HW4_TS

```
rm(list=ls())
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 3.6.3
```

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

```
## Warning: package 'xts' was built under R version 3.6.3
```

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

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

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

```
## Loading required package: TTR
```

```
## Warning: package 'TTR' was built under R version 3.6.3
```

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

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.6.3
```

```
library(fGarch)
```

```
## Warning: package 'fGarch' was built under R version 3.6.3
```

```
## Loading required package: timeDate
```

```
## Loading required package: timeSeries
```

```
## Warning: package 'timeSeries' was built under R version 3.6.3
```

```
##
## Attaching package: 'timeSeries'
```

```
## The following object is masked from 'package:zoo':
##
##     time<-
```

```
## Loading required package: fBasics
```

```
## Warning: package 'fBasics' was built under R version 3.6.3
```

```
##
## Attaching package: 'fBasics'
```

```
## The following object is masked from 'package:TTR':
##
##     volatility
```

```
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```
library(TSA)
```

```
##
## Attaching package: 'TSA'
```

```
## The following objects are masked from 'package:timeDate':
##
##     kurtosis, skewness
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
```

```
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(rugarch)
```

```
## Warning: package 'rugarch' was built under R version 3.6.3
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
fname <- file.choose()                # choose TSLA.csv
data <- read.csv(fname)
data <- log(data[,2])
data.ts <- ts(data,start=c(1,1,2012),freq=52)
data.growth = diff(data.ts)
#ts.plot(data.growth, ylab = 'The growth rate of TSLA')
```

Q1. ARIMA(p,d,q)

```
#ARIMA order
test_modelA <- function(p,d,q){
  mod = arima(data.growth, order=c(p,d,q), method="ML")
  current.aic = AIC(mod)
  df = data.frame(p,d,q,current.aic)
  names(df) <- c("p","d","q","AIC")
  print(paste(p,d,q,current.aic,sep=" "))
  return(df)
}

orders = data.frame(Inf,Inf,Inf,Inf)
names(orders) <- c("p","d","q","AIC")

for (p in 0:4){
  for (d in 0:2){
    for (q in 0:4) {
      possibleError <- tryCatch(
        orders<-rbind(orders,test_modelA(p,d,q)),
        error=function(e) e
      )
      if(inherits(possibleError, "error")) next
    }
  }
}
```

```
## [1] "0 0 0 -1001.95153390885"
## [1] "0 0 1 -999.976533799224"
## [1] "0 0 2 -998.781091149895"
## [1] "0 0 3 -997.215363657174"
## [1] "0 0 4 -997.204708031013"
## [1] "0 1 0 -743.007531101551"
## [1] "0 1 1 -993.848790218804"
## [1] "0 1 2 -991.848861232875"
## [1] "0 1 3 -990.468044137523"
## [1] "0 1 4 -988.790283849027"
## [1] "0 2 0 -326.000280490328"
## [1] "0 2 1 -732.121119803844"
## [1] "0 2 2 -974.223227676329"
## [1] "0 2 3 -972.306132578829"
## [1] "0 2 4 -970.722024644068"
## [1] "1 0 0 -999.978980972446"
## [1] "1 0 1 -999.143218168354"
## [1] "1 0 2 -997.816357940132"
## [1] "1 0 3 -995.877880742814"
## [1] "1 0 4 -998.195398558787"
## [1] "1 1 0 -859.69941165643"
## [1] "1 1 1 -991.848866912499"
```

```
## Warning in stats::arima(x = x, order = order, seasonal = seasonal, xreg =
## xreg, : possible convergence problem: optim gave code = 1
```

