Model Diagnostics & Order Selection

Kushal Kharel

10/10/2021

From a forecasting point of view, it is not advantageous to just arbitrarily choose the orders p and q. If we fit a very high order model, it will result in small estimated white noise variance. But when the fitted model is used for forecasting, the mean squared error of the forecasts will depend not only on white noise variance of the fitted model but also on the errors arising from estimation of the parameters of the model. So, to discourage the fitting of the models with many parameters, we introduce the penalty factor for every added parameters. There are many methods to do this. We will discuss few of them here.

FPE Criterion

Suppose we have a time series $X_1, X_2, ..., X_n$. The idea here is to choose the model for $\{X_t\}$ in such a way as to minimize the one-step mean squared error when the model fitted to $\{X_t\}$ is used to predict an independent realization $\{Y_t\}$ of the same process that generated $\{X_t\}$.

Let $\{X_1, X_2..., X_n\}$ be realization of $AR(p)$ process with coefficients $\phi_1, \phi_2, ..., \phi_p$, $p < n$, and $\{Y_1, Y_2, ..., Y_n\}$ be independent realization of the same process.Let $\hat{\phi}_1, \hat{\phi}_2, ..., \hat{\phi}_p$ be the maximum likelihood estimators of the coefficients and using these to compute the one-step predictor $\hat{\phi}_1 Y_n + \hat{\phi}_p Y_{n+1-p}$ of Y_{n+1}, the mean square prediction error is,

$$E\Big(Y_{n+1} - \hat{\phi}_1 Y_n - ... - \hat{\phi}_p Y_{n+1-p}\Big)^2$$

$$= E\Big[Y_{n+1} - \phi_1 Y_n - ... - \phi_p Y_{n+1-p} - (\hat{\phi}_1 - \phi_1) Y_n - ... - (\hat{\phi}_p - \phi_p) Y_{n+1-p}\Big]^2$$

$$= \sigma^2 + E\Big[(\hat{\phi}_p - \phi_p)' [Y_{n+1-i} Y_{n+1-j}]_{i,j=1}^{p} (\hat{\phi}_p - \phi)\Big]$$

where, $\phi_p' = (\phi_1, ..., \phi_p)'$, $\hat{\phi}_p' = (\hat{\phi}_1, ..., \hat{\phi}_p)'$, and σ^2 is the white noise variance of $AR(p)$ model.

$$= \sigma^2 + E\Big[(\hat{\phi}_p - \phi_p)' \Gamma_p (\hat{\phi}_p - \phi)\Big]$$

where, $\Gamma_p = E[Y_i Y_j]_{i,j=1}^{n}$

Assuming that $n^{-0.5}(\hat{\phi}_p - \phi_p)$ has its large-sample distribution $N(0, \sigma^2 \Gamma_p^{-1})$, we get;

$$E\Big(Y_{n+1} - \hat{\phi}_1 Y_n - ... - \hat{\phi}_p Y_{n+1-p}\Big)^2 \approx \sigma^2 \Big(1 + \frac{p}{n}\Big)$$

If $\hat{\sigma}^2$ is the maximum likelihood estimator of σ^2 ,then for large n, $n\frac{\hat{\sigma}^2}{\sigma^2}$ is distributed as chi-squared with $n - p$ degrees of freedom.

By replacing σ^2 by the estimator above, we get the estimated mean square prediction error of Y_{n+1} as,

$$FPE_p = \hat{\sigma}^2 \frac{n+p}{n-p}$$

AICC Criterion

AICC criterion is just the bias corrected version of AIC criterion. Let $\hat{\phi} = (\phi_1, ..., \phi_p)'$ and $\hat{\theta} = (\theta_1, ..., \theta_q)'$. The AICC is defined for an $ARMA(p, q)$ model with coefficients $\hat{\phi}, \hat{\theta}$ and σ^2 by,

$$AICC(\hat{\phi}, \hat{\theta}, \sigma^2) = -2 \ln L(\hat{\phi}, \hat{\theta}, \sigma^2) + \frac{2(p+q+1)n}{n-p-q-2}$$

where, $-2 \ln L(\hat{\phi}, \hat{\theta}, \sigma^2)$, is the log likelihood function and $\frac{2(p+q+1)n}{n-p-q-2}$, is the complexity penalty.

The approach to order selection is to fit models with varying orders $0, 1, ..., p$ for AR part and $0, 1, ..., q$ for the MA part jointly and compute AICC for each combination of orders and select the orders with the smallest AICC. If the maximum order of p and q is large, it can be computationally expensive.

BIC Criterion

We know that AIC is computed as;

$$AIC = -2ln(L) + 2k$$

and BIC is computed as;

$$BIC = -2ln(L) + k \ln(n)$$

where, L is the likelihood of the model given the data, k is the number of estimated parameters in the model, and n is the number of entries. Both of these models are cost functions which needs to be minimized. It favors models with high likelihood but penalizes the model with increase in regressors. AIC criterion is basically saying that the less information the model loses, higher its quality and BIC criterion is basically designed to maximize the posterior probability of a model given the data.

AIC or BIC for ARMA Model

We use AIC and BIC across a subset of values for p and q in order to determine the order of the series and then apply the Ljung-Box test to determine goodness of fit. Let us first simulate a $ARMA(p, q)$ process and then loop over all the pairwise values of $p \in 0, 1, 2, 3, 4$ and $a \in 1, 2, 3, 4$ and calculate the AIC. Then, we will select the model with the lowest AIC and run a test on the residuals to determine goodness of fit.

Here, we are simulating $ARMA(4, 3)$ series.

```
set.seed(100)
X = arima.sim(n=1000, model=list(ar=c(0.5,-0.25,0.4,0.2),ma=c(0.5,-0.3,0.6)))
```

Now, we will create final object to store the best model fit and lowest AIC value. We loop over the various p, q combinations and use the current object to store the fit of an $ARMA(i, j)$ mode, for the looping variables i and j. If the current AIC is less than any of the previously calculated AIC, then we set the final AIC to current value and select that order else repeat until you find the order. Upon finding the order and saving it in final.order, we will have final $ARMA$ process with selected order.

```
AIC_final = Inf # Initial guess
Order_final = c(0,0,0) # Initial guess
for (i in 0:4) for (j in 0:4){
  AIC_current = AIC(arima(X,order=c(i,0,j)))
  if (AIC_current < AIC_final){
    AIC_final = AIC_current
    Order_final = c(i,0,j)
    ARMA_final = arima(X, order=Order_final)
  }

}
```

Let us extract AIC, order and ARIMA coefficients:

```
AIC_final
```

```
## [1] 3232.971
```

```
Order_final
```

```
## [1] 4 0 3
```

