report_tsf_HW_ARIMA[1,1,1]

Kevork Sulahian

September 6, 2019

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

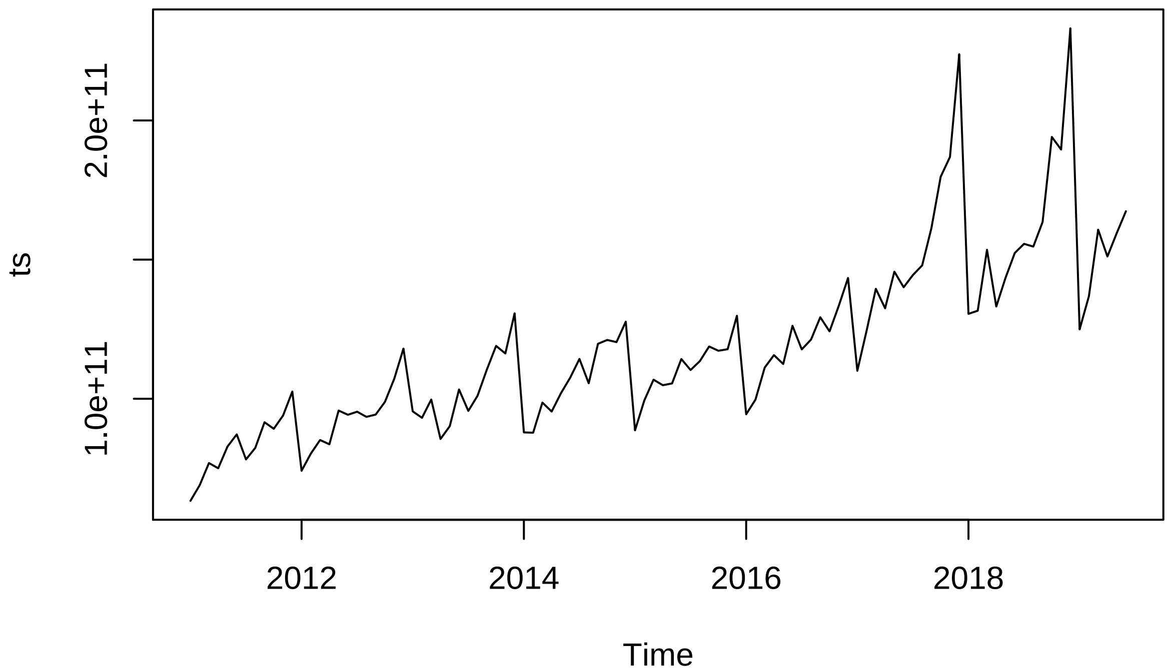
```r
df <- read_xls('economy.xls', sheet='2011-2019 NACE 2')
```

```
## New names:
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * `` -> ...6
## * ... and 99 more problems
```

```r
df = df[4,]
df = df[-c(1,3)]
rownames(df) = df[1]
```

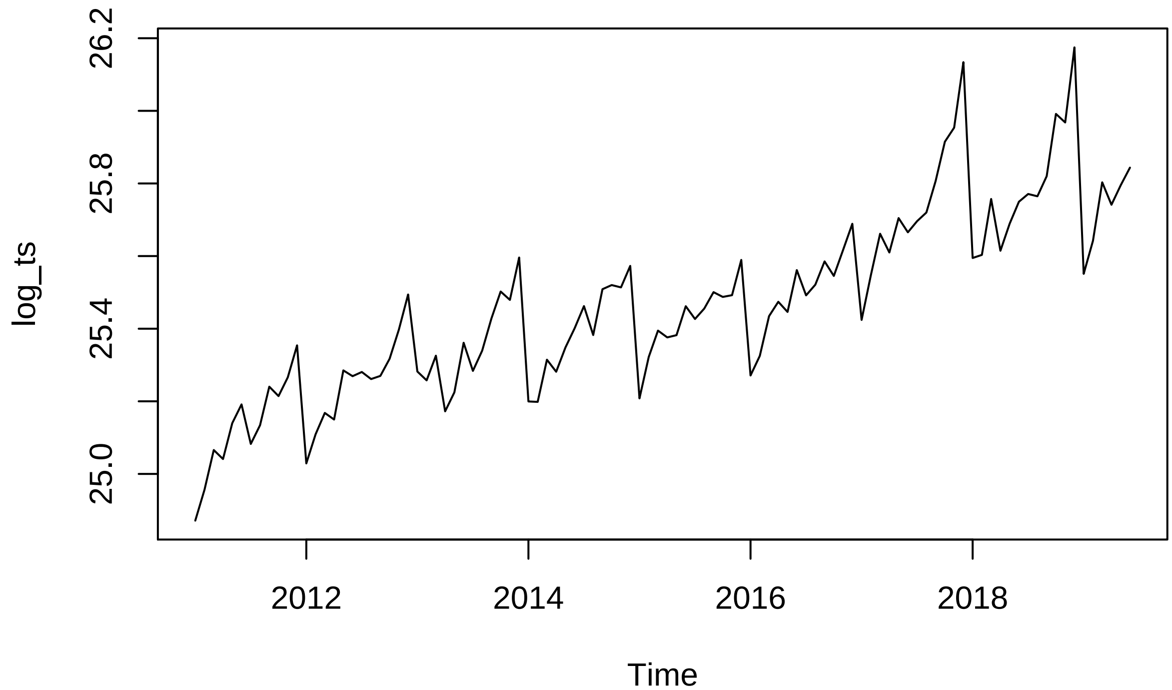
```
## Warning: Setting row names on a tibble is deprecated.
```

```r
df = df[-1]
df = t(df)
df[] <- sapply(df[],function(x) as.numeric(as.character(x)))
df = as.numeric(df)
df= df * 1000000
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:

```
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.