```
## [1] "1 1 2 -990.743965036962"
## [1] "1 1 3 -989.130917058461"
## [1] "1 1 4 -986.482092353511"
## [1] "1 2 0 -556.086756918644"
## [1] "1 2 1 -847.645839092268"
## [1] "1 2 2 -972.253440024993"
## [1] "1 2 3 -971.223059964223"
## [1] "1 2 4 -969.973995873642"
## [1] "2 0 0 -998.913427351276"
## [1] "2 0 1 -997.790707395626"
## [1] "2 0 2 -996.385041232001"
## [1] "2 0 3 -994.40479900847"
## [1] "2 0 4 -1002.0935320065"
## [1] "2 1 0 -901.028953437384"
## [1] "2 1 1 -990.553350539889"
## [1] "2 1 2 -989.130422014283"
## [1] "2 1 3 -987.744041095685"
## [1] "2 1 4 -985.895213595449"
## [1] "2 2 0 -654.99229368305"
## [1] "2 2 1 -888.290921914517"
## [1] "2 2 2 -970.644415138354"
```

```
## Warning in log(s2): NaNs produced
```

```
## [1] "2 2 3 -969.330067828764"
## [1] "2 2 4 -967.134770596984"
## [1] "3 0 0 -997.256885851907"
## [1] "3 0 1 -995.841274461945"
## [1] "3 0 2 -994.392863657056"
```

```
## Warning in log(s2): NaNs produced
```

```
## [1] "3 0 3 -992.855109486565"
## [1] "3 0 4 -1000.31583671188"
## [1] "3 1 0 -930.317317023143"
## [1] "3 1 1 -988.795182282954"
## [1] "3 1 2 -987.135492150675"
## [1] "3 1 3 -985.869986436799"
```

```
## Warning in stats::arima(x = x, order = order, seasonal = seasonal, xreg =
## xreg, : possible convergence problem: optim gave code = 1
```

```
## [1] "3 1 4 -984.312930853974"
## [1] "3 2 0 -743.08402119346"
## [1] "3 2 1 -917.006672695157"
## [1] "3 2 2 -968.771706666303"
## [1] "3 2 3 -967.70219128848"
```

```
## Warning in log(s2): NaNs produced
```

```
## [1] "3 2 4 -965.256954878143"
## [1] "4 0 0 -997.275179594166"
## [1] "4 0 1 -996.907630304939"
## [1] "4 0 2 -1003.80278179236"
```

```
## Warning in stats::arima(x = x, order = order, seasonal = seasonal, xreg =
## xreg, : possible convergence problem: optim gave code = 1
```

```
## [1] "4 0 3 -1001.77813944674"
## [1] "4 0 4 -998.32022338631"
## [1] "4 1 0 -932.817470576436"
## [1] "4 1 1 -988.643859271108"
## [1] "4 1 2 -988.300379812244"
## [1] "4 1 3 -995.302905244194"
```

```
## Warning in stats::arima(x = x, order = order, seasonal = seasonal, xreg =
## xreg, : possible convergence problem: optim gave code = 1
```

```
## [1] "4 1 4 -993.345114751789"
## [1] "4 2 0 -768.824540958537"
## [1] "4 2 1 -919.294422710676"
## [1] "4 2 2 -968.663006972023"
```

