Project Part-1

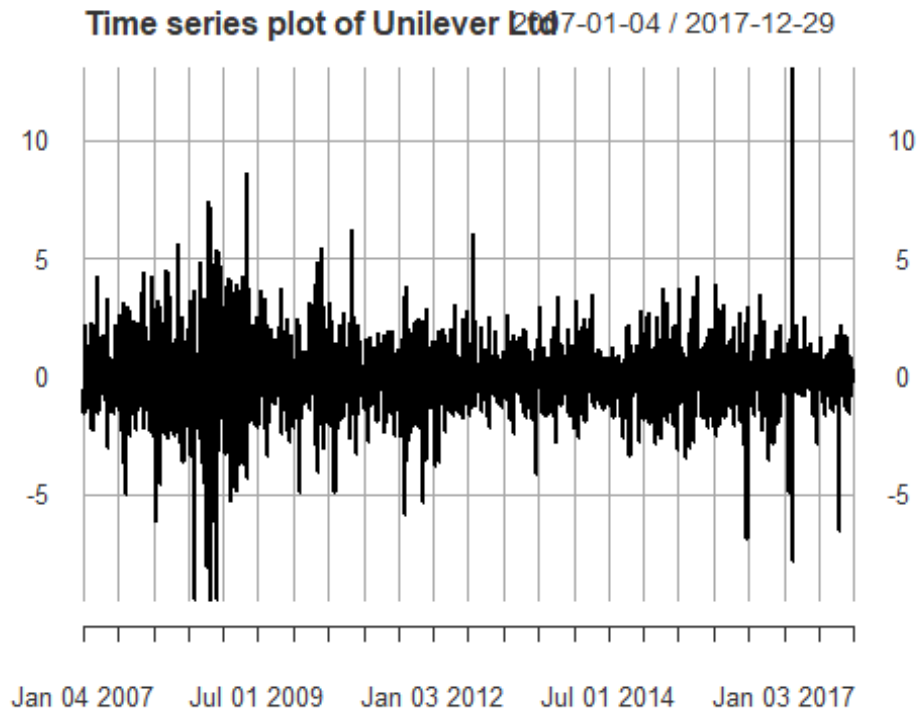
Roop Vankayalapati

2022-12-02

PART – 1

The stock selected was "Unilever"(UL), a multinational consumer goods company

```
##Time Series plot of the stock
##_____________________________________________________________________

plot(Yn, main = "Time series plot of Unilever Ltd")
```

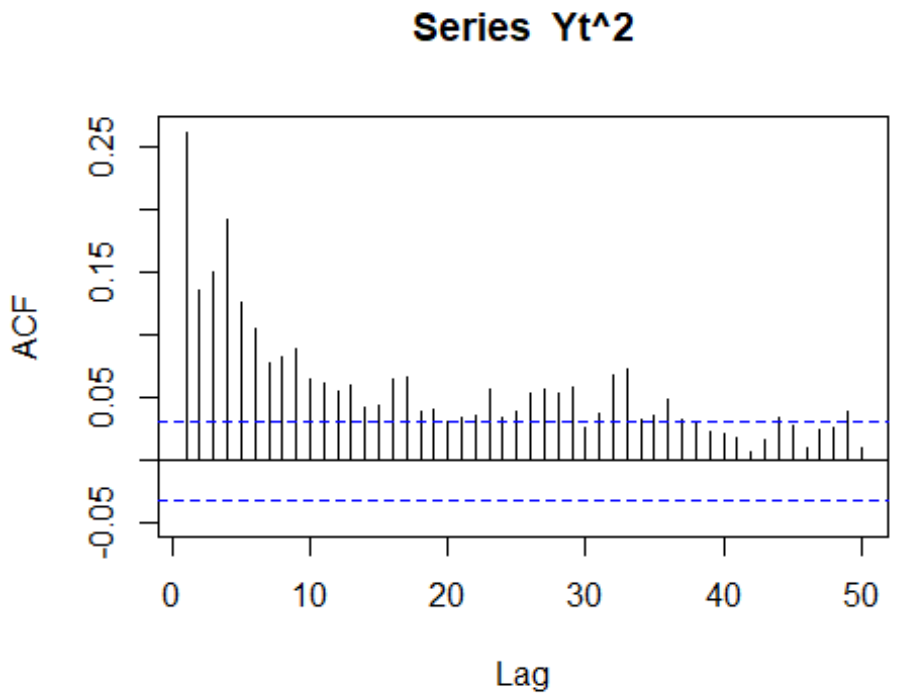



```
##Ljung-Box Test
##_____________________________________________________________________
_____________________________________________________________________
_____________________________________

Box.test(Yn, lag = 4, type = "L")

##
##  Box-Ljung test
##
```

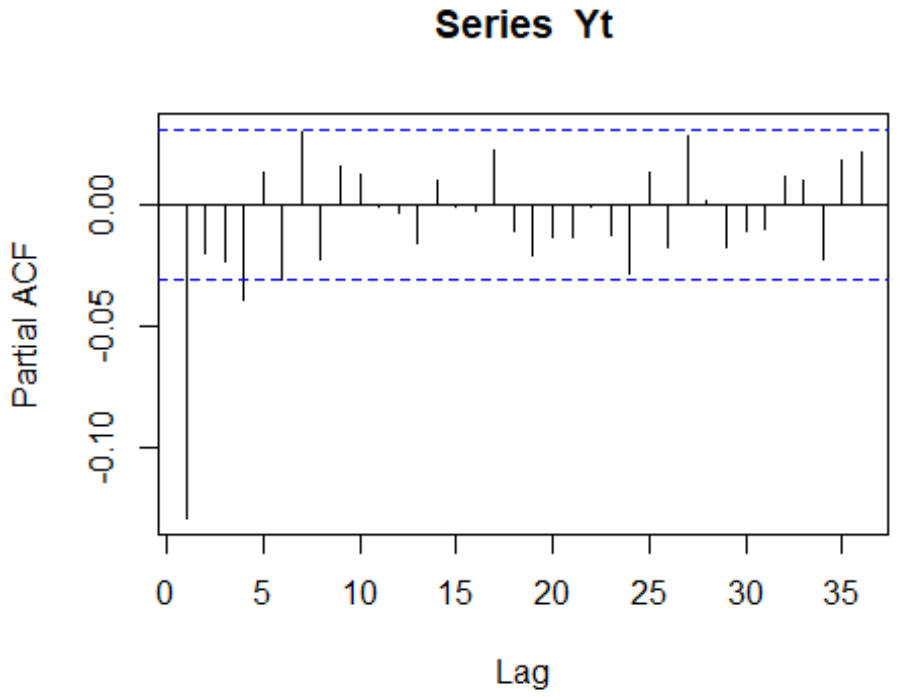
```
## data:  Yn
## X-squared = 28.027, df = 4, p-value = 1.231e-05

lags = 3*(0:5)+1;ps = c()
for(i in 1:length(lags)){
ps = cbind(ps, apply(Yn,2, function(x) Box.test(x,lags[i], "L")$p.val))
}
colnames(ps) = as.character(lags)
cat("\nP values of Ljung-Box Tests with different K:\n");

##
## P values of Ljung-Box Tests with different K:

round(ps,5)

##    1     4       7      10      13      16
## UL 0 1e-05 0.00018 0.00077 0.00282 0.00903
```

The Ljung-Box test on the data gives a p-value of 1.23e-05 which is very less We fail to reject the null hypothesis that data is white noise.