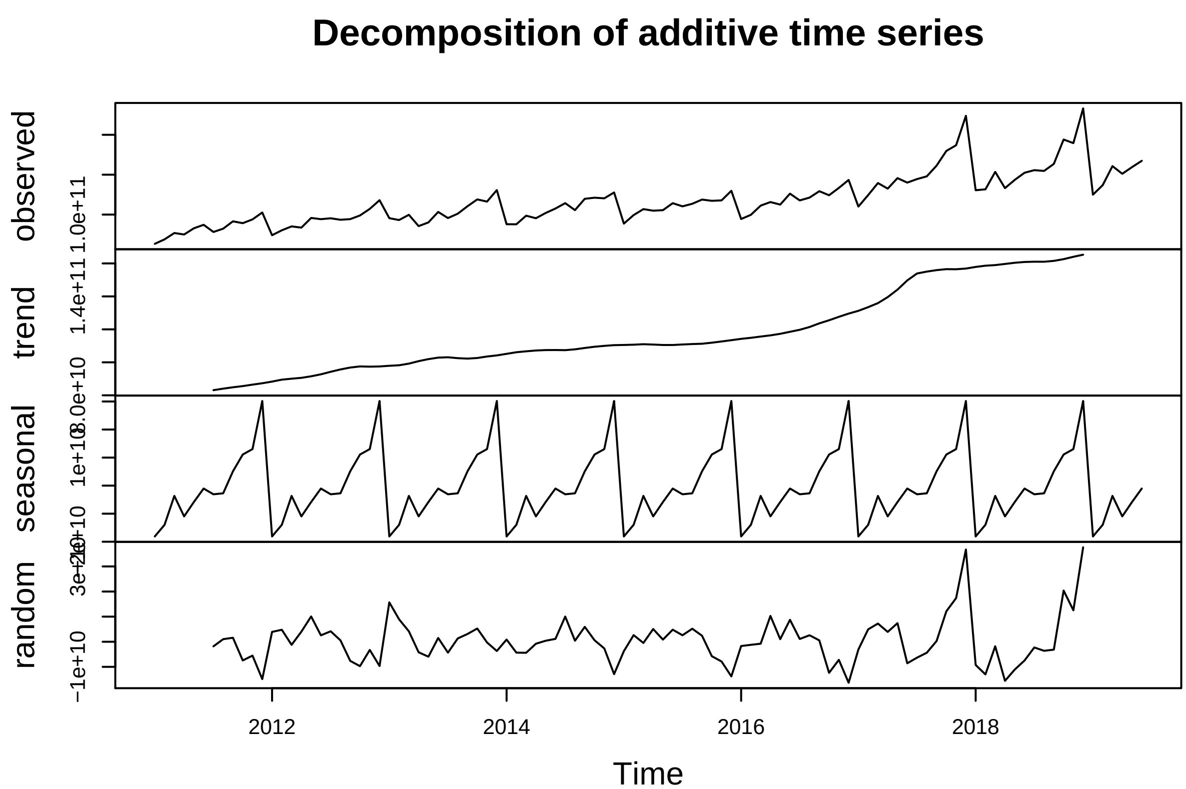
```
ts_components <- decompose(ts)
```

we can print out the estimated values of the seasonal component

```
ts_components$seasonal
```

```
##              Jan          Feb          Mar          Apr          May
## 2011 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2012 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2013 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2014 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2015 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
```

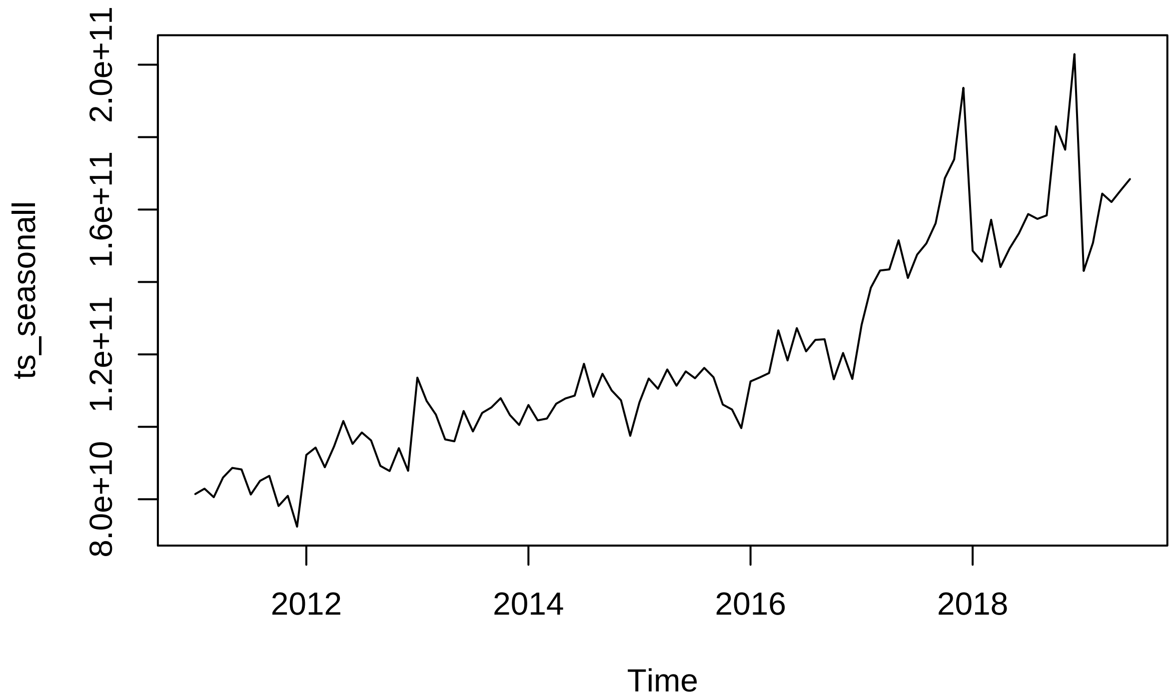
```
## 2016 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2017 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2018 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
## 2019 -18114721416 -13972761892  -3652881535 -10968033321  -5840660702
##               Jun          Jul          Aug          Sep          Oct
## 2011  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2012  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2013  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2014  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2015  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2016  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2017  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2018  -1030591654  -3088941580  -2737647830   5107804253  11091117274
## 2019  -1030591654
##               Nov          Dec
## 2011  13028793837  30178524566
## 2012  13028793837  30178524566
## 2013  13028793837  30178524566
## 2014  13028793837  30178524566
## 2015  13028793837  30178524566
## 2016  13028793837  30178524566
## 2017  13028793837  30178524566
## 2018  13028793837  30178524566
## 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

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



Holt-Winters Exponential Smoothing

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

