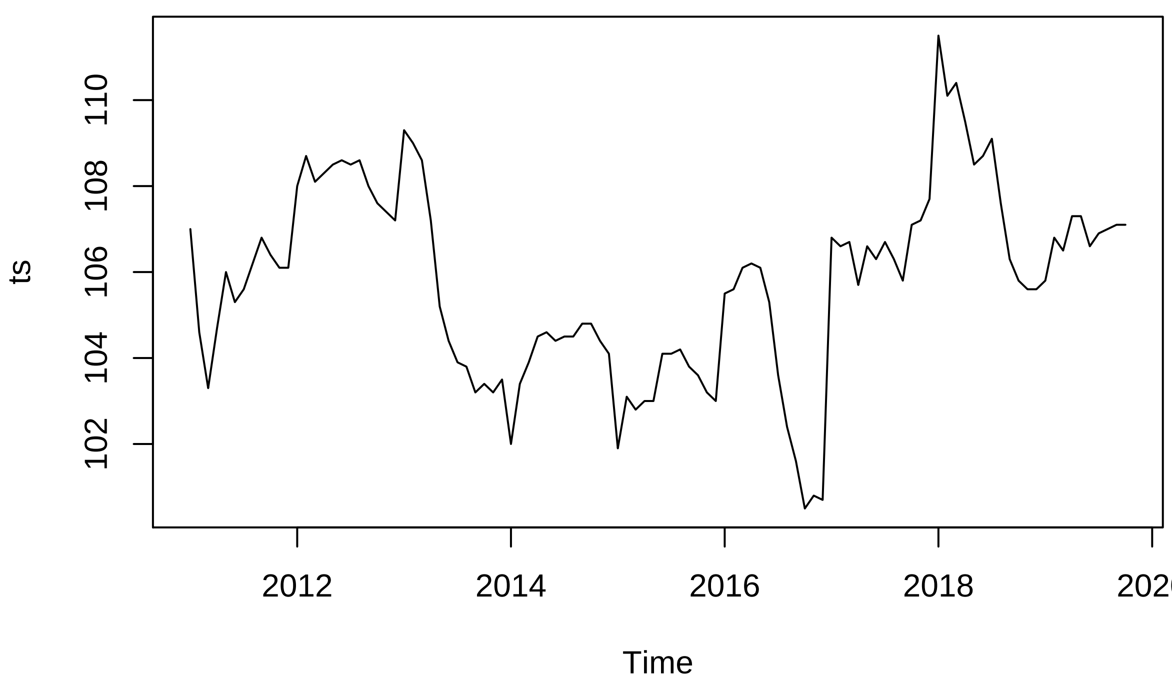
tsf third one

Kevork Sulahian

November 1, 2019

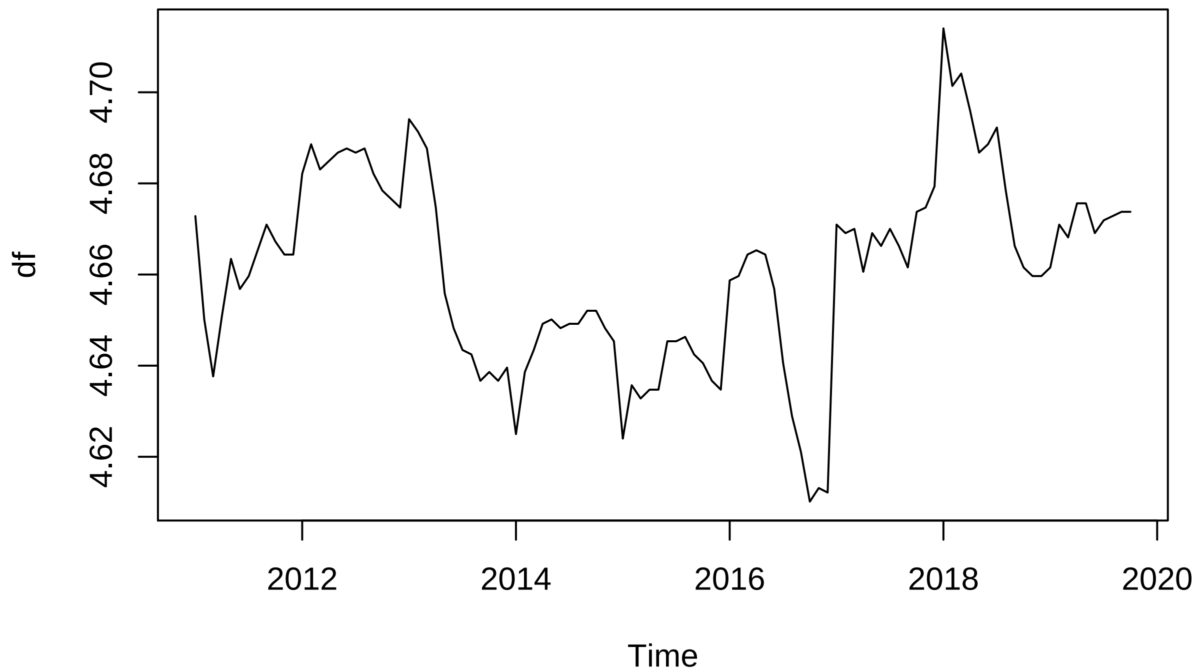
```r
library(readxl)
library(forecast)
```

```r
# library(readxl)

df <- read_xlsx("ts-3.xlsx")

ts = ts(df,start=c(2011,1), frequency = c(12))
```



In this case, it appears that an additive model is not appropriate for describing this time series, since the size of the seasonal fluctuations and random fluctuations seem to increase with the level of the time series. Thus, we may need to transform the time series in order to get a transformed time series that can be described using an additive model. For example, we can transform the time series by calculating the natural log of the original data:

```r
log_ts <- log(ts)
plot.ts(log_ts)
```



Decomposing Time Series

Decomposing a time series means separating it into its constituent components, which are usually a trend component and an irregular component, and if it is a seasonal time series, a seasonal component.

Decomposing Seasonal Data

A seasonal time series consists of a trend component, a seasonal component and an irregular component. Decomposing the time series means separating the time series into these three components: that is, estimating these three components.