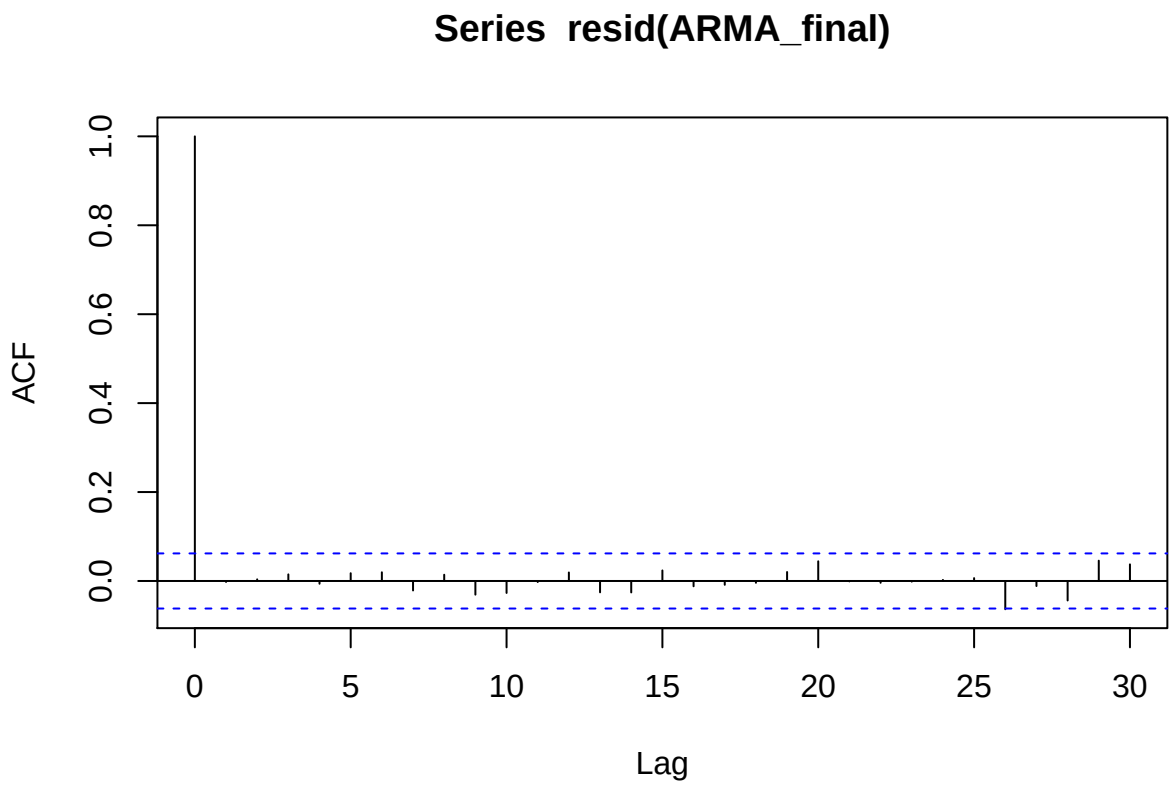
```
ARMA_final
```

```
##
## Call:
## arima(x = X, order = Order_final)
##
## Coefficients:
##          ar1      ar2     ar3     ar4     ma1      ma2     ma3  intercept
##       0.4873  -0.3209  0.4534  0.2024  0.1643  -0.0251  0.4485     0.2068
## s.e.  0.0856   0.0549  0.0464  0.0677  0.0814   0.0405  0.0359     0.3364
##
## sigma^2 estimated as 1.453:  log likelihood = -1607.49,  aic = 3232.97
```

We can see from the above results that original order of simulated ARMA model was recovered, with $p = 4$ and $q = 3$

Now, let us look at the corelogram of residuals of the model to see if they look like a realization of dicrete white noise.

```
acf(resid(ARMA_final))
```



From the plot above, we can see that the realization does lok like discrete white noise. Now, let us perform the Ljung-Box test for 20 lags to confirm this.

```
Box.test(resid(ARMA_final),lag=20,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  resid(ARMA_final)
## X-squared = 8.29, df = 20, p-value = 0.9898
```

From the results above, we can see that p-value is greater than 0.05 implying that the residuals are independent at the 95% level and hence ARMA(4,3) model provides a good fit to our simulated data.

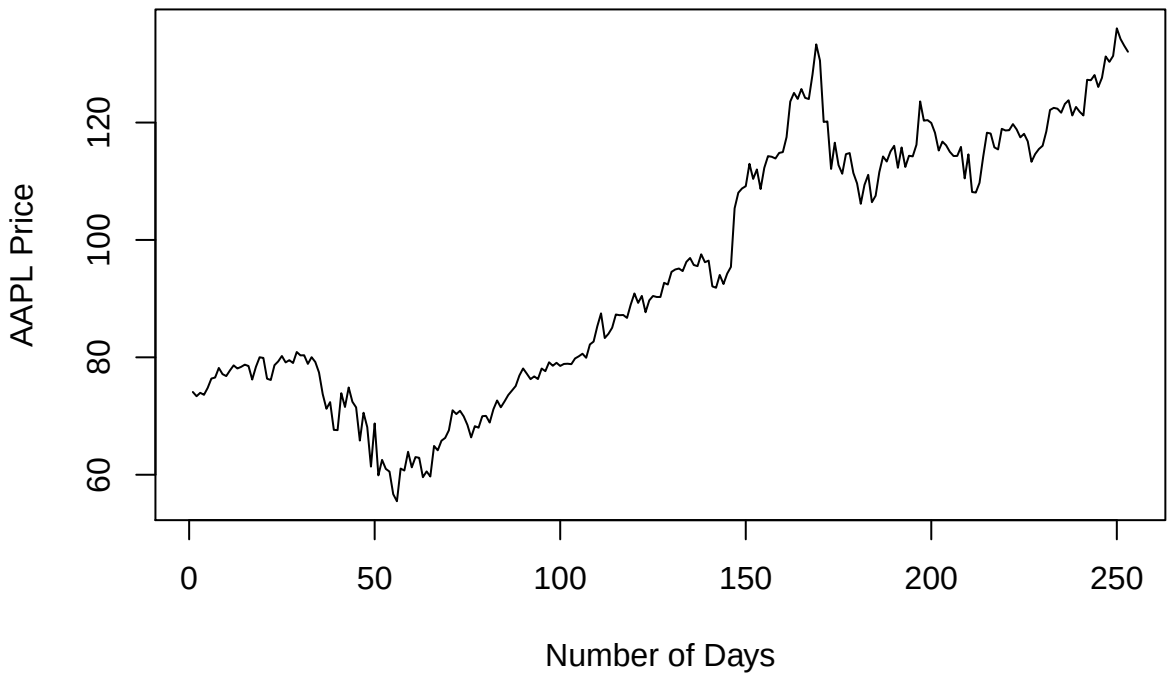
Now, let us apply the above method for AAPL stock prices from 2020-01-01 to 2021-01-01

```
library(tidyquant)
```

```
## Warning: package 'tidyquant' was built under R version 4.0.5

## Loading required package: lubridate

## Warning: package 'lubridate' was built under R version 4.0.3

##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 4.0.5

## Loading required package: xts

## Warning: package 'xts' was built under R version 4.0.2

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.0.2

##
## 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

## Loading required package: quantmod

## Warning: package 'quantmod' was built under R version 4.0.3

## Loading required package: TTR

## Warning: package 'TTR' was built under R version 4.0.2

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

## == Need to Learn tidyquant? ============================================
## Business Science offers a 1-hour course - Learning Lab #9: Performance Analysis & Portfolio Optimiza
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>
```

```
av_api_key('PQQC4A86NCLAXUSQ')
aapl = tq_get("AAPL", from = "2020-01-01", to = "2021-01-01")
plot(aapl$adjusted, type = "l", ylab = "AAPL Price", xlab = "Number of Days")
```




```
# creating log return series
aapl3 = as.ts(diff(log(aapl$adjusted)))
is.ts(aapl3)
```

```
## [1] TRUE
```

```
aaplAIC_Final = Inf
aaplOrder_final = c(0,0,0)
for (i in 0:4) for (j in 0:4){
  aaplAIC_current = AIC(arima(aapl3,order=c(i,0,j)))
  if (aaplAIC_current < aaplAIC_Final){
    aaplAIC_Final = aaplAIC_current
    aaplOrder_final = c(i,0,j)
    aaplARMA_final = arima(aapl3,order=aaplOrder_final)
  }
}
```

