo,method="mpl",start=1)
gumbelcop_est=gumbelCopula(param=summary(ft4)$coefficients[1],dim=4)
gofCopula(gumbelcop_est,data_pseudo)
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of Gumbel copula,
##  dim. d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.21535, parameter = 1.1799, p-value = 0.0004995
```

```
ft5=fitCopula(copula = joeCopula(dim=4),data=data_pseudo,method="mpl",start=1)
joecop_est=joeCopula(param=summary(ft5)$coefficients[1],dim=4)
gofCopula(joecop_est,data_pseudo)
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of Joe copula,
##  dim. d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.37422, parameter = 1.1881, p-value = 0.0004995
```

```
ft0=fitCopula(copula=normalCopula(dim=4,dispstr="un"),data=data_pseudo,method="mpl",start=c(cor_tau[1,2
normalcop_coef=summary(ft0)$coefficients
normalcop_est=normalCopula(dim=4,param=normalcop_coef[1:6,1],dispstr="un")
gofCopula(normalcop_est,data_pseudo)
```

```
##
##  Parametric bootstrap-based goodness-of-fit test of Normal copula,
##  dim. d = 4, with 'method'="Sn", 'estim.method'="mpl":
##
## data:  x
## statistic = 0.033785, parameter.rho.1 = -0.072647, parameter.rho.2
## = -0.070267, parameter.rho.3 = 0.061110, parameter.rho.4 =
## 0.702250, parameter.rho.5 = 0.487760, parameter.rho.6 = 0.567470,
## p-value = 0.1653
```

```
#####According to GoF test result, use t copula to calculate VaR and CVaR.#####

sample_num=1000
sample=rCopula(n=sample_num,copula=tcop_est)
head(sample)
```

```
##             [,1]      [,2]       [,3]      [,4]
## [1,] 0.08636557 0.1450042 0.05668753 0.0763165
## [2,] 0.60238868 0.7568719 0.42941986 0.5920689
## [3,] 0.36934467 0.8465138 0.83273357 0.8300995
## [4,] 0.69350935 0.9651634 0.93461296 0.9354833
```

```
## [5,] 0.91780383 0.8342806 0.42723728 0.4713547
## [6,] 0.46287058 0.3857469 0.58934791 0.3188136
```

```
sample_rearrange=c(sample[,1],sample[,2],sample[,3],sample[,4])
s=cbind(as.numeric(quantile(data[1:251,]$FB,sample_rearrange)),as.numeric(quantile(data[1:251,]$VZ,sampl

weight=c(0.25,0.25,0.25,0.25)#equal weighted
#weight=wts.gmv.etl
#weight=wts.tan.etl

pnl=s %*% as.matrix(weight)

alpha=c(0.01,0.05,0.1)
var_single=as.numeric(quantile(pnl,alpha))
cvar_single=c(mean(pnl[pnl<var_single[1]]),mean(pnl[pnl<var_single[2]]),mean(pnl[pnl<var_single[3]]))

#Bootstrap
B=1000
boot_var_matrix=matrix(NA,nrow=B,ncol=3)
boot_cvar_matrix=matrix(NA,nrow=B,ncol=3)
weight=c(0.25,0.25,0.25,0.25) #equal weighted
#weight=wts.gmv.etl
#weight=wts.tan.etl

for (b in 1:B){
  boot_index=sample(sample_num*4,sample_num*4,replace=T)
  boot_sample=sample_rearrange[boot_index]
  boot_s=cbind(as.numeric(quantile(data$FB,boot_sample)),as.numeric(quantile(data$VZ,boot_sample)),as.nu
  boot_pnl=boot_s %*% as.matrix(weight)

  boot_var_matrix[b,]=quantile(boot_pnl,alpha)
  boot_cvar_matrix[b,1]=mean(boot_pnl[boot_pnl<boot_var_matrix[b,1]])
  boot_cvar_matrix[b,2]=mean(boot_pnl[boot_pnl<boot_var_matrix[b,2]])
  boot_cvar_matrix[b,3]=mean(boot_pnl[boot_pnl<boot_var_matrix[b,3]])
}
#boot_var=c(mean(boot_var_matrix[,1]),mean(boot_var_matrix[,2]),mean(boot_var_matrix[,3]))
boot_var_sd=c(sd(boot_var_matrix[,1]),sd(boot_var_matrix[,2]),sd(boot_var_matrix[,3]))
#boot_cvar=c(mean(boot_cvar_matrix[,1]),mean(boot_cvar_matrix[,2]),mean(boot_cvar_matrix[,3]))
boot_cvar_sd=c(sd(boot_cvar_matrix[,1]),sd(boot_cvar_matrix[,2]),sd(boot_cvar_matrix[,3]))
output=data.frame((1-alpha)*100,var_single*100,boot_var_sd*100,cvar_single*100,boot_cvar_sd*100)
colnames(output)=c("level(%)","VaR(%)","VaR SD(%)","CVaR(%)","CVaR SD(%)")
output
```

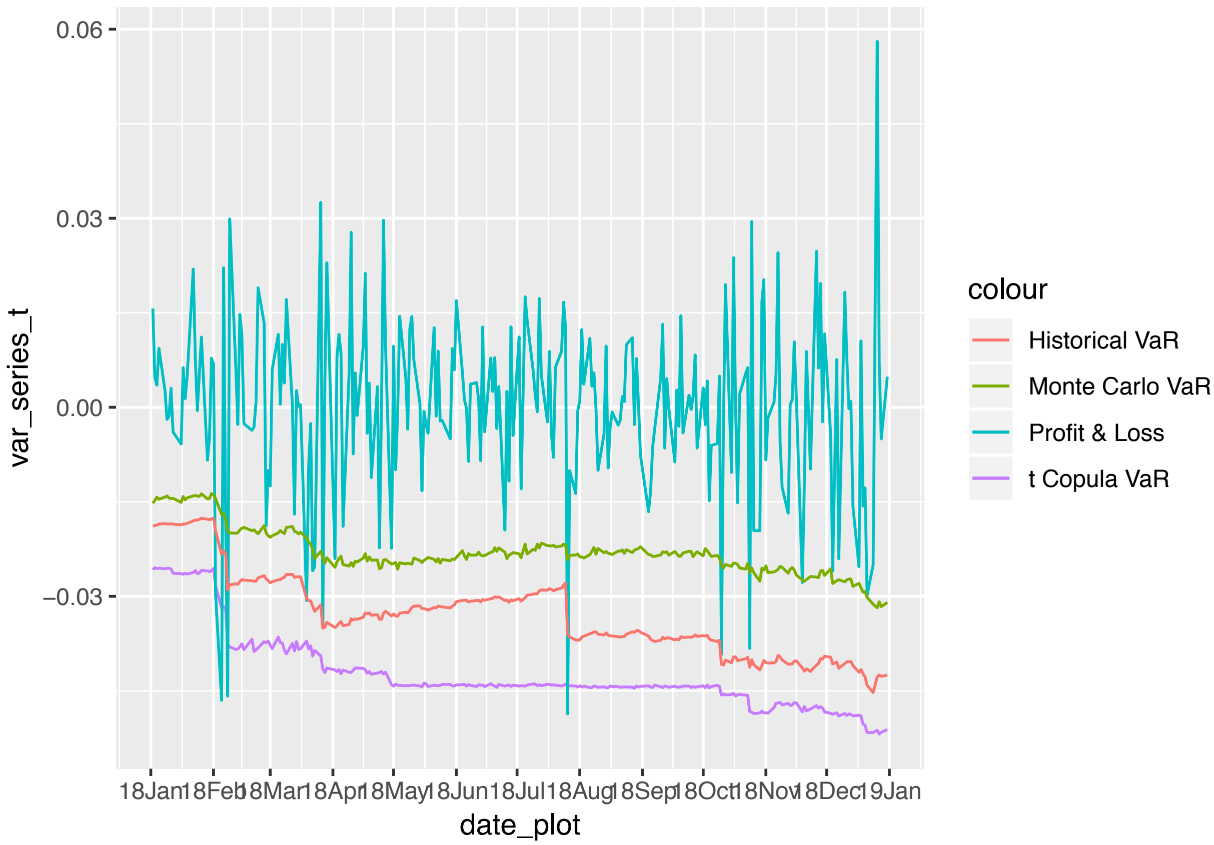
```
##   level(%)     VaR(%)  VaR SD(%)   CVaR(%) CVaR SD(%)
## 1       99 -2.5026338 0.17731567 -3.170965 0.26592487
## 2       95 -1.2734415 0.06062028 -2.047750 0.12603198
## 3       90 -0.8819875 0.05084920 -1.548653 0.08498814
```

```
fittcop=function(data,weight){
  #weight should be a 4 dimensional vector.
  t=nrow(data)
  data_pseudo=as.data.frame(cbind(rank(data$FB)/(t+1),rank(data$VZ)/(t+1),rank(data$GOOGL)/(t+1),rank(da
  names(data_pseudo)=c("FB","VZ","GOOGL","MSFT")

  cor_tau=cor(data_pseudo,method="kendall") #will be used as starting value in fitCopula function
```

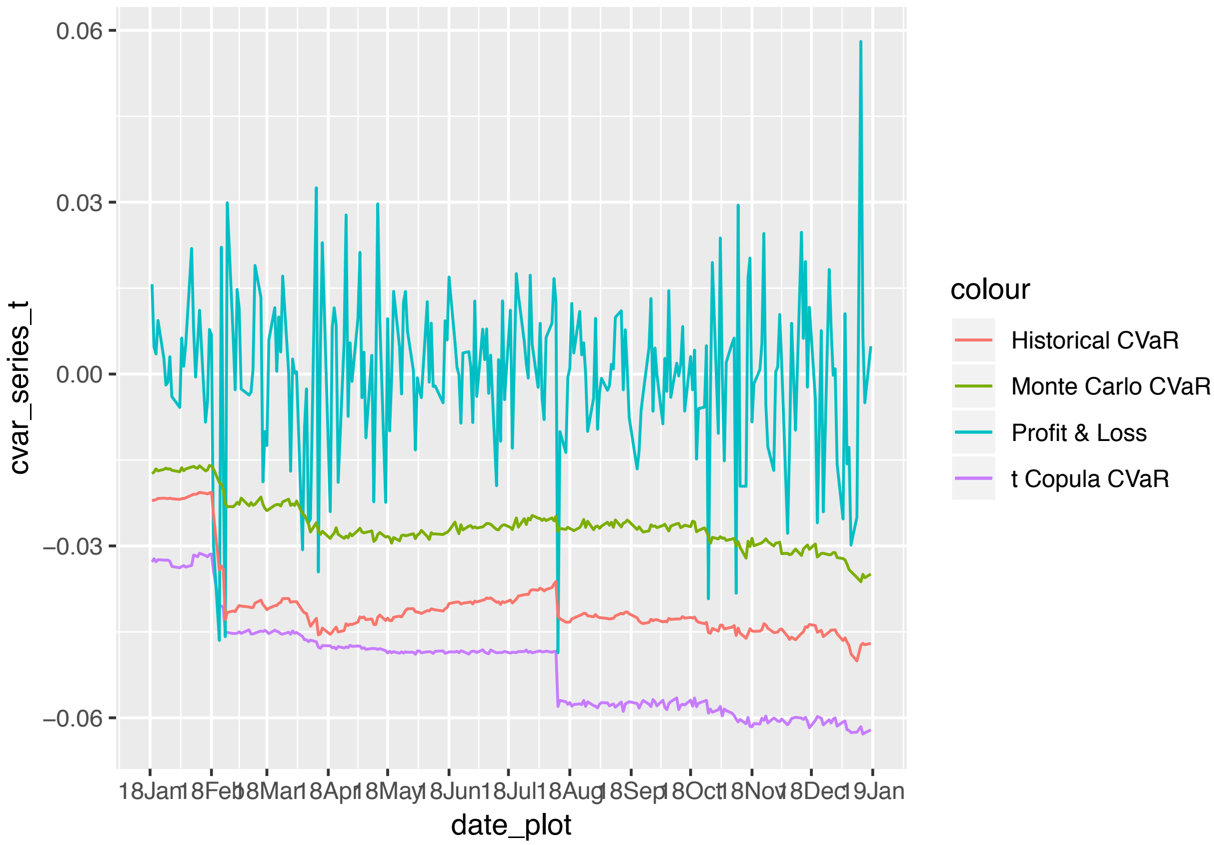
```
  tcop=tCopula(dim=4,dispstr="un")
  ft1=fitCopula(tcop,data_pseudo,method="mpl",start=c(cor_tau[1,2],cor_tau[1,3],cor_tau[1,4],cor_tau[2,
  tcop_coef=summary(ft1)$coefficients
  tcop_est=tCopula(dim=4,param=tcop_coef[1:6,1],df=tcop_coef[7,1],dispstr="un")

