# Time Series Forcasting report

Kevork Sulahian

September 26, 2019

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

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

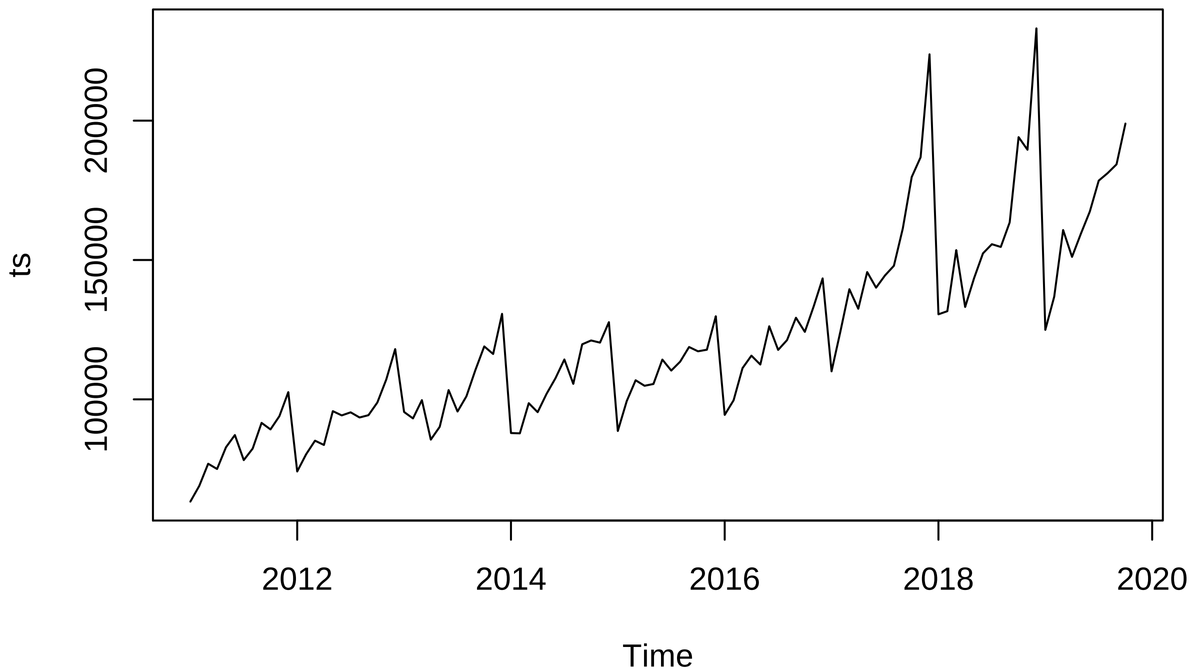
```r
df <- read_xls('economy.xls', sheet='2011-2019 NACE 2')
```

```
## New names:
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * `` -> ...6
## * ...
```

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

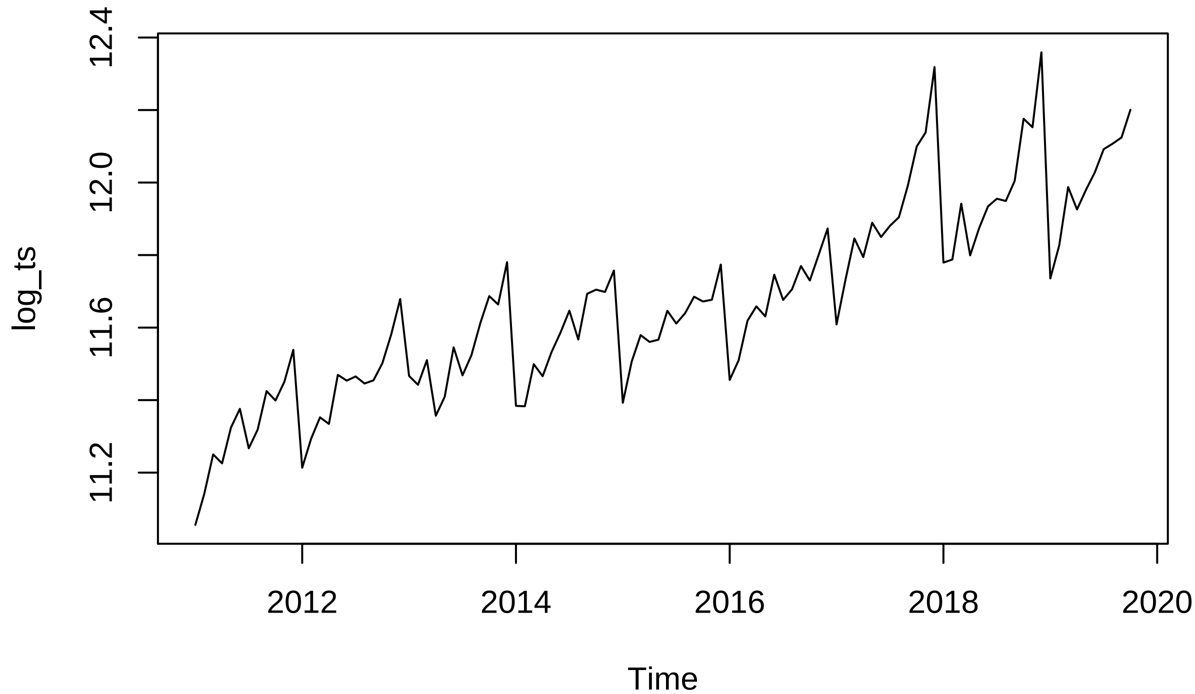
```
## Warning: Setting row names on a tibble is deprecated.
```

```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.

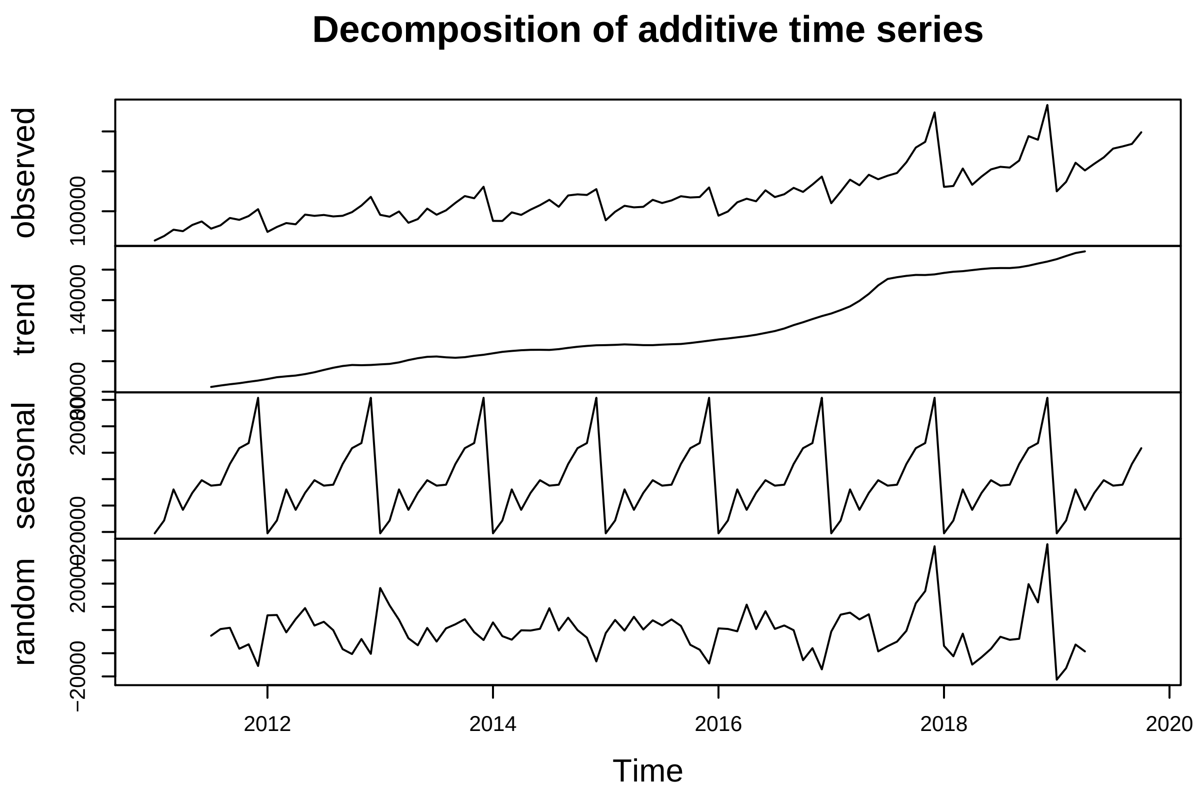
```
ts_components <- decompose(ts)
```

we can print out the estimated values of the seasonal component