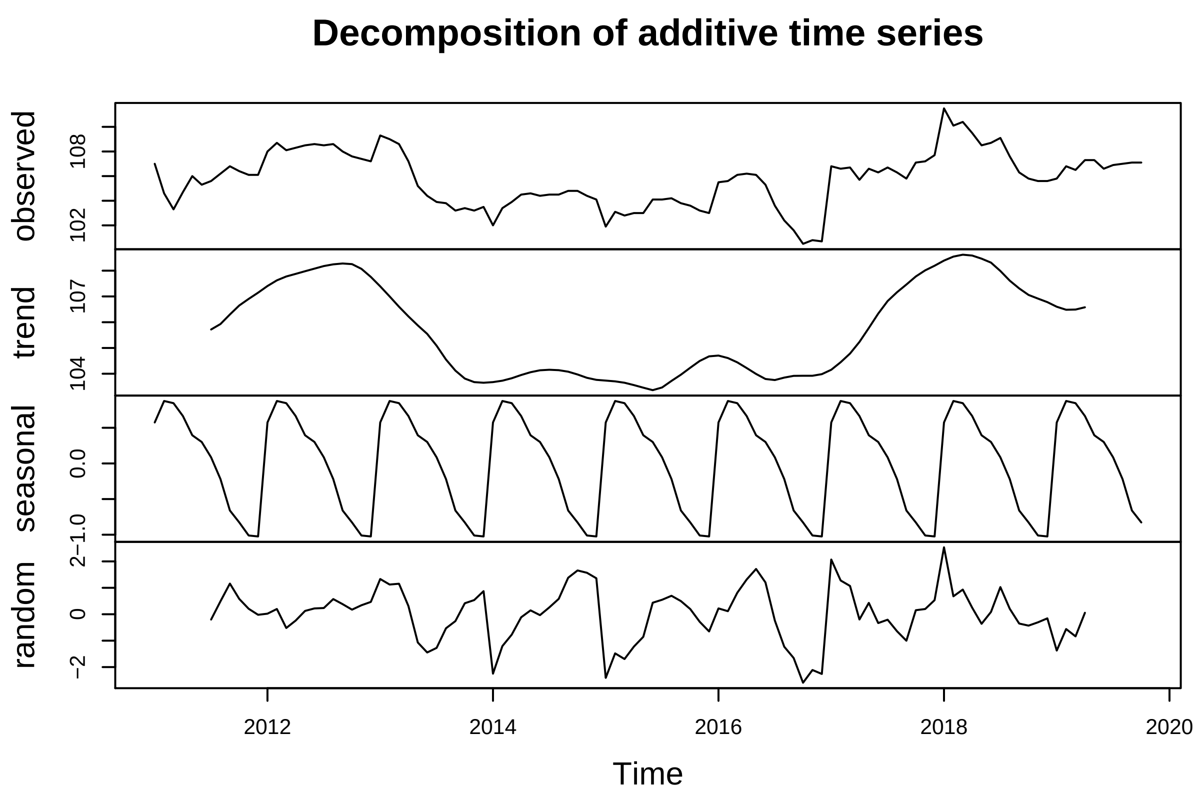
```
ts_components <- decompose(ts)
```

we can print out the estimated values of the seasonal component

```
ts_components$seasonal
```

```
##              Jan         Feb         Mar         Apr         May
## 2011  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2012  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2013  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2014  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2015  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
```

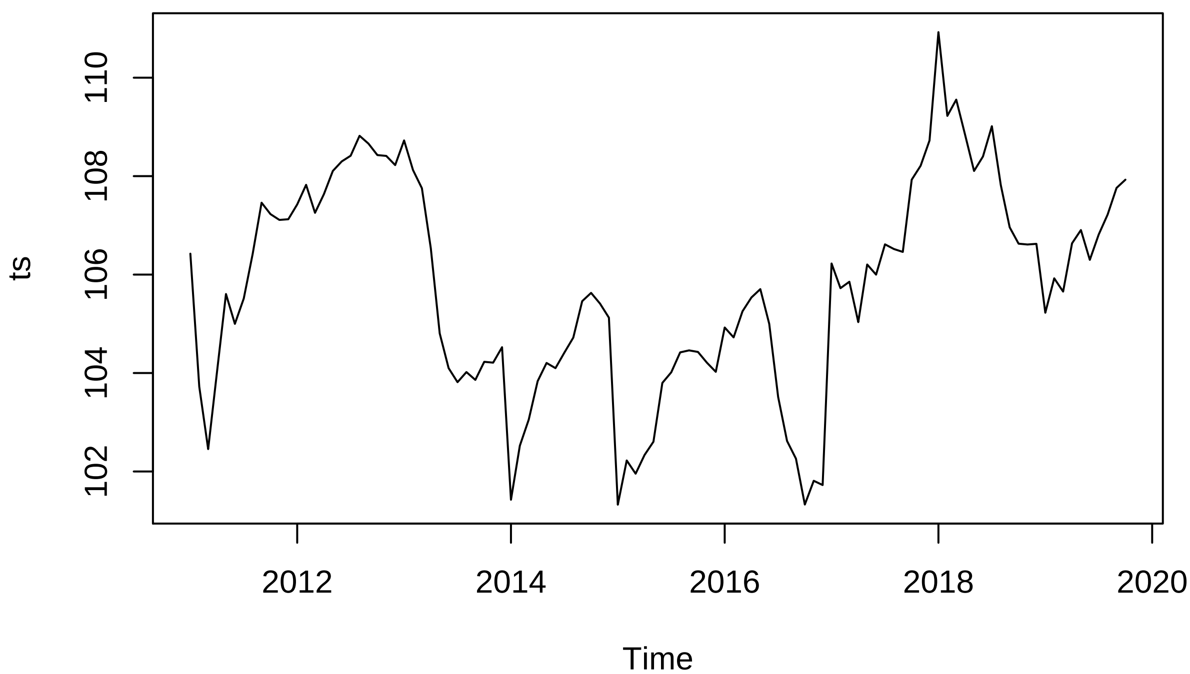
```
## 2016  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2017  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2018  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
## 2019  0.57460937  0.87617187  0.84544271  0.66523438  0.39507068
##              Jun         Jul         Aug         Sep         Oct
## 2011  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2012  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2013  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2014  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2015  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2016  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2017  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2018  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
## 2019  0.30102307  0.08554687 -0.21966146 -0.66028646 -0.82799479
##              Nov         Dec
## 2011 -1.01080729 -1.02434896
## 2012 -1.01080729 -1.02434896
## 2013 -1.01080729 -1.02434896
## 2014 -1.01080729 -1.02434896
## 2015 -1.01080729 -1.02434896
## 2016 -1.01080729 -1.02434896
## 2017 -1.01080729 -1.02434896
## 2018 -1.01080729 -1.02434896
## 2019
```



The plot above shows the original time series (top), the estimated trend component (second from top), the estimated seasonal component (third from top), and the estimated irregular component (bottom)

Seasonally Adjusting

```
ts_seasonall <- ts - ts_components$seasonal
```



Holt-Winters Exponential Smoothing

```
ts_forcaste <- HoltWinters(ts)
ts_forcaste
```

```
## Holt-Winters exponential smoothing with trend and additive seasonal component.
##
## Call:
## HoltWinters(x = ts)
##
## Smoothing parameters:
##  alpha: 0.9403495
##  beta : 0.01258609
##  gamma: 0.9381646
##
## Coefficients:
##              [,1]
## a   107.39454903
```

```
## b      0.08697109
## s1    -0.84709026
## s2    -1.15006574
## s3     0.46849601
## s4     0.77176605
## s5     0.35889154
## s6     0.15085044
## s7     0.15368619
## s8     0.18146907
## s9    -0.17345876
## s10   -0.21630702
## s11   -0.04177130
## s12   -0.29520520
```