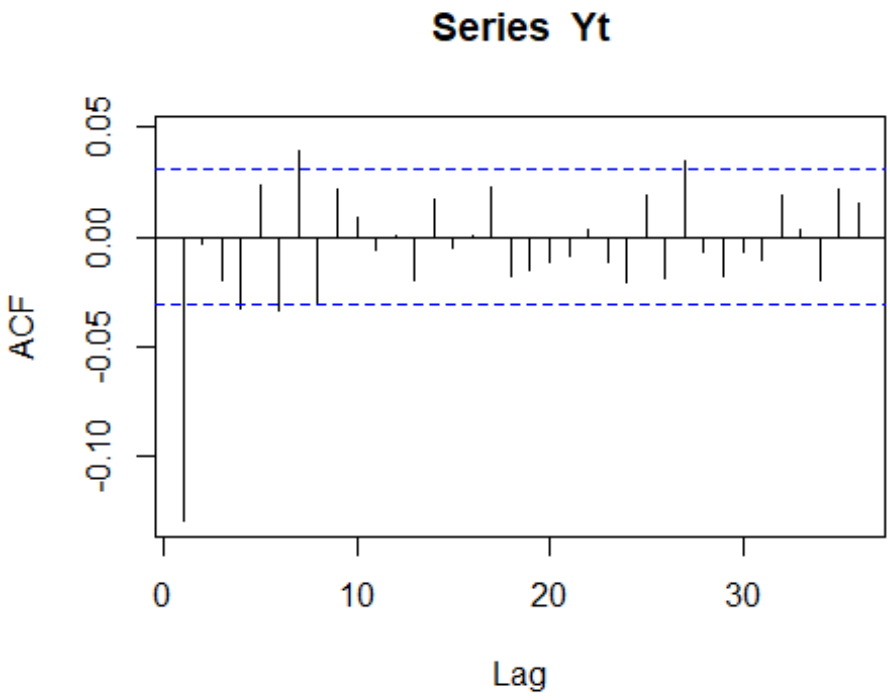
```
##Finding the number of lags from acf and pacf plots
##_____________________________________________________________________

Acf(Yt^2, lag.max = 10*log10(100000))
```




```
pacf(Yt)
```




```
Acf(Yt)
```



The ACF and PACF plots are somewhat similar and show very few significant lags as the correlation quickly decays.

I have used auto.arima() to get an initial idea about the parameters of ARMA(p,q) model. We get the (p,q) parameters as (1,1).

```
##Using auto.arima() on the data set
##_____________________________________________________________________

aic.fit = auto.arima(Yn)
aic.fit

## Series: Yn
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1      ma1    mean
##       0.3840  -0.4806  0.0385
## s.e.  0.1467   0.1393  0.0234
##
## sigma^2 = 2.136:  log likelihood = -4976.59
## AIC=9961.17   AICc=9961.19   BIC=9984.88
```

We need to be sure `auto.arima()` is not leaving out any other simpler solution which has lesser parameters. I have used different methods to filter out few other possible candidate models with different parameters to find out the final model. We have kept the max parameters to be equal to that of `auto.arima()` model

First is to use AICc and BIC test statistics to find out the best model

```
##Using Armia() to find the aic and bic values for sample models
##_____________________________________________________________________

row_names <- c("p=0", "p=1", "p=2")
col_names <- c("q=0", "q=1", "q=2")

aic = c()

for (i in 0:2) {
  for (j in 0:2) {
    aic = c(aic, Arima(Yn, order = c(j,0,i))$aicc)
  }
}

aic_arr = array(aic, dim = c(length(row_names), length(col_names)), dimnames
= list(row_names, col_names))


bic = c()

for (i in 0:2) {
  for (j in 0:2) {
    bic = c(bic, Arima(Yn, order = c(j,0,i))$bic)
  }
}

bic_arr = array(bic, dim = c(length(row_names), length(col_names)), dimnames
= list(row_names, col_names))

cat("The AICc values are : \n")

## The AICc values are :

aic_arr

##          q=0      q=1      q=2
## p=0 9987.351 9963.558 9962.052
## p=1 9965.522 9961.186 9963.236
## p=2 9962.843 9963.159 9965.201

cat("\n\nThe BIC values are : \n")

##
##
## The BIC values are :
```

```
bic_arr
```

```
##           q=0      q=1       q=2
## p=0 9999.198 9981.327  9985.741
## p=1 9983.291 9984.875  9992.844
## p=2 9986.532 9992.767 10000.726
```

From the AICc values, we can say that ARMA(1,1) has the least value but ARMA(0,1) and ARMA(1,0) have lower values of BIC than ARMA(1,1)

We now check the causality and invertability condition for the three ARMA models mentioned above.

```
##Checking the roots of the polynomial to find parameter redundancy
##_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________
```

```
ar.roots = polyroot(c(arma11$coef[c("ar1")], 1))
ma.roots = polyroot(c(arma11$coef[c("ma1")], 1))

cat("AR roots for (1,1):\t", ar.roots, "\tmodulus:", abs(ar.roots)[1]);
```

```
## AR roots for (1,1):   -0.3840131+0i  modulus: 0.3840131
```

```
cat("MA roots for (1,1):\t", ma.roots, "\tmodulus:", abs(ma.roots)[1])
```

```
## MA roots for (1,1):   0.4806338+0i   modulus: 0.4806338
```

