tsf_export

Kevork Sulahian

October 28, 2019

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

```r
# library(readxl)

df <- read_xlsx("Export_for_TS.xlsx")
```

```
## New names:
## * `` -> ...2
```

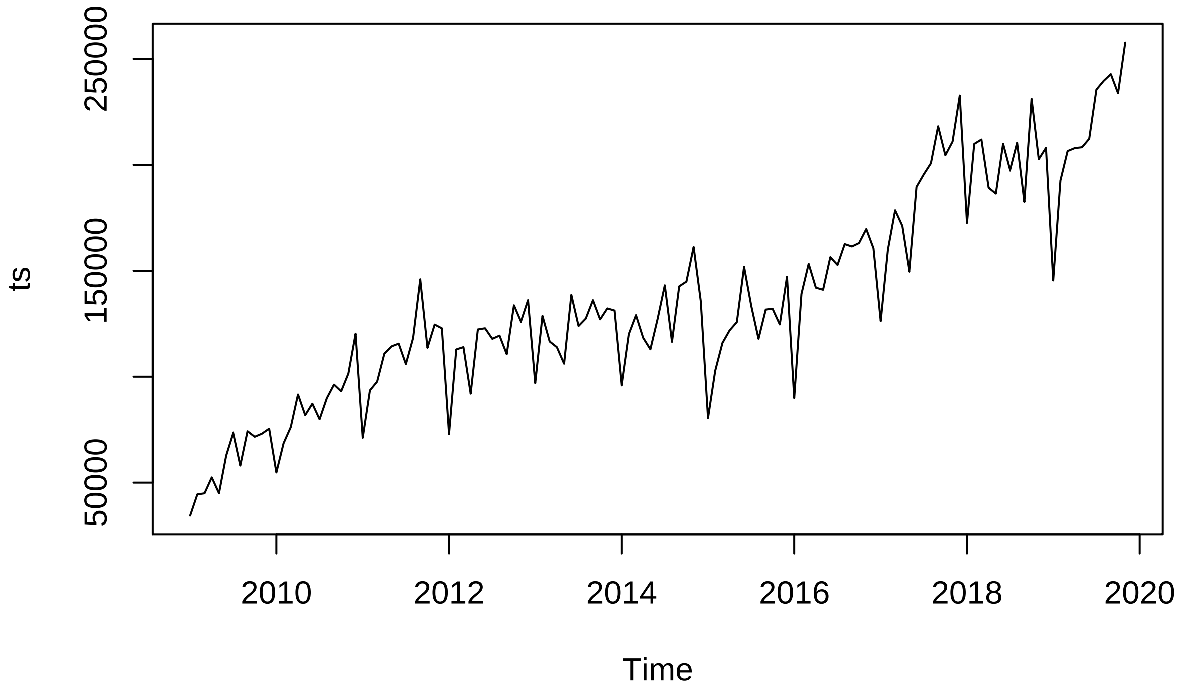
```r
df = df[1,]
df = df[-c(1,2)]

df2 = read_xlsx('export_19xlsx.xlsx')
```

```
## New names:
## * `` -> ...1
```

```r
df = t(df)
df2$...1 = c(paste0(2019,"-",1:11))
rownames(df2) =df2$...1
```