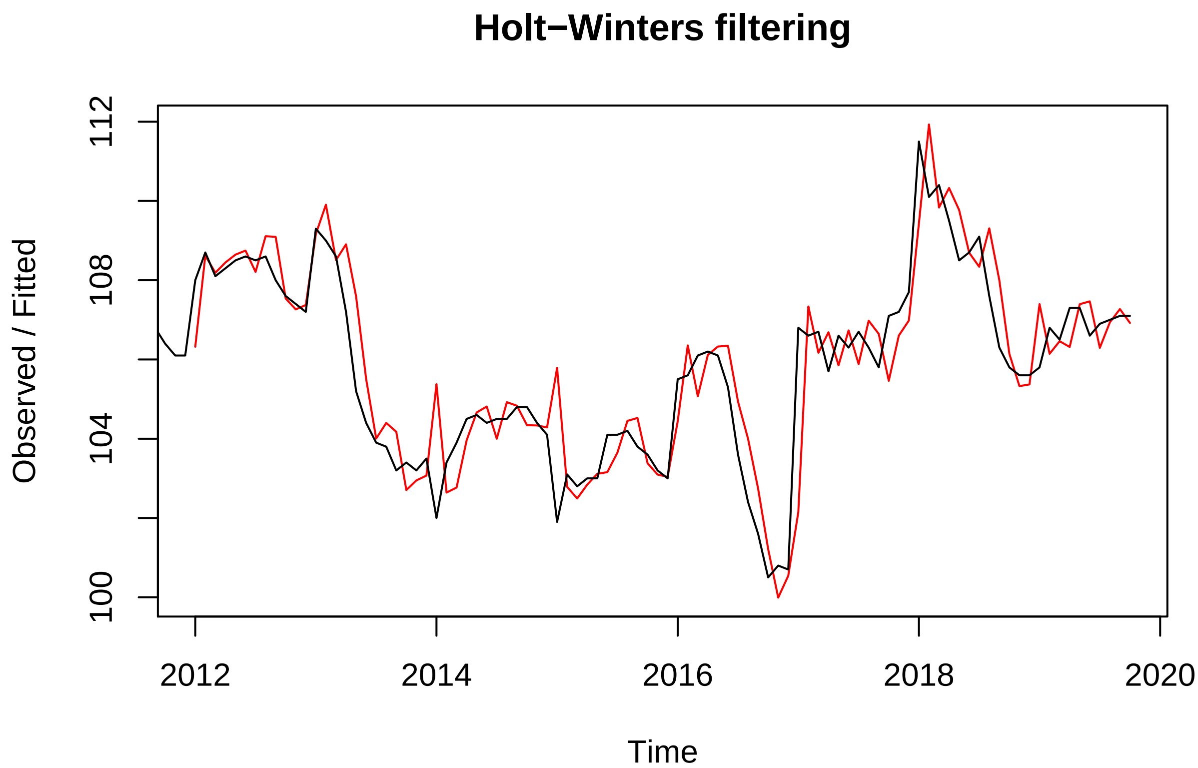
```r
#
```

this is not true i need to change

The value of alpha (0.35) is relatively low, indicating that the estimate of the level at the current time point is based upon both recent observations and some observations in the more distant past. The value of beta is 0.01, indicating that the estimate of the slope b of the trend component is updated but doesn't have much effect over the time series, and instead is set equal to its initial value. This makes good intuitive sense, as the level changes quite a bit over the time series, but the slope b of the trend component remains roughly the same. In contrast, the value of gamma (0.38) is high, indicating that the estimate of the seasonal component at the current time point is not just based upon very recent observations

```r
ts_forcaste$SSE
```

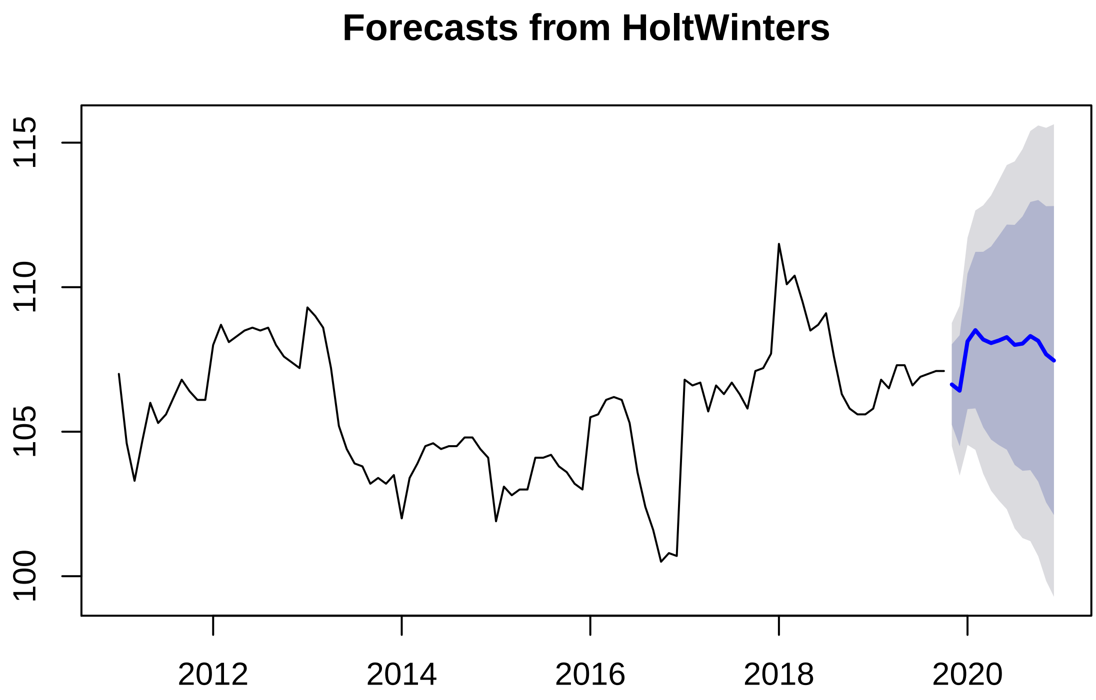
```
## [1] 111.0047
```



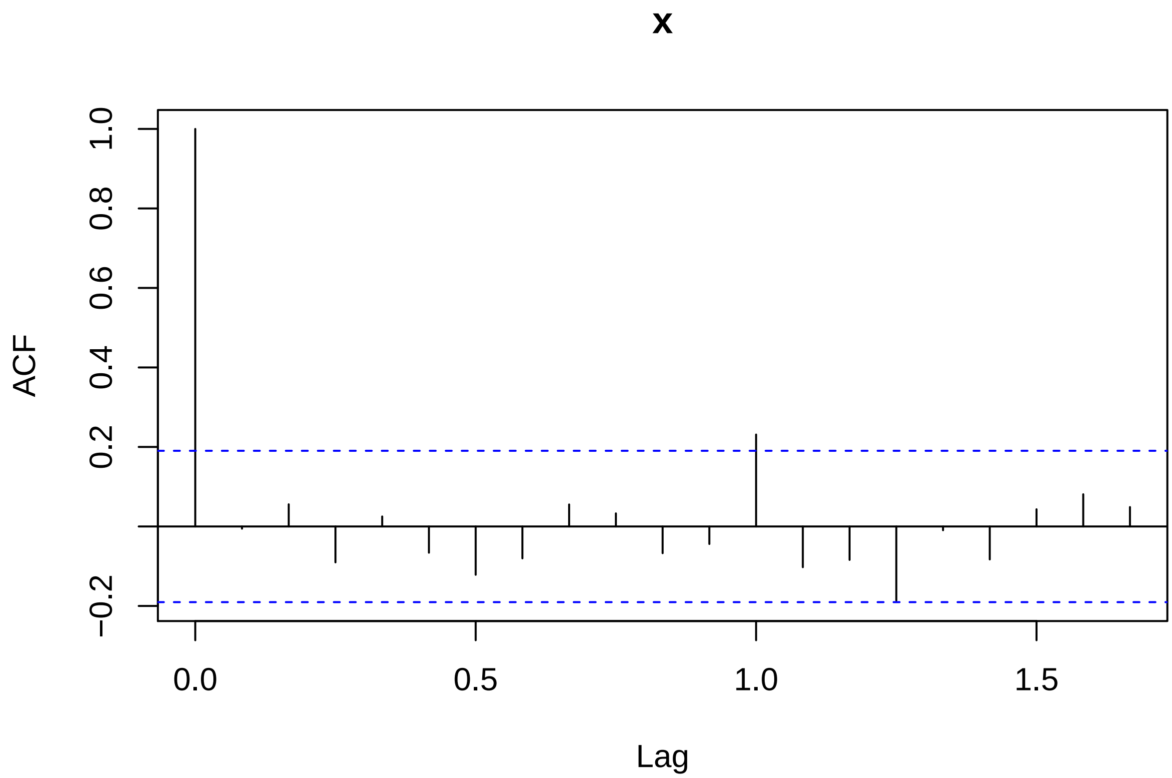

```
ts_forcaste2 = forecast:::forecast.HoltWinters(ts_forcaste, h= 14)
(as.data.frame(ts_forcaste2))[1]
```

```
##          Point Forecast
## Nov 2019       106.6344
## Dec 2019       106.4184
## Jan 2020       108.1240
## Feb 2020       108.5142
## Mar 2020       108.1883
## Apr 2020       108.0672
## May 2020       108.1570
## Jun 2020       108.2718
## Jul 2020       108.0038
## Aug 2020       108.0480
## Sep 2020       108.3095
## Oct 2020       108.1430
## Nov 2020       107.6781
## Dec 2020       107.4621
```

```
HW_pred = (as.data.frame(ts_forcaste2))[1]
HW_pred=HW_pred[3,]
```