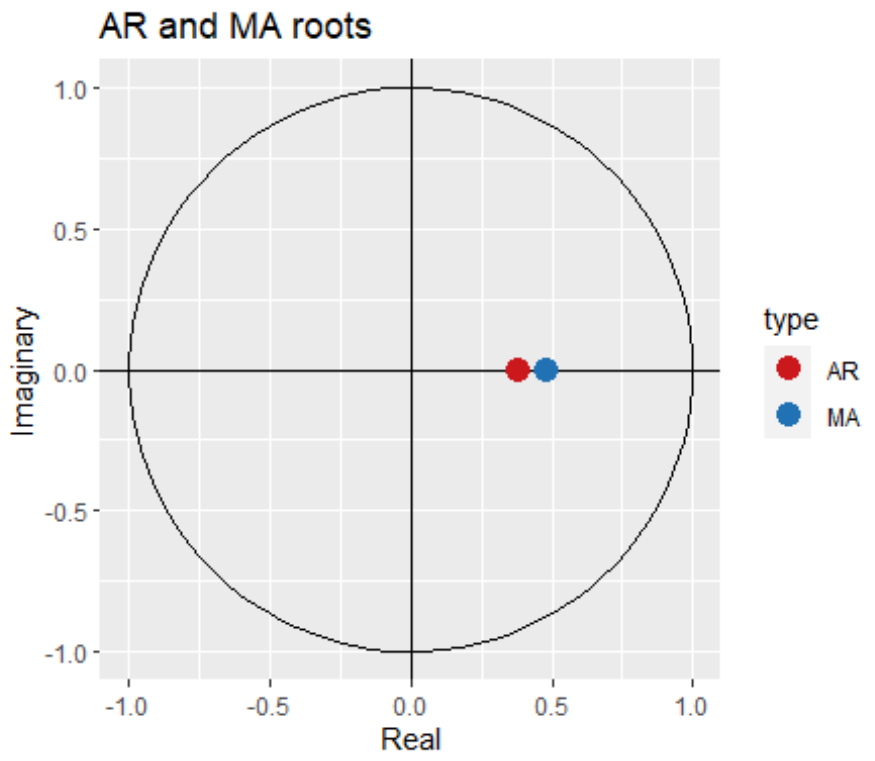
```
ar.roots = polyroot(c(arma10$coef[c("ar1")], 1))
ma.roots = polyroot(c(arma01$coef[c("ma1")], 1))

cat("AR roots for (1,0):\t", ar.roots, "\tmodulus:", abs(ar.roots)[1]);
```

```
## AR roots for (1,0):   0.09257725+0i  modulus: 0.09257725
```

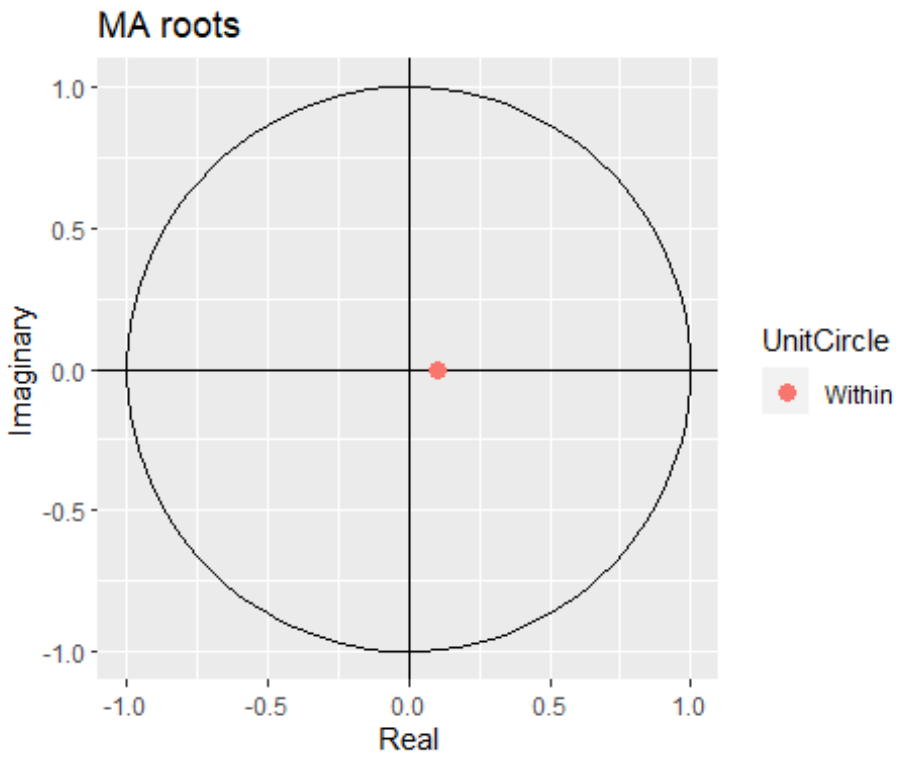
```
cat("MA roots for (0,1):\t", ma.roots, "\tmodulus:", abs(ma.roots)[1])
```

```
## MA roots for (0,1):   0.1003145+0i   modulus: 0.1003145
```

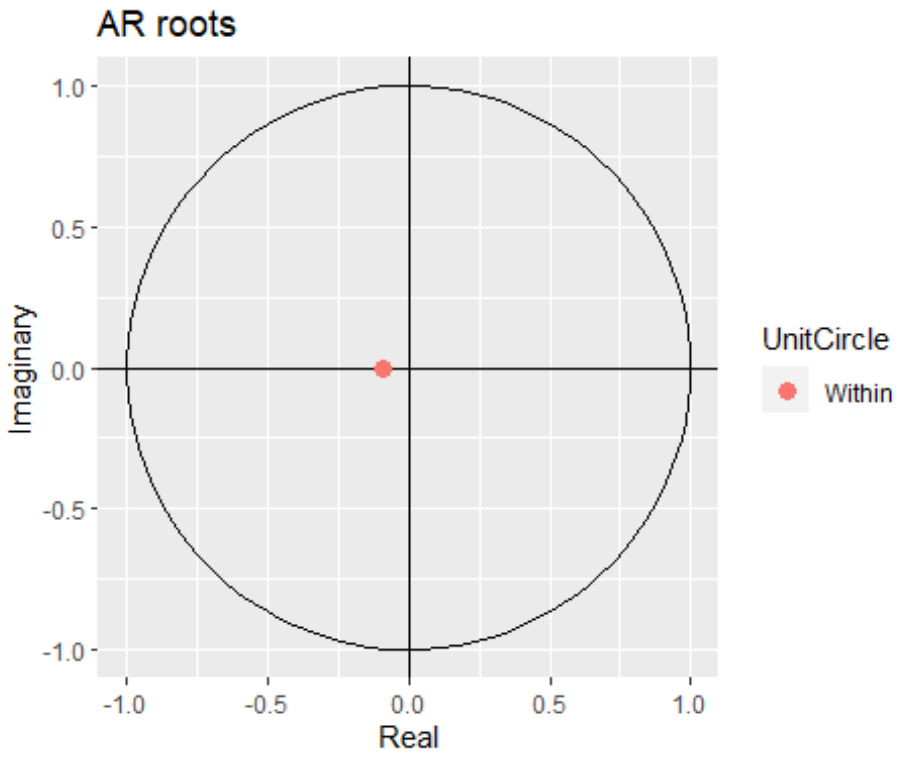
```
autoplot_roots(arma11)
```




```
autoplot_roots(arma01)
```




```
autoplot_roots(arma10)
```



We can see that all the points lie within the unit circle and the causality and invertability of the models can be proven. We can also see that the roots of ARMA(1,1) model are soo close to each other indicating some parameter redundancy. So, we cannot use ARMA(1,1). The next best model is ARMA(0,1) from the AICc and BIC values. This model also has lesser parameters than ARMA(1,1).