  sample_num=50000
  sample=rCopula(n=sample_num,copula=tcop_est)
  sample_rearrange=c(sample[,1],sample[,2],sample[,3],sample[,4])
  s=cbind(as.numeric(quantile(data$FB,sample_rearrange)),as.numeric(quantile(data$VZ,sample_rearrange))
  pnl=s %*% as.matrix(weight)

  alpha=c(0.01,0.05,0.1)
  var=as.numeric(quantile(pnl,alpha))
  cvar=c(mean(pnl[pnl<var[1]]),mean(pnl[pnl<var[2]]),mean(pnl[pnl<var[3]]))
  output=cbind(var,cvar)
  return(output)
}

fitnormalcop=function(data){
  t=length(data$FB)
  data_pseudo=as.data.frame(cbind(rank(data$FB)/(t+1),rank(data$VZ)/(t+1),rank(data$GOOGL)/(t+1),rank(da
  names(data_pseudo)=c("FB","VZ","GOOGL","MSFT")

  cor_tau=cor(data_pseudo,method="kendall")
  normalcop=normalCopula(dim=4,dispstr="un")
  ft0=fitCopula(normalcop,data_pseudo,method="mpl",start=c(cor_tau[1,2],cor_tau[1,3],cor_tau[1,4],cor_ta
  normalcop_coef=summary(ft0)$coefficients
  normalcop_est=tCopula(dim=4,param=tcop_coef[,1],dispstr="un")

  sample_num=50000
  sample=rCopula(n=sample_num,copula=normalcop_est)
  sample_rearrange=c(sample[,1],sample[,2],sample[,3],sample[,4])
  s1=as.numeric(quantile(data$FB,sample_rearrange))
  s2=as.numeric(quantile(data$VZ,sample_rearrange))
  s3=as.numeric(quantile(data$GOOGL,sample_rearrange))
  s4=as.numeric(quantile(data$MSFT,sample_rearrange))
  pnl=0.25*s1+0.25*s2+0.25*s3+0.25*s4 #equal weighted
  alpha=c(0.01,0.05,0.1)
  var=as.numeric(quantile(pnl,alpha))
  cvar=c(mean(pnl[pnl<var[1]]),mean(pnl[pnl<var[2]]),mean(pnl[pnl<var[3]]))

  output=cbind(var,cvar)
  return(output)
}
```

```
#####Simulation#####

var_series_t=c()
cvar_series_t=c()
var_series_t2=c()
cvar_series_t2=c()
#weight=wts.gmv.etl
weight=c(0.25,0.25,0.25,0.25)
#weight=wts.tan.etl
```

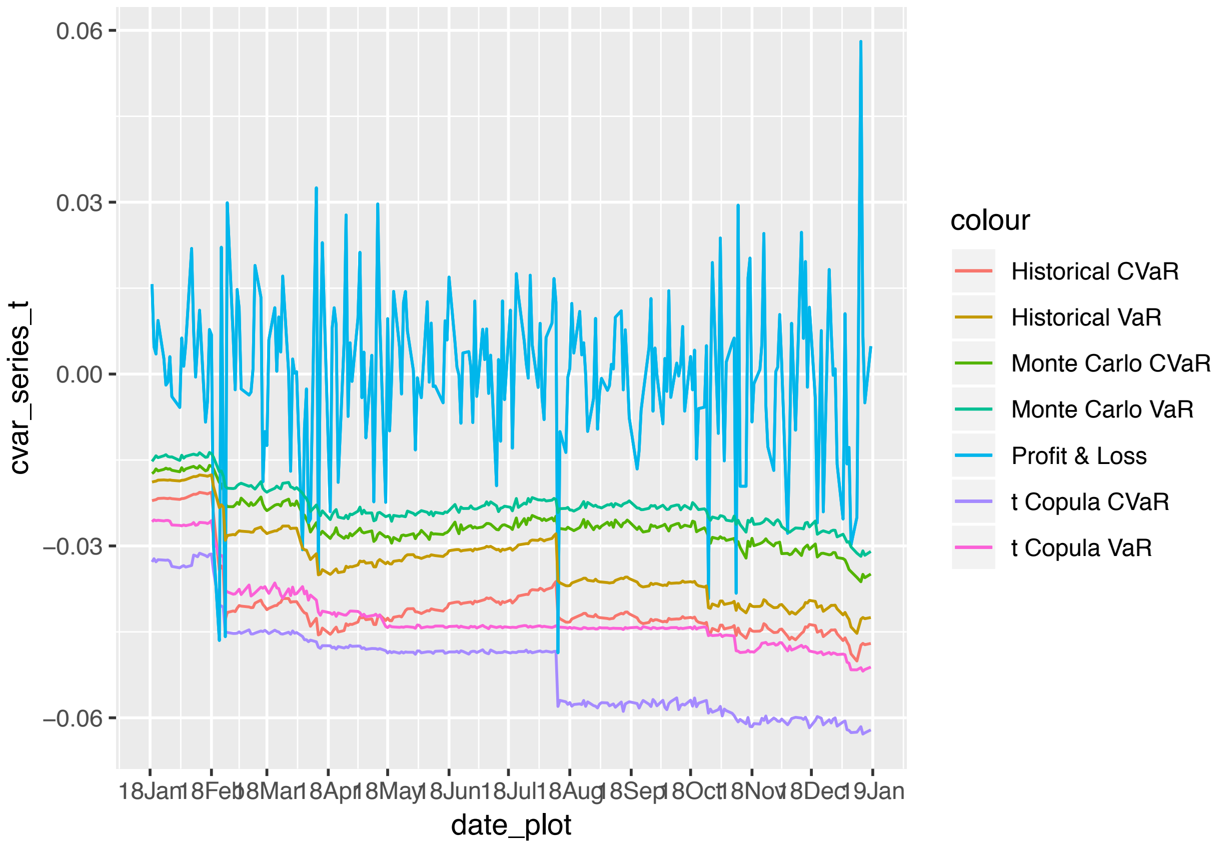
```r
for (i in 1:251) {
  data_temp=data[(1+i):(251+i),]
  result_temp1=fittcop(data_temp,weight=weight)
  #result_temp2=fitnormalcop(data_temp)
  var_series_t=c(var_series_t,as.numeric(result_temp1[1,1])) #select 99%
  cvar_series_t=c(cvar_series_t,as.numeric(result_temp1[1,2])) #select 99%
  #var_series_t2=c(var_series_t2,as.numeric(result_temp2[2,1]))
  #cvar_series_t2=c(cvar_series_t2,as.numeric(result_temp2[2,2]))
}

dat_VaR <- read.csv("vardata.csv")

truepnl=(StockValue(dat)$Stock_Value[1259:1509]-StockValue(dat)$Stock_Value[1258:1508])/StockValue(dat)$

date_plot=as.Date(data$Date[252:502])
hist_var_adjusted=-dat_VaR$HistVaR[1:251]/StockValue(dat)$Stock_Value[1259:1509]
hist_cvar_adjusted=-dat_VaR$HistES[1:251]/StockValue(dat)$Stock_Value[1259:1509]
MC_var_adjusted=-dat_VaR$MC.VaR[1:251]/StockValue(dat)$Stock_Value[1259:1509]
MC_cvar_adjusted=-dat_VaR$MC.ES[1:251]/StockValue(dat)$Stock_Value[1259:1509]

library(ggplot2)
ggplot(data.frame(date_plot,var_series_t,truepnl,hist_var_adjusted,MC_var_adjusted))+
  geom_line(mapping=aes(x=date_plot,y=var_series_t,color="t Copula VaR"))+
  geom_line(mapping=aes(x=date_plot,y=truepnl,color="Profit & Loss"))+
  geom_line(mapping=aes(x=date_plot,y=hist_var_adjusted,color="Historical VaR"))+
  geom_line(mapping=aes(x=date_plot,y=MC_var_adjusted,color="Monte Carlo VaR"))+
  scale_x_date(date_labels="%y%b",date_breaks ="1 month")
```




```r
ggplot(data.frame(date_plot,cvar_series_t,truepnl,hist_cvar_adjusted,MC_cvar_adjusted))+
  geom_line(mapping=aes(x=date_plot,y=cvar_series_t,color="t Copula CVaR"))+
  geom_line(mapping=aes(x=date_plot,y=truepnl,color="Profit & Loss"))+
  geom_line(mapping=aes(x=date_plot,y=hist_cvar_adjusted,color="Historical CVaR"))+
  geom_line(mapping=aes(x=date_plot,y=MC_cvar_adjusted,color="Monte Carlo CVaR"))+
  scale_x_date(date_labels="%y%b",date_breaks ="1 month")
```




```r
ggplot(data.frame(date_plot,cvar_series_t,truepnl,hist_cvar_adjusted,MC_cvar_adjusted))+
  geom_line(mapping=aes(x=date_plot,y=cvar_series_t,color="t Copula CVaR"))+
  geom_line(mapping=aes(x=date_plot,y=truepnl,color="Profit & Loss"))+
  geom_line(mapping=aes(x=date_plot,y=hist_cvar_adjusted,color="Historical CVaR"))+
  geom_line(mapping=aes(x=date_plot,y=MC_cvar_adjusted,color="Monte Carlo CVaR"))+
  geom_line(mapping=aes(x=date_plot,y=var_series_t,color="t Copula VaR"))+
  geom_line(mapping=aes(x=date_plot,y=truepnl,color="Profit & Loss"))+
  geom_line(mapping=aes(x=date_plot,y=hist_var_adjusted,color="Historical VaR"))+
  geom_line(mapping=aes(x=date_plot,y=MC_var_adjusted,color="Monte Carlo VaR"))+
  scale_x_date(date_labels="%y%b",date_breaks ="1 month")
```




```r
sum((var_series_t-hist_var_adjusted)>=0)
```

```
## [1] 0
```

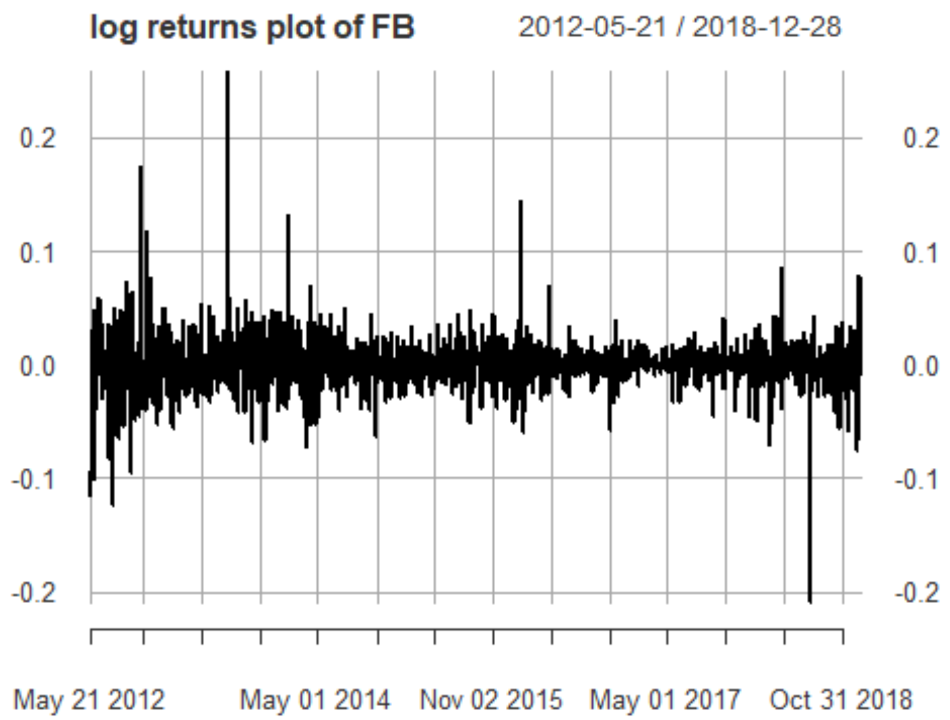
```r
sum((cvar_series_t-hist_cvar_adjusted)>=0)
```