```
## Warning in log(s2): NaNs produced
```

```
## [1] "4 2 3 -968.203943910775"
## [1] "4 2 4 -967.606352267544"
```

```
orders <- orders[order(-orders$AIC),]
tail(orders)
```

```
##    p d q       AIC
## 17 1 0 0  -999.979
## 51 3 0 4 -1000.316
## 65 4 0 3 -1001.778
## 2  0 0 0 -1001.952
## 36 2 0 4 -1002.094
## 64 4 0 2 -1003.803
```

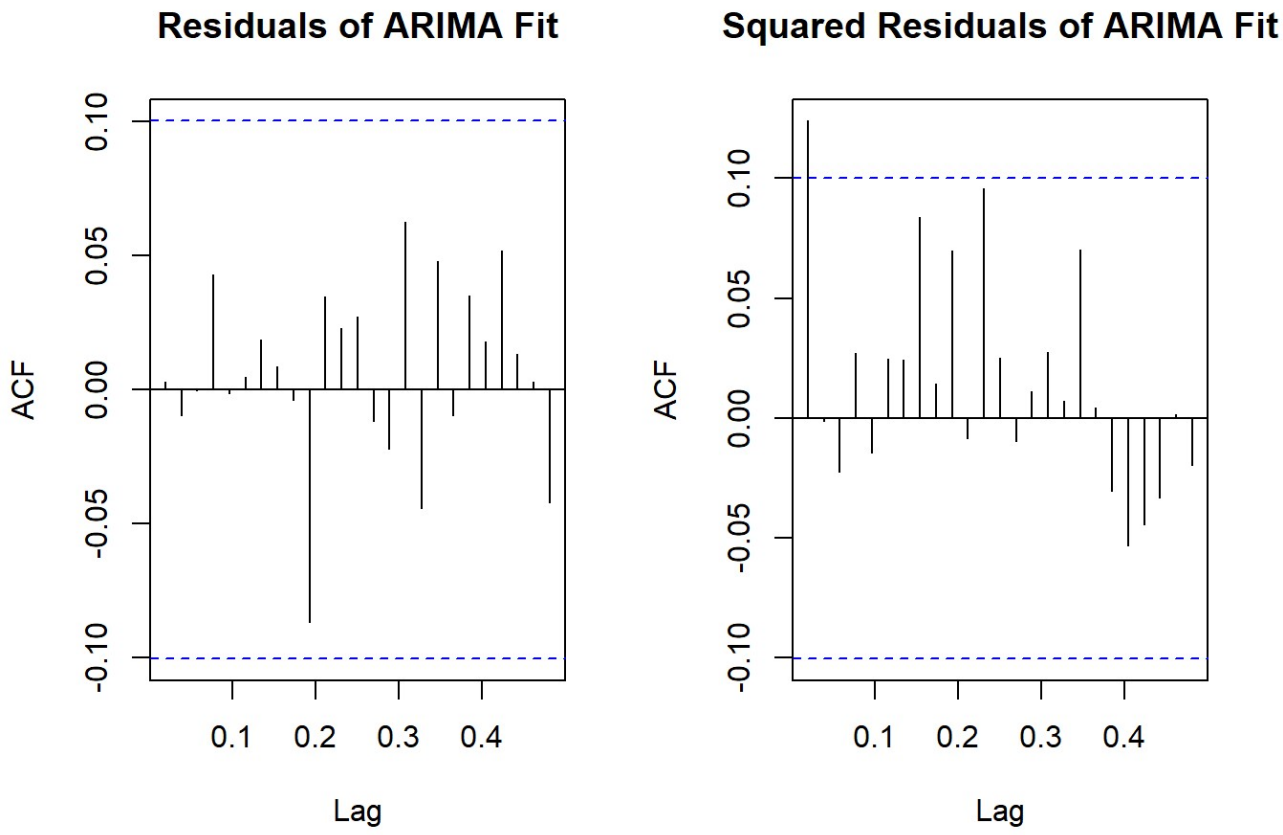
```
# 4 0 2

final.arima = arima(data.growth, order=c(4,0,2))
final.arima
```

```
##
## Call:
## arima(x = data.growth, order = c(4, 0, 2))
##
## Coefficients:
##           ar1      ar2     ar3     ar4     ma1     ma2  intercept
##       -1.2937  -0.8501  0.1048  0.1193  1.3206  0.9363     0.0057
## s.e.   0.0652   0.1153  0.0848  0.0568  0.0430  0.0840     0.0036
##
## sigma^2 estimated as 0.004049:  log likelihood = 509.9,  aic = -1005.8
```

Residual Analysis

```
resids = resid(final.arima)
par(mfrow=c(1,2))
acf(resids, main = 'Residuals of ARIMA Fit')
acf(resids^2, main = "Squared Residuals of ARIMA Fit")
```



Answer: For residuals, ACF plot looks like white noise with sample autocorrelation being small within confidence band for lags one and higher, thus, it seems to have no correlation.

For squared residuals, ACF plot indicated possible autocorrelation such as lag(1) is out of confidence band, which means it is not white noise.