```
ts_components$seasonal
```

```
##             Jan         Feb         Mar         Apr         May         Jun
## 2011 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2012 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2013 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2014 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2015 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2016 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2017 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2018 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
## 2019 -20517.7010 -15661.9010  -3889.3766 -11627.5854  -5217.1400   -407.0709
##             Jul         Aug         Sep         Oct         Nov         Dec
```

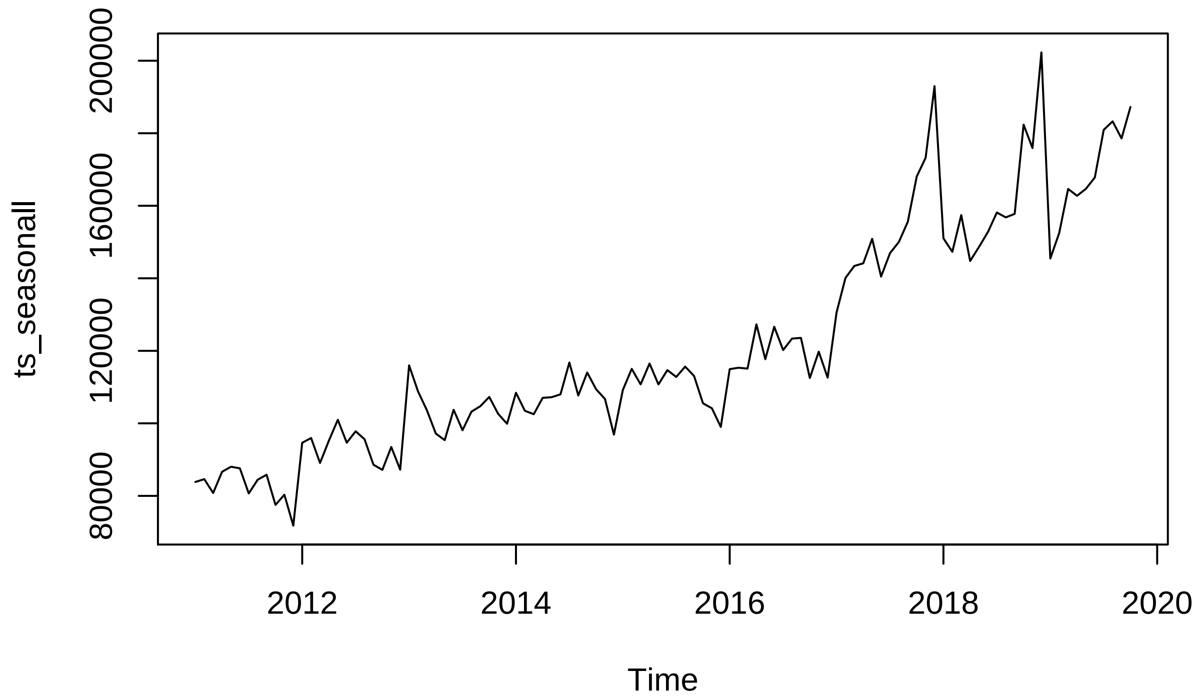
```
## 2011  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2012  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2013  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2014  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2015  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2016  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2017  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2018  -2465.4208  -2114.1271   5731.3250  11714.6380  13652.3146  30802.0453
## 2019  -2465.4208  -2114.1271   5731.3250  11714.6380
```




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

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

```
## Warning in HoltWinters(ts): optimization difficulties: ERROR:
## ABNORMAL_TERMINATION_IN_LNSRCH
```

```r
ts_forcaste
```

```
## Holt-Winters exponential smoothing with trend and additive seasonal component.
##
## Call:
## HoltWinters(x = ts)
##
## Smoothing parameters:
##  alpha: 0.3836132
##  beta : 0
##  gamma: 1
##
## Coefficients:
##             [,1]
## a   172867.8976
## b      859.0055
## s1   23294.3107
## s2   56302.6708
## s3  -47702.4669
```

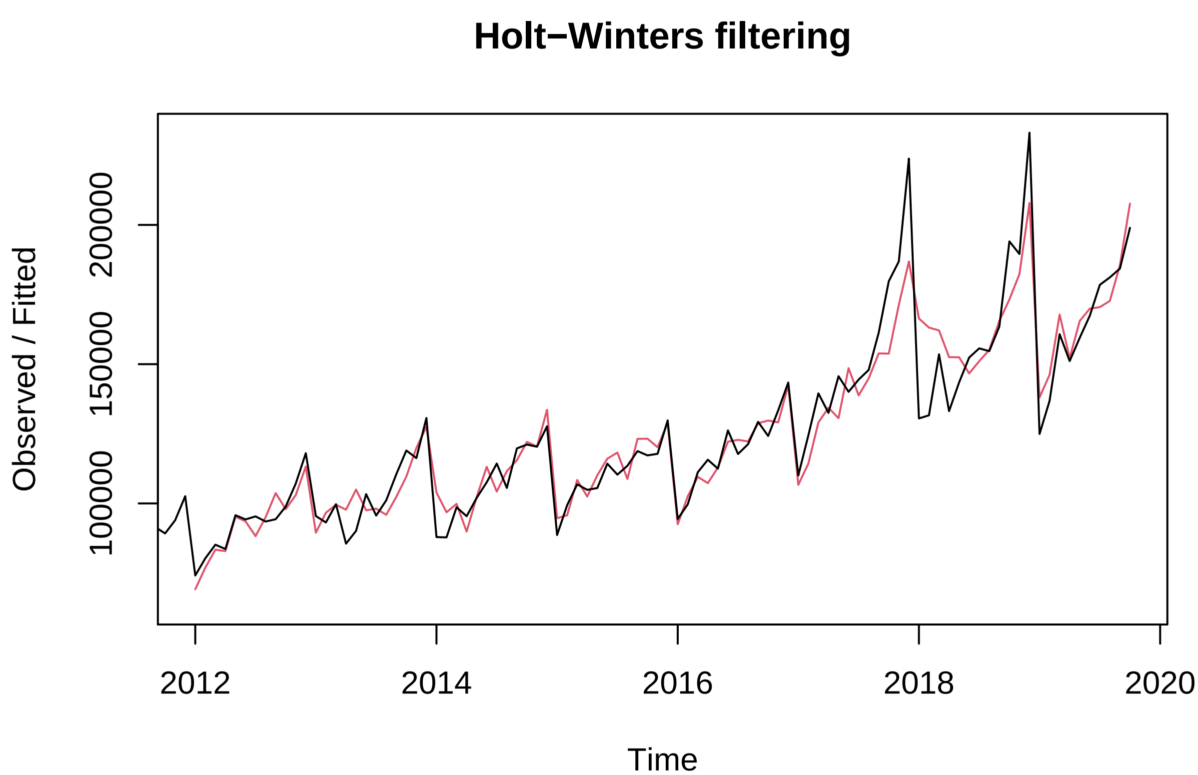
```
## s4  -33062.7461
## s5   -7303.0840
## s6  -17399.9736
## s7   -7593.9919
## s8     447.7895
## s9    7608.8518
## s10   6202.5650
## s11   8966.0791
## s12  26086.7024
```

```
#
```

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

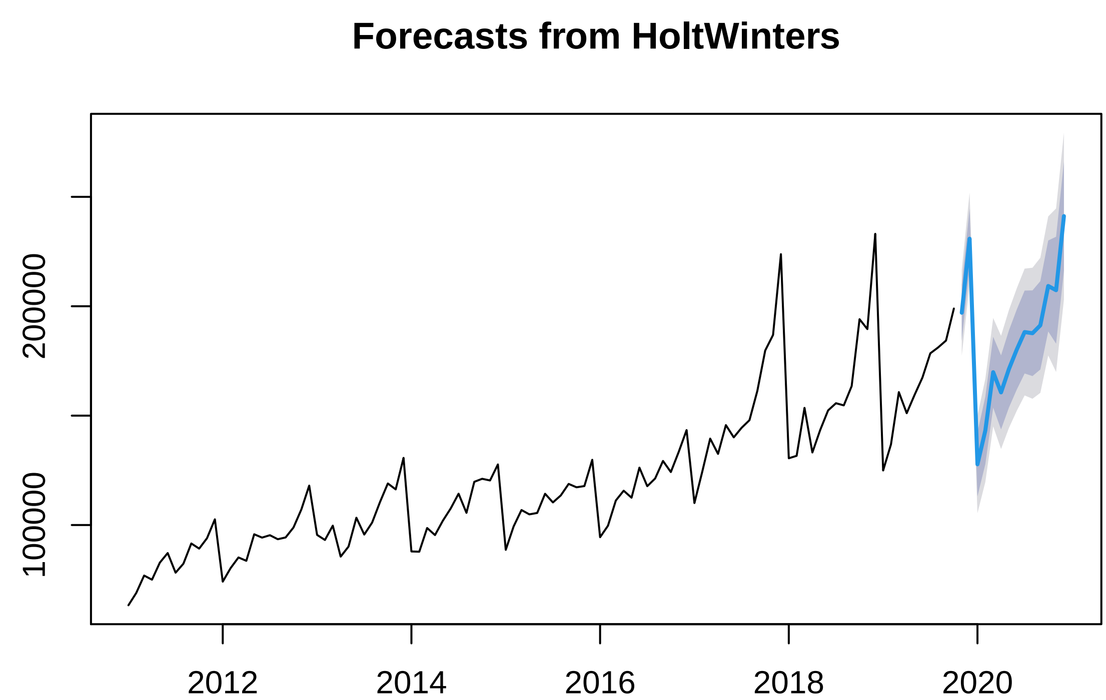
```
ts_forcaste$SSE
```