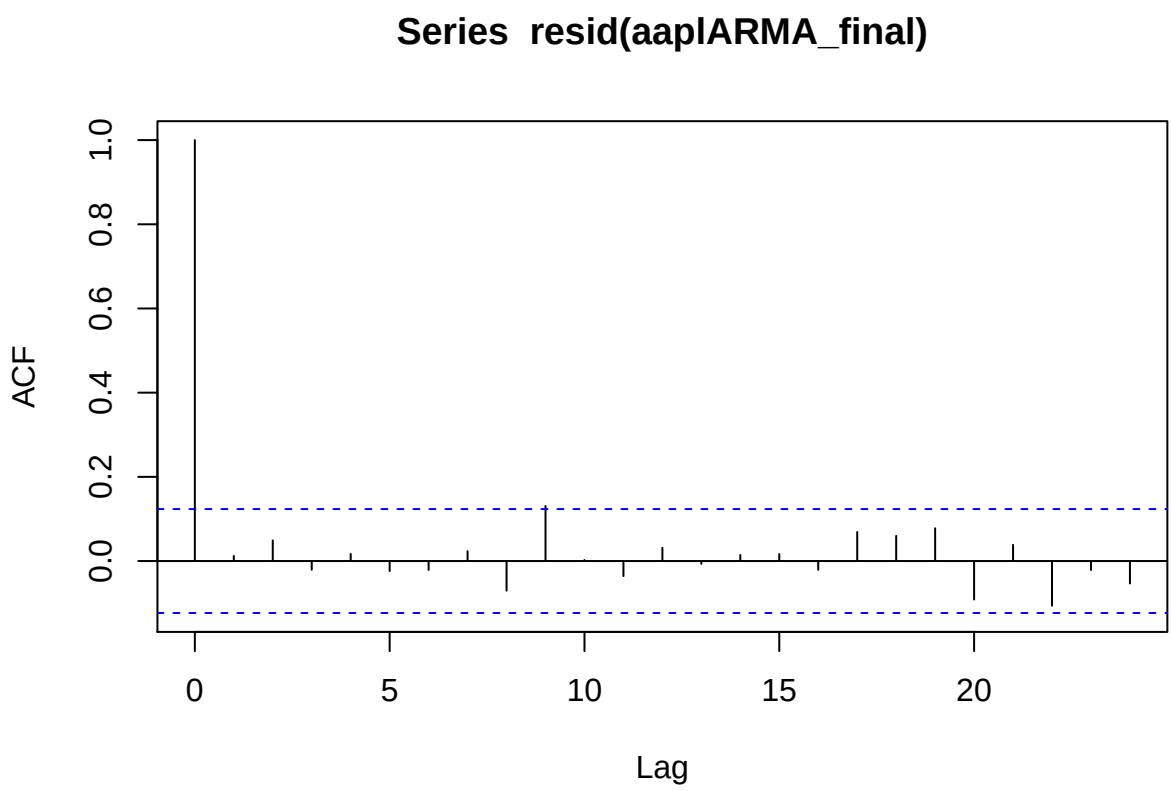
Now, let us extract final order of the model

```
aaplOrder_final
```

```
## [1] 3 0 2
```

We can see that final order of the model is $ARMA(3,2)$. Now, let us look at the residual plot of the fitted model to the aapl log daily returns.

```
acf(resid(aaplARMA_final), na.action=na.omit)
```



From the plot above, we do not notice any significant peak at higher lags which is indicating of the good fit. To confirm this, let us perform a Ljung-Box test to look for statistical evidence.

```
Box.test(resid(aaplARMA_final),lag=20,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  resid(aaplARMA_final)
## X-squared = 14.165, df = 20, p-value = 0.8221
```

We can see from the results above that p-value is greater than 0.05 indicating that the residuals are a realization of discrete white noise series. In practice, ARMA models are not generally good fits for log equity returns since it does not take into account the conditional heteroscedasticity of the series.

```
library("fUnitRoots")
```

```
## Warning: package 'fUnitRoots' was built under R version 4.0.5
```

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

```
## Warning: package 'timeDate' was built under R version 4.0.3
```

```
##
## Attaching package: 'timeDate'

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

## Loading required package: timeSeries

## Warning: package 'timeSeries' was built under R version 4.0.5

##
## Attaching package: 'timeSeries'

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

## Loading required package: fBasics

## Warning: package 'fBasics' was built under R version 4.0.5

##
## Attaching package: 'fBasics'

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