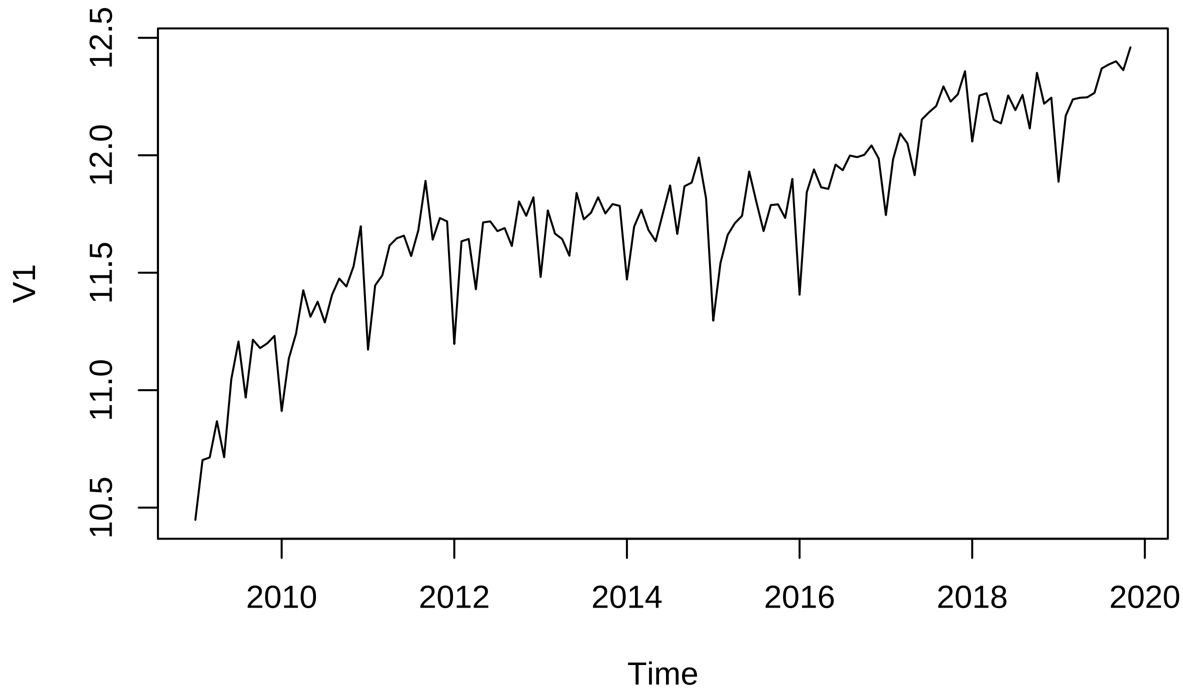
```
## Warning: Setting row names on a tibble is deprecated.
```

```r
df2$...1 = NULL
colnames(df2) = "V1"
df = as.data.frame(df)

df3 = rbind(df,df2)
ts = ts(df3,start=c(2009,1), 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.

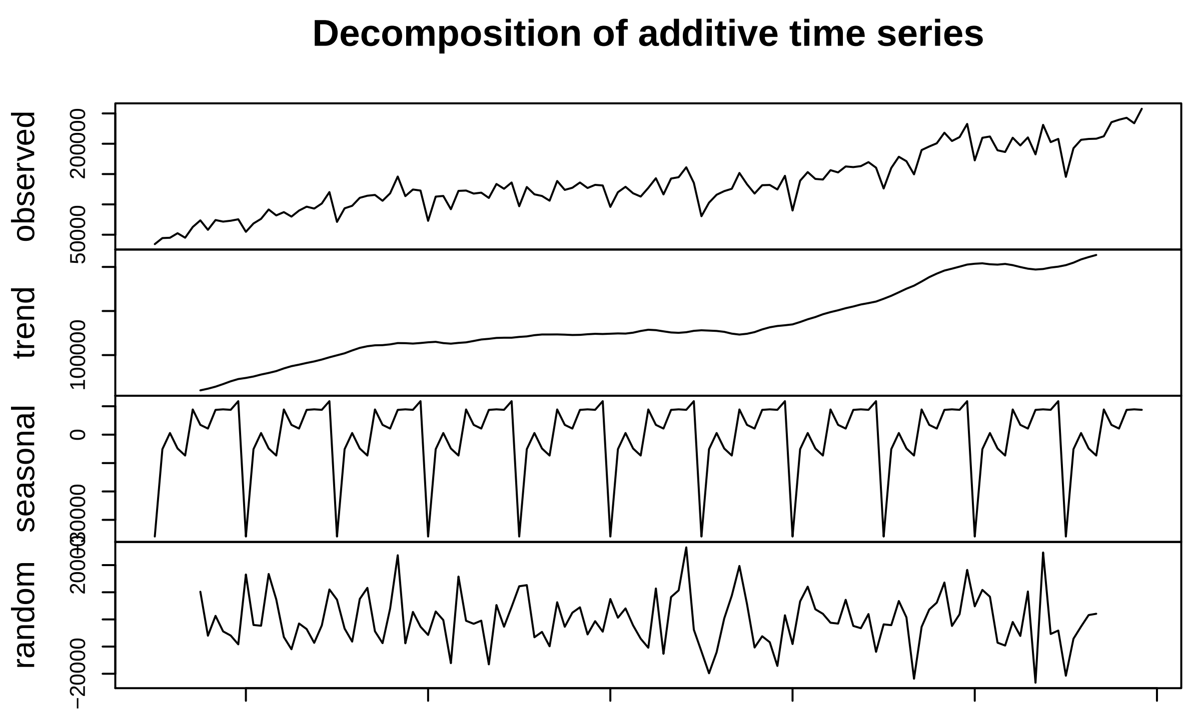
```
ts_components <- decompose(ts)
```

we can print out the estimated values of the seasonal component