```
## [1] 0
```

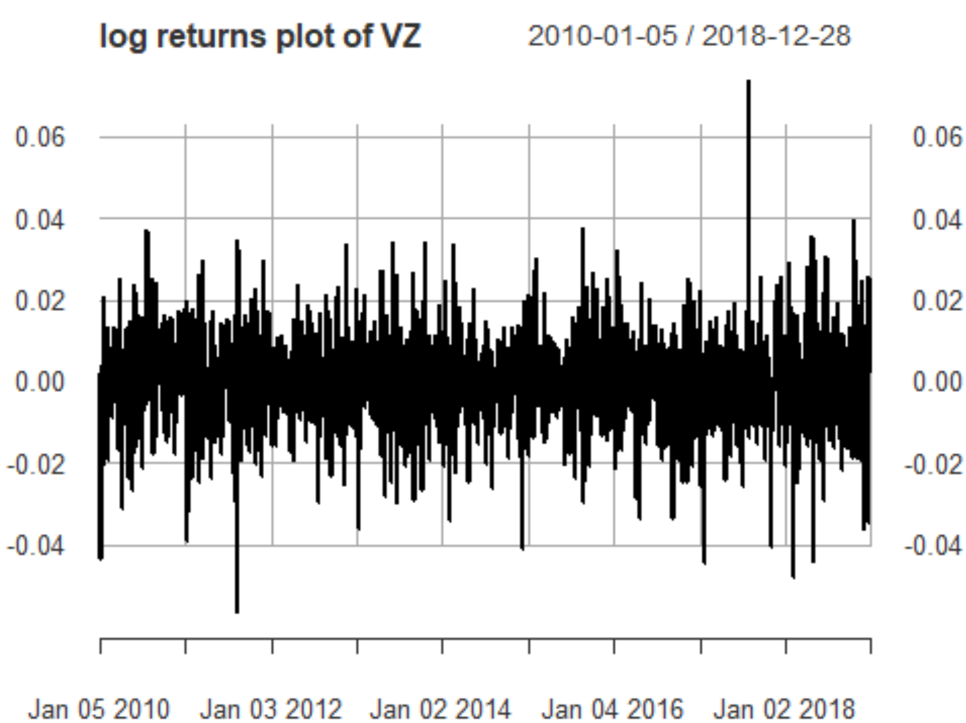
Time serise appendix

May 8, 2019

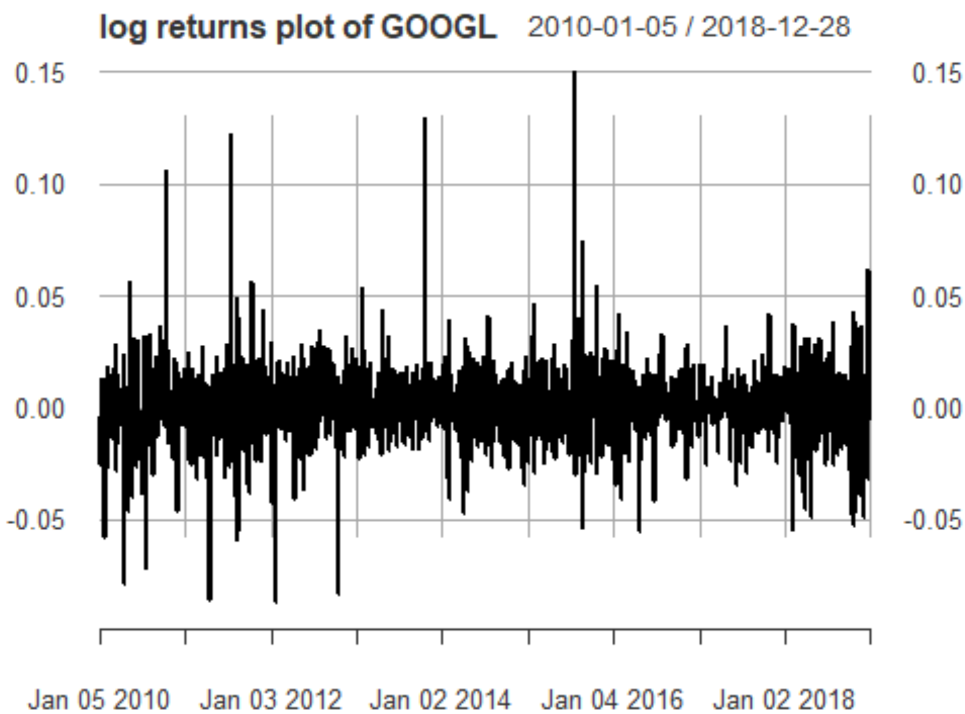
```
library(quantmod)
library(tseries)
library(forecast)

FB = getSymbols('FB', from='2010-01-01', to='2018-12-31',auto.assign = FALSE)
FB = na.omit(FB)
VZ = getSymbols('VZ', from='2010-01-01', to='2018-12-31',auto.assign = FALSE)
VZ = na.omit(VZ)
GOOGL = getSymbols('GOOGL', from='2010-01-01', to='2018-12-31',auto.assign =
FALSE)
GOOGL = na.omit(GOOGL)
MSFT = getSymbols('MSFT', from='2010-01-01', to='2018-12-31',auto.assign =
FALSE)
MSFT = na.omit(MSFT)

FB_prices = FB[,4]
VZ_prices = VZ[,4]
GOOGL_prices = GOOGL[,4]
MSFT_prices = MSFT[,4]

# log-return
r_FB = diff(log(FB_prices),lag=1)
r_FB = r_FB[!is.na(r_FB)]
plot(r_FB, type='l', main='log returns plot of FB')
```



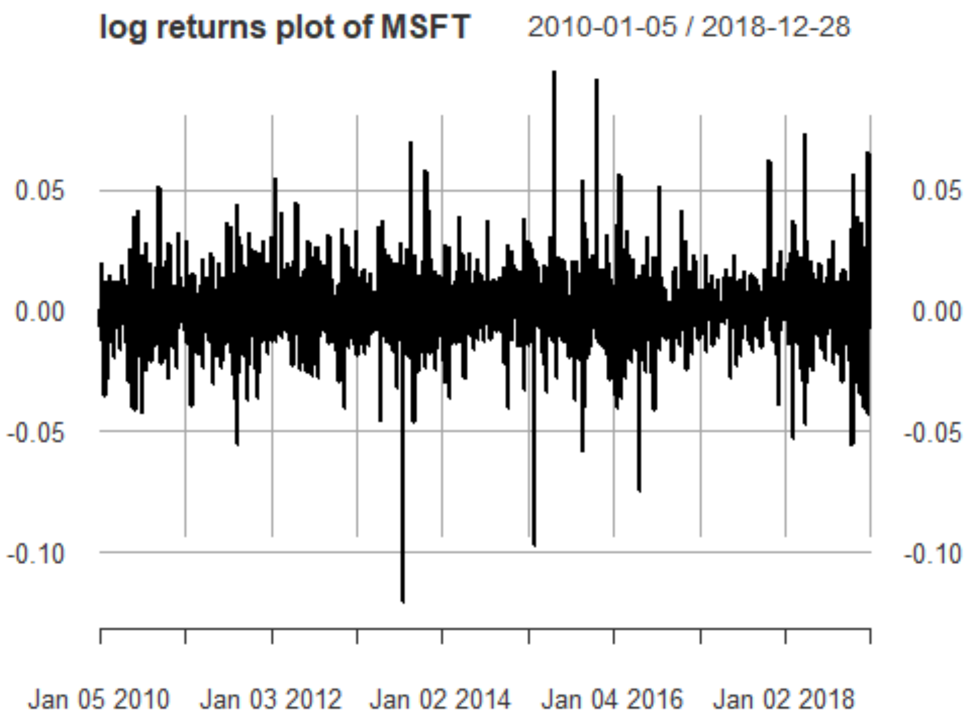

```r
r_VZ = diff(log(VZ_prices),lag=1)
r_VZ = r_VZ[!is.na(r_VZ)]
plot(r_VZ, type='l', main='log returns plot of VZ')
```



```r
r_GOOGL = diff(log(GOOGL_prices),lag=1)
r_GOOGL = r_GOOGL[!is.na(r_GOOGL)]
plot(r_GOOGL, type='l', main='log returns plot of GOOGL')
```




```r
r_MSFT = diff(log(MSFT_prices),lag=1)
r_MSFT = r_MSFT[!is.na(r_MSFT)]
plot(r_MSFT, type='l', main='log returns plot of MSFT')
```




```
FB_test = getSymbols('FB', from='2019-01-01', to='2019-04-10',auto.assign =
FALSE)
FB_test = na.omit(FB_test)
VZ_test = getSymbols('VZ', from='2019-01-01', to='2019-04-10',auto.assign =
FALSE)
VZ_test = na.omit(VZ_test)
GOOGL_test = getSymbols('GOOGL', from='2019-01-01', to='2019-04-
10',auto.assign = FALSE)
GOOGL_test = na.omit(GOOGL_test)
MSFT_test = getSymbols('MSFT', from='2019-01-01', to='2019-04-10',auto.assign
= FALSE)
MSFT_test = na.omit(MSFT_test)

FB_test_prices = FB_test[,4]
VZ_test_prices = VZ_test[,4]
GOOGL_test_prices = GOOGL_test[,4]
MSFT_test_prices = MSFT_test[,4]

r_test_FB = diff(log(FB_test_prices),lag=1)
r_test_FB = r_test_FB[!is.na(r_test_FB)]
r_test_VZ = diff(log(VZ_test_prices),lag=1)
r_test_VZ = r_test_VZ[!is.na(r_test_VZ)]
r_test_GOOGL = diff(log(GOOGL_test_prices),lag=1)
r_test_GOOGL = r_test_GOOGL[!is.na(r_test_GOOGL)]
r_test_MSFT = diff(log(MSFT_test_prices),lag=1)
r_test_MSFT = r_test_MSFT[!is.na(r_test_MSFT)]
```

Performs the Augmented Dickey-Fuller test for the null hypothesis of a unit root of a univarate time series x (equivalently, x is a non-stationary time series).

```
# check stationary of three data sets
library(aTSA)
library(forecast)
ts_FB <- ts(r_FB)
ts_VZ <- ts(r_VZ)
ts_GOOGL <- ts(r_GOOGL)
ts_MSFT <- ts(r_GOOGL)

adf.test(ts_FB)

## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -40.6    0.01
## [2,]   1 -29.2    0.01
## [3,]   2 -23.6    0.01
## [4,]   3 -20.6    0.01
## [5,]   4 -17.9    0.01
## [6,]   5 -17.4    0.01
## [7,]   6 -15.9    0.01
## [8,]   7 -14.7    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -40.7    0.01
## [2,]   1 -29.2    0.01
## [3,]   2 -23.6    0.01
## [4,]   3 -20.7    0.01
## [5,]   4 -17.9    0.01
## [6,]   5 -17.5    0.01
## [7,]   6 -16.0    0.01
## [8,]   7 -14.8    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -40.7    0.01
## [2,]   1 -29.3    0.01
## [3,]   2 -23.6    0.01
## [4,]   3 -20.7    0.01
## [5,]   4 -17.9    0.01
## [6,]   5 -17.5    0.01
## [7,]   6 -16.0    0.01
## [8,]   7 -14.9    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01

adf.test(ts_VZ)
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -47.3    0.01
## [2,]   1 -34.2    0.01
## [3,]   2 -28.1    0.01
## [4,]   3 -24.0    0.01
## [5,]   4 -20.9    0.01
## [6,]   5 -18.6    0.01
## [7,]   6 -17.2    0.01
## [8,]   7 -16.3    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -47.3    0.01
## [2,]   1 -34.2    0.01
## [3,]   2 -28.1    0.01
## [4,]   3 -24.0    0.01
## [5,]   4 -20.9    0.01
## [6,]   5 -18.6    0.01
## [7,]   6 -17.2    0.01
## [8,]   7 -16.3    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -47.3    0.01
## [2,]   1 -34.2    0.01
## [3,]   2 -28.1    0.01
## [4,]   3 -24.0    0.01
## [5,]   4 -20.9    0.01
## [6,]   5 -18.7    0.01
## [7,]   6 -17.2    0.01
## [8,]   7 -16.3    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

```
adf.test(ts_GOOGL)
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -46.6    0.01
## [2,]   1 -33.3    0.01
## [3,]   2 -27.3    0.01
## [4,]   3 -23.6    0.01
## [5,]   4 -21.8    0.01
## [6,]   5 -19.8    0.01
## [7,]   6 -18.5    0.01
```

```
## [8,]   7 -18.0    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -46.7    0.01
## [2,]   1 -33.4    0.01
## [3,]   2 -27.4    0.01
## [4,]   3 -23.7    0.01
## [5,]   4 -21.9    0.01
## [6,]   5 -19.9    0.01
## [7,]   6 -18.6    0.01
## [8,]   7 -18.1    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -46.7    0.01
## [2,]   1 -33.4    0.01
## [3,]   2 -27.4    0.01
## [4,]   3 -23.7    0.01
## [5,]   4 -21.9    0.01
## [6,]   5 -19.9    0.01
## [7,]   6 -18.6    0.01
## [8,]   7 -18.1    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

