Econ 147 Lab 2

0

MSFT is the ticker for Microsoft, a multinational tech company based in Redmond, WA. It represents a share in the company. SBUX is the ticker for Starbucks, a multinational coffehouse chain. FMAGX is a mutual fund run by fidelity. It is a managed fund, meaning there are people that actively choose investments in that fund.

1

```
options(digits=4, width=70)

library(PerformanceAnalytics)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
## Attaching package: 'PerformanceAnalytics'
```

```
## The following object is masked from 'package:graphics':
##
##     legend
```

```
library(zoo)
library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
# get monthly adjusted closing price data on MSFT, FMAGX and SBUX from Yahoo
# using the tseries function get.hist.quote(). Set sample to Jan 1998 through
# Dec 2009. Note: if you are not careful with the start and end dates
# or if you set the retclass to "ts" then results might look weird

# look at help on get.hist.quote
?get.hist.quote

# get the adjusted closing prices from Yahoo!
MSFT.prices = get.hist.quote(instrument="msft", start="1998-01-01",
                             end="2009-12-31", quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## time series ends   2009-12-01
```

```
# change class of time index to yearmon which is appropriate for monthly data
# index() and as.yearmon() are functions in the zoo package

index(MSFT.prices) = as.yearmon(index(MSFT.prices))

class(MSFT.prices)
```

```
## [1] "zoo"
```

```
colnames(MSFT.prices)
```

```
## [1] "Adjusted"
```

```
start(MSFT.prices)
```

```
## [1] "Jan 1998"
```

```
end(MSFT.prices)
```

```
## [1] "Dec 2009"
```

```
FMAGX.prices = get.hist.quote(instrument="fmagx", start="1998-01-01",
                              end="2009-12-31", quote="AdjClose",
                              provider="yahoo", origin="1970-01-01",
                              compression="m", retclass="zoo")
```

```
## time series ends   2009-12-01
```

```
index(FMAGX.prices) = as.yearmon(index(FMAGX.prices))

SBUX.prices = get.hist.quote(instrument="sbux", start="1998-01-01",
                             end="2009-12-31", quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
```

```
## time series ends   2009-12-01
```

```
index(SBUX.prices) = as.yearmon(index(SBUX.prices))

# create merged price data
lab4Prices.z = merge(MSFT.prices, FMAGX.prices, SBUX.prices)
# rename columns
colnames(lab4Prices.z) = c("MSFT", "FMAGX", "SBUX")

# calculate cc returns as difference in log prices
lab4Returns.z = diff(log(lab4Prices.z))

#
# See the document "Working with Time Series in R" on the
# class webpage for more details on zoo objects
#

# look at the return data
start(lab4Returns.z)
```

```
## [1] "Feb 1998"
```

```
end(lab4Returns.z)
```

```
## [1] "Dec 2009"
```

```
colnames(lab4Returns.z)
```

```
## [1] "MSFT"  "FMAGX" "SBUX"
```

```
head(lab4Returns.z)
```

```
##                MSFT     FMAGX      SBUX
## Feb 1998  0.127649  0.073069  0.078859
## Mar 1998  0.054533  0.049066  0.135701
## Apr 1998  0.006959  0.011513  0.060219
## May 1998 -0.060754 -0.045728 -0.002601
## Jun 1998  0.245154  0.043541  0.107312
## Jul 1998  0.014314 -0.007508 -0.243824
```

```
################################################################################
# IID normal model
################################################################################

# Create matrix of return data.

ret.mat = coredata(lab4Returns.z)
class(ret.mat)
```

```
## [1] "matrix" "array"
```

```
colnames(ret.mat)
```

```
## [1] "MSFT"  "FMAGX" "SBUX"
```

```
head(ret.mat)
```

```
##            MSFT     FMAGX      SBUX
## [1,]  0.127649  0.073069  0.078859
## [2,]  0.054533  0.049066  0.135701
## [3,]  0.006959  0.011513  0.060219
## [4,] -0.060754 -0.045728 -0.002601
## [5,]  0.245154  0.043541  0.107312
## [6,]  0.014314 -0.007508 -0.243824
```

```
summary(ret.mat)
```

```
##       MSFT               FMAGX                SBUX
##  Min.   :-0.4209   Min.   :-0.26430   Min.   :-0.4797
##  1st Qu.:-0.0555   1st Qu.:-0.02293   1st Qu.:-0.0488
##  Median : 0.0070   Median : 0.00701   Median : 0.0182
##  Mean   : 0.0048   Mean   :-0.00234   Mean   : 0.0113
##  3rd Qu.: 0.0590   3rd Qu.: 0.03625   3rd Qu.: 0.0868
##  Max.   : 0.3420   Max.   : 0.13024   Max.   : 0.2773
```