We can investigate whether the predictive model can be improved upon by checking whether the in-sample forecast errors show non-zero autocorrelations at lags 1-20, by making a correlogram and carrying out the Ljung-Box test:



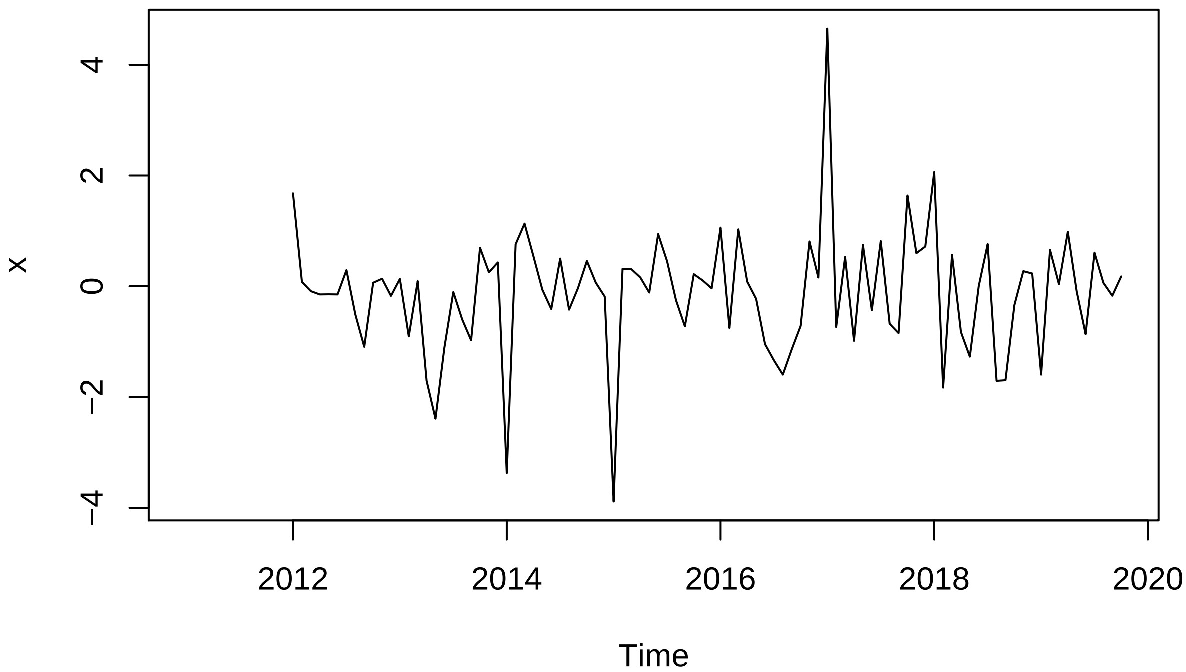

```
##
##  Box-Ljung test
##
## data:  ts_forcaste2$residuals
## X-squared = 18.88, df = 20, p-value = 0.5297
```

p-value is 0.5 instead of 0.9

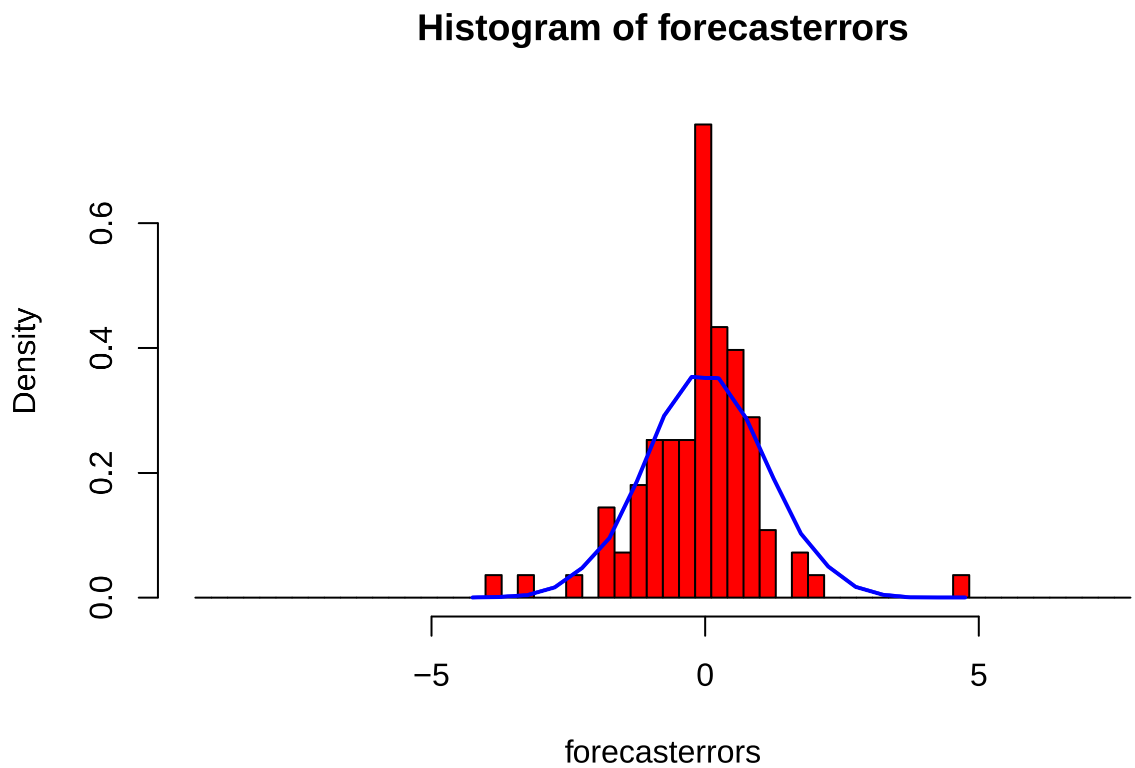
The correlogram shows that the autocorrelations for the in-sample forecast errors do not exceed the significance bounds for lags 1-20. Furthermore, the p-value for Ljung-Box test is 0.9, indicating that there is no evidence of non-zero autocorrelations at lags 1-20.