```
## [1] 9343490075
```



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

```
##          Point Forecast
## Nov 2019       197021.2
## Dec 2019       230888.6
## Jan 2020       127742.4
## Feb 2020       143241.2
## Mar 2020       169859.8
## Apr 2020       160622.0
## May 2020       171286.9
## Jun 2020       180187.7
## Jul 2020       188207.8
## Aug 2020       187660.5
## Sep 2020       191283.0
## Oct 2020       209262.7
## Nov 2020       207329.3
## Dec 2020       241196.6
```



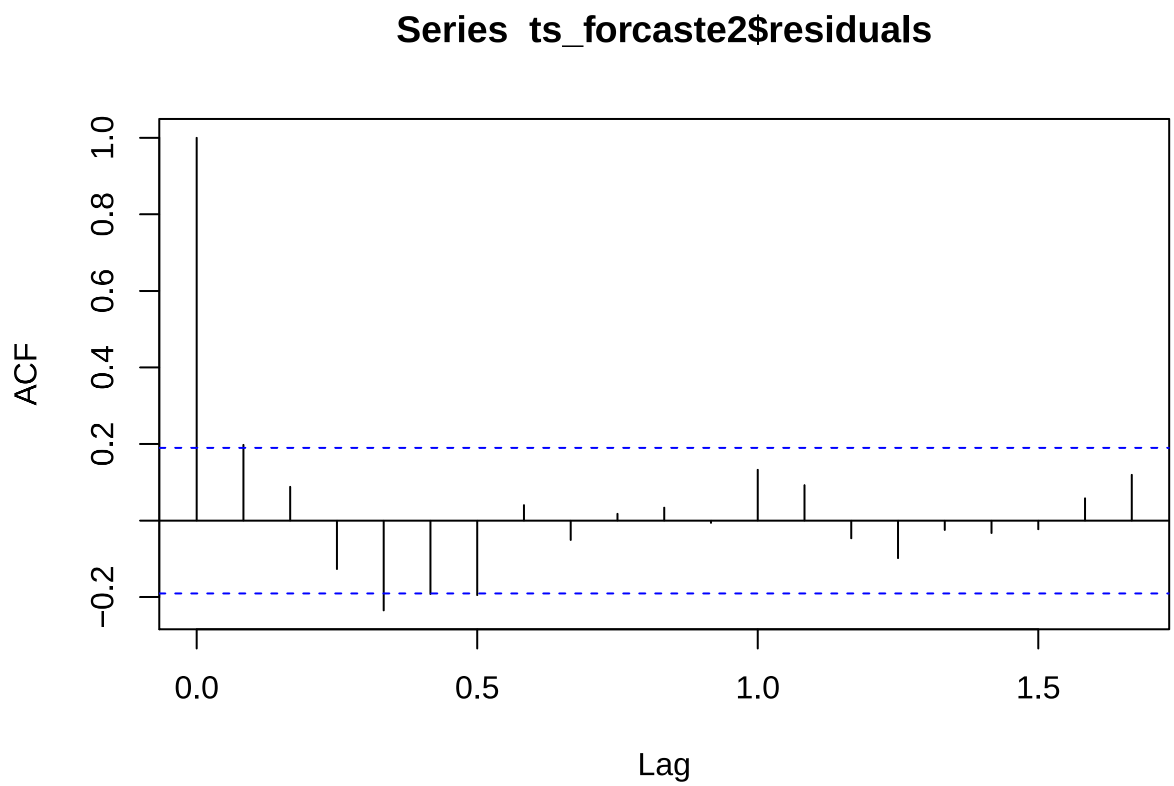

## Growth

```r
year_2019 <- window(ts, 2019)
year_2019_predict_HW <- (as.data.frame(ts_forcaste2))[1][c(1:2),]
```

```
sum_year_2019 = sum(c(year_2019,year_2019_predict_HW))
year_2020 = (as.data.frame(ts_forcaste2))[1][c(3:14),]
growth_HW <- growth(sum(year_2020),sum_year_2019)
growth_HW
```

```
## [1] 0.05145076
```

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



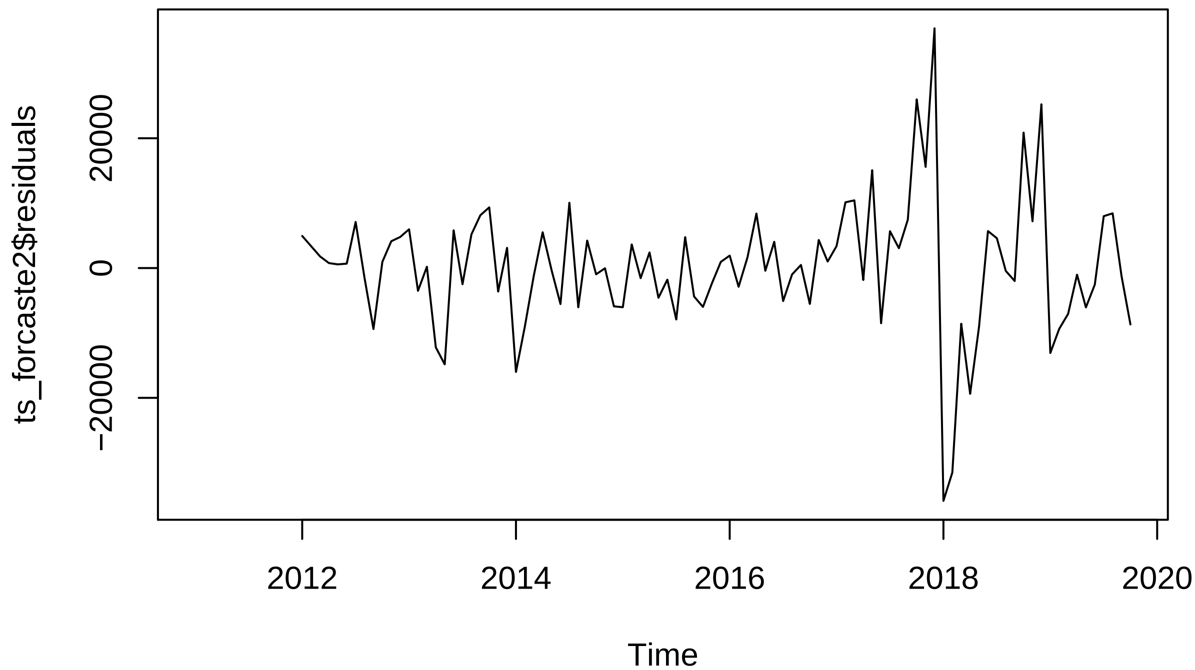

Ljung-Box test:

```
##
##  Box-Ljung test
##
## data:  ts_forcaste2$residuals
## X-squared = 26.501, df = 20, p-value = 0.1499
```

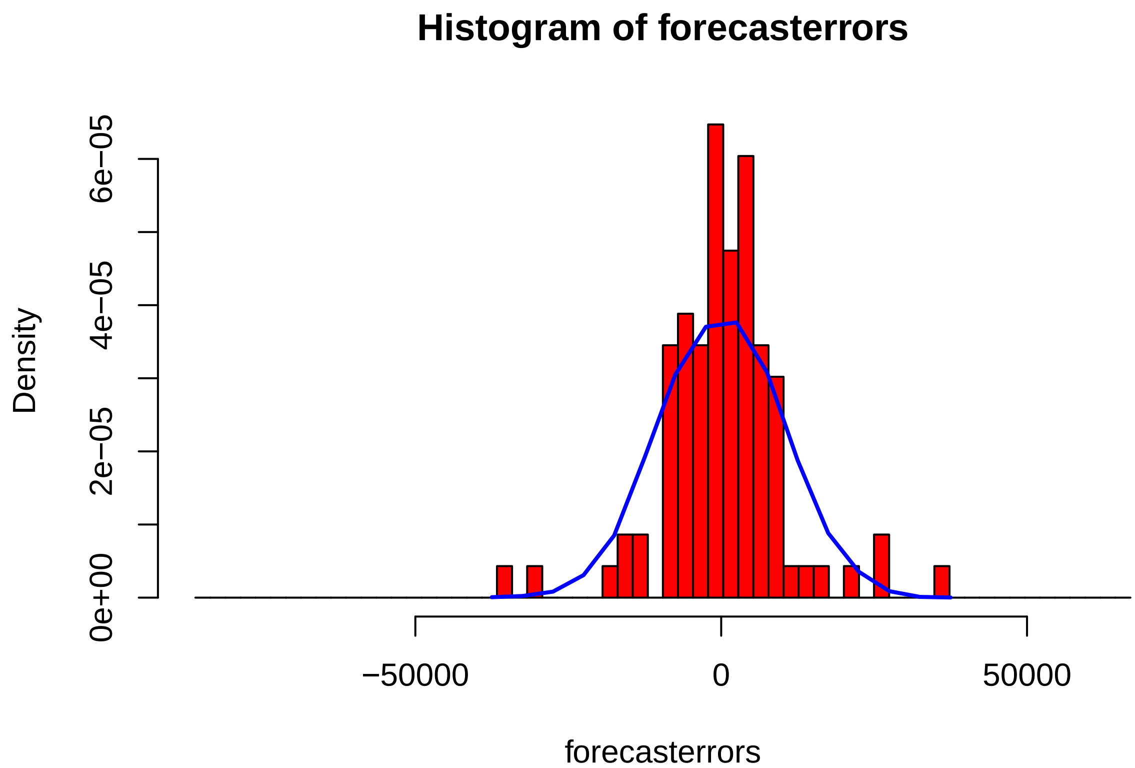
The correlogram shows that the autocorrelations for the in-sample forecast errors do not exceed the significance bounds for lags 1-20. Furthermore, the p-value for Ljung-Box test is 0.2, indicating that there is little evidence of non-zero autocorrelations at lags 1-20.

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):

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



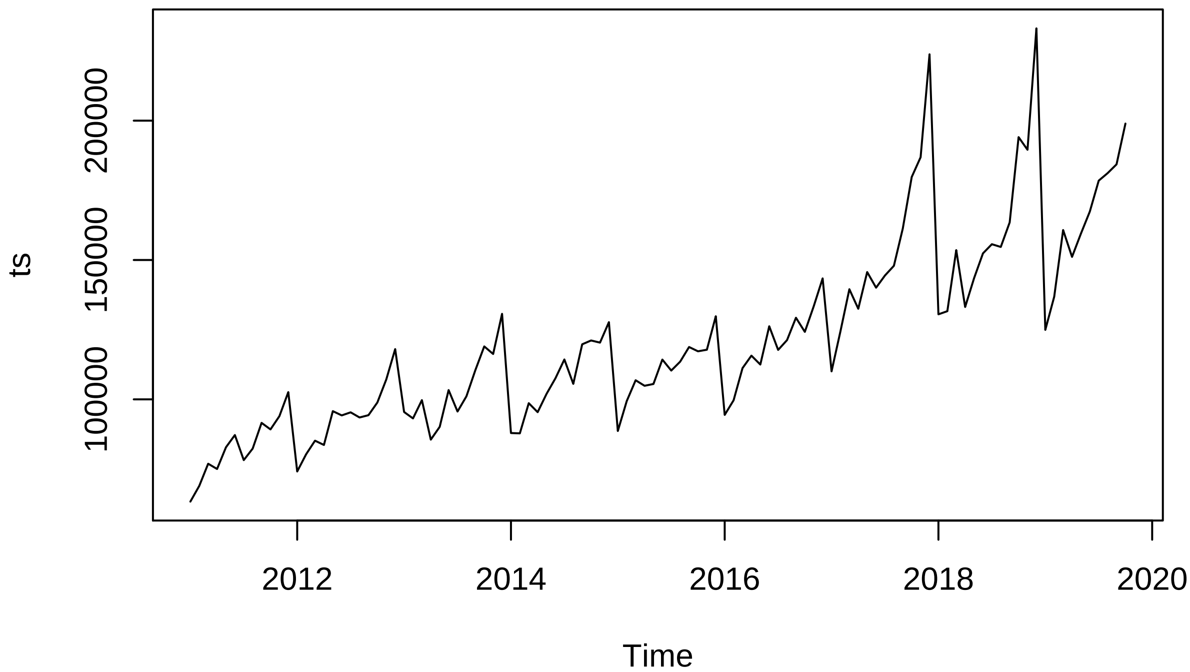

```
plotForecastErrors(ts_forcaste2$residuals)
```



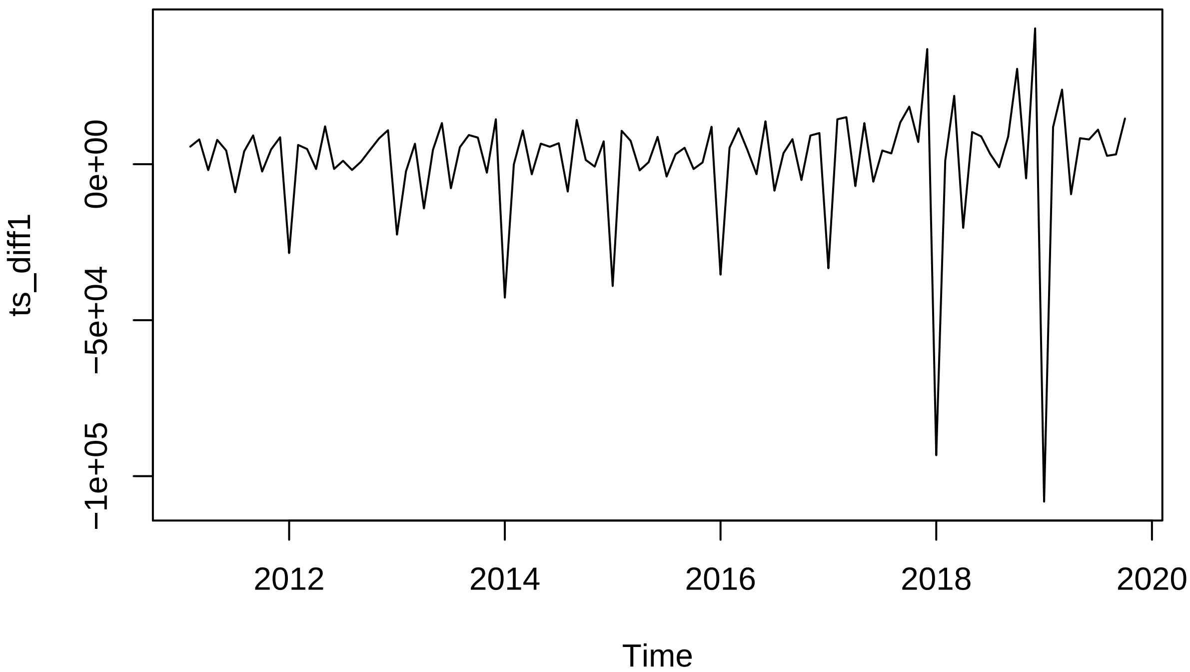

From the time plot, it appears plausible that the forecast errors have constant variance over time. From the histogram of forecast errors, it seems plausible that the forecast errors are normally distributed with mean zero.

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)
```

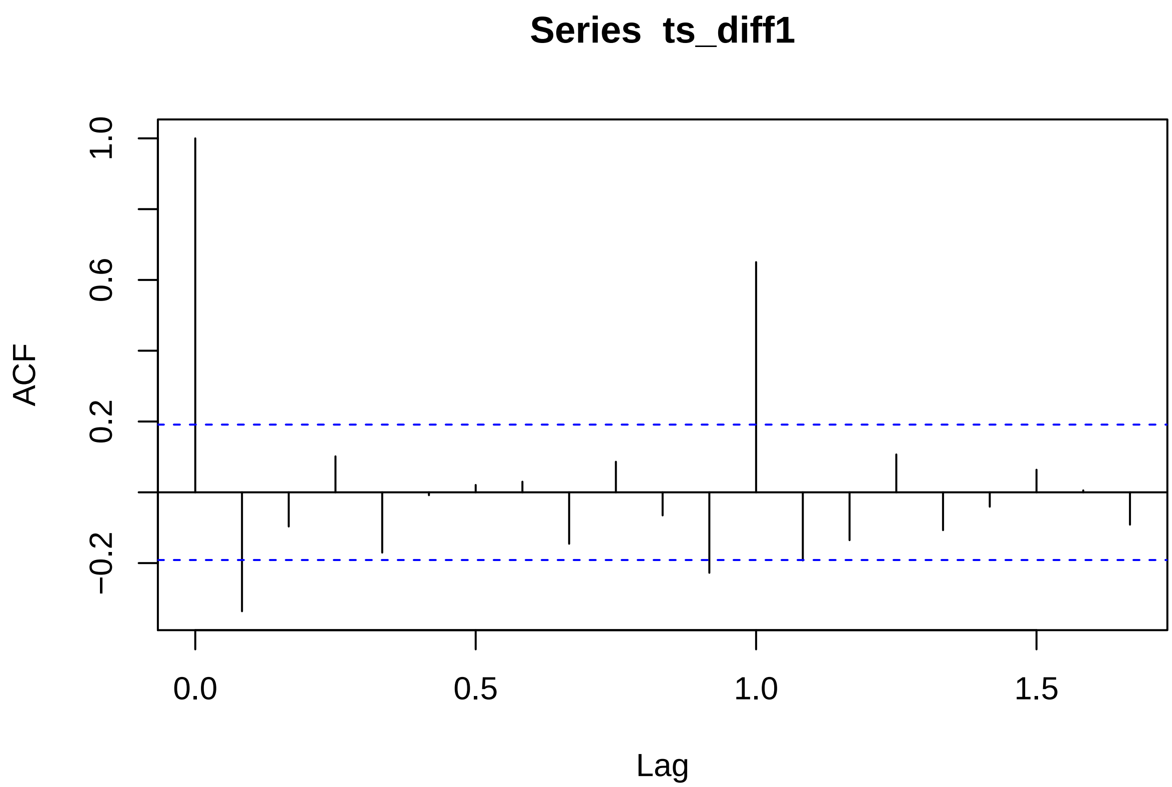



```
ts_diff1 <-  diff(ts, differences = 1)

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
```