```
## Holt-Winters exponential smoothing with trend and additive seasonal component.
##
## Call:
## HoltWinters(x = ts)
##
## Smoothing parameters:
##  alpha: 0.3841348
##  beta : 0
##  gamma: 1
##
## Coefficients:
##             [,1]
## a   166929650345
## b      859005492
## s1    2680812331
## s2    1004407035
```

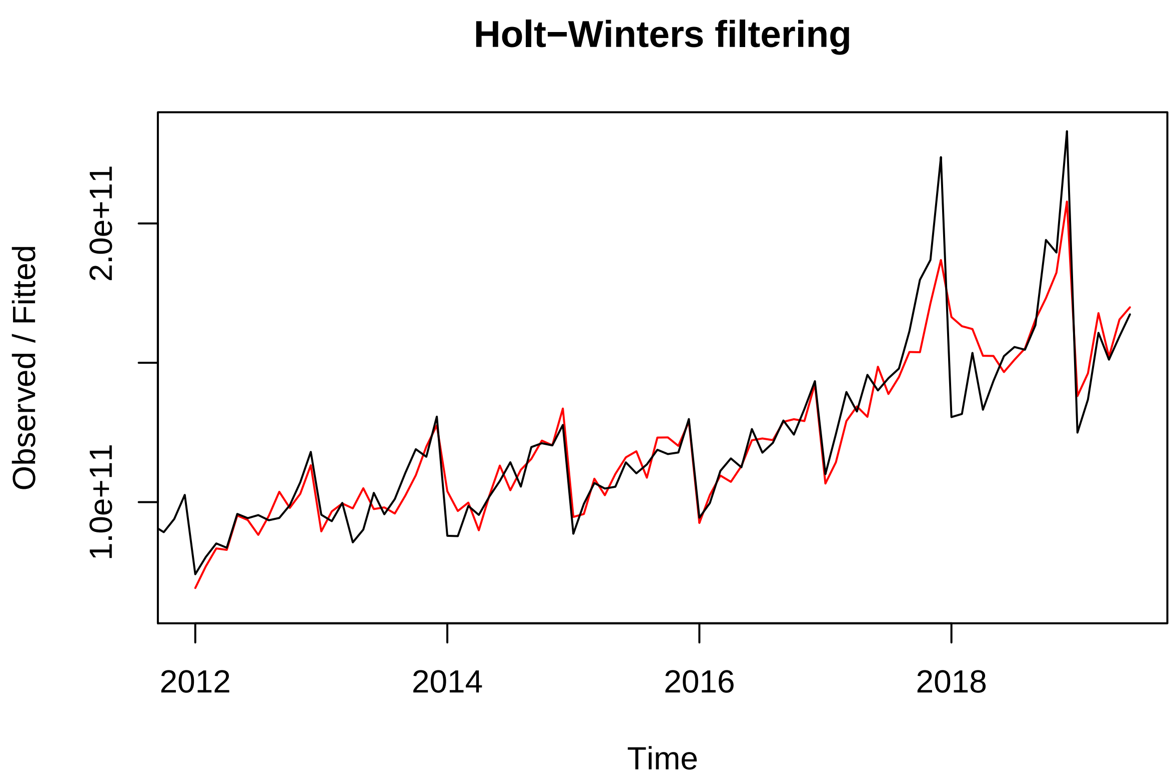
```
## s3    9728353964
## s4   31427608853
## s5   23274665015
## s6   56266579294
## s7  -47717485979
## s8  -33061264111
## s9   -7294279687
## s10 -17388238299
## s11  -7579809787
## s12    458749655
```

```
#
```

The value of alpha (0.41) 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 0.00, indicating that the estimate of the slope b of the trend component is not updated 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 (0.96) is high, indicating that the estimate of the seasonal component at the current time point is just based upon very recent observations

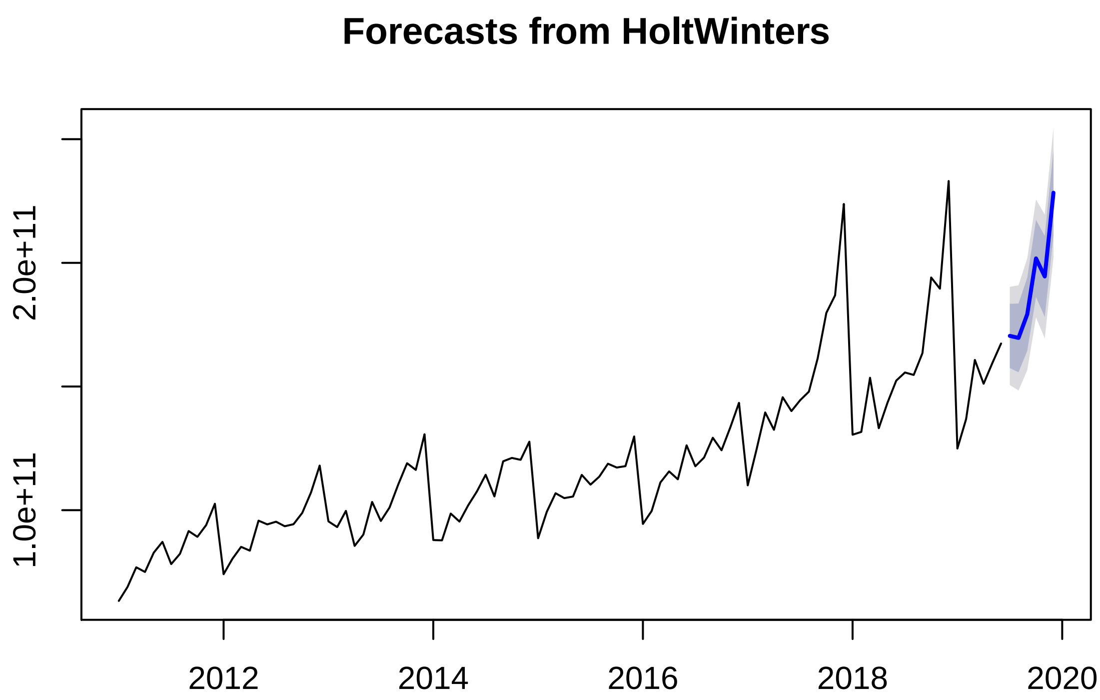
```
ts_forcaste$SSE
```

```
## [1] 9.131448e+21
```



```r
ts_forcaste2 = forecast:::forecast.HoltWinters(ts_forcaste, h= 6)
(as.data.frame(ts_forcaste2))[1]
```