We can check whether the forecast errors have constant variance over time, and are normally distributed with mean zero, by making a time plot of the forecast errors and a histogram (with overlaid normal curve):

```
plot.ts(ts_forcaste2$residuals)
```



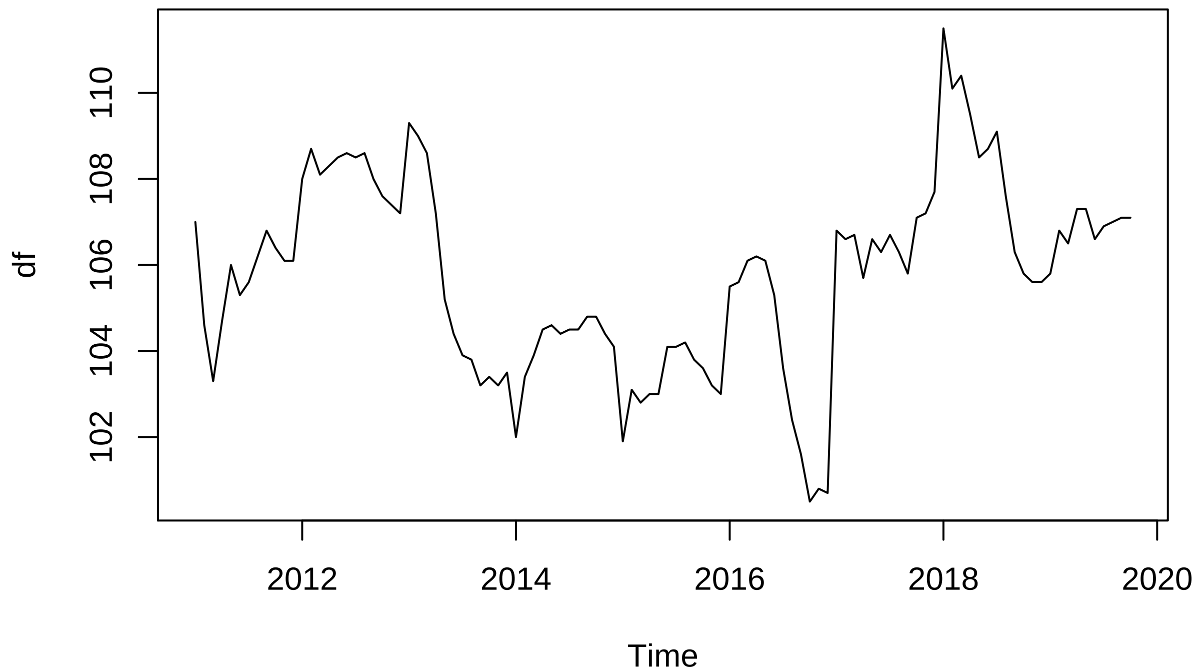

```r
plotForecastErrors(ts_forcaste2$residuals)
```



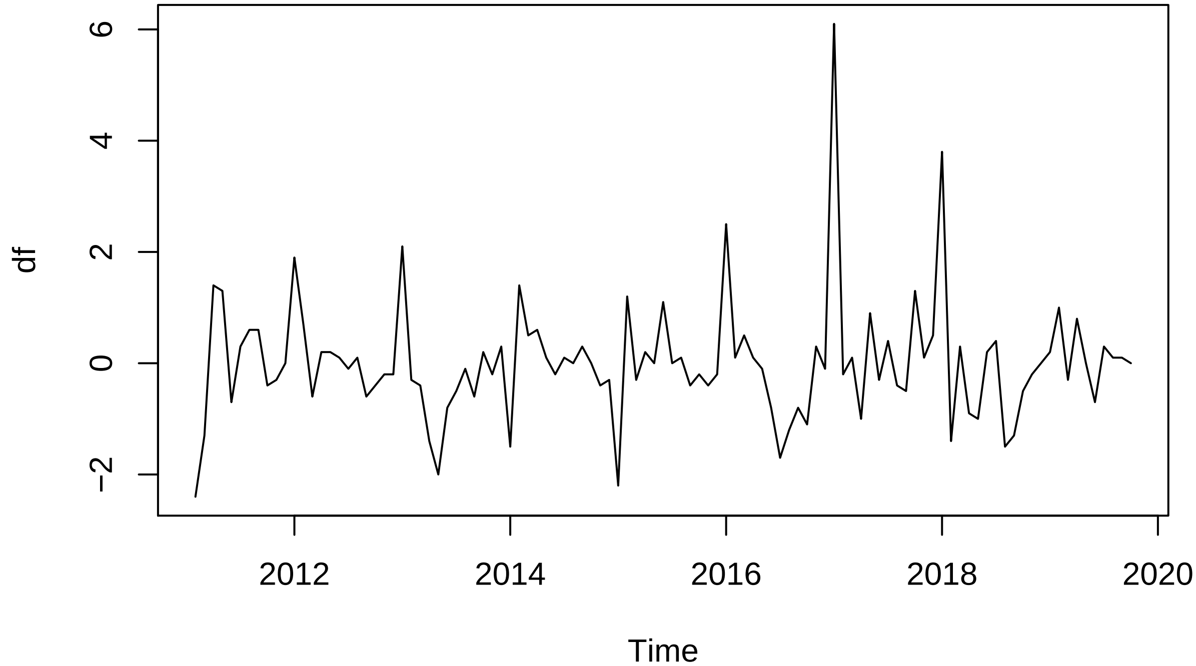
From the time plot, it appears plausible that the forecast errors have constant variance over time. From the histogram of forecast errors, it seems plausible that the forecast errors are normally distributed with mean zero.

Thus,there is little evidence of autocorrelation at lags 1-20 for the forecast errors, and the forecast errors appear to be normally distributed with mean zero and constant variance over time. This suggests that Holt-Winters exponential smoothing provides an adequate predictive model of the log of total productivity, which probably cannot be improved upon. Furthermore, the assumptions upon which the prediction intervals were based are probably valid.

```
plot.ts(ts)
```