```
ts_components$seasonal
```

```
##             Jan        Feb        Mar        Apr        May        Jun
## 2009 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2010 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2011 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2012 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2013 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
```

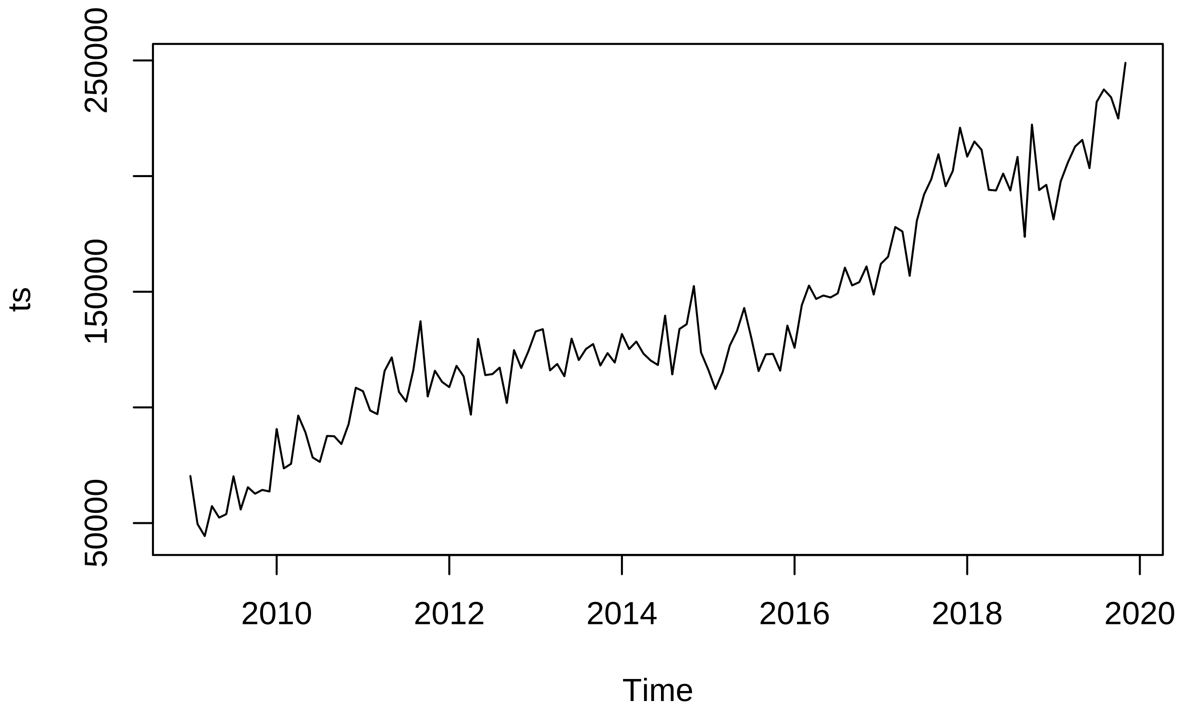
```
## 2014 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2015 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2016 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2017 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2018 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
## 2019 -35872.943  -5113.997    579.187  -4874.741  -7357.388   8874.528
##             Jul        Aug        Sep        Oct        Nov        Dec
## 2009   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2010   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2011   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2012   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2013   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2014   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2015   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2016   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2017   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2018   3440.638   2149.841   8722.117   8915.671   8744.667  11792.421
## 2019   3440.638   2149.841   8722.117   8915.671   8744.667
```



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: 0.3631002
##  beta : 0.00776954
##  gamma: 0.3604275
##
## Coefficients:
##             [,1]
## a   241120.5637
## b     1841.9013
## s1    6407.2267
```

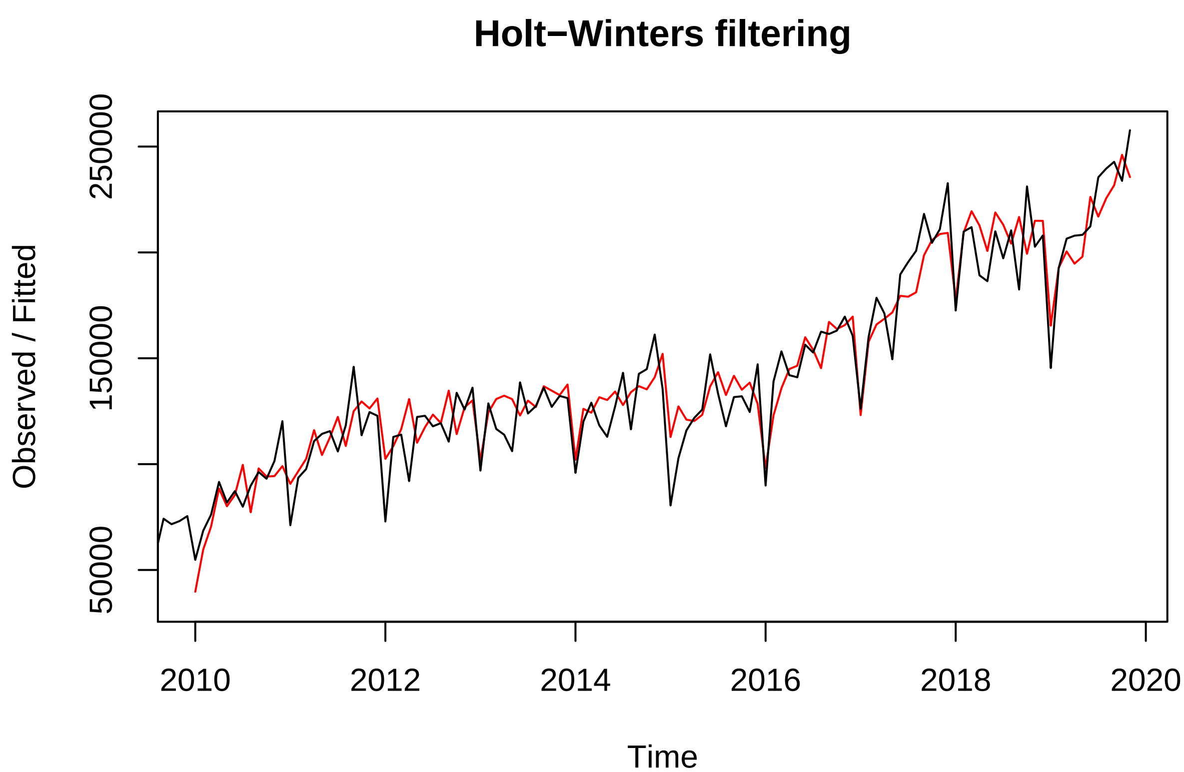
```
## s2  -45300.7926
## s3   -7832.0311
## s4    -255.7858
## s5   -8230.8329
## s6  -12108.1995
## s7    5114.0000
## s8    6623.6290
## s9    5748.0034
## s10   4318.7619
## s11   7502.7160
## s12   7565.4575
```

```
#
```