```
library(lmtest)
```

```
## Warning: package 'lmtest' was built under R version 4.0.5
```

```
library("forecast")
```

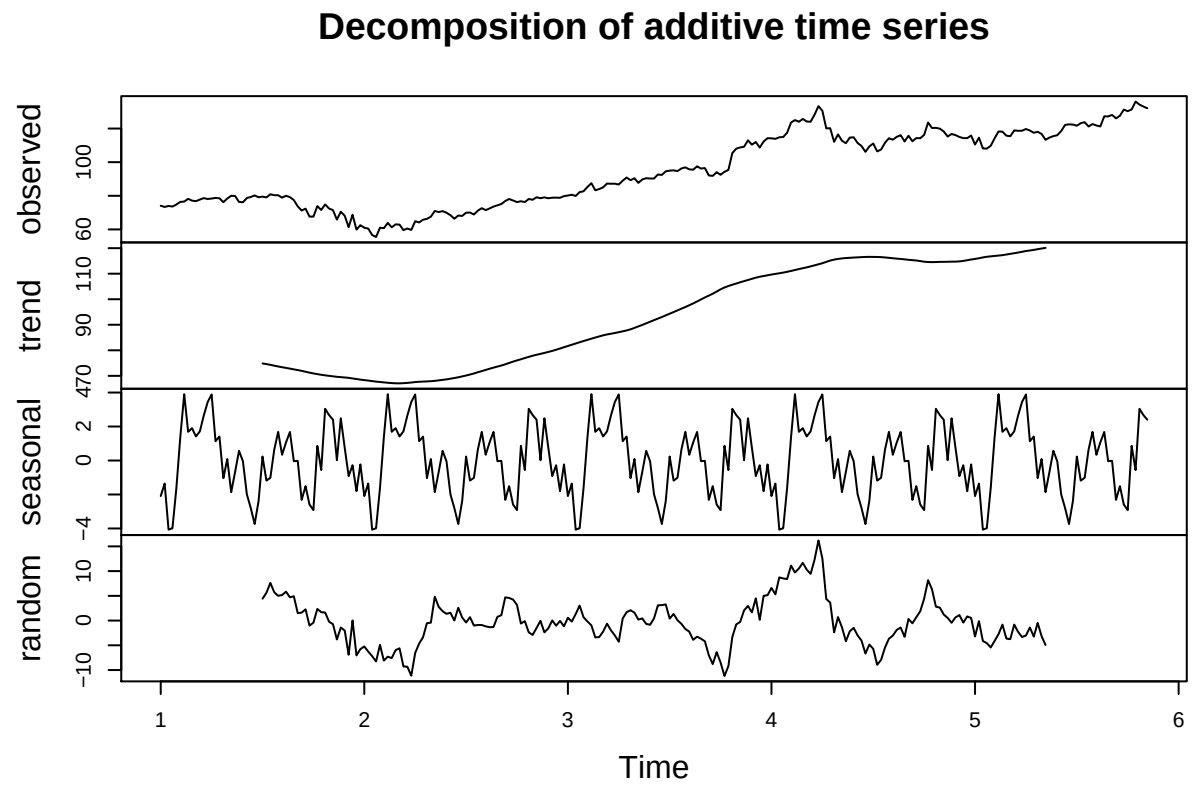
```
## Warning: package 'forecast' was built under R version 4.0.5
```

```
library(FitAR)
```

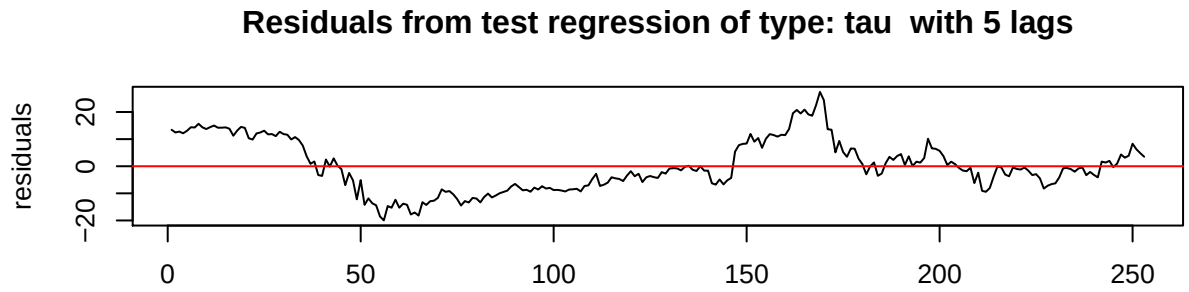
```
## Warning: package 'FitAR' was built under R version 4.0.5

## Loading required package: lattice

## Loading required package: leaps

## Warning: package 'leaps' was built under R version 4.0.2

## Loading required package: ltsa
```

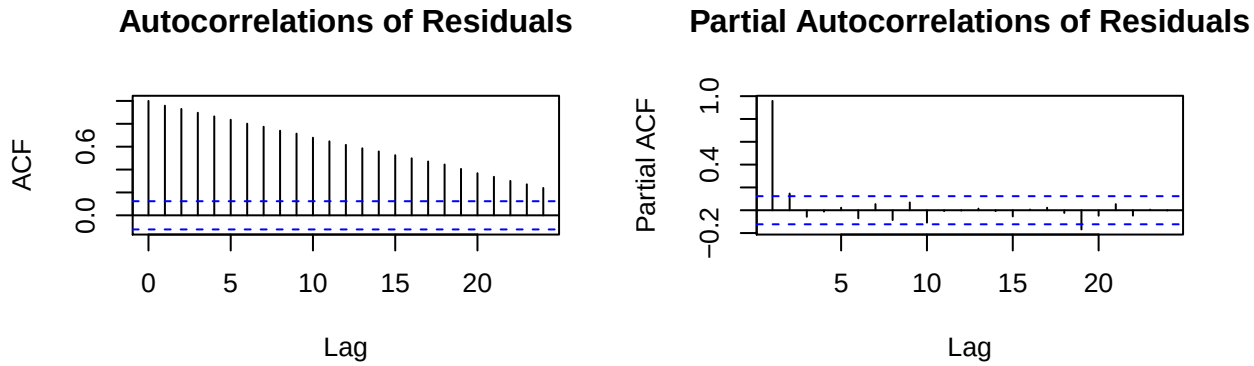
```
## Warning: package 'ltsa' was built under R version 4.0.3

## Loading required package: bestglm

## Warning: package 'bestglm' was built under R version 4.0.5

##
## Attaching package: 'FitAR'

## The following object is masked from 'package:forecast':
##
##     BoxCox
```

```
library(tidyquant)