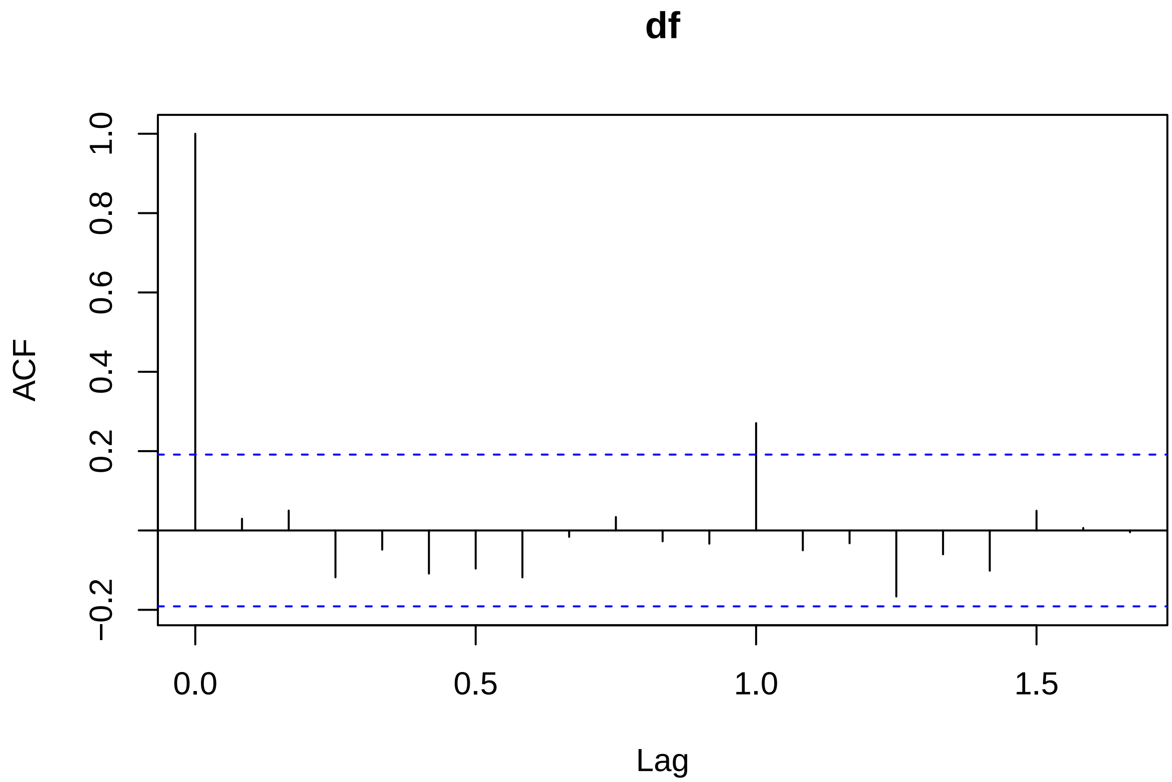



```
ts_diff1 <-  diff(ts, differences = 1)

plot.ts(ts_diff1)
```



The time series of differences (above) does appear to be stationary in mean and variance, as the level of the series stays roughly constant over time, and the variance of the series appears roughly constant over time

```
acf(ts_diff1, lag.max=20)             # plot a correlogram
```

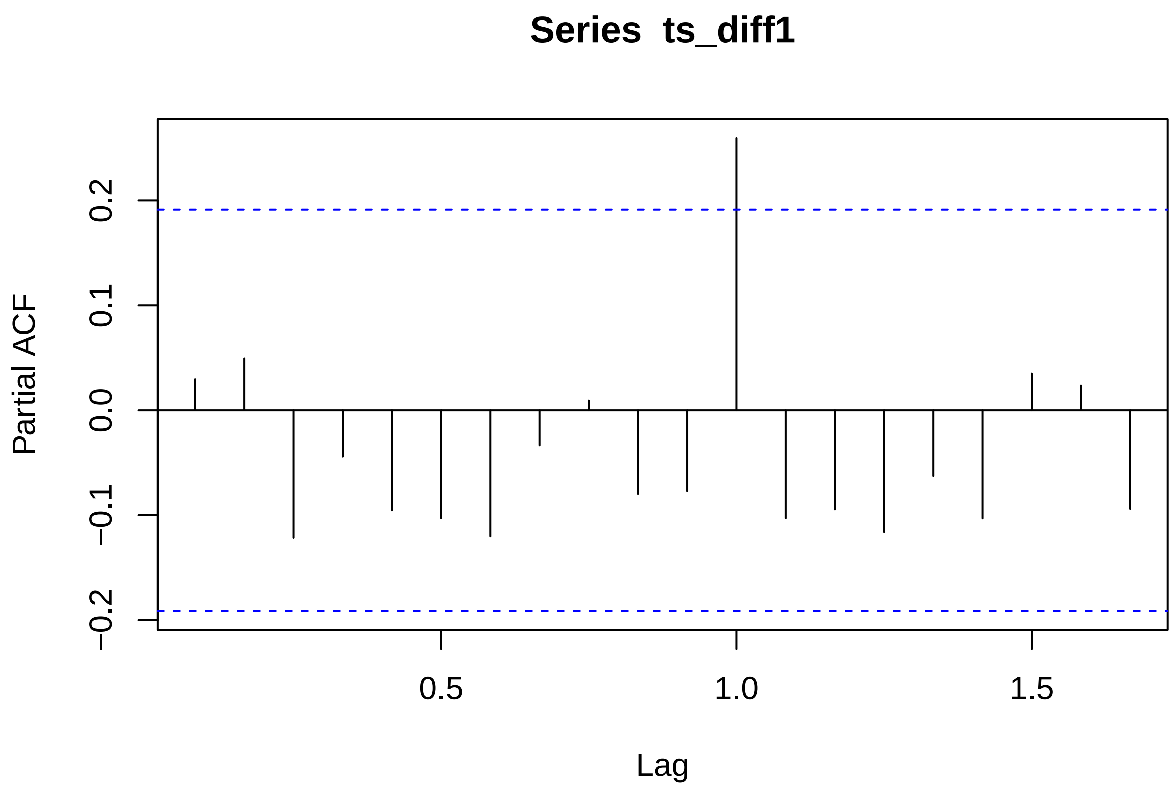


We see from the correlogram that the autocorrelation exceeds the significance bound 3 times but all the others do not exceed

```r
acf(ts_diff1, lag.max=20, plot=FALSE) # get the autocorrelation values
```

```
##
## Autocorrelations of series 'ts_diff1', by lag
##
## 0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500
##  1.000  0.030  0.050 -0.118 -0.048 -0.109 -0.096 -0.118 -0.016  0.034
## 0.8333 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833
## -0.027 -0.033  0.271 -0.050 -0.032 -0.166 -0.060 -0.101  0.050  0.006
## 1.6667
## -0.004
```

```r
pacf(ts_diff1, lag.max=20)             # plot a partial correlogram
```




```r
pacf(ts_diff1, lag.max=20, plot=FALSE) # get the partial autocorrelation values
```