```
# For serial correlation
Box.test(resids, lag=7, type='Ljung', fitdf=6)
```

```
##
##  Box-Ljung test
##
## data:  resids
## X-squared = 0.89344, df = 1, p-value = 0.3445
```

```
Box.test((resids)^2, lag=7, type='Ljung', fitdf=6)
```

```
##
##  Box-Ljung test
##
## data:  (resids)^2
## X-squared = 6.9288, df = 1, p-value = 0.008482
```

Answer: For residuals, p value of Box-Ljung test is big (>0.05), which can not reject null hypothesis of no correlation, so there has no correlation.

For squared residuals, p value of Box-Ljung test is small (<0.05), which reject null hypothesis of no correlation, so there has correlation.

Q2. ARMA(p,q)-GARCH(m,n) joint model

```
# ARIMA-GARCH GARCH order
# GARCH update
test_modelAGG <- function(m,n){
  spec = ugarchspec(variance.model=list(garchOrder=c(m,n)),
                    mean.model=list(armaOrder=c(4,2),
                      include.mean=T), distribution.model="std")
                    fit = ugarchfit(spec, data.growth, solver = 'hybrid')
                    current.bic = infocriteria(fit)[2]
                    df = data.frame(m,n,current.bic)
                    names(df) <- c("m","n","BIC")
                    print(paste(m,n,current.bic,sep=" "))
                    return(df)
}

orders = data.frame(Inf,Inf,Inf)
names(orders) <- c("m","n","BIC")


for (m in 0:2){
     for (n in 0:2){
          possibleError <- tryCatch(
            orders<-rbind(orders,test_modelAGG(m,n)),
            error=function(e) e
          )
          if(inherits(possibleError, "error")) next
          }
}
```

```
## [1] "0 0 -2.56390336438806"
## [1] "0 1 -2.54828047735607"
## [1] "0 2 -2.52640253908174"
## [1] "1 0 -2.44543643661524"
## [1] "1 1 -2.54946776356549"
## [1] "1 2 -2.58530066319158"
## [1] "2 0 -2.53658372933802"
## [1] "2 1 -2.53390384643259"
## [1] "2 2 -2.51887509720818"
```

```
orders <- orders[order(-orders$BIC),]
tail(orders)  # 1,2
```

```
##   m n       BIC
## 9 2 1 -2.533904
## 8 2 0 -2.536584
## 3 0 1 -2.548280
## 6 1 1 -2.549468
## 2 0 0 -2.563903
## 7 1 2 -2.585301
```

```
# ARMA update
# ARIMA-GARCH ARIMA order
test_modelAGA <- function(p,q){
  spec = ugarchspec(variance.model=list(garchOrder=c(1,2)),
    mean.model=list(armaOrder=c(p,q),
                    include.mean=T), distribution.model="std")
    fit = ugarchfit(spec, data.growth, solver = 'hybrid')
    current.bic = infocriteria(fit)[2]
    df = data.frame(p,q,current.bic)
    names(df) <- c("p","q","BIC")
    print(paste(p,q,current.bic,sep=" "))
    return(df)
}

orders = data.frame(Inf,Inf,Inf)
names(orders) <- c("p","q","BIC")

for (p in 0:4){
     for (q in 0:4){
          possibleError <- tryCatch(
            orders<-rbind(orders,test_modelAGA(p,q)),
            error=function(e) e
          )
          if(inherits(possibleError, "error")) next
          }
}
```