```
##          Point Forecast
## Jul 2019   170469468169
## Aug 2019   169652068366
## Sep 2019   179235020787
## Oct 2019   201793281168
## Nov 2019   194499342823
## Dec 2019   228350262594
```

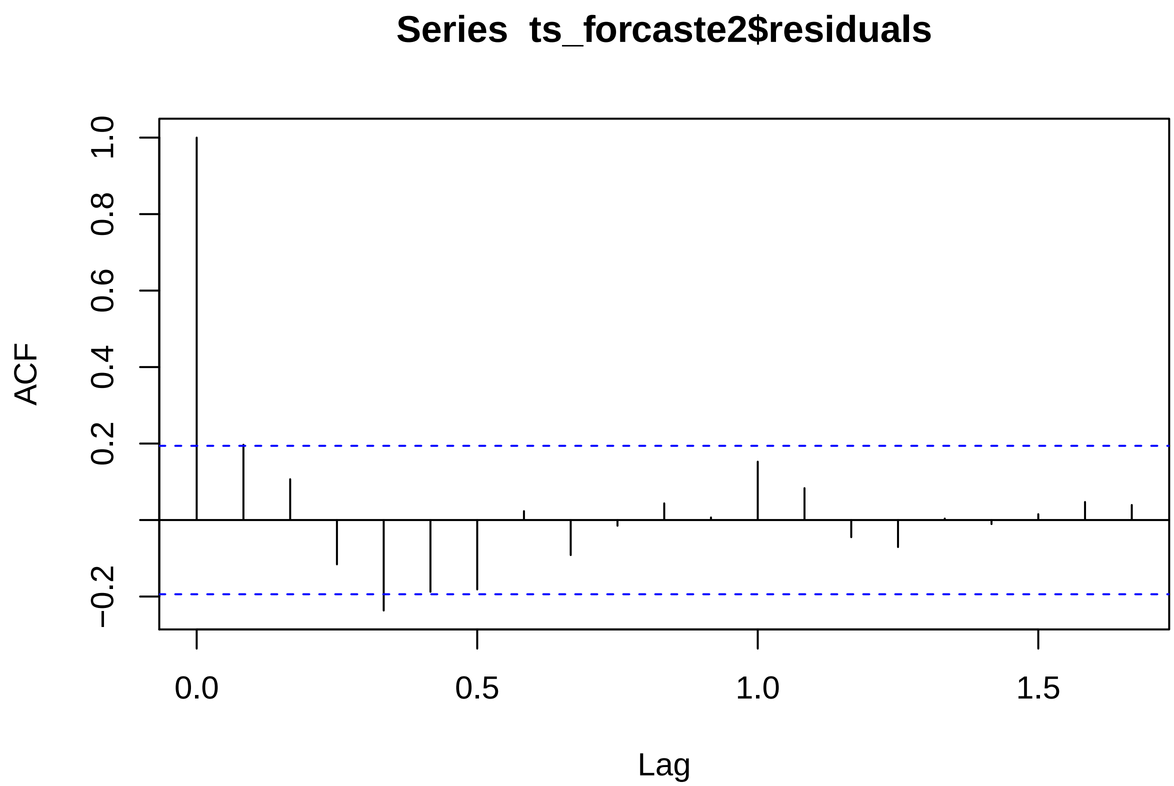


Growth

```r
last_year <- window(ts, 2018, c(2018,12))
this_year <- window(ts, 2019)
this_year_predict_HW <- (as.data.frame(ts_forcaste2))[1]

growth_HW <- growth(sum(c(this_year,as.numeric(this_year_predict_HW$`Point Forecast`))), sum(last_year))
growth_HW
```

```
## [1] 0.05644614
```

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



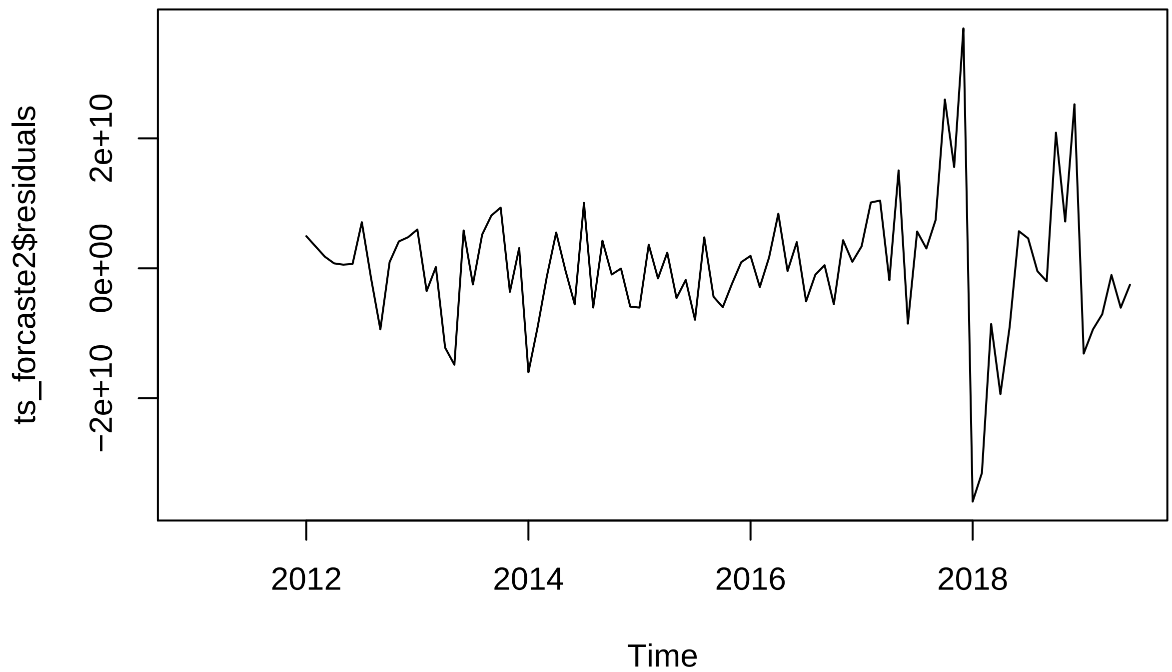
Ljung-Box test:

```
##
##  Box-Ljung test
##
## data:  ts_forcaste2$residuals
## X-squared = 23.626, df = 20, p-value = 0.2591
```

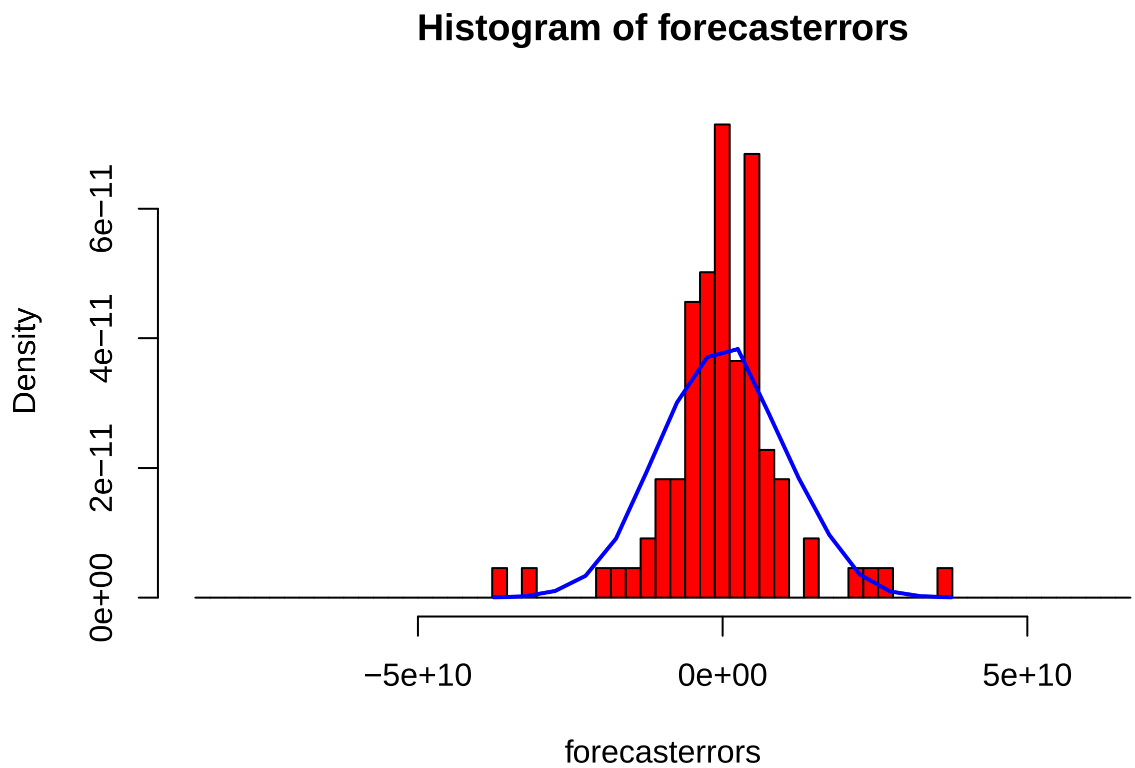
The correlogram shows that the autocorrelations for the in-sample forecast errors do not exceed the significance bounds for lags 1-20. Furthermore, the p-value for Ljung-Box test is 0.2, indicating that there is little evidence of non-zero autocorrelations at lags 1-20.

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):

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



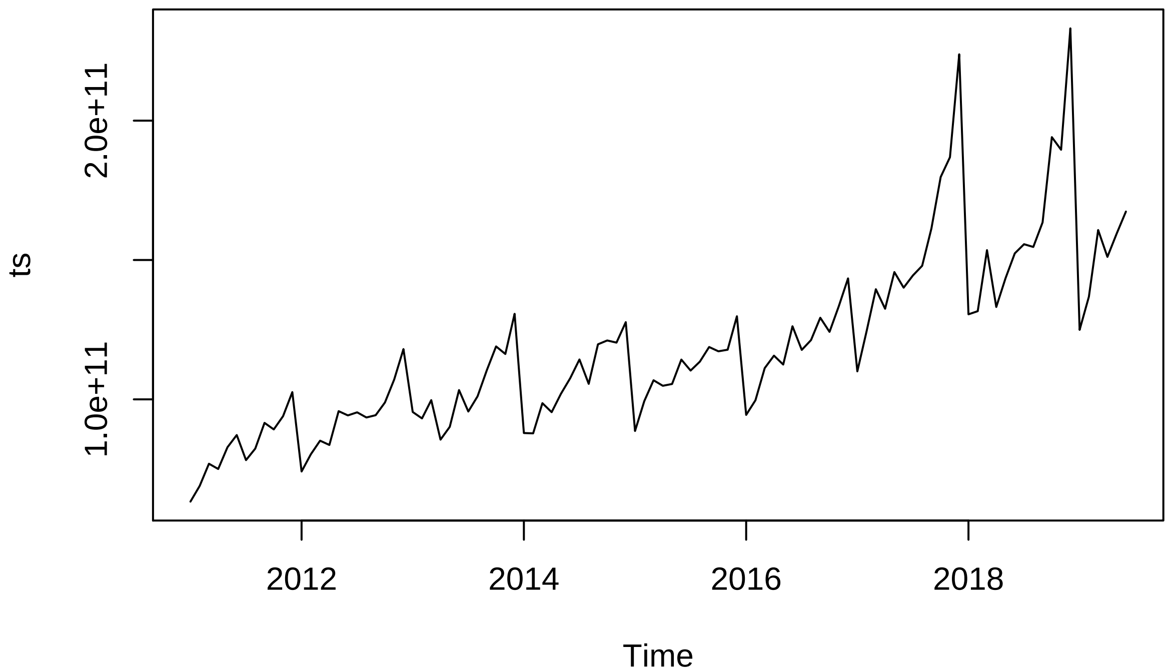

```r
plotForecastErrors(ts_forcaste2$residuals)
```



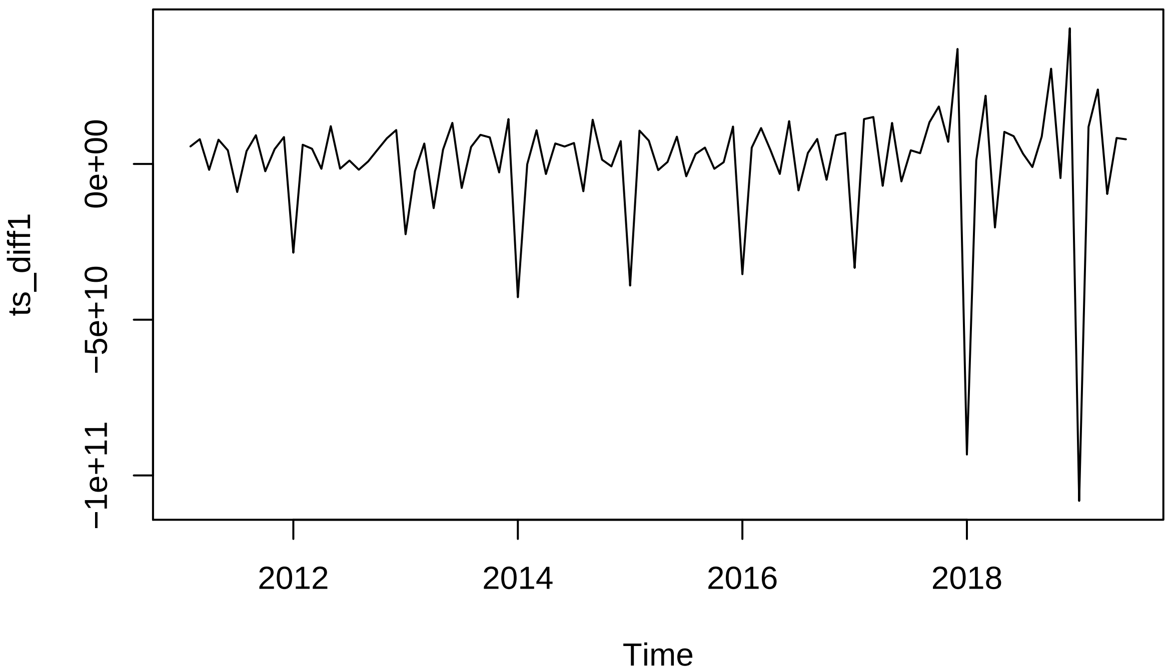
From the time plot, it appears plausible that the forecast errors have constant variance over time. From the histogram of forecast errors, it seems plausible that the forecast errors are normally distributed with mean zero.

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)
```