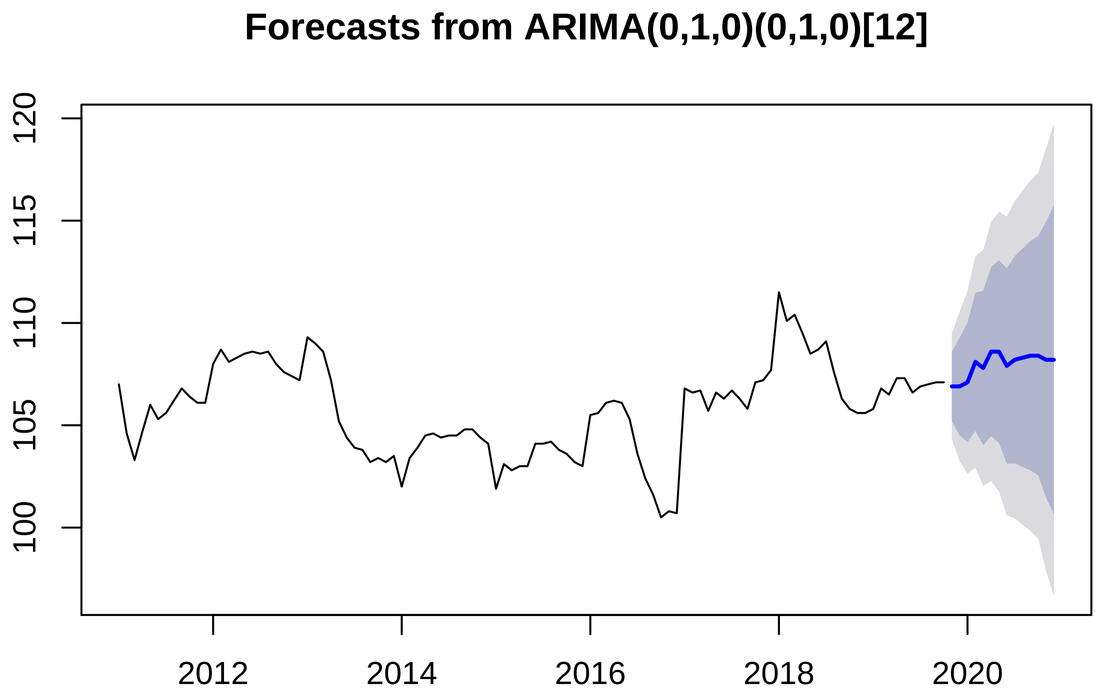
```
##
## Partial autocorrelations of series 'ts_diff1', by lag
##
## 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333
##  0.030  0.049 -0.121 -0.044 -0.095 -0.103 -0.120 -0.033  0.009 -0.080
## 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667
## -0.077  0.259 -0.103 -0.094 -0.116 -0.063 -0.103  0.035  0.024 -0.094
```

Arima, 0,1,0

```r
ts_arima = Arima(ts, order=c(0,1,0),seasonal = list(order = c(0,1,0)))
ts_arima
```

```
## Series: ts
## ARIMA(0,1,0)(0,1,0)[12]
##
## sigma^2 estimated as 1.735:  log likelihood=-157.55
## AIC=317.11   AICc=317.15   BIC=319.64
```

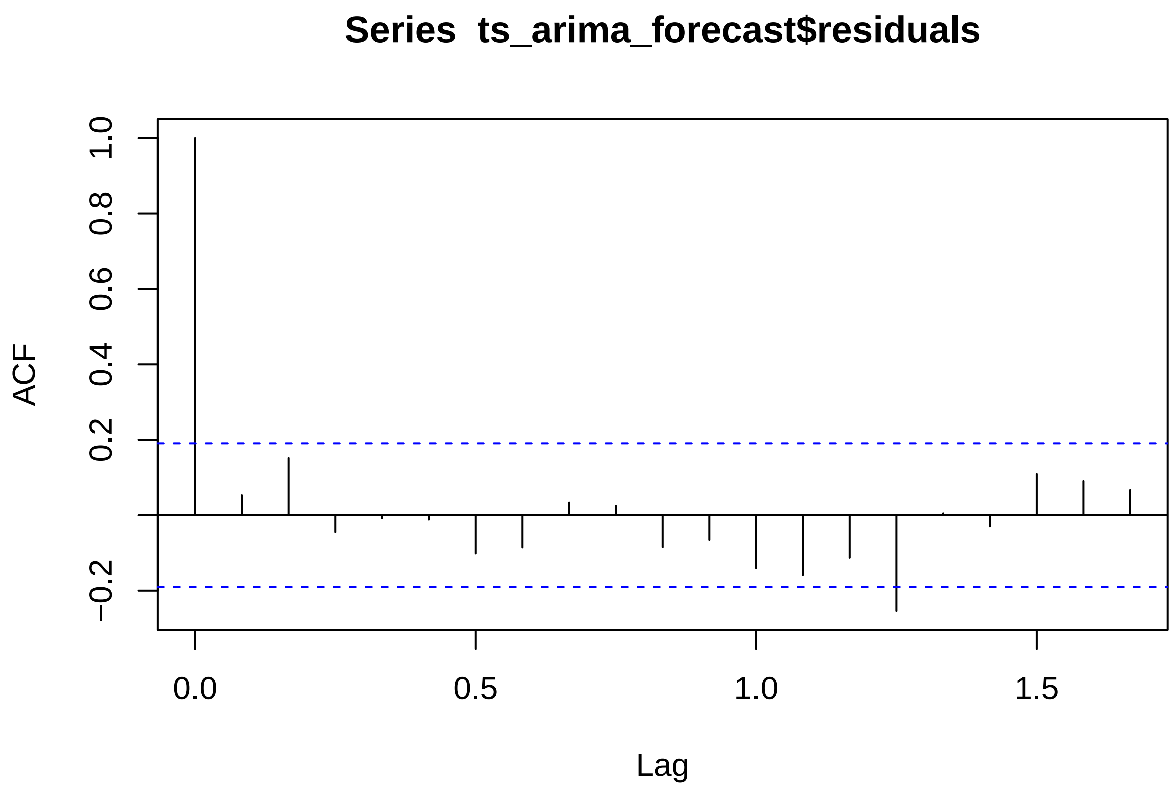
```
ts_arima_forecast = forecast(ts_arima,h = 14)
arima_pred = ts_arima_forecast[4]
arima_pred = arima_pred$mean[14]
forecast:::plot.forecast(ts_arima_forecast)
```



As in the case of exponential smoothing models, it is a good idea to investigate whether the forecast errors of an ARIMA model are normally distributed with mean zero and constant variance, and whether the are correlations between successive forecast errors.

For example, we can make a correlogram of the forecast errors for our ARIMA(0,1,1) model, and perform the Ljung-Box test for lags 1-20, by typing:

```
acf(ts_arima_forecast$residuals, lag.max=20)
```

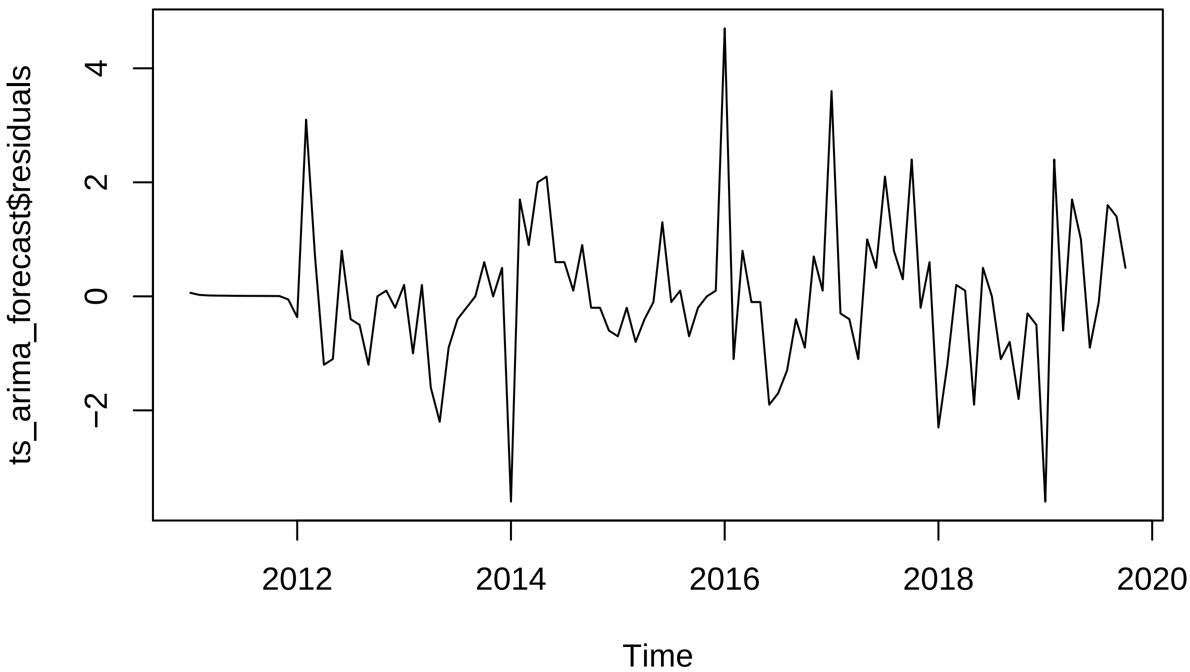



```r
Box.test(ts_arima_forecast$residuals, lag=20, type="Ljung-Box")
```