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

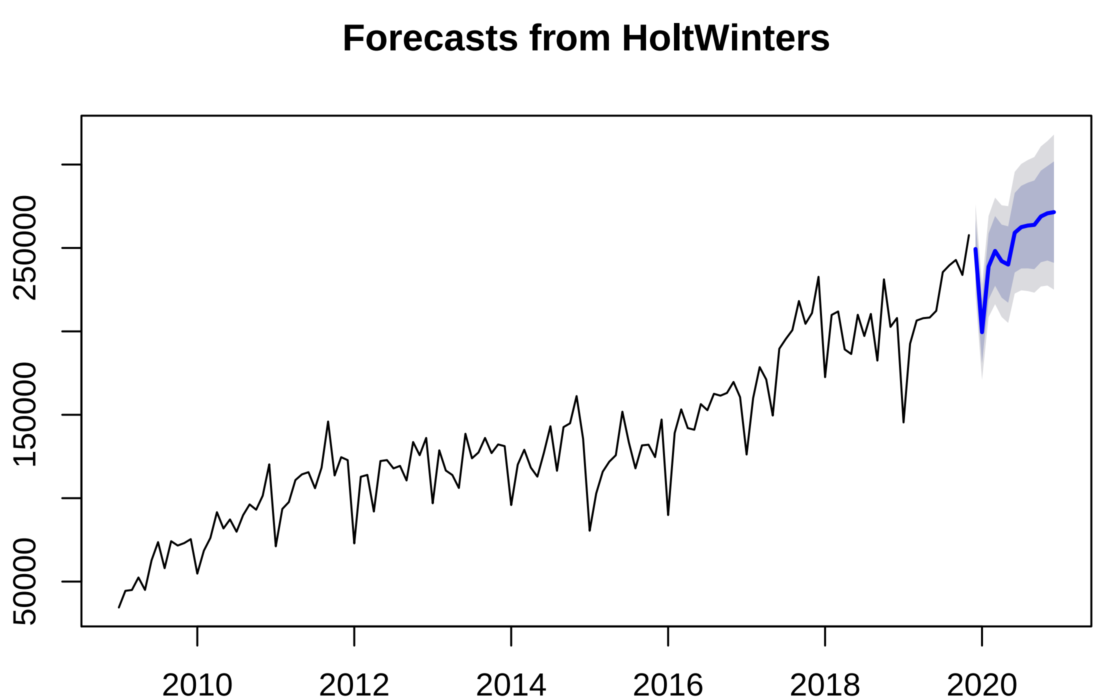
```
ts_forcaste$SSE
```

```
## [1] 22399824125
```



```
ts_forcaste2 = forecast:::forecast.HoltWinters(ts_forcaste, h= 13)
(as.data.frame(ts_forcaste2))[1]
```

```
##          Point Forecast
## Dec 2019       249369.7
## Jan 2020       199503.6
## Feb 2020       238814.2
## Mar 2020       248232.4
## Apr 2020       242099.2
## May 2020       240063.8
## Jun 2020       259127.9
## Jul 2020       262479.4
## Aug 2020       263445.7
## Sep 2020       263858.3
## Oct 2020       268884.2
## Nov 2020       270788.8
## Dec 2020       271472.5
```