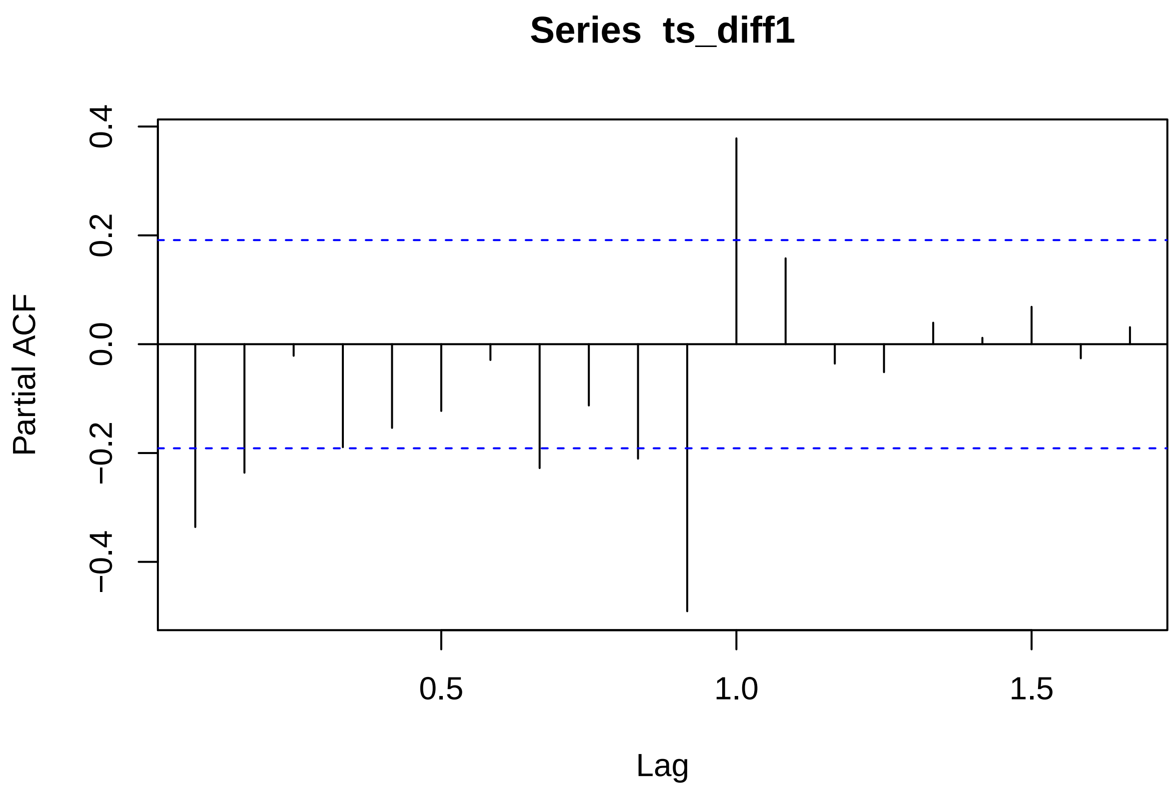



We see from the correlogram that the autocorrelation exceeds the significance bound 3 times but all the others do not exceed

```
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 0.8333
##  1.000 -0.336 -0.097  0.102 -0.170 -0.008  0.021  0.030 -0.145  0.086 -0.065
## 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667
## -0.227  0.650 -0.193 -0.135  0.107 -0.107 -0.041  0.064  0.005 -0.091
```

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




```
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 0.9167
## -0.336 -0.236 -0.021 -0.189 -0.154 -0.123 -0.029 -0.228 -0.113 -0.210 -0.491
## 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667
##  0.378  0.158 -0.036 -0.051  0.040  0.012  0.069 -0.026  0.031
```

## Arima, 1,1,1

```
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:
```

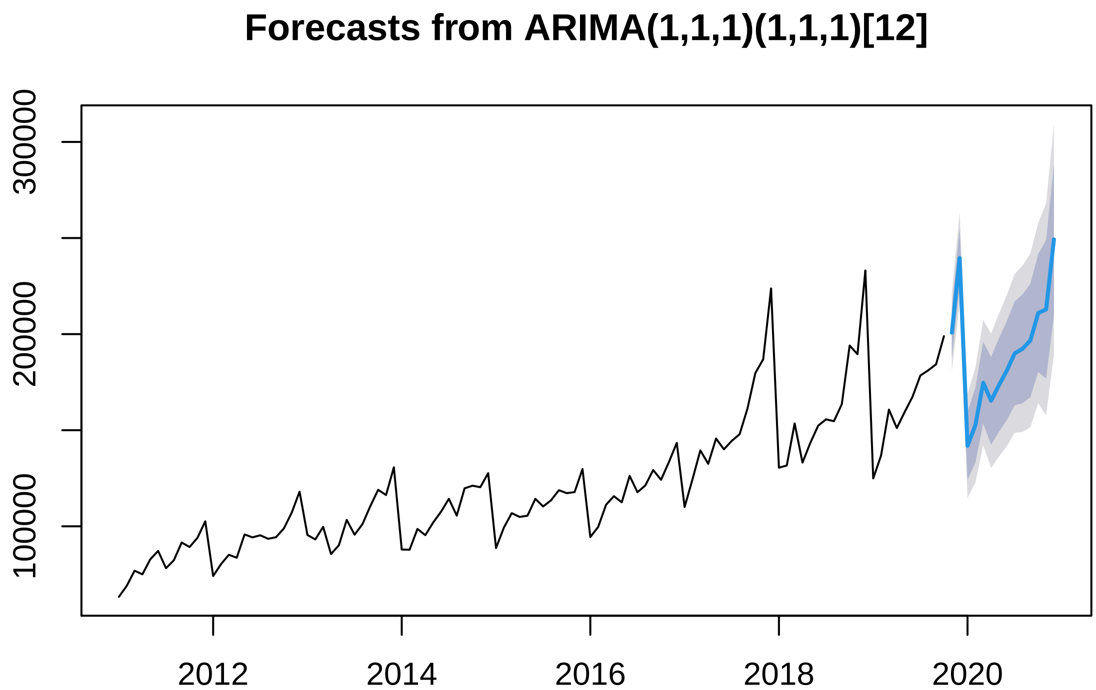
```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##           ar1      ma1    sar1     sma1
##       -0.0038  -0.3631  0.5113  -0.6981
## s.e.      NaN   0.0047  0.0060   0.0059
##
## sigma^2 estimated as 105590908:  log likelihood=-989.51
## AIC=1989.02   AICc=1989.71   BIC=2001.68
```

```r
ts_arima_forecast = forecast(ts_arima,h = 14)
ts_arima_forecast
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Nov 2019       200747.7 187577.0 213918.5 180604.8 220890.6
## Dec 2019       239599.9 224011.9 255187.9 215760.2 263439.6
## Jan 2020       141878.3 124191.9 159564.6 114829.3 168927.2
## Feb 2020       152638.6 133077.9 172199.4 122723.1 182554.2
## Mar 2020       174734.4 153463.8 196005.0 142203.9 207264.9
## Apr 2020       165337.3 142484.5 188190.1 130386.9 200287.6
## May 2020       173469.8 149137.4 197802.2 136256.6 210683.0
## Jun 2020       181002.0 155274.9 206729.0 141655.8 220348.1
## Jul 2020       189982.2 162932.4 217032.1 148613.0 231351.4
## Aug 2020       192378.5 164067.6 220689.5 149080.6 235676.4
## Sep 2020       196699.0 167180.8 226217.2 151554.8 241843.2
## Oct 2020       211008.4 180330.4 241686.4 164090.5 257926.3
## Nov 2020       212839.1 176717.8 248960.4 157596.4 268081.8
## Dec 2020       249303.2 210138.3 288468.0 189405.7 309200.6
```