(a). Compute estimates of IID normal model parameters

```
muhat.vals = apply(ret.mat, 2, mean)
muhat.vals
```

```
##      MSFT     FMAGX      SBUX
##  0.004846 -0.002342  0.011318
```

```
sigma2hat.vals = apply(ret.mat, 2, var)
sigma2hat.vals
```

```
##     MSFT    FMAGX     SBUX
## 0.011206 0.003705 0.014254
```

```
sigmahat.vals = apply(ret.mat, 2, sd)
sigmahat.vals
```

```
##    MSFT   FMAGX    SBUX
## 0.10586 0.06086 0.11939
```

```
cov.mat = var(ret.mat)
cov.mat
```

```
##           MSFT    FMAGX     SBUX
## MSFT  0.011206 0.003519 0.004235
## FMAGX 0.003519 0.003705 0.003189
## SBUX  0.004235 0.003189 0.014254
```

```
cor.mat = cor(ret.mat)
cor.mat
```

```
##         MSFT  FMAGX   SBUX
## MSFT  1.0000 0.5461 0.3351
## FMAGX 0.5461 1.0000 0.4388
## SBUX  0.3351 0.4388 1.0000
```

```
covhat.vals = cov.mat[lower.tri(cov.mat)]
rhohat.vals = cor.mat[lower.tri(cor.mat)]
names(covhat.vals) <- names(rhohat.vals) <-
  c("MSFT,FMAGX","MSFT,SBUX","FMAGX,SBUX")
covhat.vals
```

```
## MSFT,FMAGX  MSFT,SBUX FMAGX,SBUX
##   0.003519   0.004235   0.003189
```

```
rhohat.vals
```

```
## MSFT,FMAGX  MSFT,SBUX FMAGX,SBUX
##     0.5461     0.3351     0.4388
```

```
cbind(muhat.vals,sigma2hat.vals,sigmahat.vals)
```

```
##       muhat.vals sigma2hat.vals sigmahat.vals
## MSFT    0.004846       0.011206       0.10586
## FMAGX  -0.002342       0.003705       0.06086
## SBUX    0.011318       0.014254       0.11939
```

```
cbind(covhat.vals,rhohat.vals)
```

```
##            covhat.vals rhohat.vals
## MSFT,FMAGX    0.003519      0.5461
## MSFT,SBUX     0.004235      0.3351
## FMAGX,SBUX    0.003189      0.4388
```

Microsoft stock is less volatile than Starbucks but more than FMAGX, as shown by its variance. Fidelity is relatively stable with low variance but shows a negative mu value. Starbucks has the highest mu and the highest variance, suggesting room to grow.

(b),(c). Compute standard errors for estimated parameters

estimated standard error for mean

```
nobs = nrow(ret.mat)
nobs
```

```
## [1] 143
```

```
se.muhat = sigmahat.vals/sqrt(nobs)
se.muhat
```

```
##     MSFT    FMAGX     SBUX
## 0.008853 0.005090 0.009984
```

```
cbind(muhat.vals,se.muhat)
```

```
##       muhat.vals se.muhat
## MSFT    0.004846 0.008853
## FMAGX  -0.002342 0.005090
## SBUX    0.011318 0.009984
```

compute approx 95% confidence intervals

```
mu.lower = muhat.vals - 2*se.muhat
mu.upper = muhat.vals + 2*se.muhat
cbind(mu.lower,mu.upper)
```

```
##       mu.lower mu.upper
## MSFT  -0.01286 0.022551
## FMAGX -0.01252 0.007838
## SBUX  -0.00865 0.031286
```