```
## [1] "0 0 -2.59414251605655"
## [1] "0 1 -2.57906022121838"
## [1] "0 2 -2.56455428020891"
## [1] "0 3 -2.54865607784683"
## [1] "0 4 -2.53911915443343"
## [1] "1 0 -2.57909651003785"
## [1] "1 1 -2.56726202872953"
## [1] "1 2 -2.55239126566091"
## [1] "1 3 -2.53740320989982"
## [1] "1 4 -2.53246708497704"
## [1] "2 0 -2.56431658368596"
## [1] "2 1 -2.54888609334393"
## [1] "2 2 -2.55967439946073"
## [1] "2 3 -2.53365148310193"
## [1] "2 4 -2.52790492979997"
## [1] "3 0 -2.54958786221549"
## [1] "3 1 -2.53500788485522"
## [1] "3 2 -2.52197398267646"
## [1] "3 3 -2.55488673412531"
## [1] "3 4 -2.51304792718752"
## [1] "4 0 -2.53980325581717"
## [1] "4 1 -2.52880332724522"
## [1] "4 2 -2.58530066319158"
## [1] "4 3 -2.58191370189517"
## [1] "4 4 -2.63168512525666"
```

```
orders <- orders[order(-orders$BIC),]
tail(orders)  # 4,4
```

```
##    p q       BIC
## 3  0 1 -2.579060
## 7  1 0 -2.579097
## 25 4 3 -2.581914
## 24 4 2 -2.585301
## 2  0 0 -2.594143
## 26 4 4 -2.631685
```

```
# Final Garch Order
# GARCH update
test_modelAGG <- function(m,n){
  spec = ugarchspec(variance.model=list(garchOrder=c(m,n)),
                    mean.model=list(armaOrder=c(4,4),
                      include.mean=T), distribution.model="std")
                    fit = ugarchfit(spec, data.growth, solver = 'hybrid')
                    current.bic = infocriteria(fit)[2]
                    df = data.frame(m,n,current.bic)
                    names(df) <- c("m","n","BIC")
                    print(paste(m,n,current.bic,sep=" "))
                    return(df)
}

orders = data.frame(Inf,Inf,Inf)
names(orders) <- c("m","n","BIC")


for (m in 0:2){
     for (n in 0:2){
          possibleError <- tryCatch(
            orders<-rbind(orders,test_modelAGG(m,n)),
            error=function(e) e
          )
          if(inherits(possibleError, "error")) next
          }
}
```

```
## [1] "0 1 -2.5183468941466"
## [1] "0 2 -2.54579358676056"
## [1] "1 0 -2.55958143101487"
## [1] "1 1 -2.6501023714207"
## [1] "1 2 -2.63168512525666"
## [1] "2 0 -2.3827911407877"
## [1] "2 1 -2.50353044553978"
## [1] "2 2 -2.57430872227522"
```

```
orders <- orders[order(-orders$BIC),]
tail(orders)   # 1,1
```

```
##   m n       BIC
## 2 0 1 -2.518347
## 3 0 2 -2.545794
## 4 1 0 -2.559581
## 9 2 2 -2.574309
## 6 1 2 -2.631685
## 5 1 1 -2.650102
```

```
# Final Arima-Garch Model
spec = ugarchspec(variance.model=list(garchOrder=c(1,1)),
                  mean.model=list(armaOrder=c(4,4),
                                  include.mean=T), distribution.model="std")
final.model = garchFit(~ arma(4,4)+ garch(1,1), data=data.growth, trace = FALSE)
```

```
## Warning in arima(.series$x, order = c(u, 0, v), include.mean = include.mean):
## possible convergence problem: optim gave code = 1
```

```
summary(final.model)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(4, 4) + garch(1, 1), data = data.growth,
##     trace = FALSE)
##
## Mean and Variance Equation:
##  data ~ arma(4, 4) + garch(1, 1)
## <environment: 0x000000002bee4400>
##  [data = data.growth]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##         mu          ar1          ar2          ar3          ar4          ma1
##  0.00426134  -0.75959668  -0.19646790   0.57980707   0.09368805   0.76189873
##         ma2          ma3          ma4        omega       alpha1        beta1
##  0.20718093  -0.54108488   0.01292577   0.00026663   0.06429873   0.87028504
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.0042613   0.0049026    0.869  0.38474
## ar1    -0.7595967   0.0848621   -8.951  < 2e-16 ***
## ar2    -0.1964679   0.1382944   -1.421  0.15542
## ar3     0.5798071   0.1484487    3.906 9.39e-05 ***
## ar4     0.0936881   0.1006034    0.931  0.35172
## ma1     0.7618987   0.1004590    7.584 3.35e-14 ***
## ma2     0.2071809   0.1584895    1.307  0.19114
## ma3    -0.5410849   0.1672439   -3.235  0.00122 **
## ma4     0.0129258   0.1073348    0.120  0.90415
## omega   0.0002666   0.0001437    1.856  0.06352 .
## alpha1  0.0642987   0.0311171    2.066  0.03880 *
## beta1   0.8702850   0.0469443   18.539  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  522.7252    normalized:  1.368391
##
## Description:
##  Wed Apr 08 14:59:46 2020 by user: Sealion
##
```