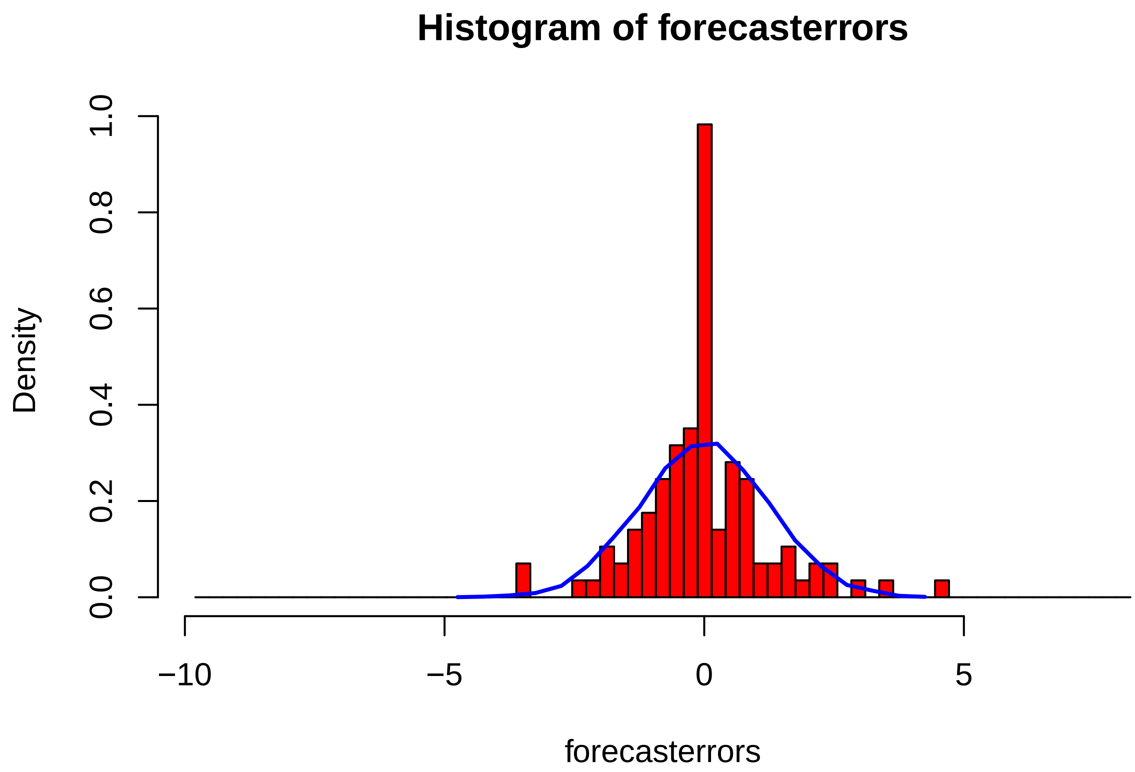
```
##
##  Box-Ljung test
##
## data:  ts_arima_forecast$residuals
## X-squared = 25.235, df = 20, p-value = 0.1926
```

p value too high to reject

```r
plot.ts(ts_arima_forecast$residuals)            # make time plot of forecast errors
```




```
plotForecastErrors(ts_arima_forecast$residuals)
```

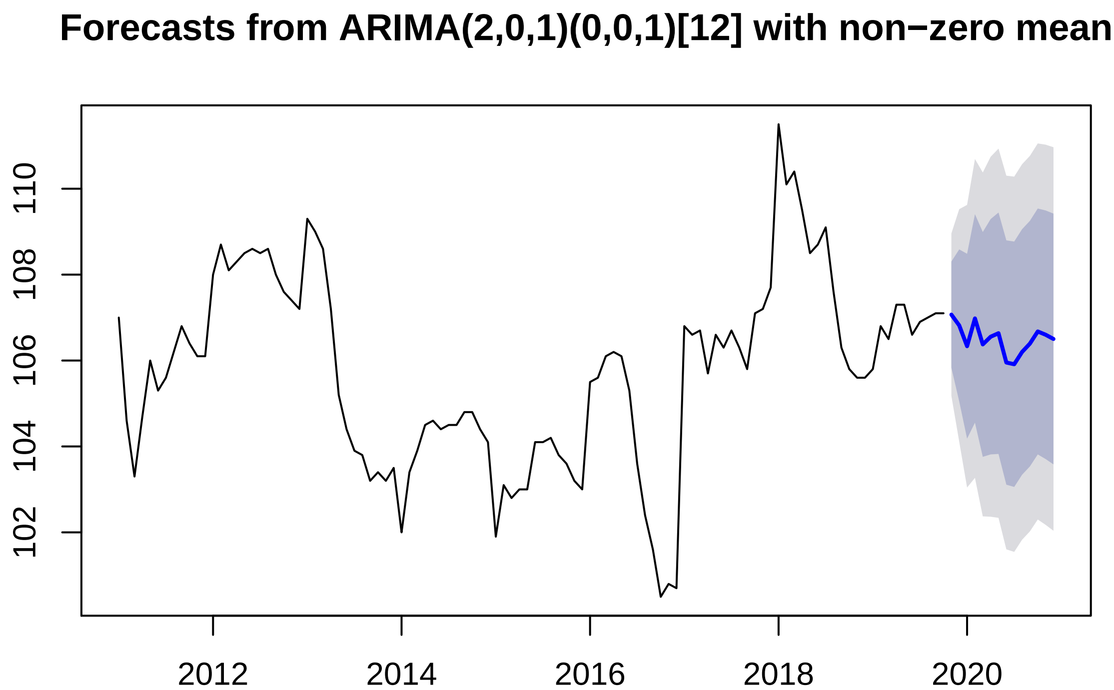


A model chosen automatically

```
fit3 <- auto.arima(ts, seasonal = T)
fit3
```

```
## Series: ts
## ARIMA(2,0,1)(0,0,1)[12] with non-zero mean
##
## Coefficients:
##          ar1      ar2      ma1    sma1      mean
##       1.7749  -0.8282  -0.7506  0.5355  105.9343
## s.e.  0.1036   0.0933   0.1282  0.1159    0.6408
##
## sigma^2 estimated as 0.93:  log likelihood=-146.68
## AIC=305.35   AICc=306.2   BIC=321.33
```

```
fit_forecast = forecast(fit3,h=14)
plot(fit_forecast)
```




```
# str(fit)
auto.arima_pred = (as.data.frame(fit_forecast))[1][c(14),]
```