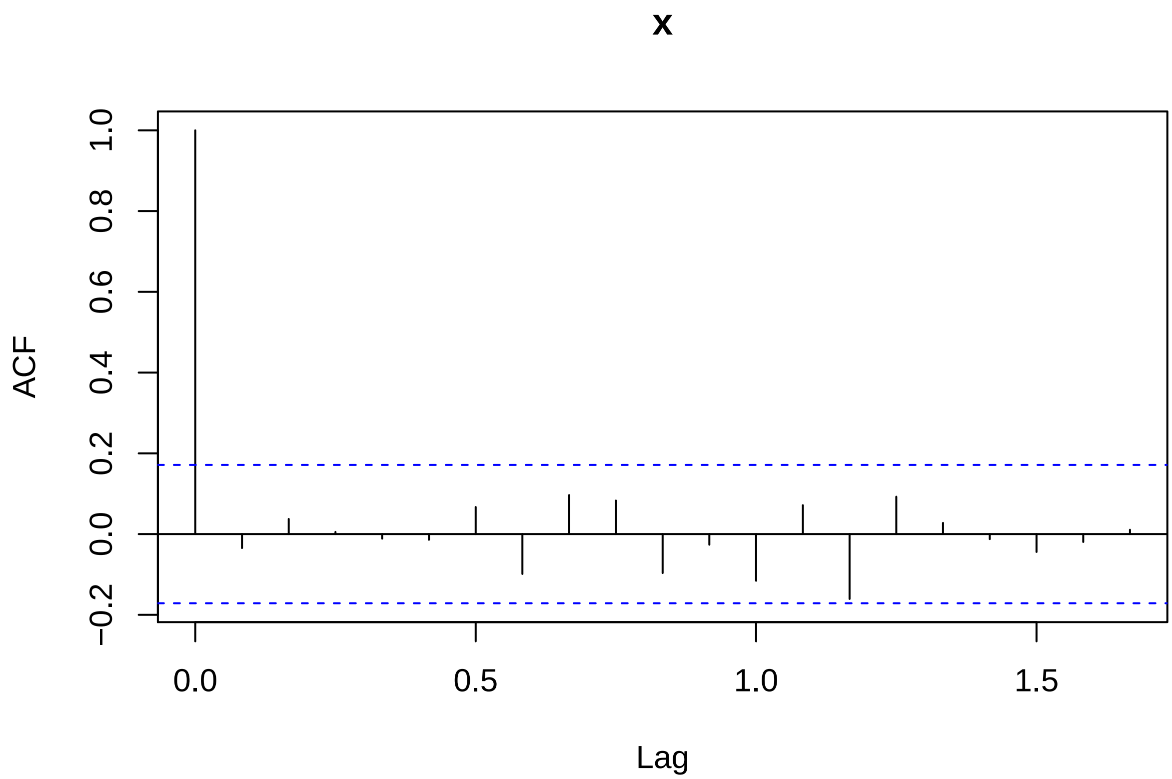


Growth

```
year_2019 <- window(ts, 2019)
year_2019_predict_HW <- (as.data.frame(ts_forcaste2))[1][c(1),]
sum_year_2019 = sum(c(year_2019,year_2019_predict_HW))
year_2020 = (as.data.frame(ts_forcaste2))[1][c(2:13),]
growth_HW <- growth(sum(year_2020),sum_year_2019)
growth_HW
```

```
## [1] 0.1508492
```

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:

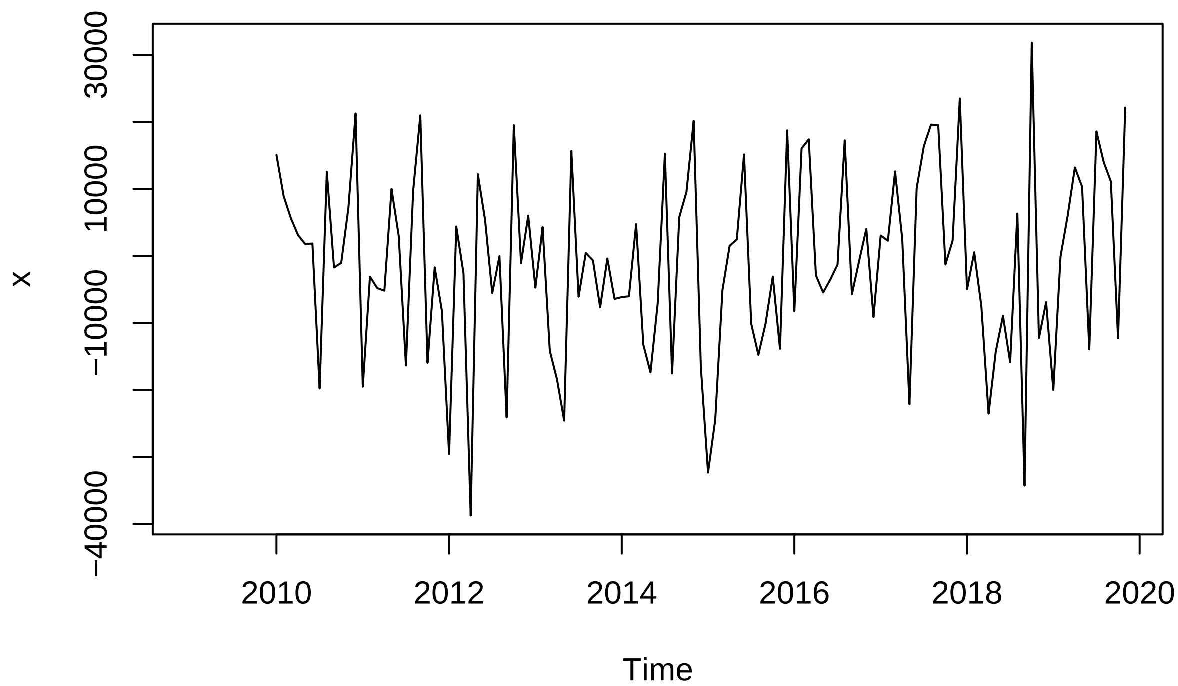



```
##
##  Box-Ljung test
##
## data:  ts_forcaste2$residuals
## X-squared = 13.33, df = 20, p-value = 0.8628
```