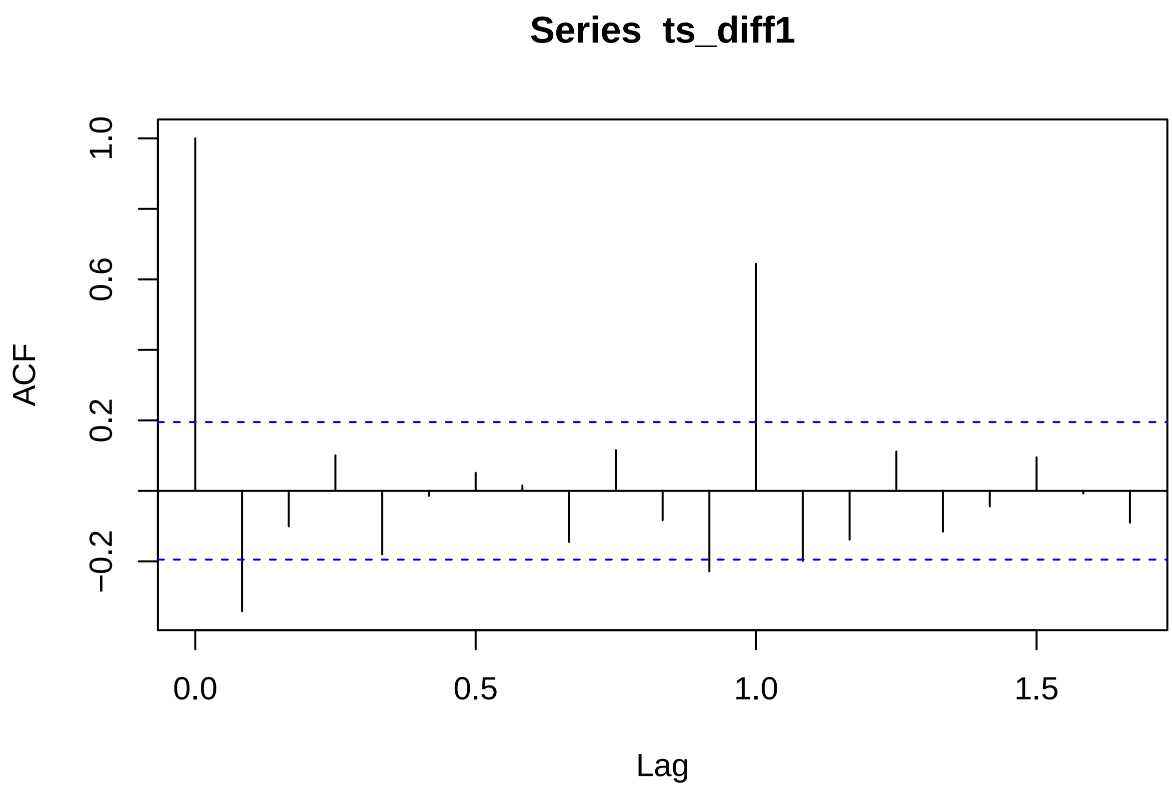



```r
ts_diff1 <-  diff(ts, differences = 1)

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
```

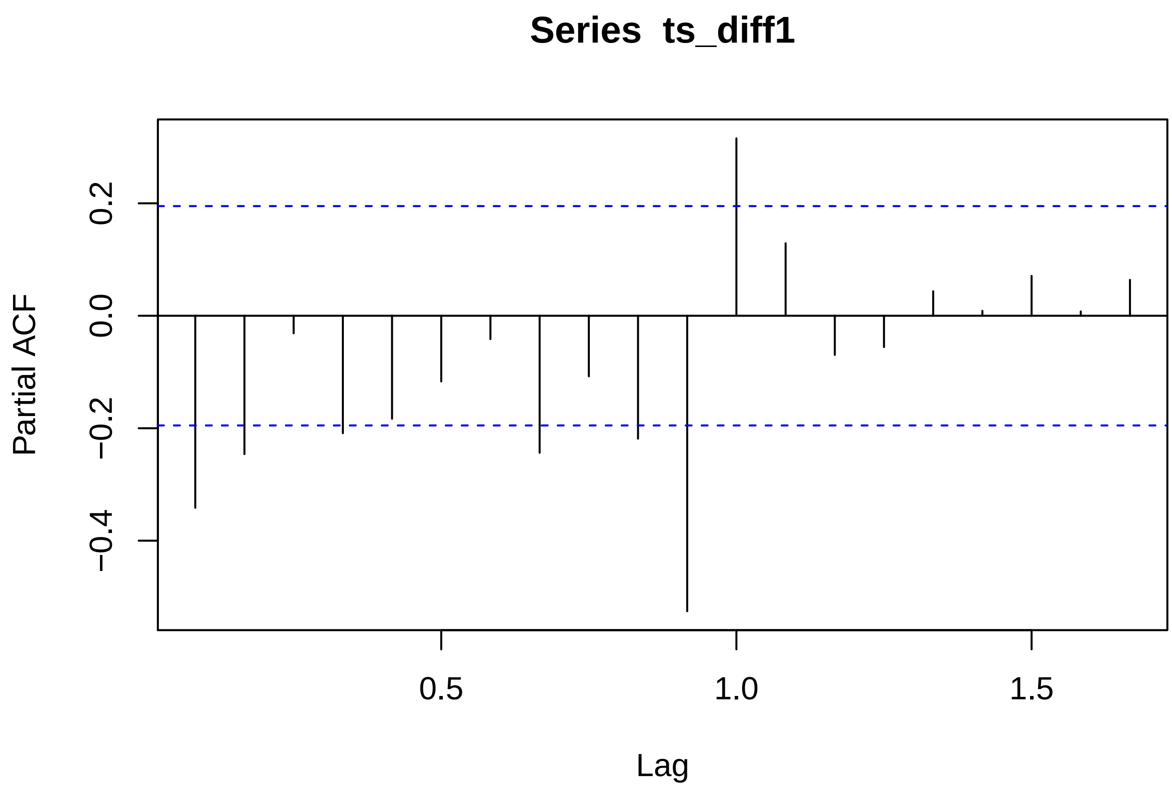


We see from the correlogram that the autocorrelation exceeds the significance bound 3 times but all the others do not exceed

```
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 -0.342 -0.101  0.101 -0.180 -0.014  0.051  0.015 -0.145  0.115
## 0.8333 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833
## -0.083 -0.228  0.644 -0.199 -0.138  0.112 -0.116 -0.044  0.095 -0.007
## 1.6667
## -0.090
```

```
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
## -0.342 -0.246 -0.031 -0.209 -0.183 -0.117 -0.041 -0.244 -0.108 -0.219
## 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667
## -0.525  0.315  0.129 -0.070 -0.056  0.044  0.009  0.071  0.008  0.064
```