compute estimated standard errors for variance and sd

```
se.sigma2hat = sigma2hat.vals/sqrt(nobs/2)
se.sigma2hat
```

```
##      MSFT     FMAGX      SBUX
## 0.0013253 0.0004381 0.0016858
```

```
se.sigmahat = sigmahat.vals/sqrt(2*nobs)
se.sigmahat
```

```
##     MSFT    FMAGX     SBUX
## 0.006260 0.003599 0.007060
```

```
cbind(sigma2hat.vals,se.sigma2hat)
```

```
##       sigma2hat.vals se.sigma2hat
## MSFT        0.011206    0.0013253
## FMAGX       0.003705    0.0004381
## SBUX        0.014254    0.0016858
```

```
cbind(sigmahat.vals,se.sigmahat)
```

```
##       sigmahat.vals se.sigmahat
## MSFT        0.10586    0.006260
## FMAGX       0.06086    0.003599
## SBUX        0.11939    0.007060
```

compute approx 95% confidence intervals

```
sigma2.lower = sigma2hat.vals - 2*se.sigma2hat
sigma2.upper = sigma2hat.vals + 2*se.sigma2hat
cbind(sigma2.lower,sigma2.upper)
```

```
##       sigma2.lower sigma2.upper
## MSFT      0.008556     0.013857
## FMAGX     0.002828     0.004581
## SBUX      0.010883     0.017626
```

```
sigma.lower = sigmahat.vals - 2*se.sigmahat
sigma.upper = sigmahat.vals + 2*se.sigmahat
cbind(sigma.lower,sigma.upper)
```

```
##       sigma.lower sigma.upper
## MSFT      0.09334     0.11838
## FMAGX     0.05367     0.06806
## SBUX      0.10527     0.13351
```

```
# compute estimated standard errors for correlation
se.rhohat = (1-rhohat.vals^2)/sqrt(nobs)
se.rhohat
```

```
## MSFT,FMAGX  MSFT,SBUX FMAGX,SBUX
##    0.05869    0.07423    0.06752
```

```
cbind(rhohat.vals,se.rhohat)
```

```
##            rhohat.vals se.rhohat
## MSFT,FMAGX      0.5461   0.05869
## MSFT,SBUX       0.3351   0.07423
## FMAGX,SBUX      0.4388   0.06752
```

```
# compute approx 95% confidence intervals
rho.lower = rhohat.vals - 2*se.rhohat
rho.upper = rhohat.vals + 2*se.rhohat
cbind(rho.lower,rho.upper)
```

```
##            rho.lower rho.upper
## MSFT,FMAGX    0.4287    0.6635
## MSFT,SBUX     0.1866    0.4836
## FMAGX,SBUX    0.3038    0.5739
```

Since we are using a 95% confidence interval, we can say that we are 95% confident that mu, sigma^2, sigma, and correlation are between the above confidence intervals.

(d). Compute 5% and 1% Value at Risk

```r
# function to compute Value-at-Risk
# note: default values are selected for
# the probability level (p) and the initial
# wealth (w). These values can be changed
# when calling the function. Highlight the entire
# function, right click and select run line or selection

Value.at.Risk = function(x,p=0.05,w=100000) {
  x = as.matrix(x)
  q = apply(x, 2, mean) + apply(x, 2, sd)*qnorm(p)
  VaR = (exp(q) - 1)*w
  VaR
}
# 5% and 1% VaR estimates based on W0 = 100000

Value.at.Risk(ret.mat,p=0.05,w=100000)
```

```
##   MSFT  FMAGX   SBUX
## -15573  -9738 -16895
```

```r
Value.at.Risk(ret.mat,p=0.01,w=100000)
```

```
##   MSFT  FMAGX   SBUX
## -21449 -13406 -23389
```

Fidelity has the lowest VaR, consistent with its low variance. Microsoft has the median VaR, consistent with is median variance. Starbucks has the highest VaR, consistent with its high variance.