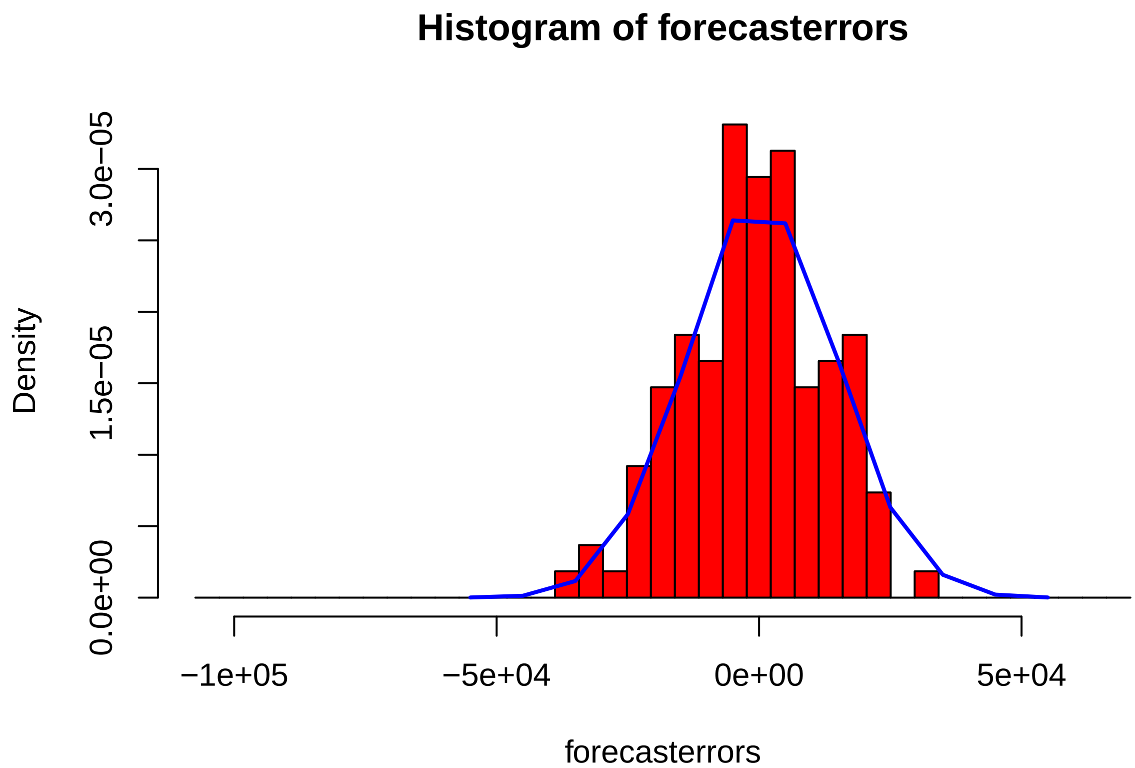
The correlogram shows that the autocorrelations for the in-sample forecast errors do not exceed the significance bounds for lags 1-20. Furthermore, the p-value for Ljung-Box test is 0.9, indicating that there is no evidence of non-zero autocorrelations at lags 1-20.

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)
```



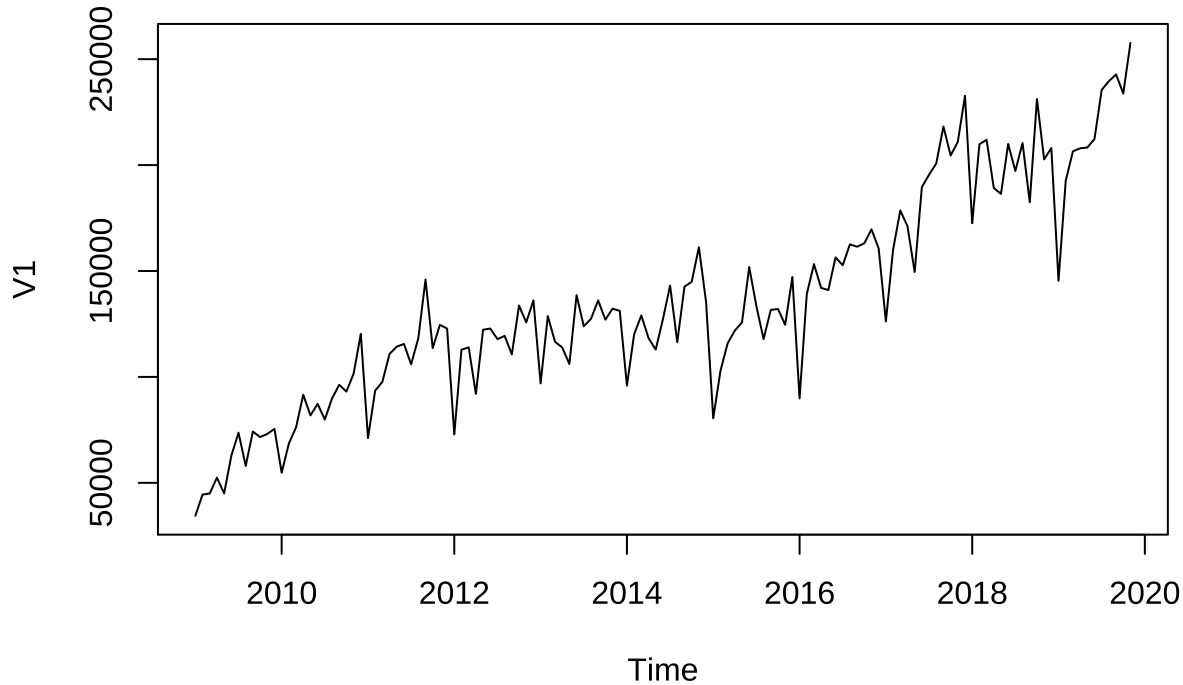