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



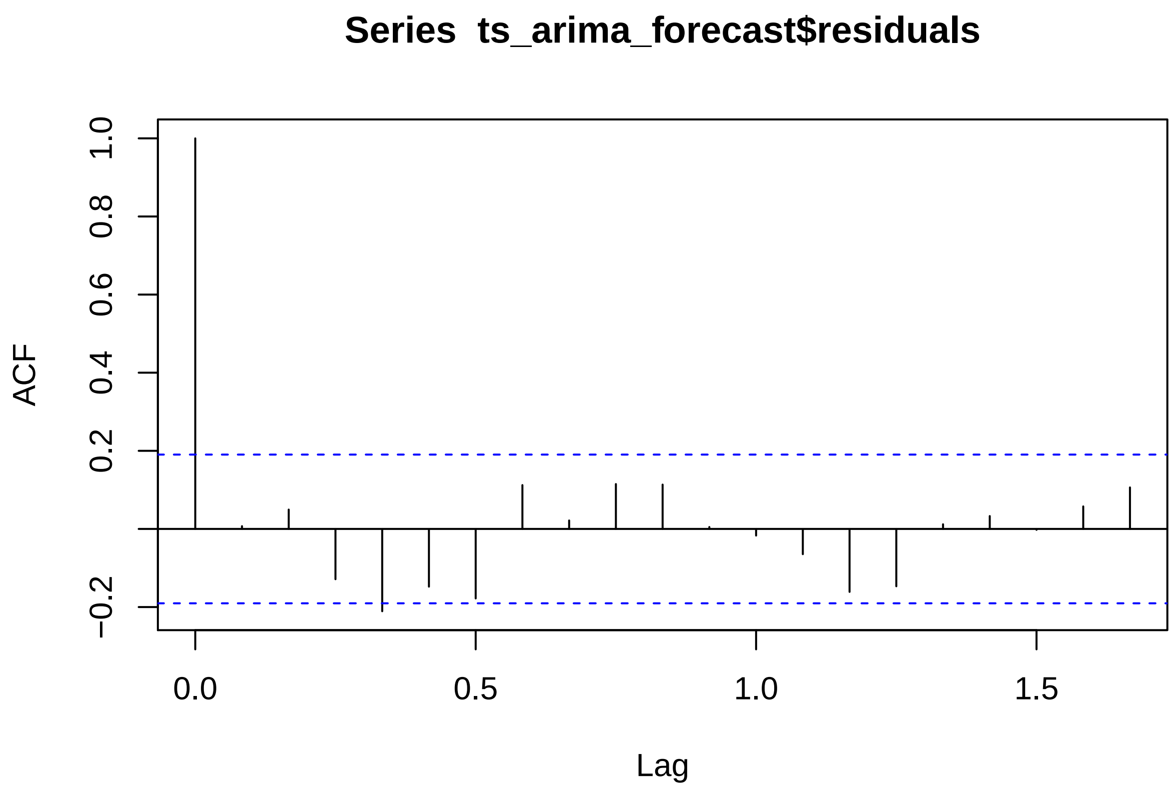

## Growth

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


year_2019_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1][c(1:2),]
sum_year_2019 = sum(c(year_2019,year_2019_predict_ARIMA))
year_2020 = (as.data.frame(ts_arima_forecast))[1][c(3:14),]
growth_ARIMA <- growth(sum_year_2019, sum(year_2020))
growth_ARIMA=-growth_ARIMA
```

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)
```

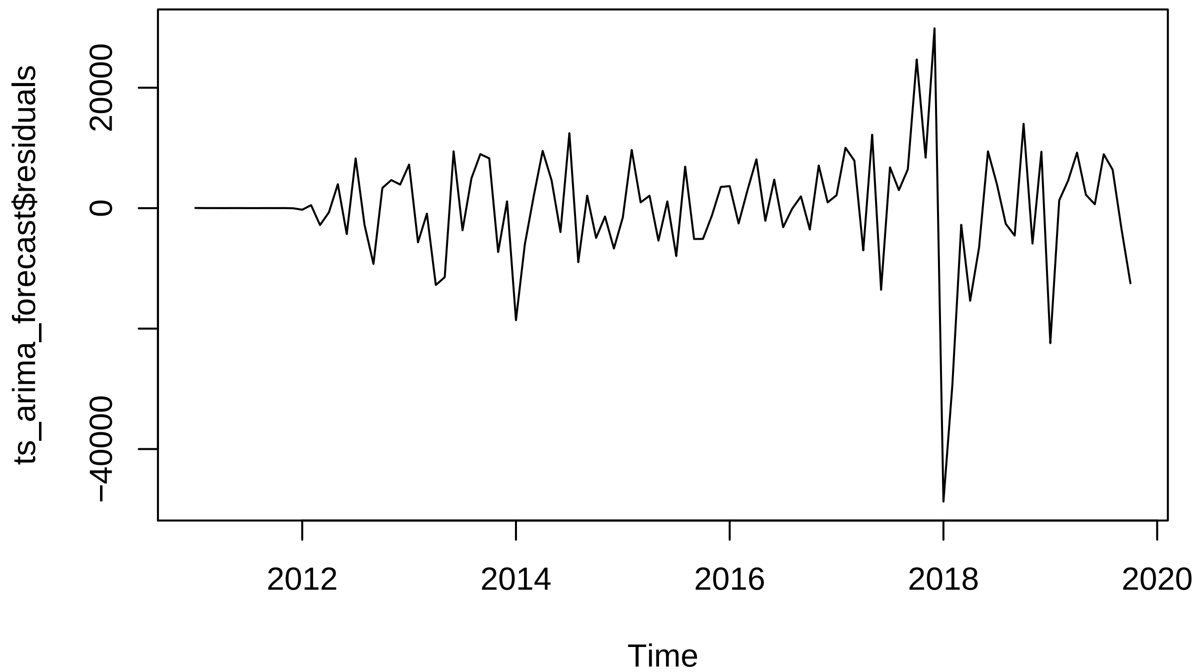



```r
Box.test(ts_arima_forecast$residuals, lag=20, type="Ljung-Box")
```

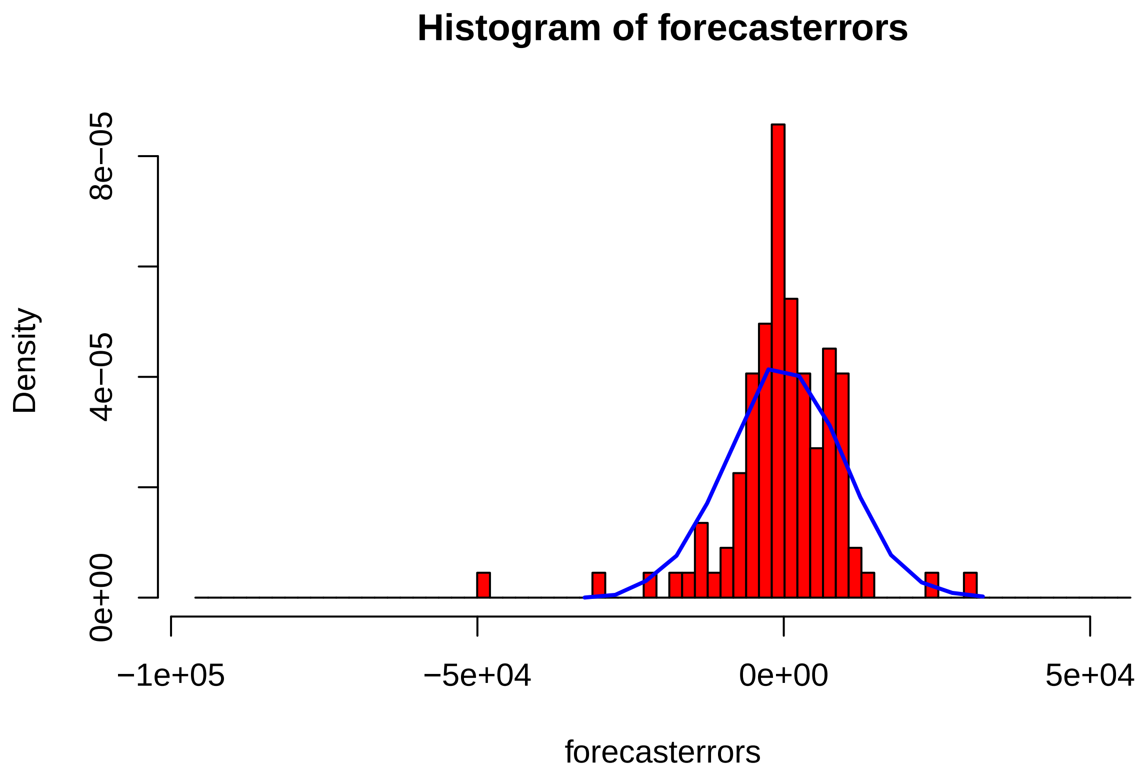
```
##
##  Box-Ljung test
##
## data:  ts_arima_forecast$residuals
## X-squared = 26.409, df = 20, p-value = 0.1527
```

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

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




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




# Arima, 0,1,0 as given from the loop

```
ts_arima = Arima(ts, order=c(2,1,1),seasonal = list(order = c(2,1,0)))
ts_arima
```