```
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  55.67742  8.124612e-13
##  Shapiro-Wilk Test  R    W      0.9822455 0.0001214266
##  Ljung-Box Test     R    Q(10)  4.683363  0.9113019
##  Ljung-Box Test     R    Q(15)  5.972257  0.98021
##  Ljung-Box Test     R    Q(20)  8.220803  0.9903032
##  Ljung-Box Test     R^2  Q(10)  2.410858  0.992112
##  Ljung-Box Test     R^2  Q(15)  5.080218  0.9914252
##  Ljung-Box Test     R^2  Q(20)  6.10085   0.9987543
##  LM Arch Test       R    TR^2   4.568772  0.9708563
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -2.673954 -2.550014 -2.675849 -2.624784
```

Answer: Equation for ARMA(4,4)-GARCH(1,1)

Y_t = 0.00426134 - 0.75959668*Y_t-1 - 0.1964679*Y_t-2 + 0.57980707*Y_t-3 + 0.09368805*Y_t-4 + Z_t + 0.76189873*Z_t-1 + 0.20718093* Z_t-2 - 0.54108488*Z_t-3 + 0.01292577*Z_t-4

(a_t)^2 = 0.00026663 + 0.06429873*(Z_t-1)^2 + 0.87028504* (a_t-1)^2

Y_t is modeled by ARMA(4,4) with AR coefficient (ar1,ar2,ar3,ar4) and MA coefficient (ma1,ma2,ma3,ma4). Z_t is modeled by a GARCH(1,1) model, which means that the variance of Z_t sigma t squared follows that formula of a GARCH(1,1) model, the coefficients are alpha1 and beta1.

```
# Goodness of fit (evaluation of model)
# summary(final.model)
```

Answer: Evaluation the joint model with goodness of fit. From joint model of ARMA(4,4)-GARCH(1,1), p-values of coefficients of ar1, ar3, ma1, ma3, alpha1 and beta1 are small, which means statistically significant. In addition, for Ljung-Box Test, the p-value of residuals is big, so we can not reject the null of uncorrelated residuals, which means uncorrelated residuals. The p-value of squared residuals are also big, so we can not reject the null hypothesis of uncorrelated squared residuals, which means uncorrelated squared residuals.

Q3. Forecasting

```
#  Predictions
## Prediction of the return time series
n=length(data.growth)                                    # n = 382
data.growth.test = data.growth[343:n]                    # length = 40
data.growth.train =  data.growth[-c(343:n)]

nfore = length(data.growth.test)                         # length = 40 (test)
fore.series = NULL

for(f in 1: nfore){
  ## Fit models
  data = data.growth.train
  if(f>=2)
    data = c(data.growth.train, data.growth.test[1:(f-1)])

  final.model = ugarchfit(spec, data, solver = 'hybrid')