```r
plotForecastErrors(ts_forcaste2$residuals)
```



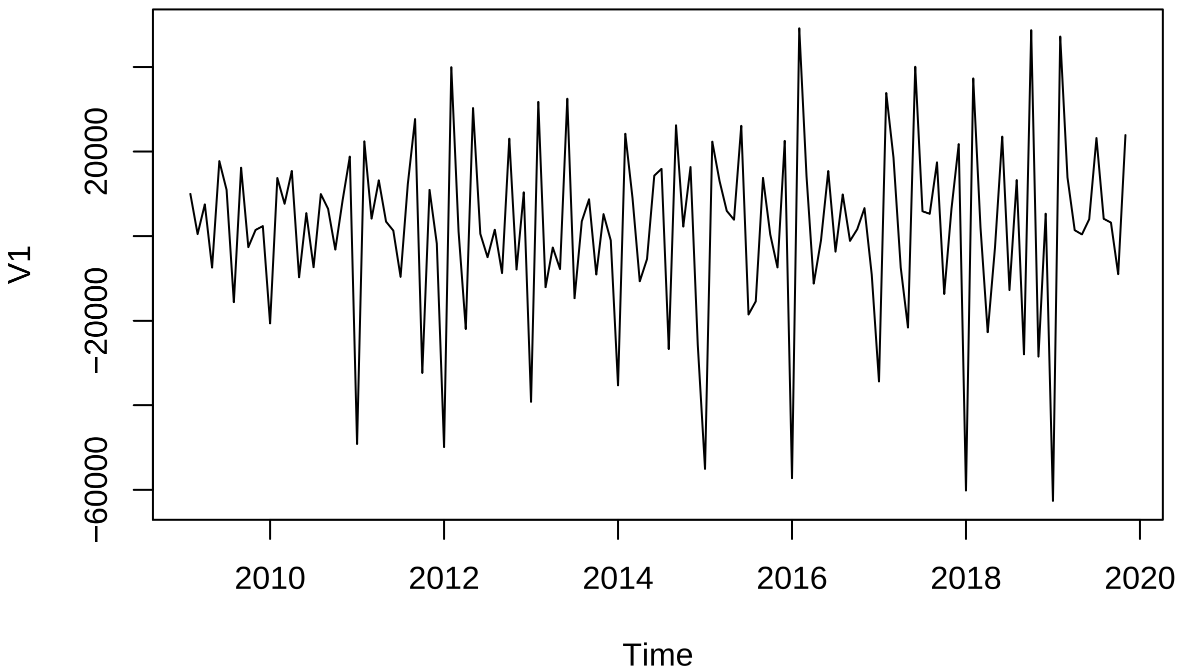
From the time plot, it appears plausible that the forecast errors have constant variance over time. From the histogram of forecast errors, it seems plausible that the forecast errors are normally distributed with mean zero.

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)
```

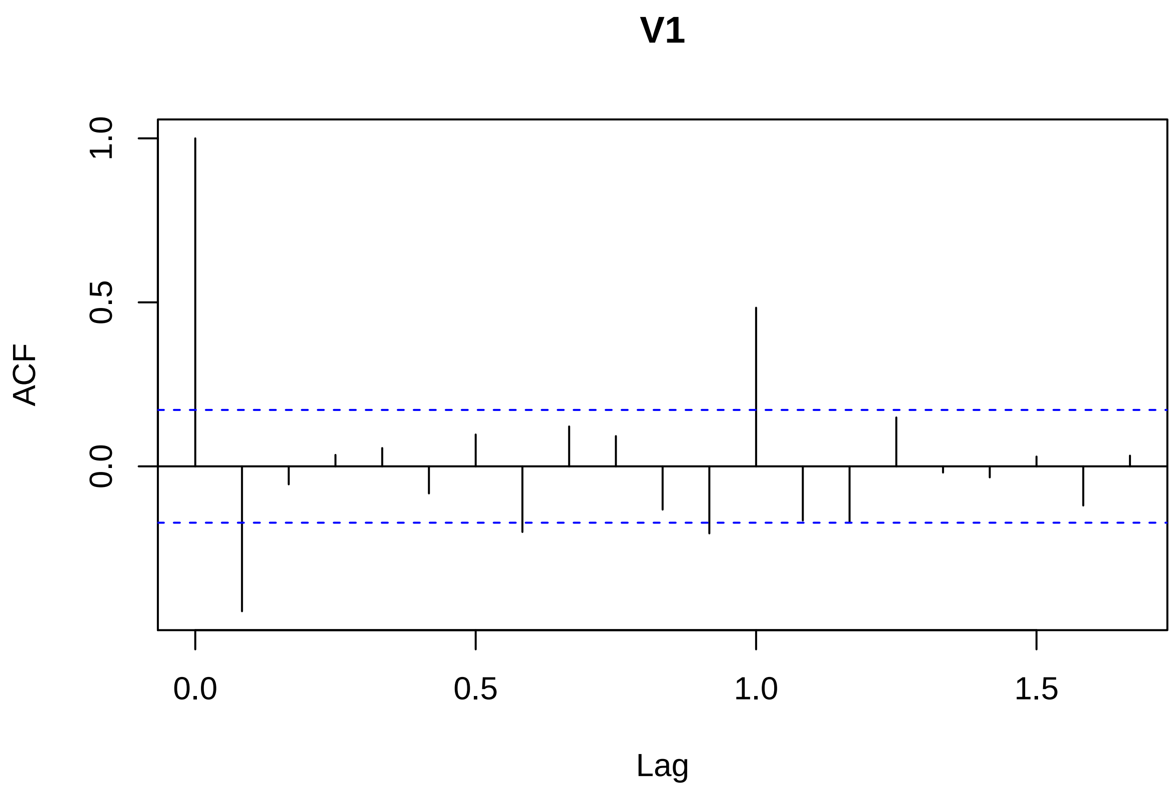



```
ts_diff1 <-  diff(ts, differences = 1)

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
```