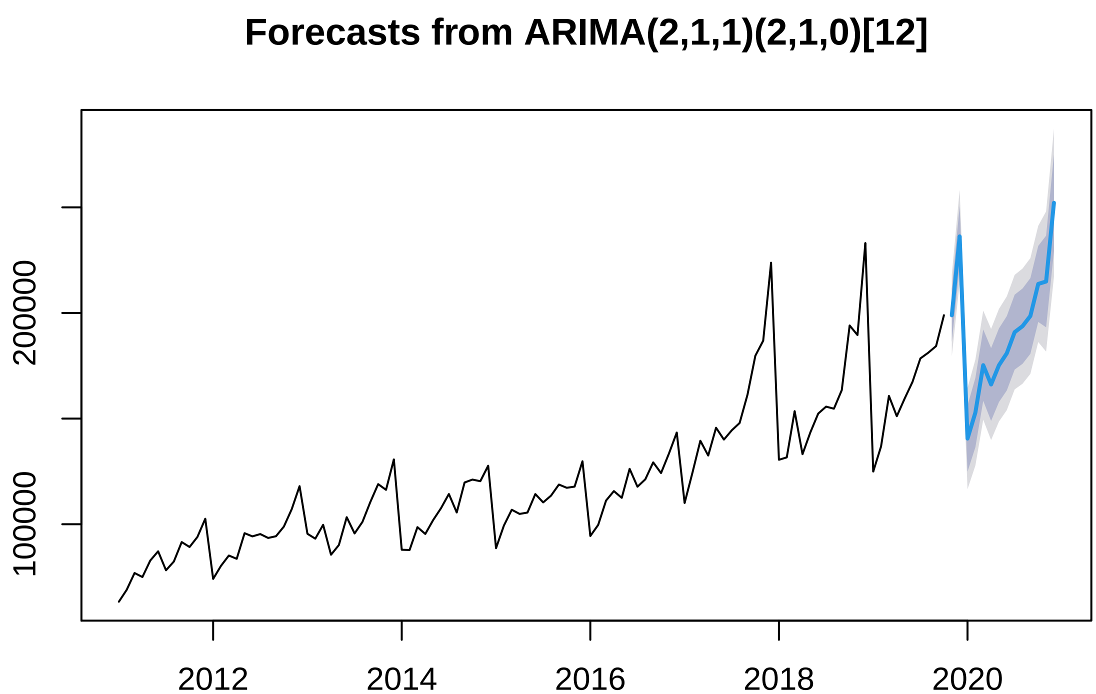
```
## Series: ts
## ARIMA(2,1,1)(2,1,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1     sar1     sar2
##       0.5187  0.1770  -0.9699  -0.1603  -0.1667
## s.e.  0.1138  0.1093   0.0578   0.1085   0.1423
##
## sigma^2 estimated as 96893567:  log likelihood=-985.7
## AIC=1983.39   AICc=1984.37   BIC=1998.59
```

```
ts_arima_forecast = forecast(ts_arima,h = 14)
ts_arima_forecast
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Nov 2019       198884.4 186268.0 211500.8 179589.3 218179.5
## Dec 2019       236259.6 221866.9 250652.4 214247.8 258271.4
## Jan 2020       140545.4 124870.8 156220.0 116573.2 164517.7
## Feb 2020       152866.6 136465.3 169267.9 127782.9 177950.3
## Mar 2020       175294.4 158415.9 192172.8 149481.0 201107.7
## Apr 2020       166146.1 148947.6 183344.6 139843.2 192449.0
```

```
## May 2020       175261.5 157836.7 192686.4 148612.5 201910.6
## Jun 2020       180913.6 163321.8 198505.3 154009.3 207817.8
## Jul 2020       190922.9 173202.7 208643.1 163822.2 218023.6
## Aug 2020       193755.1 175932.0 211578.2 166497.0 221013.2
## Sep 2020       198569.6 180660.8 216478.3 171180.5 225958.6
## Oct 2020       213742.6 195760.3 231725.0 186241.0 241244.2
## Nov 2020       214895.4 193202.7 236588.2 181719.2 248071.6
## Dec 2020       252153.2 229270.1 275036.2 217156.5 287149.8
```

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




## Growth

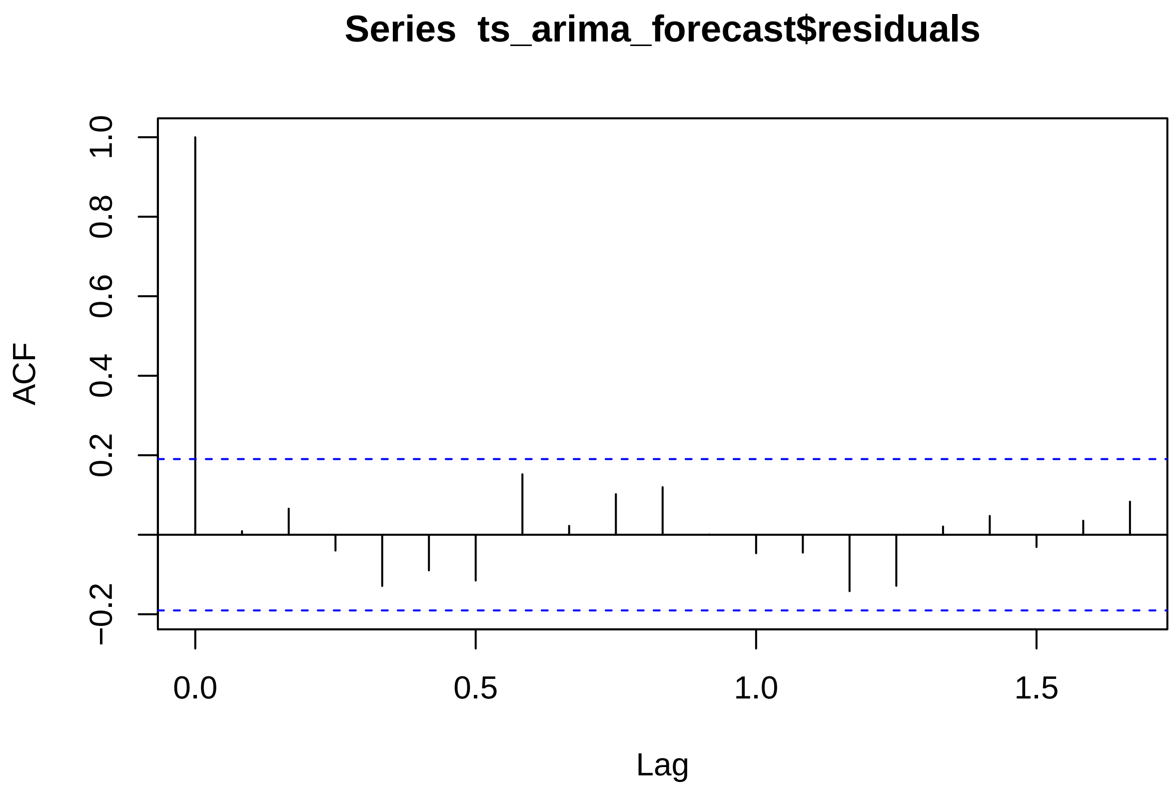
```
this_year_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1]

year_2019_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1][c(1:2),]
sum_year_2019 = sum(c(year_2019,year_2019_predict_ARIMA))
year_2020 = (as.data.frame(ts_arima_forecast))[1][c(3:14),]
growth_ARIMA2 <- growth(sum(year_2020),sum_year_2019)
growth_ARIMA2
```

```
## [1] 0.08492606
```

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)
```

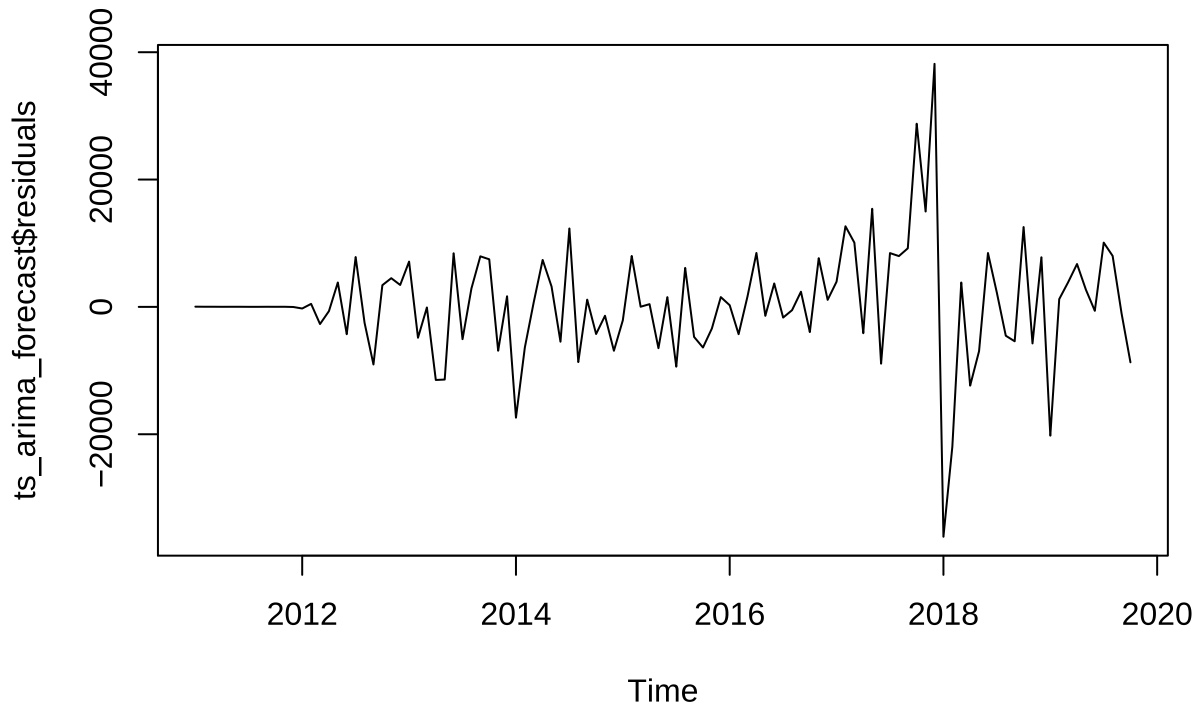