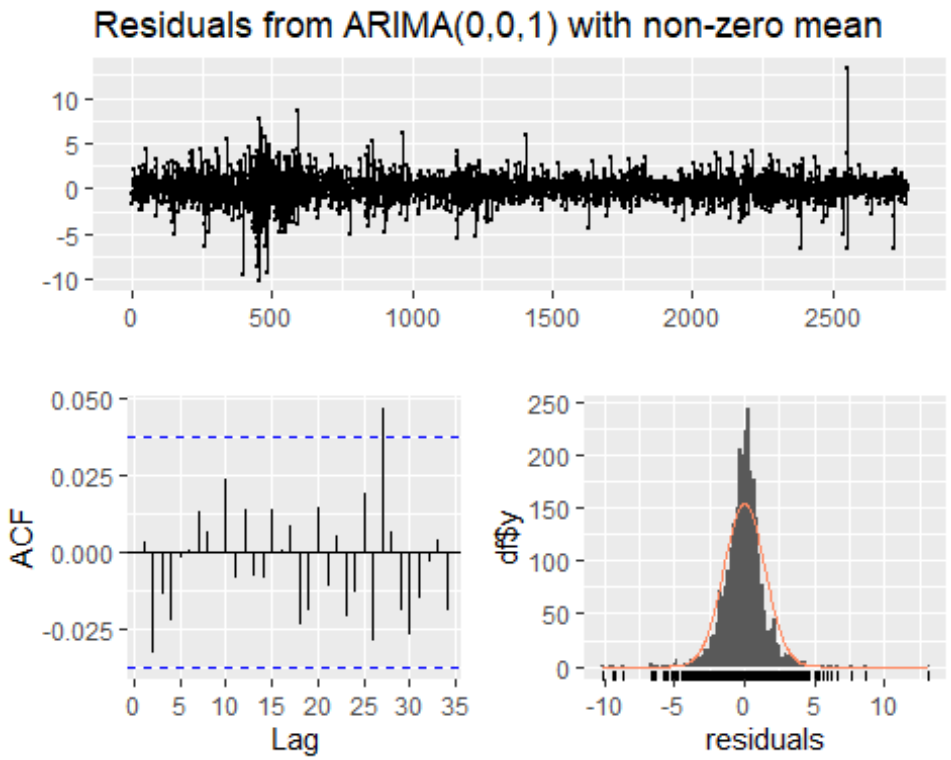
Now performing residual analysis to check the correlations of residuals

```
##Residual analysis using Ljung-Box Test
##_____________________________________________________________________


res11 = arma11$residuals
Box.test(res11, round(log(n))+2, "L", fitdf = 2)

##
##  Box-Ljung test
##
## data:  res11
## X-squared = 3.0623, df = 8, p-value = 0.9304

res10 = arma10$residuals
Box.test(res10, round(log(n))+2, "L", fitdf = 1)

##
##  Box-Ljung test
##
```

```
## data:  res10
## X-squared = 8.9754, df = 9, p-value = 0.4395

res01 = arma01$residuals
Box.test(res01, round(log(n))+1, "L", fitdf = 1)

##
##  Box-Ljung test
##
## data:  res01
## X-squared = 7.2097, df = 9, p-value = 0.6153

cat("\nResidual values for arma01:\n")

##
## Residual values for arma01:

lags = 2:6
ps = c()
for(i in 1:length(lags))
  ps[i] = Box.test(res01,lags[i],"L",1)$p.val
names(ps) = as.character(lags-1); round(ps,4)

##      1      2      3      4      5
## 0.0786 0.1615 0.1703 0.2843 0.4121

cat("\nResidual values for arma10:\n")

##
## Residual values for arma10:

lags = 2:6
ps = c()
for(i in 1:length(lags))
  ps[i] = Box.test(res10,lags[i],"L",1)$p.val
names(ps) = as.character(lags-1); round(ps,4)

##      1      2      3      4      5
## 0.0257 0.0640 0.0779 0.1454 0.2338

cat("\nResidual values for arma11:\n")

##
## Residual values for arma11:

lags = 2:6
ps = c()
for(i in 1:length(lags))
  ps[i] = Box.test(res11,lags[i],"L",1)$p.val
names(ps) = as.character(lags-1); round(ps,4)

##      1      2      3      4      5
## 0.8782 0.9675 0.8835 0.9554 0.9844
```

The p-value of ARMA(0,1) model is so small which proves the null hypothesis that they are indeed from white noise.

```
checkresiduals(arma01)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,1) with non-zero mean
## Q* = 7.2094, df = 9, p-value = 0.6153
##
## Model df: 1.   Total lags used: 10
```

The residuals are plotted over a normal curve with mean = 0 which proves the fit of our model.

Prediction using ARMA(0,1) model or the MA(1) model.