Arima, 1,1,1

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

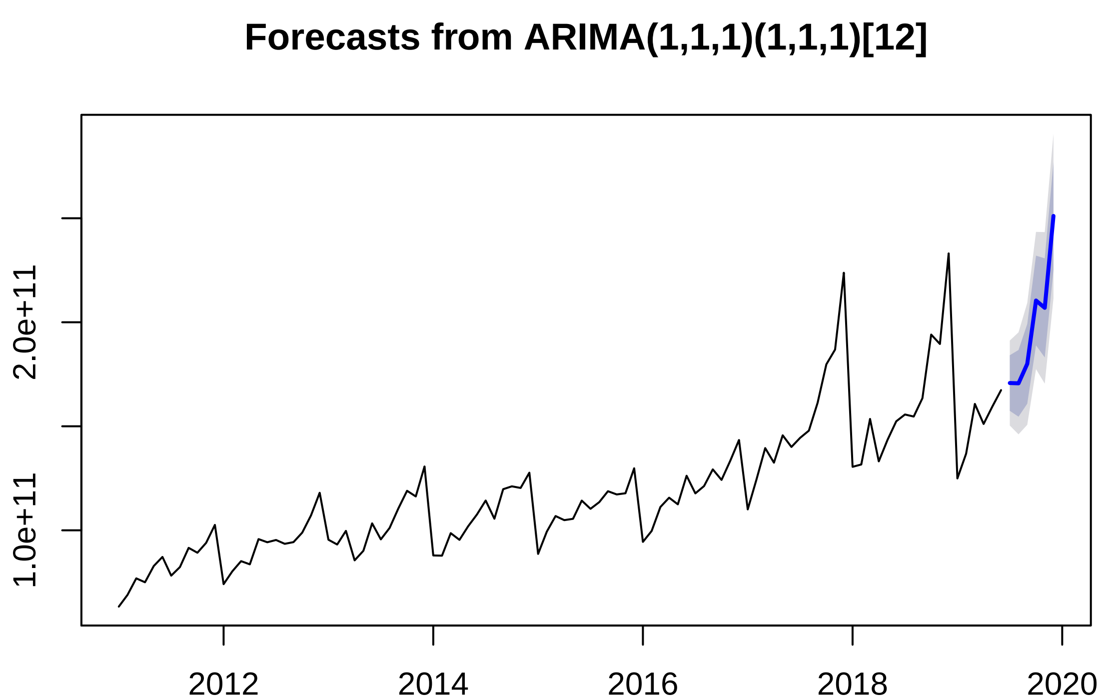
```
## Series: ts
## ARIMA(1,1,1)(1,1,1)[12]
##
## Coefficients:
##           ar1     ma1     sar1    sma1
##       -0.3595  0.0227  -0.5334  0.4228
## s.e.   0.3126  0.3363   1.1785  1.2365
```

```
##
## sigma^2 estimated as 1.087e+20:  log likelihood=-2177.43
## AIC=4364.86   AICc=4365.58   BIC=4377.3
```

```r
ts_arima_forecast = forecast(ts_arima,h = 6)
ts_arima_forecast
```

```
##          Point Forecast         Lo 80        Hi 80        Lo 95
## Jul 2019   170788319965 157428067448 184148572483 150355576982
## Aug 2019   170645005267 154613295424 186676715109 146126620875
## Sep 2019   180021974811 160869582624 199174366999 150730918630
## Oct 2019   210457121225 188898689473 232015552977 177486345071
## Nov 2019   206924998308 183115365432 230734631185 170511307141
## Dec 2019   251110539040 225274163882 276946914198 211597213035
##                 Hi 95
## Jul 2019 191221062948
## Aug 2019 195163389659
## Sep 2019 209313030992
## Oct 2019 243427897379
## Nov 2019 243338689476
## Dec 2019 290623865044
```

```r
forecast:::plot.forecast(ts_arima_forecast)
```



Growth

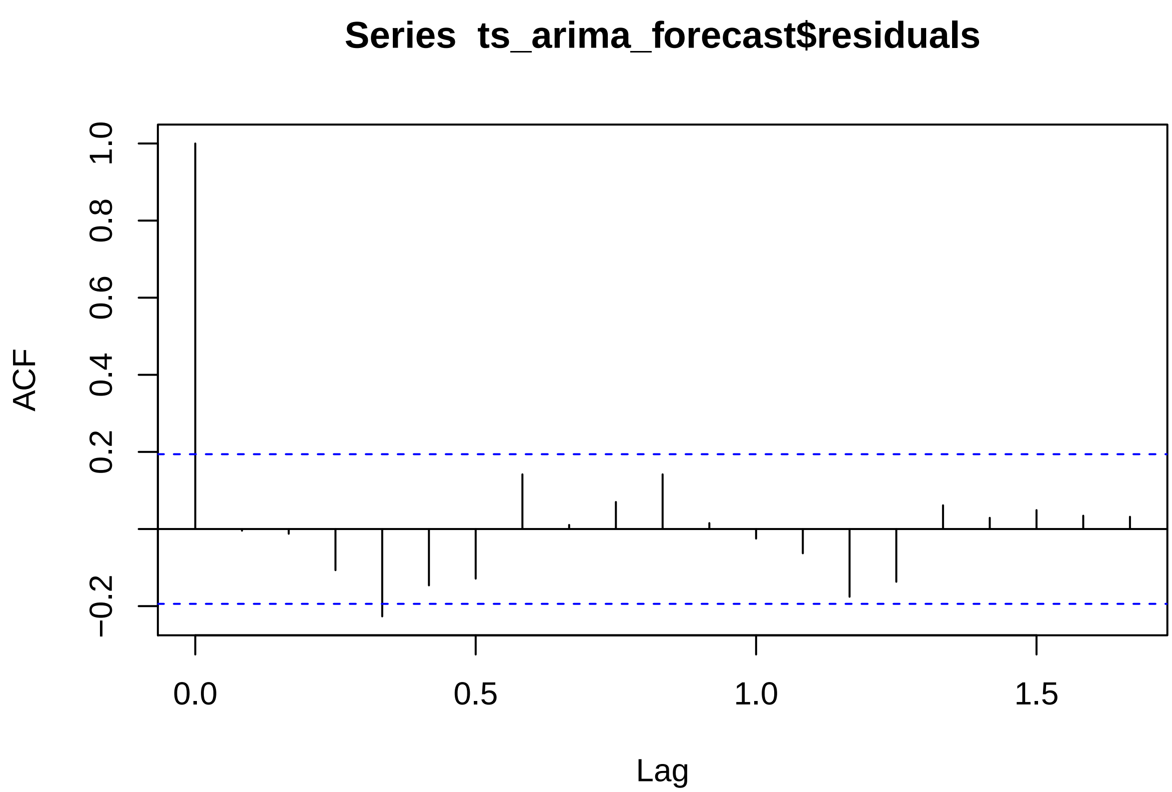
```r
this_year_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1]

growth_ARIMA <- growth(sum(c(this_year,as.numeric(this_year_predict_ARIMA$`Point Forecast`))), sum(last_
growth_ARIMA
```

```
## [1] 0.08018893
```