  ## Forecast
  fore = ugarchforecast(final.model, n.ahead=1)
  fore.series = c(fore.series, fore@forecast$seriesFor)
}
```

Compute Accuracy Measures

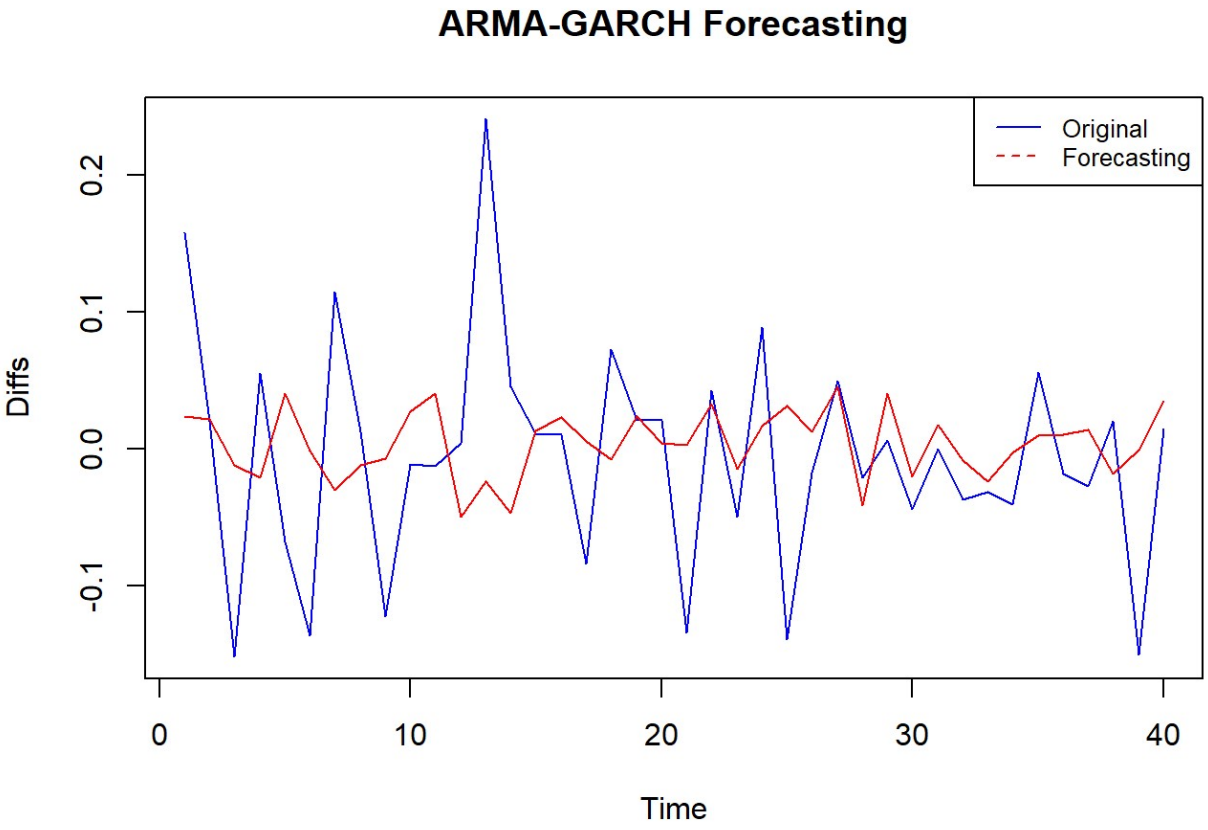
```
### Mean Absolute Prediction Error (MAPE)
100*mean(abs(fore.series - data.growth.test)/abs(data.growth.test))
```

```
## [1] 311.6466
```

```
### Precision Measure (PM)
sum((fore.series - data.growth.test)^2)/sum((data.growth.test-mean(data.growth.test))^
2)
```

```
## [1] 1.136645
```

```
# Overlay the draw of the predicted points on the original original series.
par(mfrow=c(1,1))
{ts.plot(data.growth.test, ylab = 'Diffs', col = 'blue', main='ARMA-GARCH Forecastin
g')
lines(fore.series, col="red")
legend("topright", c('Original', 'Forecasting'), lty=1:2, cex = 0.8, col = c("blue",
'red'))}
```


ARMA-GARCH Forecasting
Original
Forecasting
Diffs
0.2
0.1
0.0
-0.1
0
10
20
30
40
Time