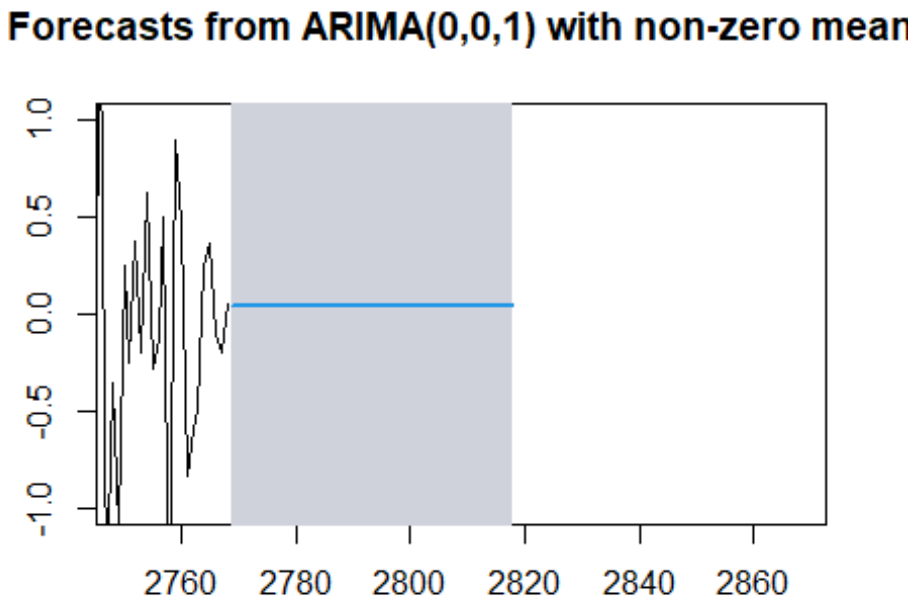
```
##Making the prediction using the forecast() function
##_____________________________________________________________________


pred = predict(arma01, n.ahead = 10)
pred
```

```
## $pred
## Time Series:
## Start = 2769
## End = 2778
## Frequency = 1
##  [1] 0.03909688 0.03820545 0.03820545 0.03820545 0.03820545 0.03820545
##  [7] 0.03820545 0.03820545 0.03820545 0.03820545
##
## $se
## Time Series:
## Start = 2769
## End = 2778
## Frequency = 1
##  [1] 1.462447 1.469787 1.469787 1.469787 1.469787 1.469787 1.469787 1.4697
87
##  [9] 1.469787 1.469787
```

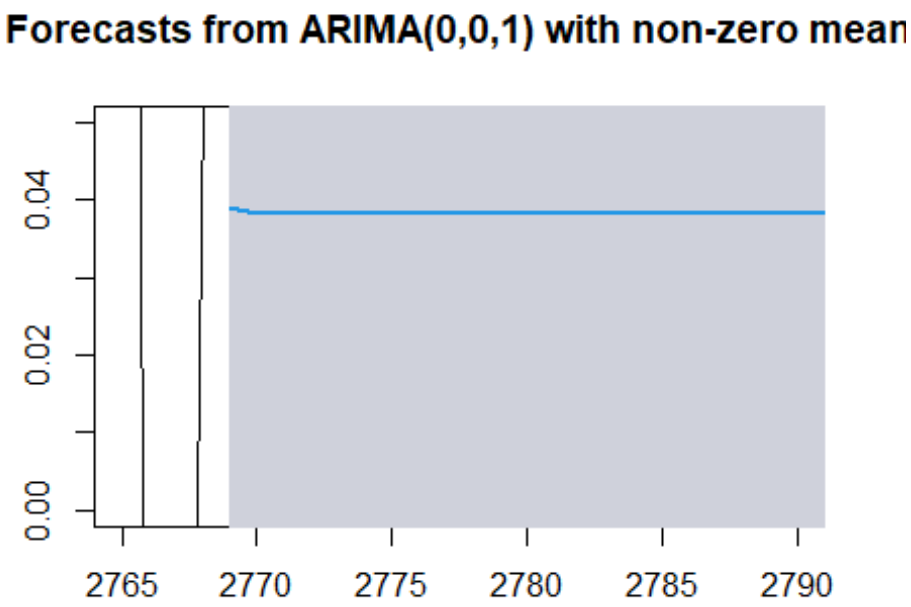
```
fore = forecast(arma01, h = 50, level = c(90, 95))
```

```
##Plotting the predictions and zooming in at the end to check for convergence
##_____________________________________________________________________________
```

```
plot(fore, xlim = c(2750, 2868), ylim = c(-1,1))
```




```
plot(fore, xlim = c(2765, 2790), ylim = c(-0.0001,0.05))
```



We can clearly see from the second plot that the graph converges to the mean value from the model that is 0.385. To further get a clearer picture, we plot the forecasting to see the convergence

We have made two plots to get a better look at the convergence. The first plot shows that the forecasting converges onto a value and the second plot gives a closer look at the convergence and we can clearly see it converging to 0.385 which is the mean.

PART - 2

We now start modeling the GARCH model GARCH(1,1) is said to have the best fot most of the financial data and we start with this distribution to find out the best possible distribution

```
#Checking lowest aic and bic values to find the distribution model

library(rugarch)
ma1 = Arima(Yn, order = c(0,0,1))
spec = ugarchspec(mean.model = list(armaOrder = c(0,1)),
variance.model = list(garchOrder = c(1,1)))
fit = ugarchfit(data=Yn, spec=spec)

at = resid(ma1)
```

```
et = fit@fit$z
dists = c("std", "sstd", "ged", "sged", "norm", "snorm", "nig" ,"jsu")
fits = vector("list", 8)
for (i in 1:8) {
  fits[[i]] = fitdist(dists[i], at)
}

ml = c()
p = c()
for(i in 1:8){
  fit = fits[[i]]
  ml[i] = fit$values[length(fit$values)]
  p[i] = length(fit$pars)
}

aic = 2*ml + 2*p
names(aic) = dists
bic = 2*ml + log(n)*p
names(bic) = dists
rbind(AIC = aic)
```

```
##          std     sstd      ged     sged    norm    snorm      nig      jsu
## AIC 9423.239 9418.088 9452.995 9442.795 9961.54 9957.558 9417.858 9415.103
```

```
cat("\nAIC of residuals from MA(1)+GARCH(1,1) is for the distribution : ", di
sts[which.min(aic)])
```

```
##
## AIC of residuals from MA(1)+GARCH(1,1) is for the distribution :  jsu
```

We now use stardardized residuals “et” to find the best fitting distribution.

```
for (i in 1:8) {
  fits[[i]] = fitdist(dists[i], et)
}

ml = c()
p = c()
for(i in 1:8){
  fit = fits[[i]]
  ml[i] = fit$values[length(fit$values)]
  p[i] = length(fit$pars)
}

aic = 2*ml + 2*p
names(aic) = dists
bic = 2*ml + log(n)*p
```

```
names(bic) = dists
rbind(AIC = aic)
```

```
##          std     sstd      ged     sged     norm    snorm      nig      js
u
## AIC 7524.635 7518.275 7569.738 7561.627 7855.508 7856.753 7532.718 7523.31
4
```

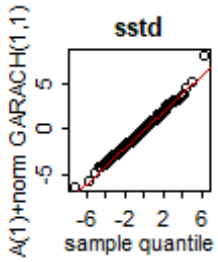
```
cat("\nAIC of standardized residuals from MA(1) + GARCH(1,1) is for the distr
ibution: ", dists[which.min((aic))], "\n")
```

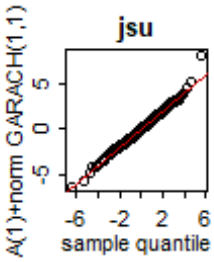
```
##
## AIC of standardized residuals from MA(1) + GARCH(1,1) is for the distribut
ion:  sstd
```

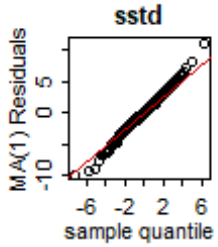
We can observe that jsu and sstd distributions are teh best fitting distributions among all of them. We can further gain insight from QQ-plots to see the best fitting one.