```
Box.test(ts_arima_forecast$residuals, lag=20, type="Ljung-Box")
```

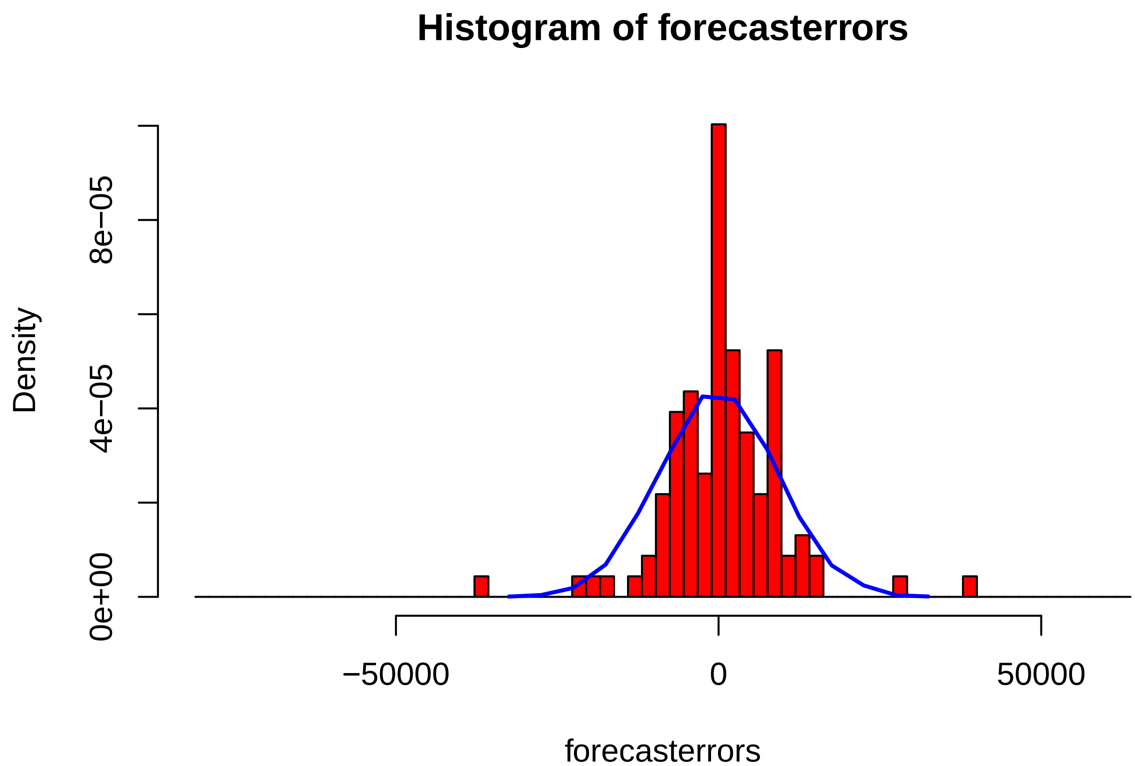
```
##
##  Box-Ljung test
##
## data:  ts_arima_forecast$residuals
## X-squared = 17.298, df = 20, p-value = 0.6335
```

# 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
```




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

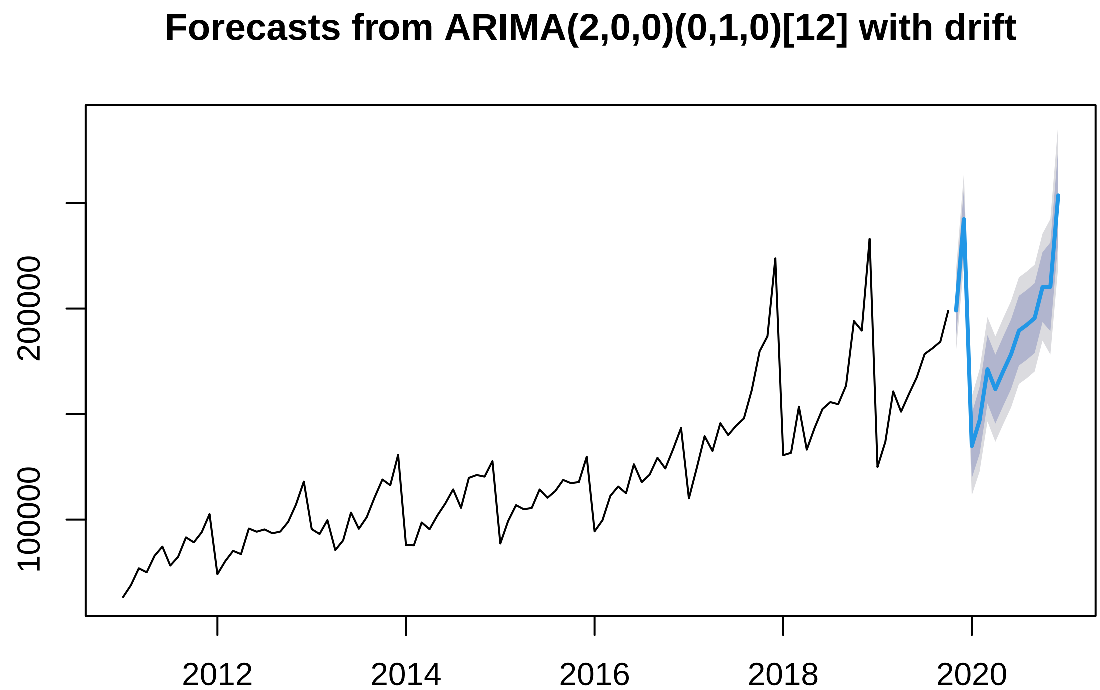



## A model chosen automatically

```
fit <- auto.arima(ts,max.p = 5,max.q = 5,max.P = 5,max.Q = 5,max.d = 3,seasonal = TRUE)
fit
```

```
## Series: ts
## ARIMA(2,0,0)(0,1,0)[12] with drift
##
## Coefficients:
##          ar1     ar2     drift
##       0.5279  0.1712  942.9126
## s.e.  0.1016  0.1017  267.7789
##
## sigma^2 estimated as 96306581:  log likelihood=-996.15
## AIC=2000.3   AICc=2000.75   BIC=2010.47
```

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




```
# str(fit)
```

### Growth

```
year_2019_predict_auto.arima <- (as.data.frame(fit_forecast))[1][c(1:2),]
year_2019_predict_auto.arima_95_low <- (as.data.frame(fit_forecast))[4][c(1:2),]
year_2019_predict_auto.arima_95_high <- (as.data.frame(fit_forecast))[5][c(1:2),]

sum_year_2019 = sum(c(year_2019,year_2019_predict_auto.arima))
sum_year_2019_low = sum(c(year_2019,year_2019_predict_auto.arima_95_low))
sum_year_2019_high = sum(c(year_2019,year_2019_predict_auto.arima_95_high))


year_2020_predict_auto.arima <- (as.data.frame(fit_forecast))[1][c(3:14),]
year_2020_predict_auto.arima_95_low <- (as.data.frame(fit_forecast))[4][c(3:14),]
year_2020_predict_auto.arima_95_high <- (as.data.frame(fit_forecast))[5][c(3:14),]

growth_auto.arima <- growth(sum(year_2020_predict_auto.arima),sum_year_2019)
growth_auto.arima_95_low <- growth(sum(year_2020_predict_auto.arima_95_low),sum_year_2019_low)
growth_auto.arima_95_high <- growth(sum(year_2020_predict_auto.arima_95_high),sum_year_2019_high)

growth_auto.arima
```

```
## [1] 0.06246265
```

```r
growth_auto.arima_95_low
```

```
## [1] -0.07022863
```

```r
growth_auto.arima_95_high
```

```
## [1] 0.1900376
```

## all the growths

```r
# growth_ARIMA =  -growth_ARIMA
#
# growth_ARIMA2 = -growth_ARIMA2
#
# growth_auto.arima
#
# growth_HW
```