```
adf.test(ts_MSFT )
```

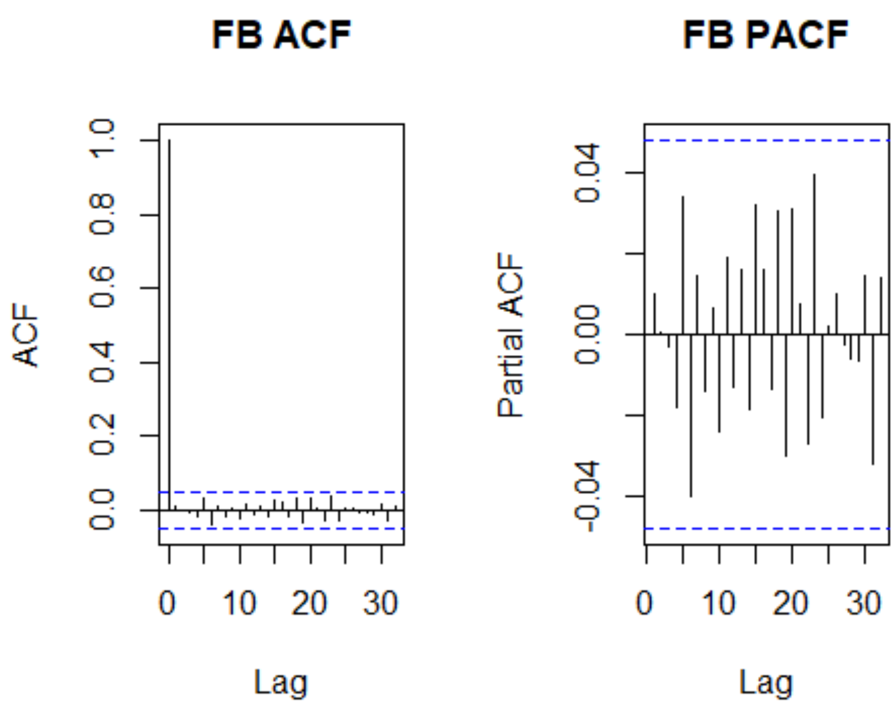
```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -46.6    0.01
## [2,]   1 -33.3    0.01
## [3,]   2 -27.3    0.01
## [4,]   3 -23.6    0.01
## [5,]   4 -21.8    0.01
## [6,]   5 -19.8    0.01
## [7,]   6 -18.5    0.01
## [8,]   7 -18.0    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -46.7    0.01
## [2,]   1 -33.4    0.01
## [3,]   2 -27.4    0.01
## [4,]   3 -23.7    0.01
## [5,]   4 -21.9    0.01
## [6,]   5 -19.9    0.01
## [7,]   6 -18.6    0.01
## [8,]   7 -18.1    0.01
## Type 3: with drift and trend
```

```
##       lag   ADF p.value
## [1,]    0 -46.7    0.01
## [2,]    1 -33.4    0.01
## [3,]    2 -27.4    0.01
## [4,]    3 -23.7    0.01
## [5,]    4 -21.9    0.01
## [6,]    5 -19.9    0.01
## [7,]    6 -18.6    0.01
## [8,]    7 -18.1    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

Diagnosing the ACF and PACF Plots

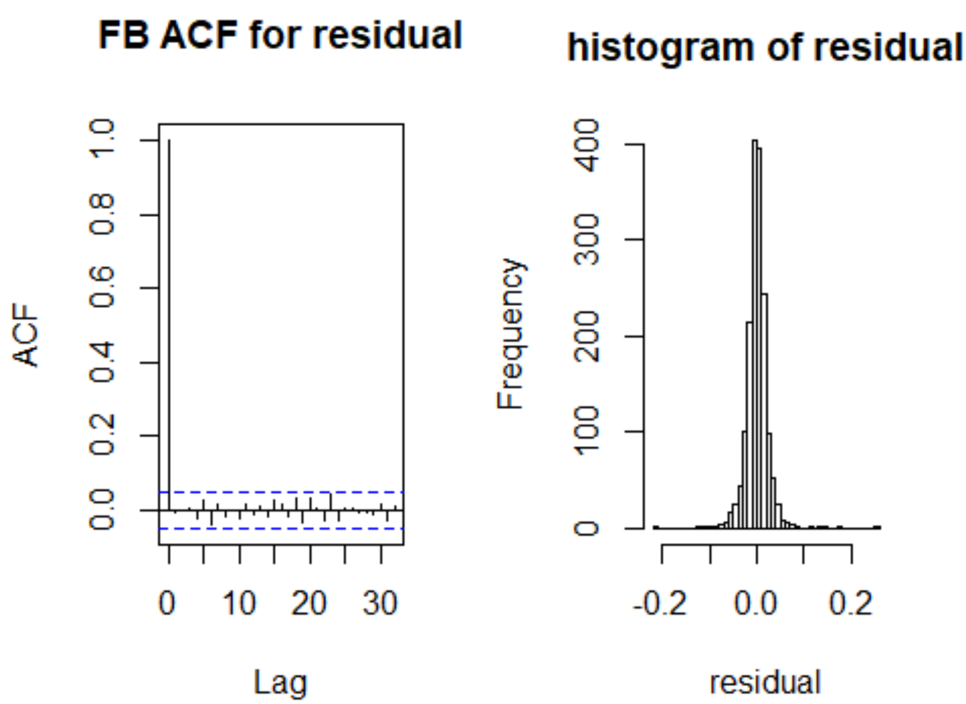
FB.

```
fit.fb <- auto.arima(na.omit(r_FB), seasonal = TRUE, max.p = 10, max.q = 10,
max.order = 10, ic = "aic")
par(mfrow=c(1,2))
acf((na.omit(r_FB)), main="FB ACF") ##autocorelation
pacf(na.omit(r_FB), main="FB PACF") ##partial acf
```

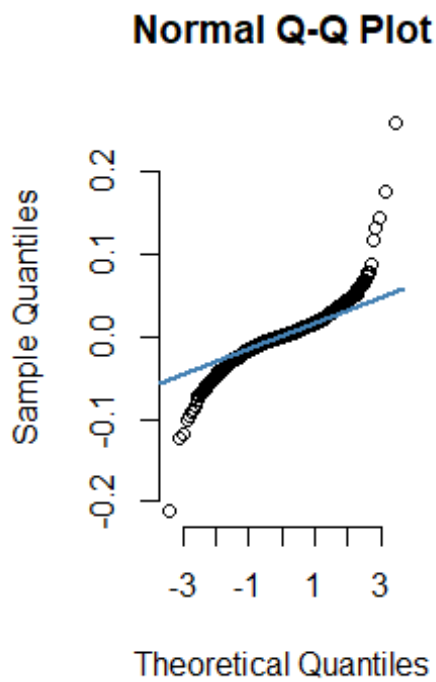



```
auto.fit.fb <- Arima(na.omit(r_FB), order = c(2, 0, 2))
hand.fit.fb <- Arima(na.omit(r_FB), order = c(0, 0, 0))
AIC(auto.fit.fb)

## [1] -7782.765
```

```r
AIC(hand.fit.fb)

## [1] -7790.071

# autofit is better

acf(auto.fit.fb$residuals, main="FB ACF for residual")
Box.test(auto.fit.fb$residuals, lag=10, fitdf=0, type="Box-Pierce")  # >0.05
in lag 10  Box-P >0.05

##
##  Box-Pierce test
##
## data:  auto.fit.fb$residuals
## X-squared = 6.1483, df = 10, p-value = 0.8027

hist(t(as.matrix(auto.fit.fb$residuals)),breaks = 40, main = "histogram of
residual", xlab = "residual")
```

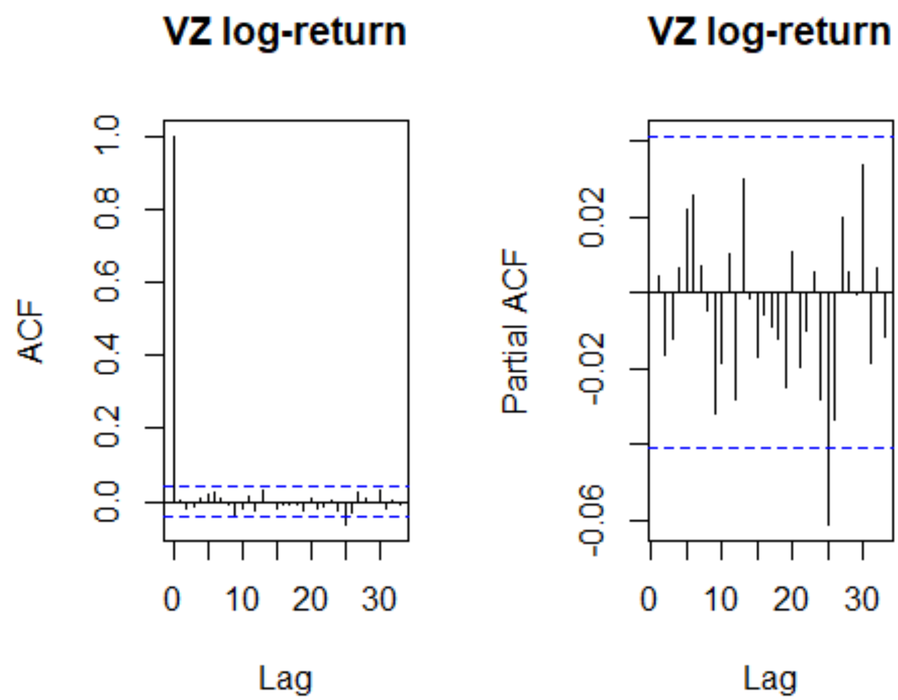



```r
qqnorm(auto.fit.fb$residuals, pch = 1, frame = FALSE)
qqline(auto.fit.fb$residuals, col = "steelblue", lwd = 2)
```

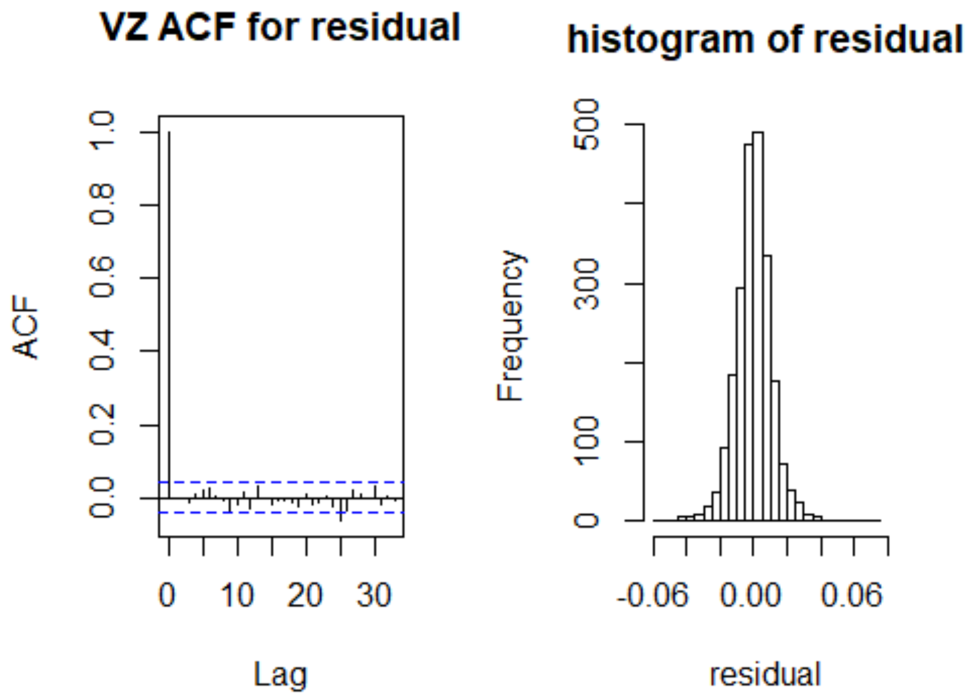


VZ

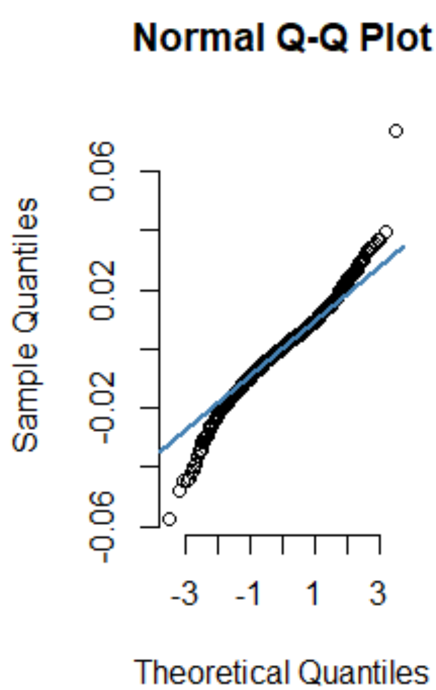
```
fit.vz <- auto.arima(na.omit(r_VZ),seasonal = TRUE, max.p = 10, max.q = 10,
max.order = 10, ic = "aic")
par(mfrow=c(1,2))
acf((na.omit(r_VZ)), main="VZ log-return")
pacf((na.omit(r_VZ)), main="VZ log-return")
```




```
auto.fit.vz <- Arima(na.omit(r_VZ), order = c(2, 0, 0))
hand.fit.vz <- Arima(na.omit(r_VZ), order = c(0, 0, 0))
AIC(auto.fit.vz)

## [1] -14076.64

AIC(hand.fit.vz)   # autofit is better

## [1] -14079.99

acf(auto.fit.vz$residuals, main="VZ ACF for residual")
Box.test(fit.fb$residuals,lag=10, fitdf=0, type="Box-Pierce")  # >0.05 in lag
10 Box-P >0.05

##
##  Box-Pierce test
##
## data:  fit.fb$residuals
## X-squared = 6.1483, df = 10, p-value = 0.8027

hist(t(as.matrix(auto.fit.vz$residuals)),breaks = 40, main = "histogram of
residual", xlab = "residual")
```