#import data
av_api_key('PQQC4A86NCLAXUSQ')
aapl = tq_get("AAPL", from = "2020-01-01", to = "2021-01-01")
#convert to time series
aapl2 = as.ts(aapl$adjusted)
tsData = ts(aapl2, frequency = 52)
#decompose into time series components
timeseriescomponents <- decompose(tsData)
plot(timeseriescomponents)
```



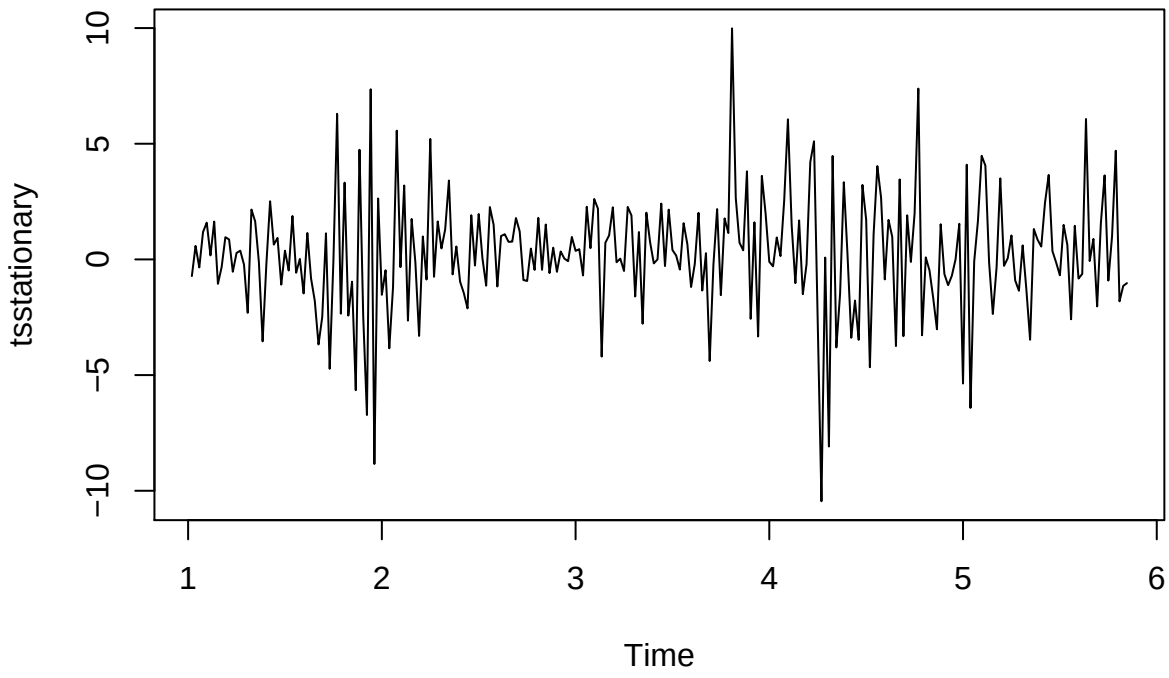
```
#detemine stationarity of data
urkpssTest(tsData, type = c("tau"), lags = c("short"),use.lag = NULL, doplot = TRUE)
```



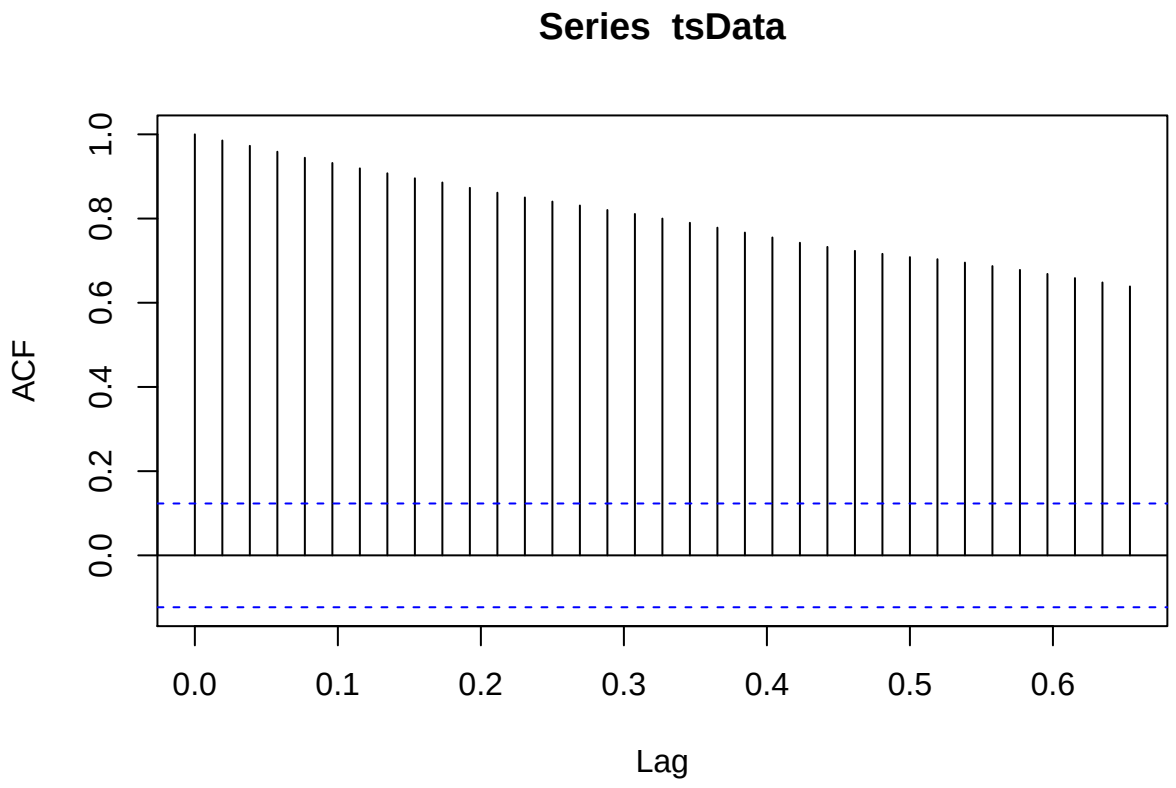