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:

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

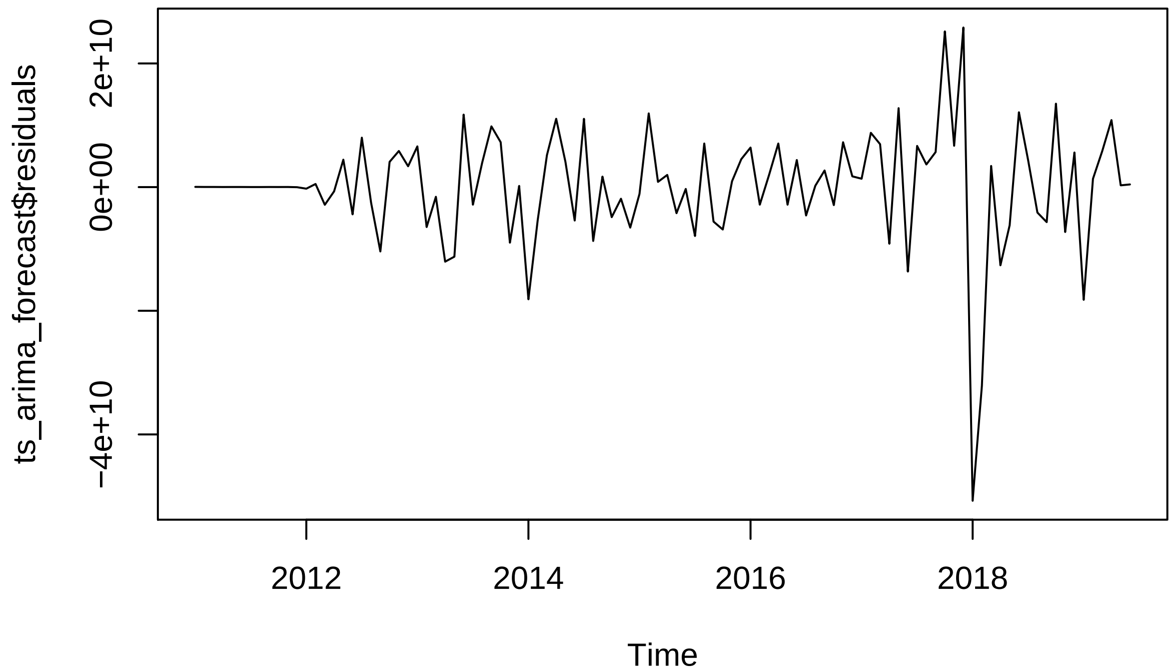



```r
Box.test(ts_arima_forecast$residuals, lag=20, type="Ljung-Box")
```

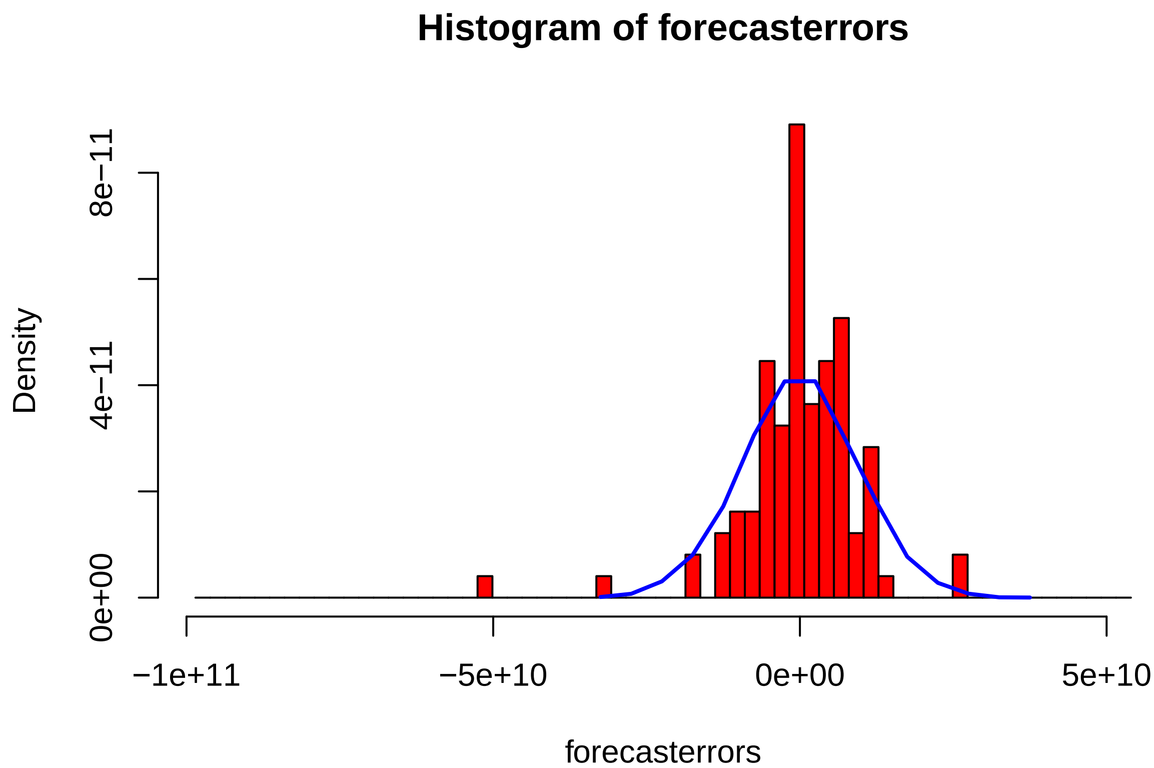
```
##
##  Box-Ljung test
##
## data:  ts_arima_forecast$residuals
## X-squared = 23.833, df = 20, p-value = 0.2498
```

we can reject the null hypothesis, it's rather similar to the HW

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




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



Since successive forecast errors do not seem to be correlated, and the forecast errors seem to be normally distributed with mean zero and constant variance, the ARIMA(0,1,1) does seem to provide an adequate predictive model

testing best arima

```
# library(forecast)
# modelAIC <- data.frame()
# for(d in 0:1){
#   for(p in 0:9){
#     for(q in 0:9){
#
#       fit=Arima(mid.ts,order=c(p,d,q))
#       modelAIC <- rbind(modelAIC, c(d,p,q,AIC(fit))) #
#     }
#   }
# }
# names(modelAIC) <- c("d", "p", "q",  "AIC")
# rowNum <- which(modelAIC$AIC==max(modelAIC$AIC))
# modelAIC[rowNum,]#Required model parameters
```