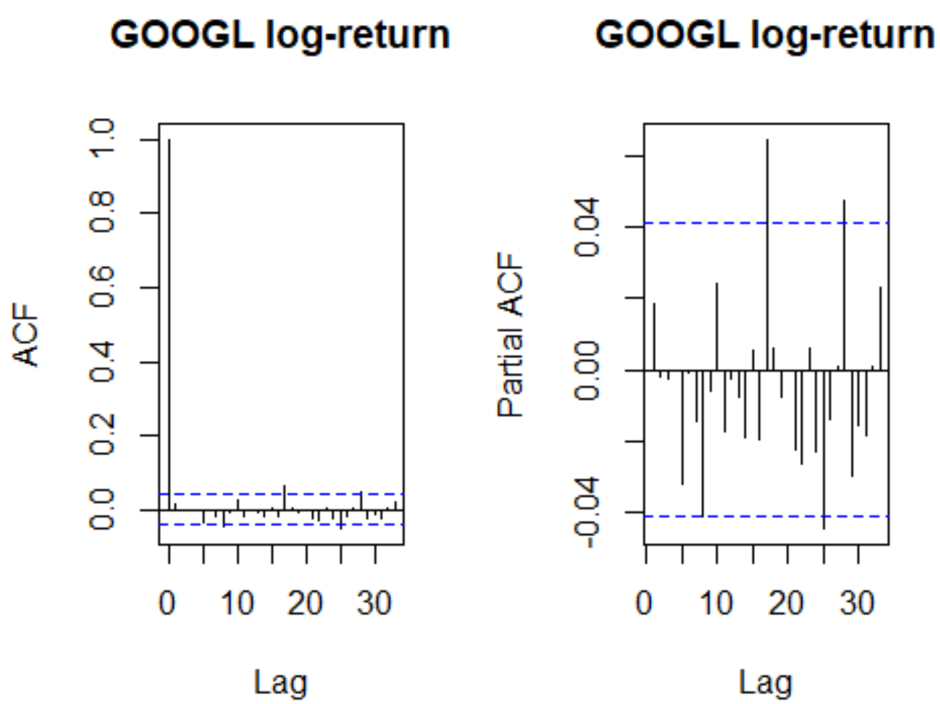



```
qqnorm(auto.fit.vz$residuals, pch = 1, frame = FALSE)
qqline(auto.fit.vz$residuals, col = "steelblue", lwd = 2)
```



googl

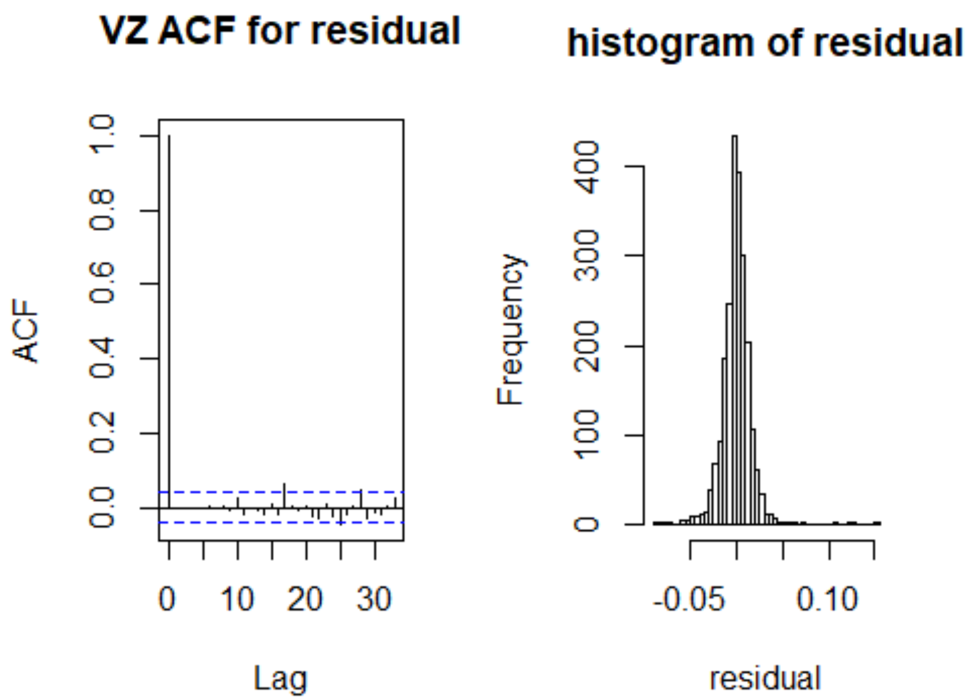
```
fit.googl <- auto.arima(na.omit(r_GOOGL), seasonal = TRUE, max.p = 10, max.q
= 10, max.order = 10, ic = "aic")

par(mfrow=c(1,2))
acf((na.omit(r_GOOGL)), main="GOOGL log-return")
pacf((na.omit(r_GOOGL)),main="GOOGL log-return")
```



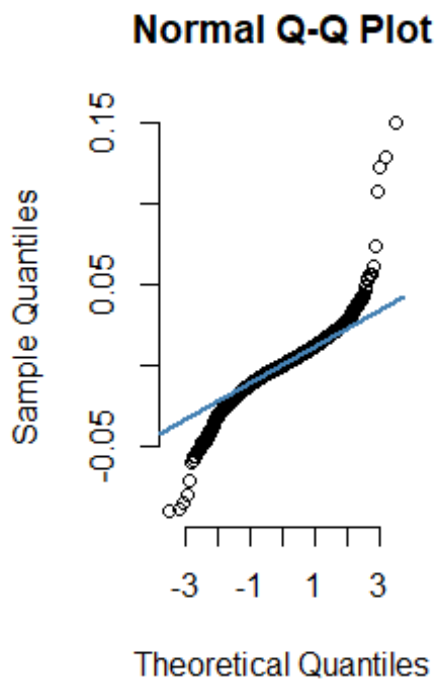

```
auto.fit.googl <- Arima(na.omit(r_GOOGL), order = c(0, 0, 0))
hand.fit.googl <- Arima(na.omit(r_GOOGL), order = c(0, 0, 8))
AIC(auto.fit.googl)

## [1] -12488.56

AIC(hand.fit.googl)

## [1] -12480.25

# However, only handfit is meanful


acf(hand.fit.googl$residuals, main="VZ ACF for residual")
Box.test(hand.fit.googl$residuals,lag=10, fitdf=0, type="Box-Pierce")

##
##  Box-Pierce test
##
```

```
## data:  hand.fit.googl$residuals
## X-squared = 1.7342, df = 10, p-value = 0.998
```

```r
hist(t(as.matrix(hand.fit.googl$residuals)),breaks = 40, main = "histogram of residual", xlab = "residual")
```

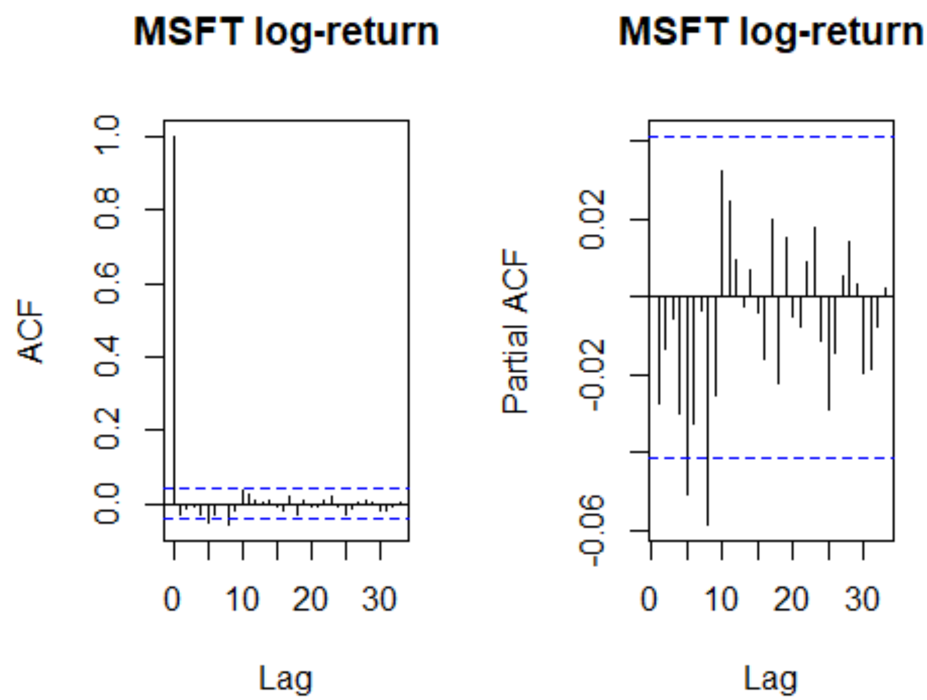



```r
qqnorm(hand.fit.googl$residuals, pch = 1, frame = FALSE)
qqline(hand.fit.googl$residuals, col = "steelblue", lwd = 2)
```