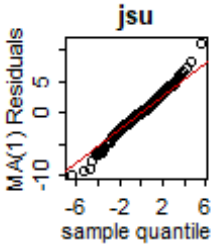
```
##QQ Plot of the model
par(mfrow = c(3,2), pty = "s", mex = 0.5)
q = ((1:n) - 0.5)/n;
qy = quantile(et,q)
est = fits[[2]]$pars
qx = qdist(dists[2], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[2], xlab = "sample quantile",
       ylab = "MA(1)+norm GARACH(1,1) res")
abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")

est = fits[[8]]$pars
qx = qdist(dists[8], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[8], xlab = "sample quantile",
       ylab = "MA(1)+norm GARACH(1,1) res")
abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")


qy = quantile(at,q)
est = fits[[2]]$pars
qx = qdist(dists[2], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[2], xlab = "sample quantile",
       ylab = "MA(1) Residuals")
abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")
```

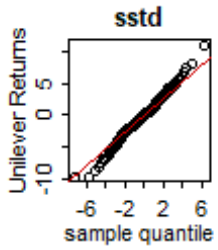
```
est = fits[[8]]$pars
qx = qdist(dists[8], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[8], xlab = "sample quantile",
       ylab = "MA(1) Residuals")
abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")

qy = quantile(Yn,q)
est = fits[[2]]$pars
qx = qdist(dists[2], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[2], xlab = "sample quantile",
       ylab = "Unilever Returns")
abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")
est = fits[[8]]$pars
qx = qdist(dists[8], q, mu = est["mu"],
             sigma = est["sigma"], skew = est["skew"],shape = est["shape"])
plot(qx, qy, main = dists[8], xlab = "sample quantile",
       ylab = "Unilever Returns")

abline(lsfit(qx[round(c(0.2,0.8)*n)],
               qy[round(c(0.2,0.8)*n)] )$coef, col = "red3")
```

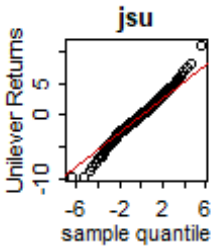












We can clearly see that the "sstd" distribution has the almost perfect fit among the distributions with "jsu" coming close to it.

We further check the 2 distributions using the 4 information criterion with higher order GARCH models to make sure we did not leave any other best possible model.

```
##Checking Garch models with sstd
sstd.ic = c();
ind = 0
sstd.fits = vector("list", 4)
for(p in 1:2)
  {for(q in 1:2)
    {ind = ind +1
    spec = ugarchspec(mean.model = list(armaOrder = c(0,1)),
                      variance.model = list(garchOrder = c(p,q)),
                      distribution.model = "sstd")
    garch = ugarchfit(data=Yn, spec=spec)
    sstd.ic = cbind(sstd.ic, infocriteria(garch))
    sstd.fits[[ind]] = garch
    names(sstd.fits)[ind] = paste0("garch",p,q)
    }
  }
garch.pq = names(sstd.fits)
colnames(sstd.ic) = garch.pq;
sstd.ic

##                garch11  garch12  garch21  garch22
## Akaike       3.287378 3.287837 3.288110 3.288559
## Bayes        3.302364 3.304963 3.305236 3.307827
## Shibata      3.287365 3.287820 3.288093 3.288538
## Hannan-Quinn 3.292791 3.294022 3.294295 3.295518

cat("\nModel Selected with sstd distribution:\n");

##
## Model Selected with sstd distribution:

apply(sstd.ic,1,function(u) garch.pq[which.min(u)])

##       Akaike        Bayes      Shibata Hannan-Quinn
##    "garch11"    "garch11"    "garch11"    "garch11"
```

All the information criterion choose GARCH(1,1) model

```
##Checking Various Garch models with jsu

jsu.ic = c();
ind = 0
jsu.fits = vector("list", 2)
for(q in 1:2)
    {ind = ind +1
```

```
    spec = ugarchspec(mean.model = list(armaOrder = c(0,1)),
                      variance.model = list(garchOrder = c(1,q)),
                      distribution.model = "jsu")
    garch = ugarchfit(data=Yn, spec=spec)
    jsu.ic = cbind(jsu.ic, infocriteria(garch))
    jsu.fits[[ind]] = garch
    names(jsu.fits)[ind] = paste0("garch",1,q)
  }
garch.pq = names(jsu.fits)
colnames(jsu.ic) = garch.pq;
jsu.ic
```

```
##                garch11  garch12
## Akaike       3.289256 3.289676
## Bayes        3.304242 3.306803
## Shibata      3.289243 3.289660
## Hannan-Quinn 3.294669 3.295862
```

```
cat("\nModel Selected with jsu distribution:\n");
```

```
##
## Model Selected with jsu distribution:
```

```
apply(jsu.ic,1,function(u) garch.pq[which.min(u)])
```

```
##       Akaike        Bayes      Shibata Hannan-Quinn
##    "garch11"    "garch11"    "garch11"    "garch11"
```

Even with "jsu", all the criterion pont towards the GARCH(1,1)model Comparing the information criterion values, we find that "sstd" distribution has a better fit. We can observe that from the lower values of ic

```
##With white noise and GARCH with sstd error

sstd.ic = c();
ind = 0
sstd.fits = vector("list", 2)
for(q in 1:2)
    {ind = ind +1
    spec = ugarchspec(mean.model = list(armaOrder = c(0,1)),
                                                distribution.model = "sstd")
    garch = ugarchfit(data=Yn, spec=spec)
    sstd.ic = cbind(sstd.ic, infocriteria(garch))
    sstd.fits[[ind]] = garch
    names(sstd.fits)[ind] = paste0("garch",1,q)
  }
garch.pq = names(sstd.fits)
```

```
colnames(sstd.ic) = garch.pq;
sstd.ic
```

```
##                garch11  garch12
## Akaike       3.287378 3.287378
## Bayes        3.302364 3.302364
## Shibata      3.287365 3.287365
## Hannan-Quinn 3.292791 3.292791
```

```
cat("\nModel Selected with sstd distribution:\n");
```