```
##
## Title:
##  KPSS Unit Root Test
##
## Test Results:
##   NA
##
## Description:
##  Sun Oct 10 17:17:48 2021 by user: kushk
```

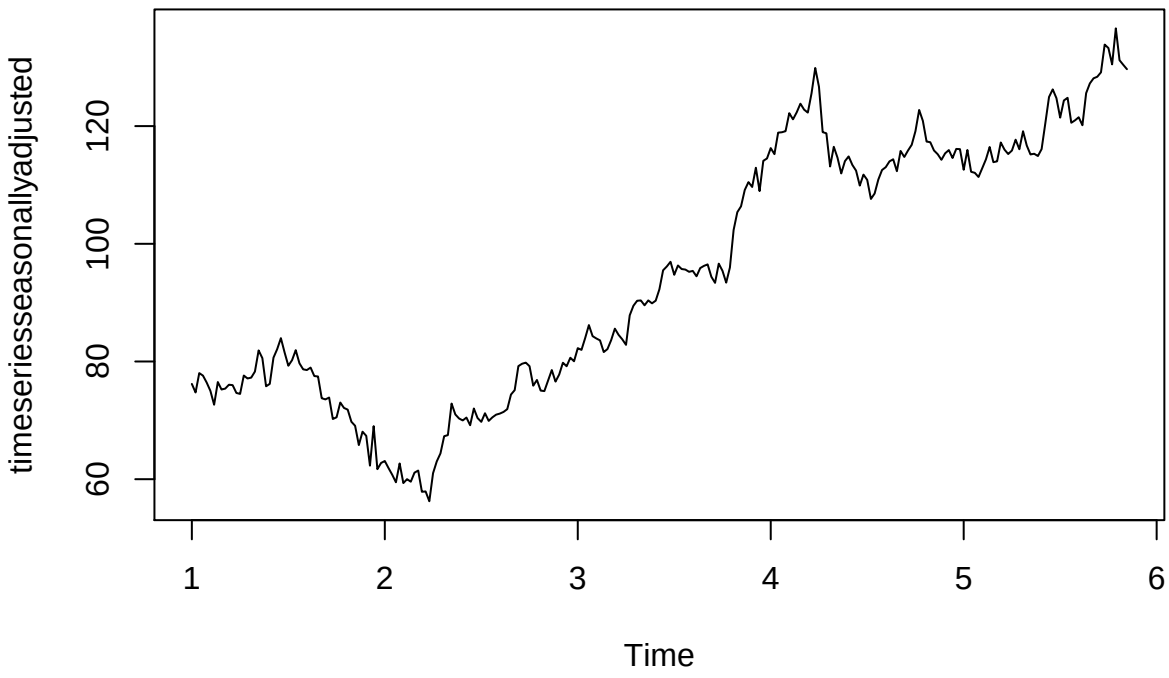
```
tsstationary<-diff(tsData, differences=1)
plot(tsstationary)
```



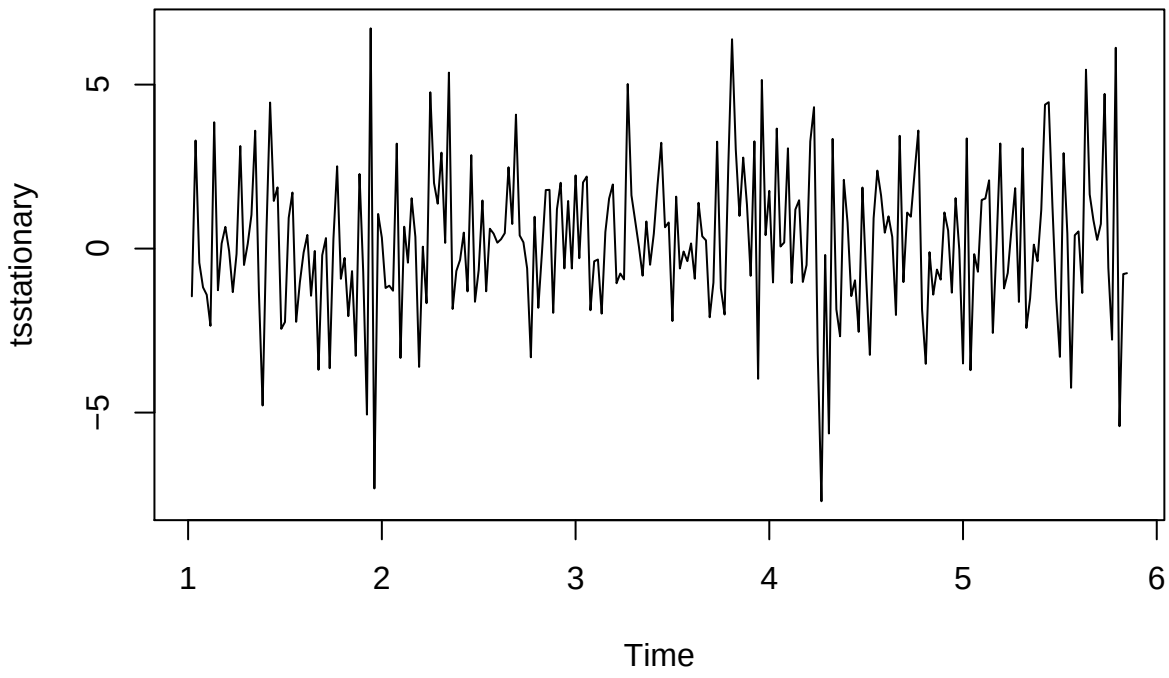

```
acf(tsData,lag.max=34)
```



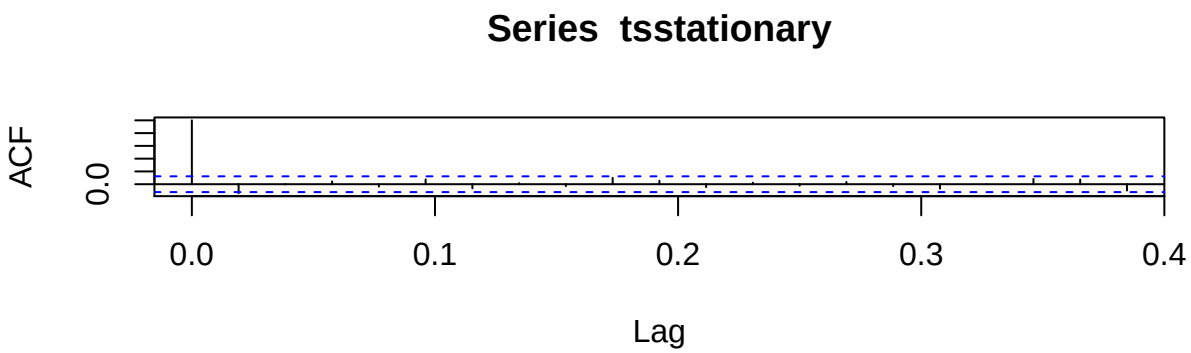

```
#remove seasonality
timeseriesseasonallyadjusted <- tsData- timeseriescomponents$seasonal
plot(timeseriesseasonallyadjusted)
```

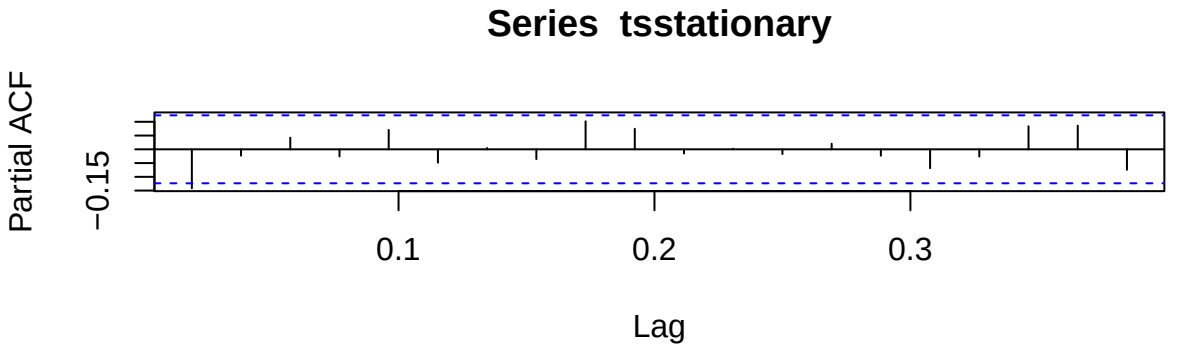