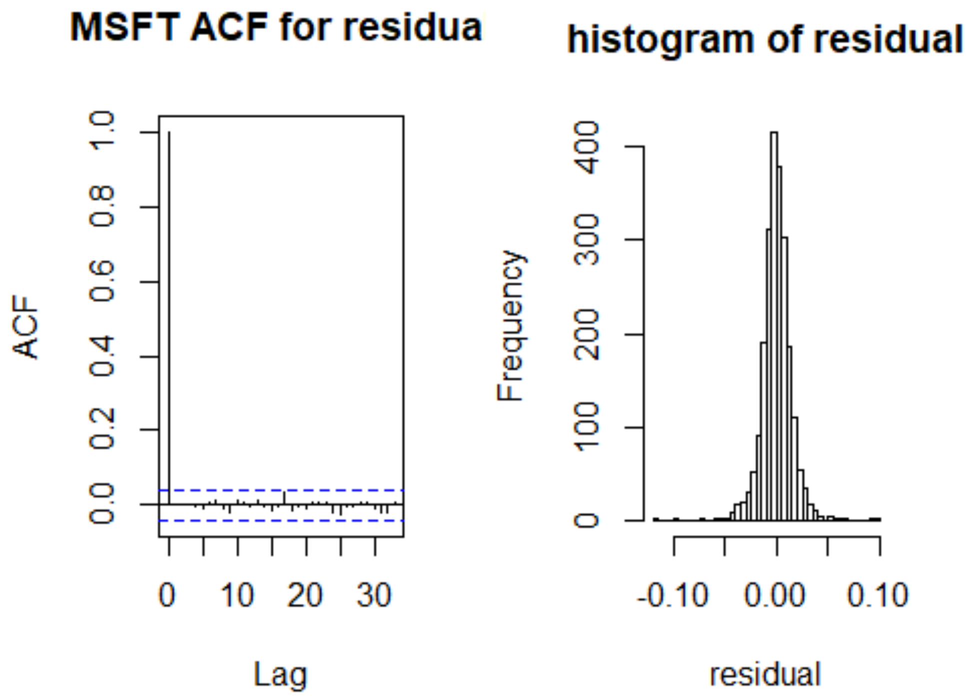


MSFT

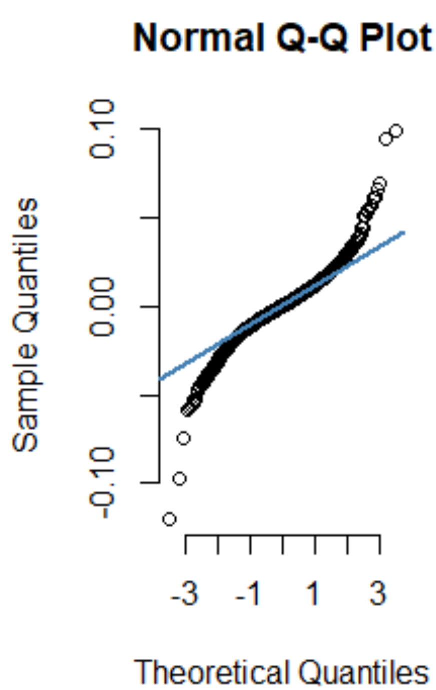
```
fit.msft <- auto.arima(na.omit(r_MSFT),seasonal = TRUE, max.p = 10, max.q =
10, max.order = 10, ic = "aic")
par(mfrow=c(1,2))
acf((na.omit(r_MSFT)),main="MSFT log-return")
pacf((na.omit(r_MSFT)),main="MSFT log-return")
```




```
auto.fit.msft <- Arima(na.omit(r_MSFT), order = c(1, 0, 1))
hand.fit.msft <- Arima(na.omit(r_MSFT), order = c(8, 0, 8))
AIC(auto.fit.msft)

## [1] -12730.98

AIC(hand.fit.msft)     # auto is better, but not meanful

## [1] -12723.19

acf(hand.fit.msft$residuals, main="MSFT ACF for residual")

Box.test(hand.fit.msft$residuals,lag=10, fitdf=0, type="Box-Pierce")

##
##  Box-Pierce test
##
## data:  hand.fit.msft$residuals
## X-squared = 2.1529, df = 10, p-value = 0.995

hist(t(as.matrix(hand.fit.msft$residuals)),breaks = 40, main = "histogram of
residual", xlab = "residual")
```




```r
qqnorm(hand.fit.msft$residuals, pch = 1, frame = FALSE)
qqline(hand.fit.msft$residuals, col = "steelblue", lwd = 2)
```

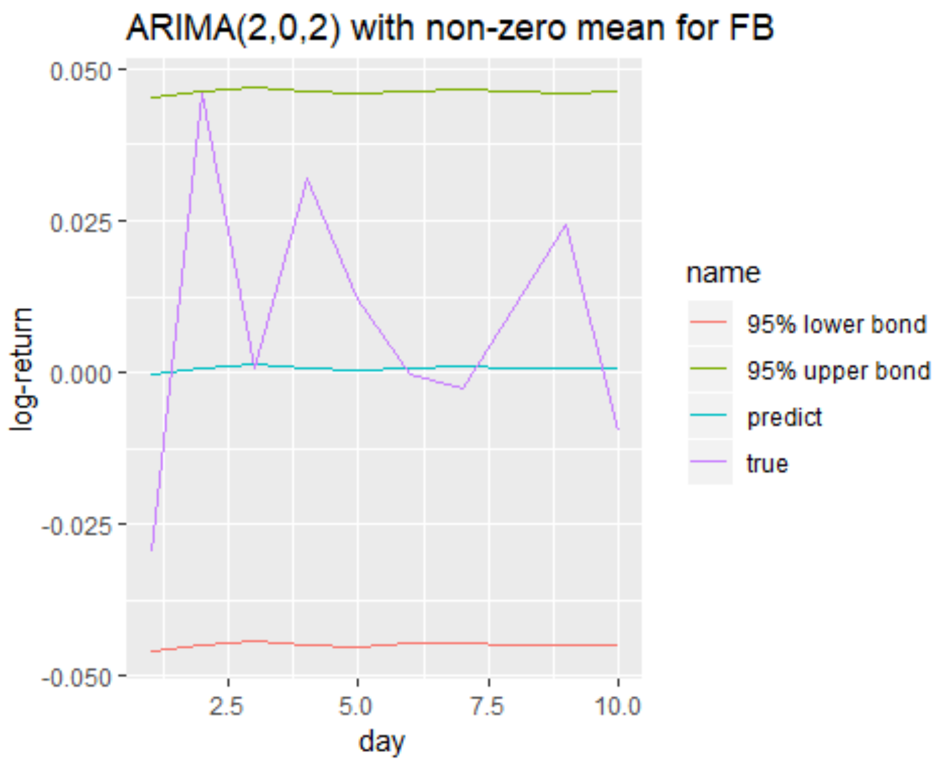


arima is our baseline

arima for FB

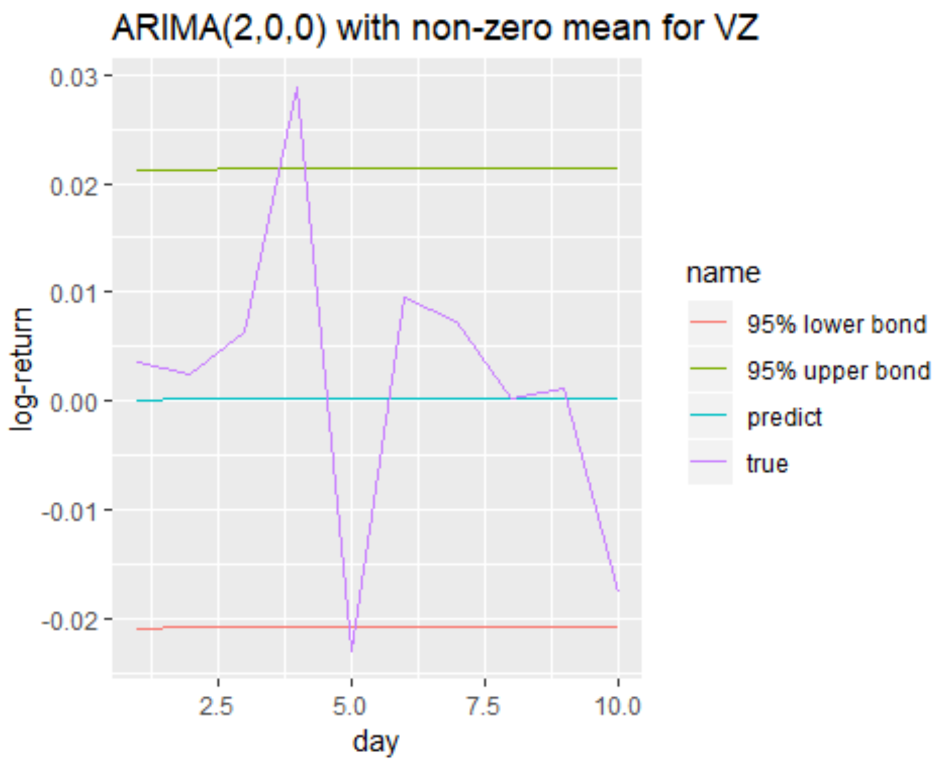
fit a arima model for FB data

```
# please unload aTSA,itsmr before running this code
library(aTSA)
library(itsmr)

detach("package:aTSA", unload=TRUE)
detach("package:itsmr", unload=TRUE)

library(forecast)
library(ggplot2)

a <- data.frame(r_test_FB[1:10,])
time <- unlist(rownames(a))
data <- a$FB.Close

#test <- forecast(auto.fit.fb, h = length(data))
test <- forecast(auto.fit.fb, h = length(data))


arima.predict <- test$mean
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x,x,x),
                        y=c(data,arima.predict,test$upper[,2],test$lower[,2]),
                        name =
c(rep("true",length(x)),rep("predict",length(x)),rep("95% upper
bond",length(x)),
                                    rep("95% lower bond",length(x))))
ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(2,0,2) with non-zero mean for FB", x = "day", y = "log-
return")
```



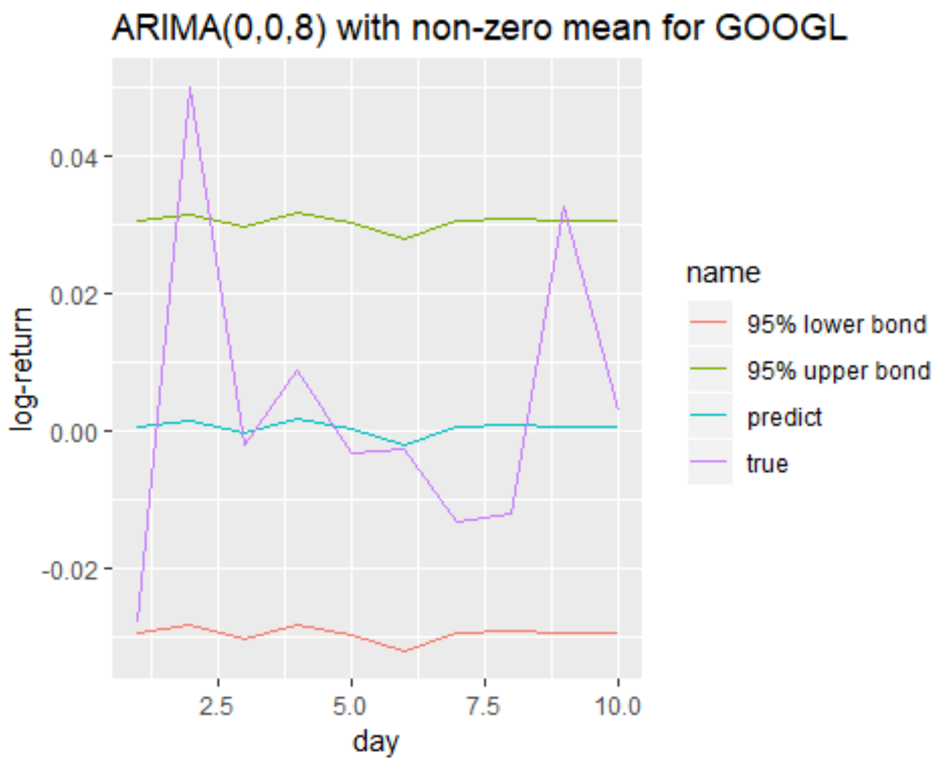
fit a arima model for VZ data

```
# please unload aTSA,itsmr before running this code
library(forecast)

a <- data.frame(r_test_VZ[1:10,])
time <- unlist(rownames(a))
data <- a$VZ.Close

test <- forecast(auto.fit.vz, h = length(data))

arima.predict <- test$mean
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x,x,x),
                   y=c(data,arima.predict,test$upper[,2],test$lower[,2]),
                   name =
c(rep("true",length(x)),rep("predict",length(x)),rep("95% upper
bond",length(x)),
                         rep("95% lower bond",length(x))))
ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(2,0,0) with non-zero mean for VZ", x = "day", y = "log-
return")
```