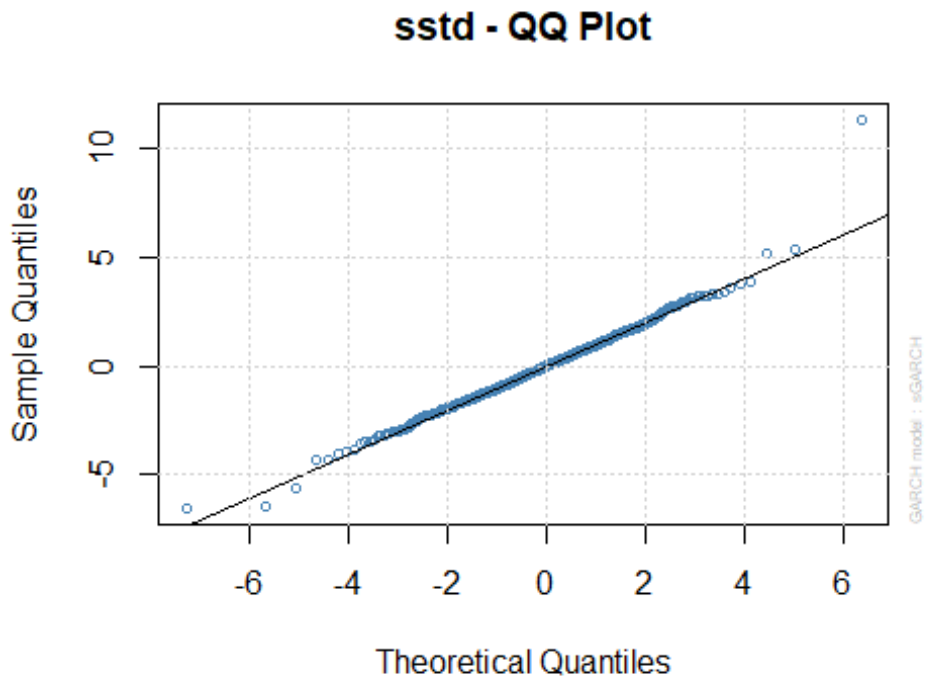
```
##
## Model Selected with sstd distribution:
```

```
apply(sstd.ic,1,function(u) garch.pq[which.min(u)])
```

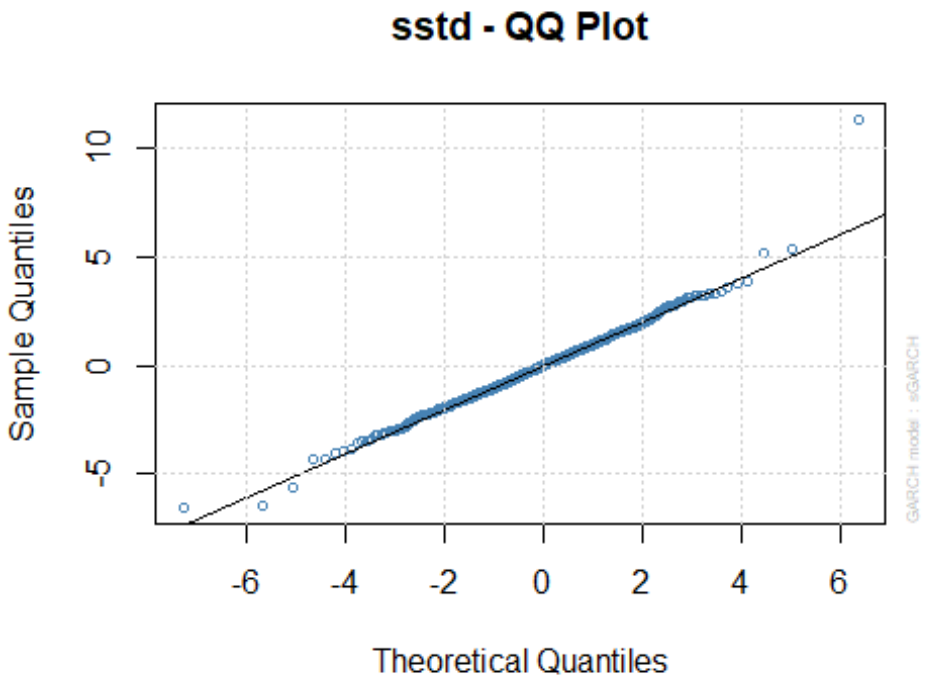
```
##       Akaike        Bayes      Shibata Hannan-Quinn
##    "garch11"    "garch11"    "garch11"    "garch11"
```

The last models we look at are MA(1)+GARCH(1,1) and MA(1)+GARCH(1,2) with "sstd" errors

```
plot(sstd.fits$garch11, which = 9)
```




```
plot(sstd.fits$garch12, which = 9)
```



The Q-Q plots of both GARCH(1,1) and GARCH(1,2) look almost same, hance we choose the simpler model that is GARCH(1,1)

Forecasting with the finalised MA(1)+GARCH(1,1) model with sstd dist:

Retrieving the data

```
n = dim(Yn)[1]
nh = dim(Yt)[1]
n.fore = nh-n
cat("\nDates of the Data:\n", paste(c("from", "to"), time(Yt)[c(1,nh)]))

##
## Dates of the Data:
##  from 2007-01-04 to 2022-10-31

cat("\nDates of Modeling:\n",paste(c("from", "to"), time(Yt)[c(1,n)]), "Sampl
e size:",n);

##
## Dates of Modeling:
##  from 2007-01-04 to 2017-12-29 Sample size: 2768
```

```
cat("\nDates of forecast:\n", paste(c("from","to"), time(Yt)[c(n+1,nh)]),"len
gth:", n.fore)

##
## Dates of forecast:
##  from 2018-01-02 to 2022-10-31 length: 1217
```