```
tsstationary <- diff(timeseriesseasonallyadjusted, differences=1)
plot(tsstationary)
```




```
par(mfrow=c(2,1))
acf(tsstationary, lag.max=20)
pacf(tsstationary, lag.max=20)
```






```
#fit the model
fitARIMA<-arima(tsData, order=c(3,0,2),seasonal = list(order = c(3,0,2), period = 12),method="ML")
fitARIMA
```

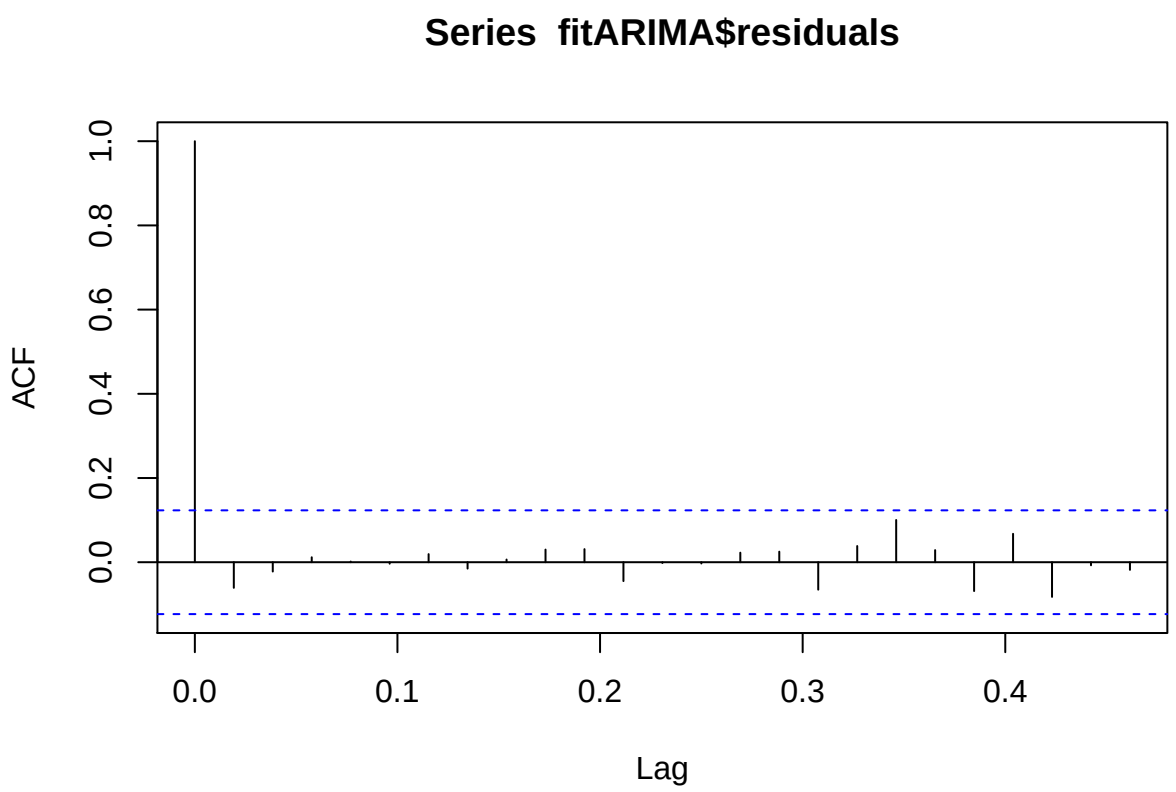
```
##
## Call:
## arima(x = tsData, order = c(3, 0, 2), seasonal = list(order = c(3, 0, 2), period = 12),
##     method = "ML")
##
## Coefficients:
##           ar1     ar2     ar3     ma1     ma2    sar1    sar2    sar3     sma1
##       -0.7746  0.8572  0.9006  1.7049  0.8167  0.3761  0.2141  0.1448  -0.3608
## s.e.   0.1638  0.0396  0.1281  0.2473  0.2137  0.3745  0.3182  0.0821   0.3722
##          sma2  intercept
##       -0.3418   100.7646
## s.e.   0.3065    21.7713
##
## sigma^2 estimated as 6.073:  log likelihood = -590.22,  aic = 1204.44
```

```
#significance of coefficients
coeftest(fitARIMA)
```

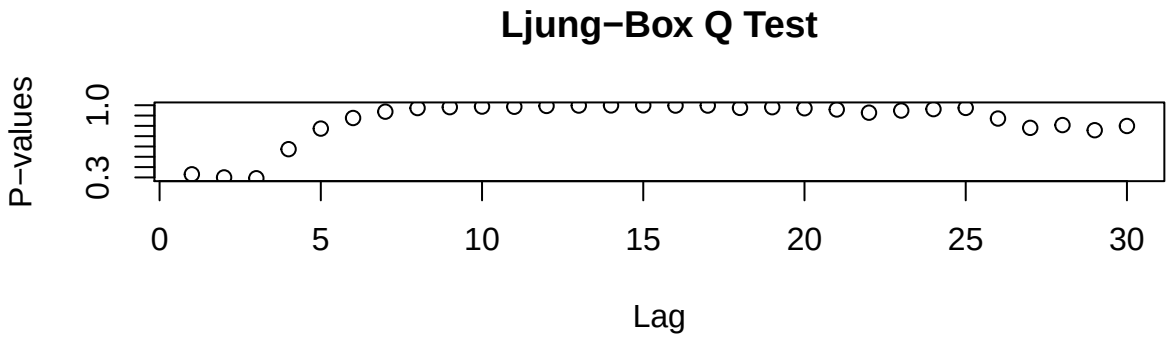
```
##
## z test of coefficients:
##
```

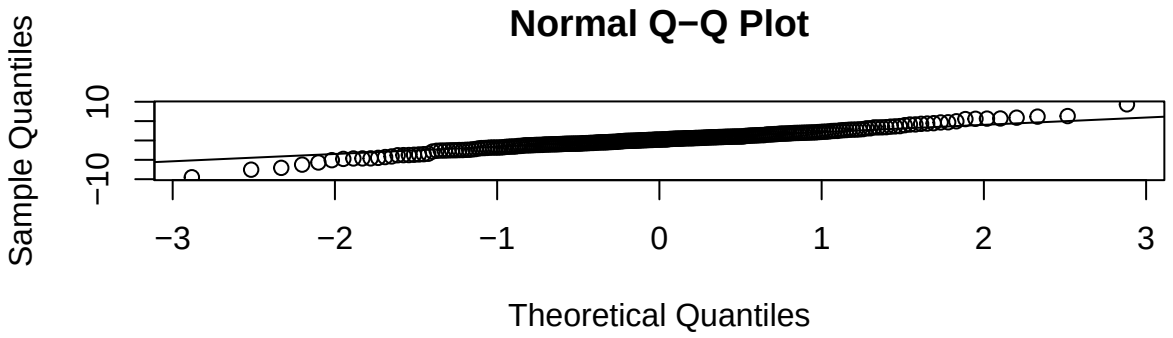
```
##             Estimate Std. Error z value  Pr(>|z|)
## ar1        -0.774600   0.163754 -4.7303 2.242e-06 ***
## ar2         0.857189   0.039562 21.6667 < 2.2e-16 ***
## ar3         0.900560   0.128098  7.0302 2.062e-12 ***
## ma1         1.704895   0.247271  6.8948 5.392e-12 ***
## ma2         0.816695   0.213717  3.8214 0.0001327 ***
## sar1        0.376095   0.374526  1.0042 0.3152878
## sar2        0.214133   0.318206  0.6729 0.5009866
## sar3        0.144760   0.082147  1.7622 0.0780348 .
## sma1       -0.360838   0.372174 -0.9695 0.3322751
## sma2       -0.341780   0.306497 -1.1151 0.2648008
## intercept 100.764631  21.771277  4.6283 3.686e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
par(mfrow=c(1,1))
acf(fitARIMA$residuals)
```