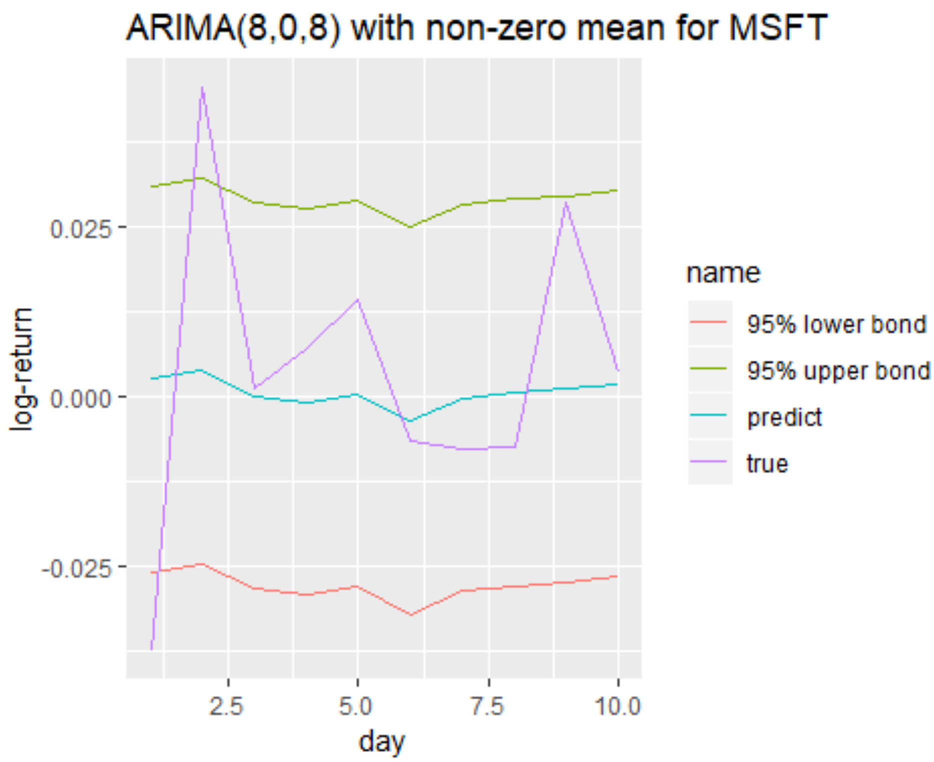
fit a arima model for GOOGL data

```
# please unload aTSA,itsmr before running this code
library(forecast)

a <- data.frame(r_test_GOOGL[1:10,])
time <- unlist(rownames(a))
data <- a$GOOGL.Close

test <- forecast(hand.fit.googl, h = length(data))

arima.predict <- test$mean
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x,x,x),
                   y=c(data,arima.predict,test$upper[,2],test$lower[,2]),
                   name =
c(rep("true",length(x)),rep("predict",length(x)),rep("95% upper
bond",length(x)),
                            rep("95% lower bond",length(x))))
ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(0,0,8) with non-zero mean for GOOGL", x = "day", y =
"log-return")
```



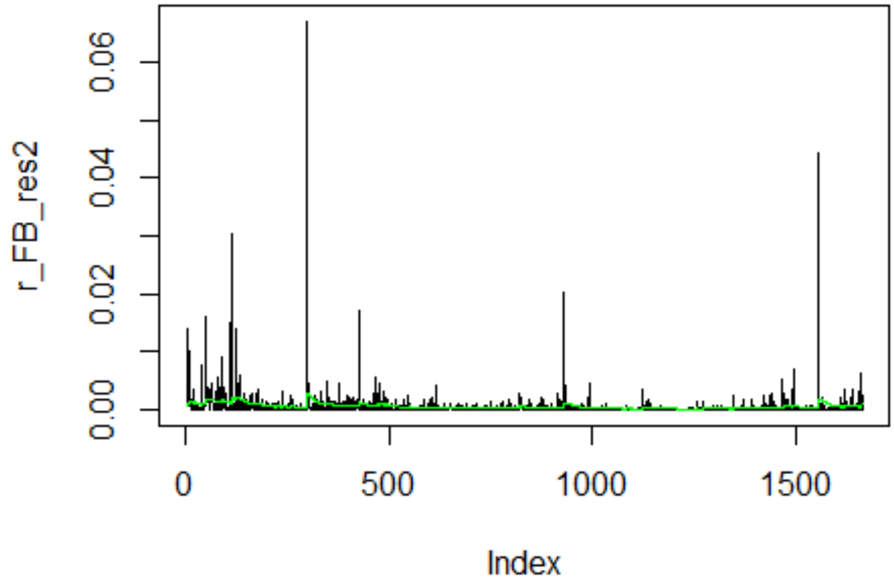
fit a arima model for MSFT data

```
# please unload aTSA,itsmr before running this code
library(forecast)

a <- data.frame(r_test_MSFT[1:10,])
time <- unlist(rownames(a))
data <- a$MSFT.Close

#test <- forecast(auto.fit.msft, h = length(data))
test <- forecast(hand.fit.msft, h = length(data))


arima.predict <- test$mean
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x,x,x),
                        y=c(data,arima.predict,test$upper[,2],test$lower[,2]),
                        name =
c(rep("true",length(x)),rep("predict",length(x)),rep("95% upper
bond",length(x)),
                                rep("95% lower bond",length(x))))
ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(8,0,8) with non-zero mean for MSFT", x = "day", y =
"log-return")
```

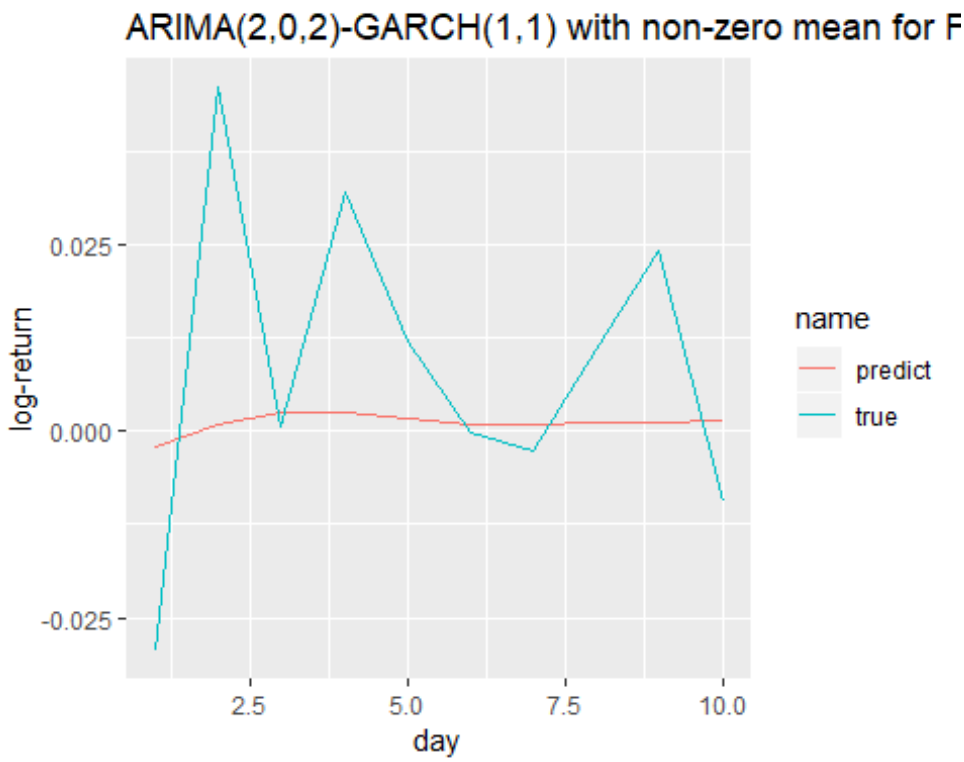


fit arima-garch(1,1) for FB

```
library(rugarch)
myspec <- ugarchspec(variance.model = list(garchOrder = c(1, 1), submodel =
NULL,
                                        external.regressors = NULL,
variance.targeting = FALSE),
                  mean.model = list(armaOrder = c(2, 2), include.mean = TRUE,
archm = FALSE,
                                    archpow = 1, arfima = FALSE,
external.regressors = NULL,
                                    archex = FALSE),
                  distribution.model="std")

r_FB_fit <- ugarchfit(spec = myspec,data=r_FB, solver="solnp")
r_FB_fit@fit$coef

##            mu           ar1           ar2           ma1           ma2
##  1.226839e-03  8.222295e-01 -4.832339e-01 -8.460434e-01  4.525421e-01
##         omega        alpha1         beta1         shape
##  1.819504e-06  3.599169e-02  9.614638e-01  3.720024e+00

r_FB_var <- r_FB_fit@fit$var
r_FB_res2 <- (r_FB_fit@fit$residuals)^2
plot(r_FB_res2, type = "l")
lines(r_FB_var, col = "green")
```

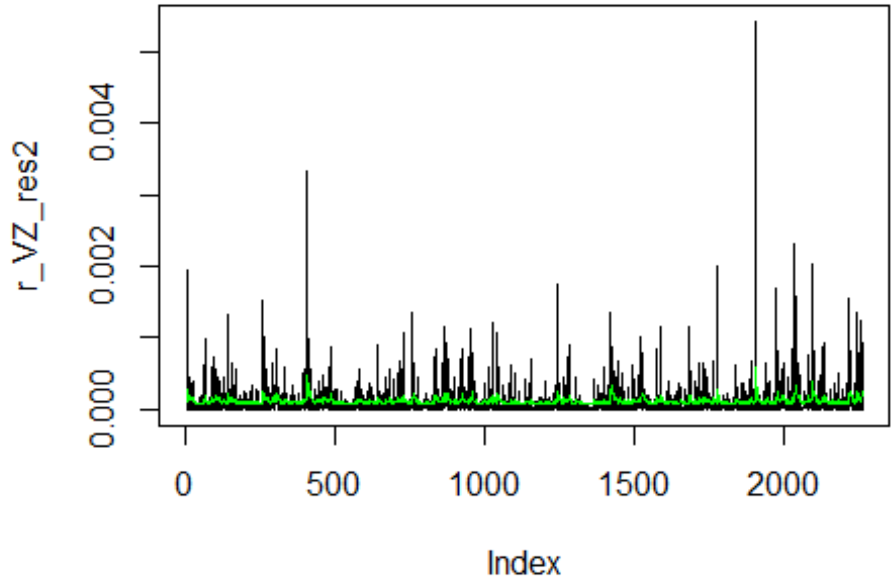



```r
# plot(r_FB_fit)  plot with options cannot be knitted

test <- ugarchforecast(r_FB_fit,n.ahead=10,data=r_FB)
# plot(test)  plot with options cannot be knitted

# std
predict <- c(-
0.0020092,0.0007549,0.0024026,0.0024216,0.0016411,0.0009901,0.0008320,0.00101
66,0.0012448 ,0.0013432)

a <- data.frame(r_test_FB[1:10,])
time <- unlist(rownames(a))
data <- a$FB.Close


x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x),
                        y=c(data,predict),
                        name = c(rep("true",length(x)),rep("predict",length(x))))

ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(2,0,2)-GARCH(1,1) with non-zero mean for FB", x =
"day", y = "log-return")
```

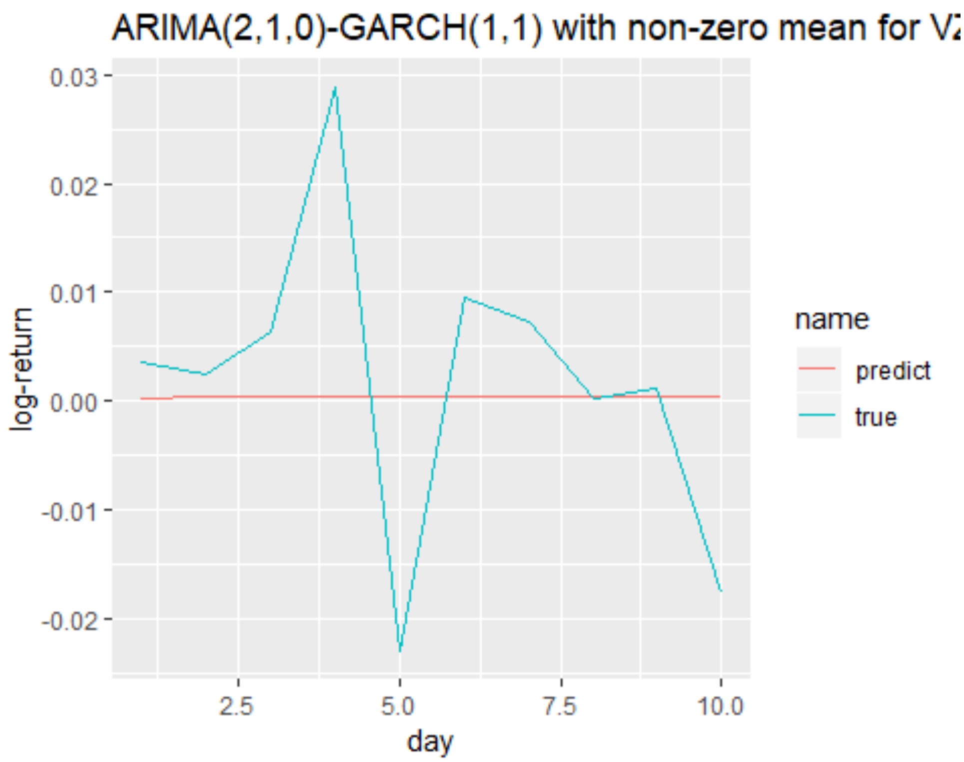


fit arima-garch(1,1) for VZ

```
library(rugarch)
myspec <- ugarchspec(variance.model = list(garchOrder = c(1, 1), submodel =
NULL,
                                        external.regressors = NULL,
variance.targeting = FALSE),
                  mean.model = list(armaOrder = c(2, 0), include.mean = TRUE,
archm = FALSE,
                                    archpow = 1, arfima = FALSE,
external.regressors = NULL,
                                    archex = FALSE),
                  distribution.model = "std")

r_VZ_fit <- ugarchfit(spec = myspec,data=r_VZ, solver="solnp")
r_VZ_fit@fit$coef

##            mu          ar1          ar2        omega       alpha1
##  4.099562e-04  1.108811e-02 -2.132502e-02  1.516539e-05  9.333735e-02
##         beta1        shape
##  7.781598e-01  6.182830e+00

r_VZ_var <- r_VZ_fit@fit$var
r_VZ_res2 <- (r_VZ_fit@fit$residuals)^2
plot(r_VZ_res2, type = "l")
lines(r_VZ_var, col = "green")
```