Rolling Forecast

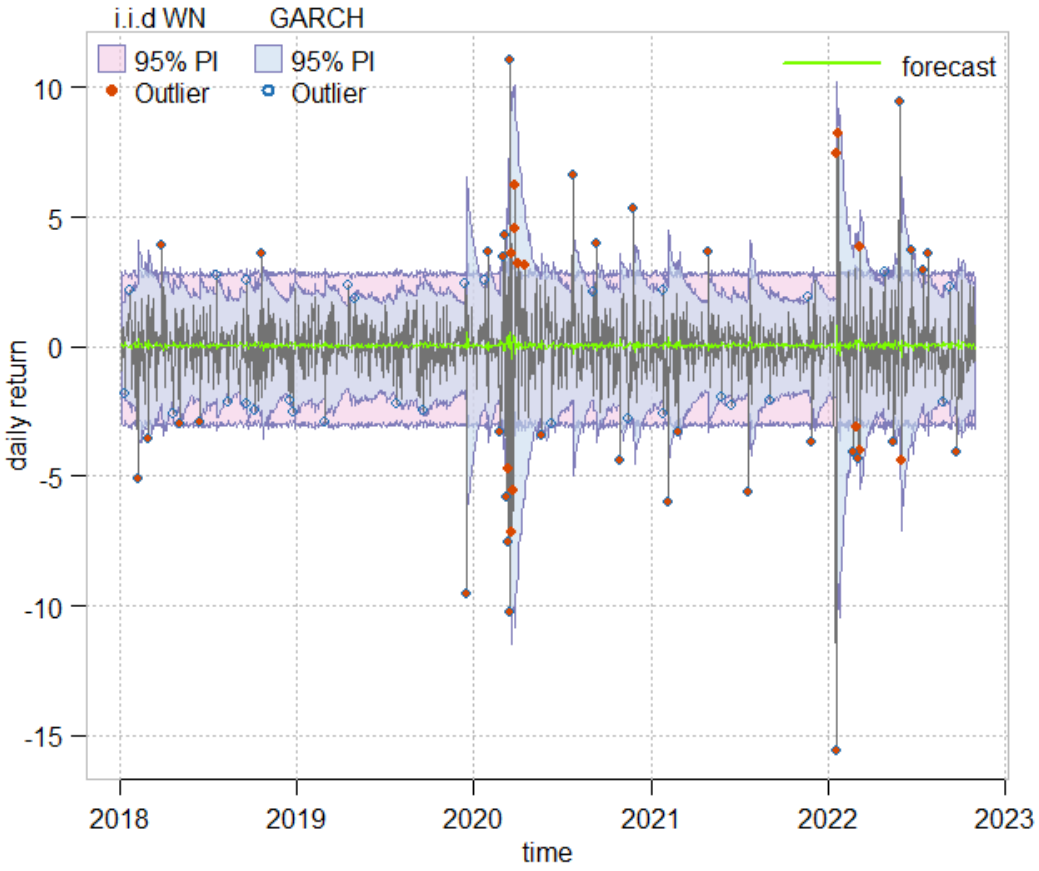
```
spec.11 = ugarchspec(mean.model = list(armaOrder = c(0,1)),
                     variance.model = list(garchOrder = c(1,1)),
                     distribution.model = "sstd")
fit.11 = ugarchfit(data = Yt, spec = spec.11, out.sample = n.fore)
fore.g = ugarchforecast(fit.11, n.ahead = 1, n.roll = n.fore-1)

cat("MA(1) + GARCH(1,1)");

## MA(1) + GARCH(1,1)

rate(fore.g)

##
##       One-Step Rolling Forecast
## ---------------------------------
##        coverage below PI beyond PI
## 95% PI   0.9482   0.0304    0.0214
## 90% PI   0.9154   0.0444    0.0403
```

We did a one-step rolling forecast. The coverage rates are symmetric and can be observed in beyond and below PI values.

Just to be sure, we still compare it with MA(1) + iid

```
spec.0 = arfimaspec(mean.model = list(armaOrder = c(0, 1)),
                    distribution.model = "sstd")
fit.0 = arfimafit(data = Yt, spec = spec.0, out.sample = n.fore)
fore.0 = arfimaforecast(fit.0, n.roll = n.fore-1)
#showShort0(fit.0)

cat("MA(1) + i.i.d. sstd Noise");

## MA(1) + i.i.d. sstd Noise

rate0(fore.0)

##
##       One-Step Rolling Forecast
## ---------------------------------
##        coverage below PI beyond PI
## 95% PI   0.9606   0.0214    0.0181
## 90% PI   0.9236   0.0362    0.0403
```

```
plot_PI(fore.0,fore.g)
```



We can see from the diagram that the MA(1)+GARCH(1,1) model does a better job at capturing the volatility of the data when compared to the iid white noise model.

```
alpha = 0.05;
S = 10000/100
today = which(colnames(fitted(fore.g)) == "2019-12-16")
today = today:(today+2)

# ARMA + GARCH
mu = fitted(fore.g)["T+1", today]
sig = sigma(fore.g)["T+1", today]
q = qdist("sstd", p = alpha, skew = coef(fit.11)["skew"]
          ,shape = coef(fit.11)["shape"])
VaR = -S*(mu + q*sig);
cat("\nVaR with ARMA + GARCH model from Dec 16th :\n");VaR
```

```
##
## VaR with ARMA + GARCH model from Dec 16th :

## 2019-12-16 2019-12-17 2019-12-18
##   192.7494   449.3472   465.8640

#with iid noise
mu.iid = fitted(fore.0)["T+1", today]
q.iid = qdist("sstd", p = alpha, skew = coef(fit.0)["skew"], shape = coef(fit
.0)["shape"])
VaR.iid = -S*(mu.iid+q.iid)
cat("one day value at risk with iid noise from Dec 16th : \n" )

## one day value at risk with iid noise from Dec 16th :

VaR.iid

## 2019-12-16 2019-12-17 2019-12-18
##  156.81738   72.22445  132.94135
```

During normal times, we can see that iid model performs better. But that's not the case when huge amount of volatility is present in the data.

To test that we take data from period such as covid time where there is high level of volatility.

```
alpha = 0.05;
S = 10000/100
covid_date = which(colnames(fitted(fore.g)) == "2020-03-16")
covid_date = covid_date:(covid_date+2)
# ARMA + GARCH
mu = fitted(fore.g)["T+1", covid_date]
sig = sigma(fore.g)["T+1", covid_date]
q = qdist("sstd", p = alpha, skew = coef(fit.11)["skew"]
          ,shape = coef(fit.11)["shape"])
VaR = -S*(mu + q*sig);
cat("\nVaR with ARMA + GARCH model:\n");VaR

##
## VaR with ARMA + GARCH model:

## 2020-03-16 2020-03-17 2020-03-18
##   661.2746   899.7105   790.8520

mu.iid = fitted(fore.0)["T+1", covid_date]
q.iid = qdist("sstd", p = alpha, skew = coef(fit.0)["skew"], shape = coef(fit
.0)["shape"])
```

```
VaR.iid = -S*(mu.iid+q.iid)
cat("one day value at risk with iid noise : \n")

## one day value at risk with iid noise :

VaR.iid

## 2020-03-16 2020-03-17 2020-03-18
##   66.94021  221.58828  126.48796
```

This clearly indicates that GARCH model is much better when it comes to modelling volatility.

Therefore we can say that MA(1)+GARCH(1,1) with sstd distribution is well suited to forecast Unilever stock data.