```
#residual diagnostics
boxresult<-LjungBoxTest (fitARIMA$residuals,k=2,StartLag=1)
par(mfrow=c(2,1))
plot(boxresult[,3],main="Ljung-Box Q Test", ylab="P-values", xlab="Lag")
qqnorm(fitARIMA$residuals)
qqline(fitARIMA$residuals)
```






```
auto.arima(tsData, trace=TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,1,2)(1,0,1)[52] with drift         : 1184.596
##  ARIMA(0,1,0)            with drift         : 1194.536
##  ARIMA(1,1,0)(1,0,0)[52] with drift         : 1180.702
##  ARIMA(0,1,1)(0,0,1)[52] with drift         : 1190.326
##  ARIMA(0,1,0)                               : 1194.503
##  ARIMA(1,1,0)            with drift         : 1189.325
##  ARIMA(1,1,0)(1,0,1)[52] with drift         : 1182.712
##  ARIMA(1,1,0)(0,0,1)[52] with drift         : 1190.577
##  ARIMA(0,1,0)(1,0,0)[52] with drift         : 1179.8
##  ARIMA(0,1,0)(1,0,1)[52] with drift         : 1181.793
##  ARIMA(0,1,0)(0,0,1)[52] with drift         : 1195.333
##  ARIMA(0,1,1)(1,0,0)[52] with drift         : 1179.592
##  ARIMA(0,1,1)            with drift         : 1189.208
##  ARIMA(0,1,1)(1,0,1)[52] with drift         : 1181.605
##  ARIMA(1,1,1)(1,0,0)[52] with drift         : 1182.001
##  ARIMA(0,1,2)(1,0,0)[52] with drift         : 1181.665
##  ARIMA(1,1,2)(1,0,0)[52] with drift         : 1183.615
##  ARIMA(0,1,1)(1,0,0)[52]                    : 1183.025
##
##  Now re-fitting the best model(s) without approximations...
```

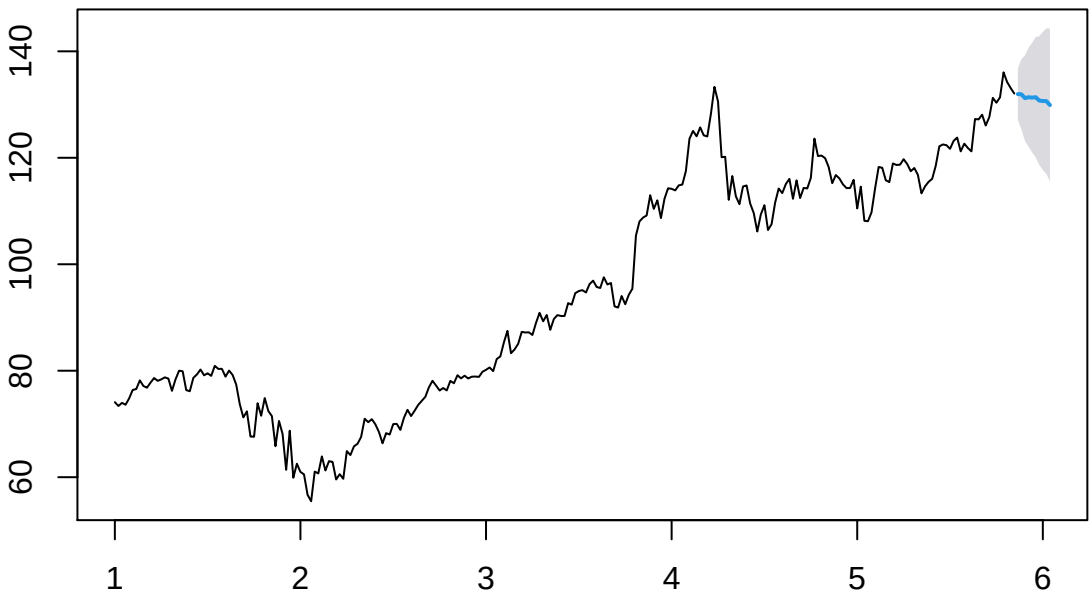
```
##
##  ARIMA(0,1,1)(1,0,0)[52] with drift         : 1192.147
##
##  Best model: ARIMA(0,1,1)(1,0,0)[52] with drift
```

```
## Series: tsData
## ARIMA(0,1,1)(1,0,0)[52] with drift
##
## Coefficients:
##          ma1    sar1   drift
##       -0.1594  0.0725  0.2245
## s.e.   0.0593  0.0706  0.1427
##
## sigma^2 estimated as 6.497:  log likelihood=-591.99
## AIC=1191.99   AICc=1192.15   BIC=1206.1
```

```
#forcast future values
par(mfrow=c(1,1))
predict(fitARIMA,n.ahead = 10, type = "o")
```

```
## $pred
## Time Series:
## Start = c(5, 46)
## End = c(6, 3)
## Frequency = 52
##  [1] 131.9694 131.9403 131.2014 131.3800 131.3002 131.3885 130.7484 130.6683
##  [9] 130.6538 129.9025
##
## $se
## Time Series:
## Start = c(5, 46)
## End = c(6, 3)
## Frequency = 52
##  [1] 2.464251 3.365710 4.104430 4.736878 5.242583 5.766805 6.170174 6.614234
##  [9] 6.970140 7.344621
```

```
futurVal <- forecast(fitARIMA,h=10, level=c(95))
plot(futurVal, type = "l")
```


Forecasts from ARIMA(3,0,2)(3,0,2)[12] with non−zero mean
60
80
100
120
140
1
2
3
4
5
6