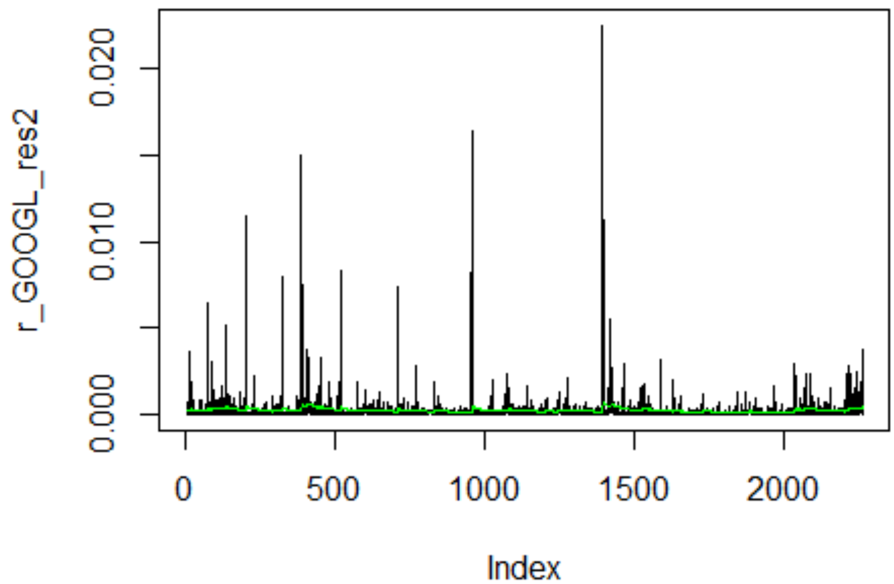

```
#plot(r_VZ_fit)  plot with options cannot be knitted

test <- ugarchforecast(r_VZ_fit,n.ahead=10,data=r_VZ)
#plot(test)  plot with options cannot be knitted

predict<-
c(0.0001619,0.0003696,0.0004148,0.0004109,0.0004099,0.0004099,0.0004100,0.000
4100,0.0004100,0.0004100)

a <- data.frame(r_test_VZ[1:10,])
time <- unlist(rownames(a))
data <- a$VZ.Close
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x),
                   y=c(data,predict),
                   name = c(rep("true",length(x)),rep("predict",length(x))))

ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(2,1,0)-GARCH(1,1) with non-zero mean for VZ", x =
"day", y = "log-return")
```



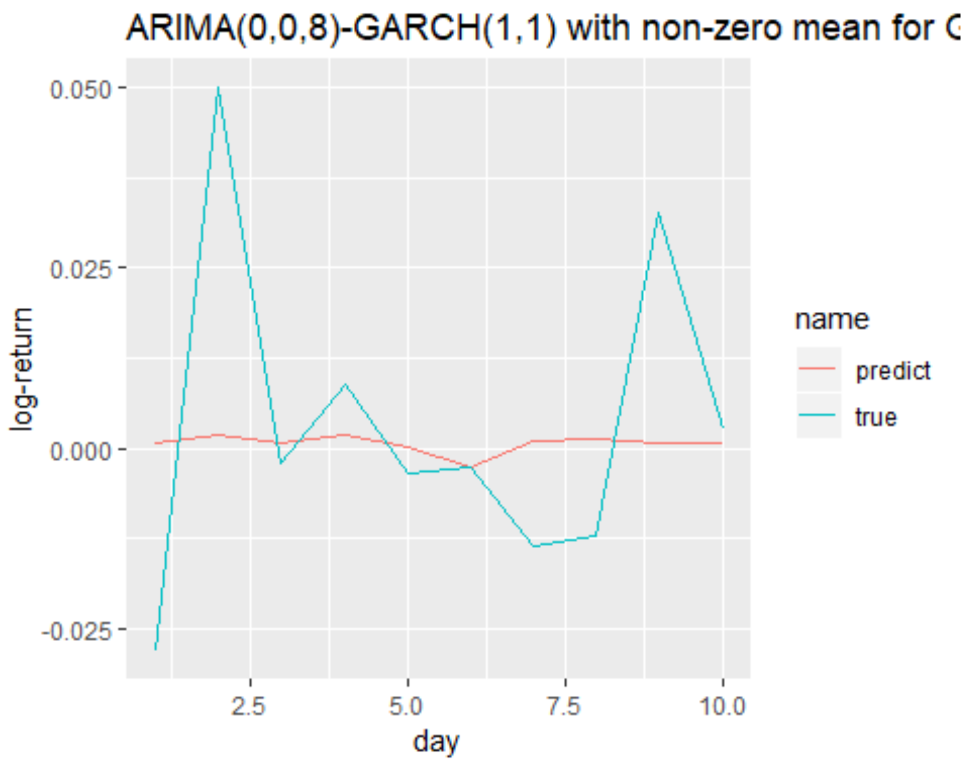
fit arima-garch(1,1) for googl

```
library(rugarch)
myspec <- ugarchspec(variance.model = list(garchOrder = c(1, 1), submodel = 
NULL,
                                         external.regressors = NULL, 
variance.targeting = FALSE),
                  mean.model = list(armaOrder = c(0, 8), include.mean = TRUE, 
archm = FALSE,
                                    archpow = 1, arfima = FALSE, 
external.regressors = NULL,
                                    archex = FALSE),
                  distribution.model = "std")

r_GOOGL_fit <- ugarchfit(spec = myspec,data=r_GOOGL, solver="solnp")
r_GOOGL_fit@fit$coef

##            mu           ma1           ma2           ma3           ma4 
##  8.433024e-04  2.656700e-02  3.835647e-03 -3.727838e-03 -9.768139e-03 
##           ma5           ma6           ma7           ma8         omega 
## -2.975267e-02 -1.734688e-02 -2.227947e-02 -6.000378e-02  2.163071e-06 
##        alpha1         beta1         shape 
##  2.289137e-02  9.679212e-01  3.781436e+00

r_GOOGL_var <- r_GOOGL_fit@fit$var
r_GOOGL_res2 <- (r_GOOGL_fit@fit$residuals)^2
```

```r
plot(r_GOOGL_res2, type = "l")
lines(r_GOOGL_var, col = "green")
```




```r
# plot(r_GOOGL_fit) plot with option cannot be knitted

test <- ugarchforecast(r_GOOGL_fit,n.ahead=10,data=r_GOOGL)
# plot(test) plot with option cannot be knitted

predict<- c(0.0007698,0.0018607,0.0006213,0.0019938,0.0001841,-
0.0026608,0.0010379,0.0013933,0.0008433,0.0008433)

a <- data.frame(r_test_GOOGL[1:10,])
time <- unlist(rownames(a))
data <- a$GOOGL.Close

x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x),
                    y=c(data,predict),
                    name = c(rep("true",length(x)),rep("predict",length(x))))

ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(0,0,8)-GARCH(1,1) with non-zero mean for GOGL", x =
"day", y = "log-return")
```

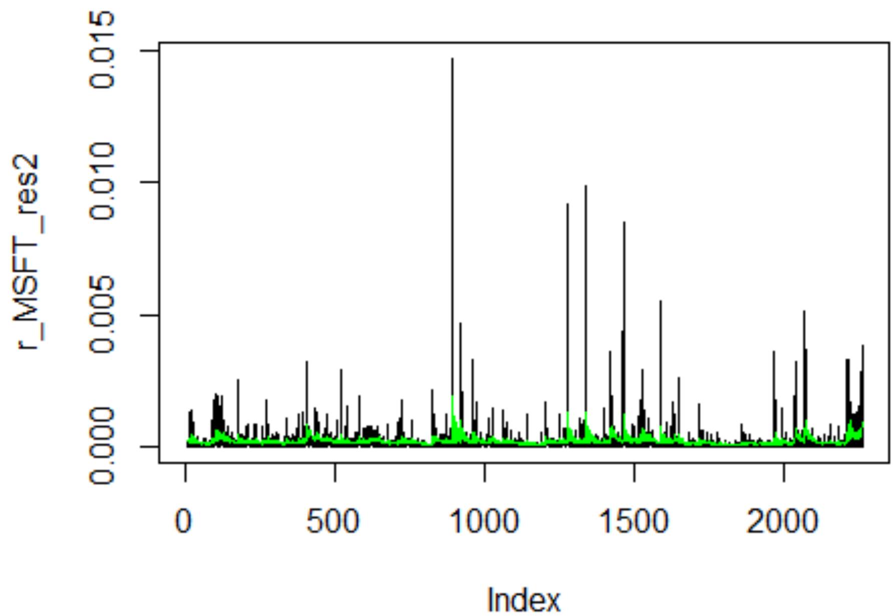


fit arima-garch(1,1) for MSFT

```
library(rugarch)
myspec <- ugarchspec(variance.model = list(garchOrder = c(10, 10), submodel = NULL,
                                        external.regressors = NULL,
variance.targeting = FALSE),
                  mean.model = list(armaOrder = c(8, 8), include.mean = TRUE,
archm = FALSE,
                                    archpow = 1, arfima = FALSE,
external.regressors = NULL,
                                    archex = FALSE),
                  distribution.model = "std")

r_MSFT_fit <- ugarchfit(spec = myspec,data=r_MSFT, solver="solnp")
r_MSFT_fit@fit$coef

##            mu           ar1           ar2           ar3           ar4
##  7.967988e-04  6.858379e-01 -6.647892e-01  6.211483e-02  1.173999e-01
##           ar5           ar6           ar7           ar8           ma1
## -5.805110e-01  9.122712e-01 -4.495788e-01 -4.403046e-02 -6.974982e-01
##           ma2           ma3           ma4           ma5           ma6
##  6.796358e-01 -9.890330e-02 -1.312685e-01  5.676126e-01 -9.666008e-01
##           ma7           ma8         omega        alpha1        alpha2
##  4.513791e-01 -4.300620e-03  1.956116e-05  1.195146e-01  9.989315e-02
##        alpha3        alpha4        alpha5        alpha6        alpha7
##  1.335222e-08  6.593657e-02  5.052002e-13  2.289803e-02  1.583184e-07
```

```
##        alpha8        alpha9       alpha10         beta1         beta2
##  3.187762e-03  2.875906e-08  2.083221e-02  6.806350e-08  4.782505e-08
##         beta3         beta4         beta5         beta6         beta7
##  7.628352e-08  5.159703e-08  2.709911e-01  1.276705e-07  4.092441e-08
##         beta8         beta9        beta10         shape
##  1.330274e-07  8.376178e-08  3.263776e-01  4.237392e+00
```

```
r_MSFT_var <- r_MSFT_fit@fit$var
r_MSFT_res2 <- (r_MSFT_fit@fit$residuals)^2
plot(r_MSFT_res2, type = "l")
lines(r_MSFT_var, col = "green")
```




```
#plot(r_MSFT_fit) plot with options cannot be knitted

test <- ugarchforecast(r_MSFT_fit,n.ahead=10,data=r_GOOGL)
#plot(test) plot with options cannot be knitted

predict <- c(0.0024086,0.0035608,0.0013704,-0.0006616,0.0024898,0.0003487,-
0.0021382,0.0009073,0.0030510,-0.0006585)

a <- data.frame(r_test_MSFT[1:10,])
time <- unlist(rownames(a))
data <- a$MSFT.Close
x <- c(1:length(data))
plot.data <- data.frame(x=c(x,x),
                   y=c(data,predict),
                   name = c(rep("true",length(x)),rep("predict",length(x))))
```

```r
ggplot(data = plot.data) +
  geom_line(mapping = aes(x = x, y = y, col= name)) +
  labs(title = "ARIMA(8,0,8)-GARCH(1,1) with non-zero mean for MSFT", x =
"day", y = "log return")
```