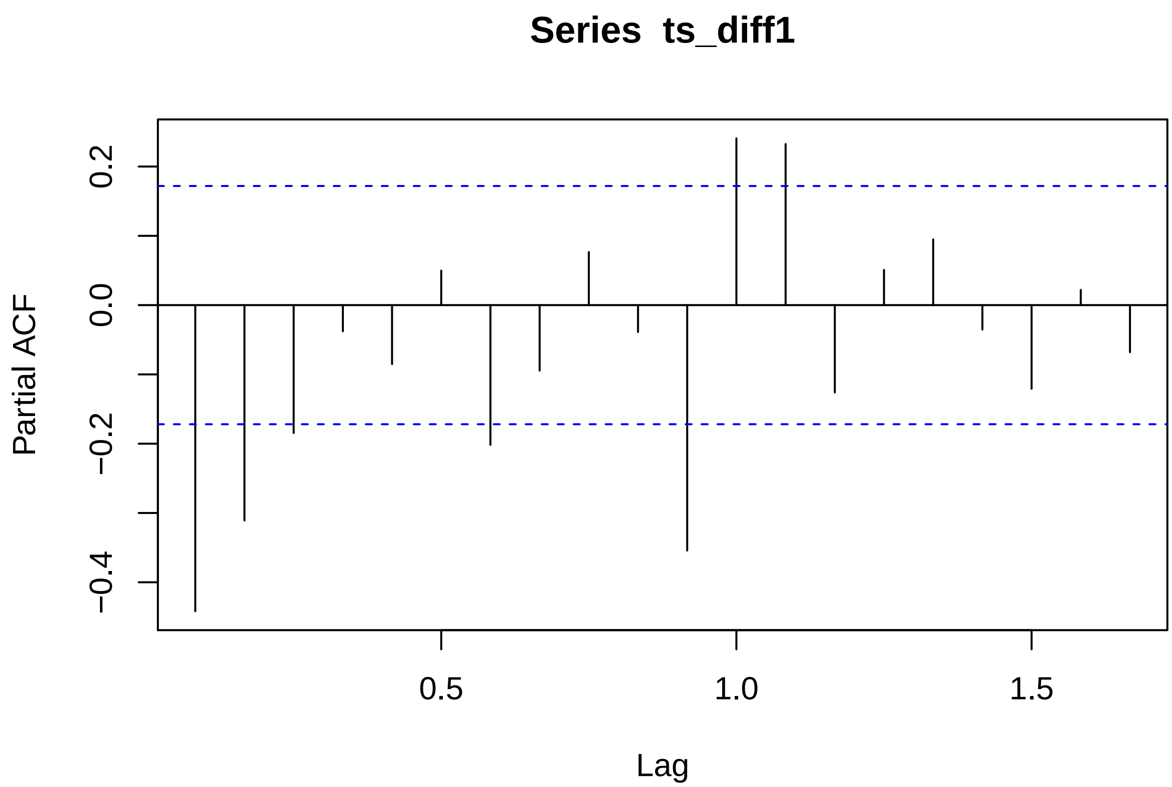


We see from the correlogram that the autocorrelation exceeds the significance bound 3 times but all the others do not exceed

```
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 -0.442 -0.055  0.035  0.056 -0.083  0.097 -0.200  0.122  0.092
## 0.8333 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833
## -0.132 -0.204  0.484 -0.165 -0.168  0.149 -0.019 -0.034  0.030 -0.119
## 1.6667
##  0.033
```

```
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
## -0.442 -0.311 -0.185 -0.038 -0.085  0.050 -0.202 -0.095  0.077 -0.039
## 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667
## -0.354  0.241  0.232 -0.126  0.051  0.095 -0.035 -0.121  0.022 -0.068
```

Arima, 0,1,0

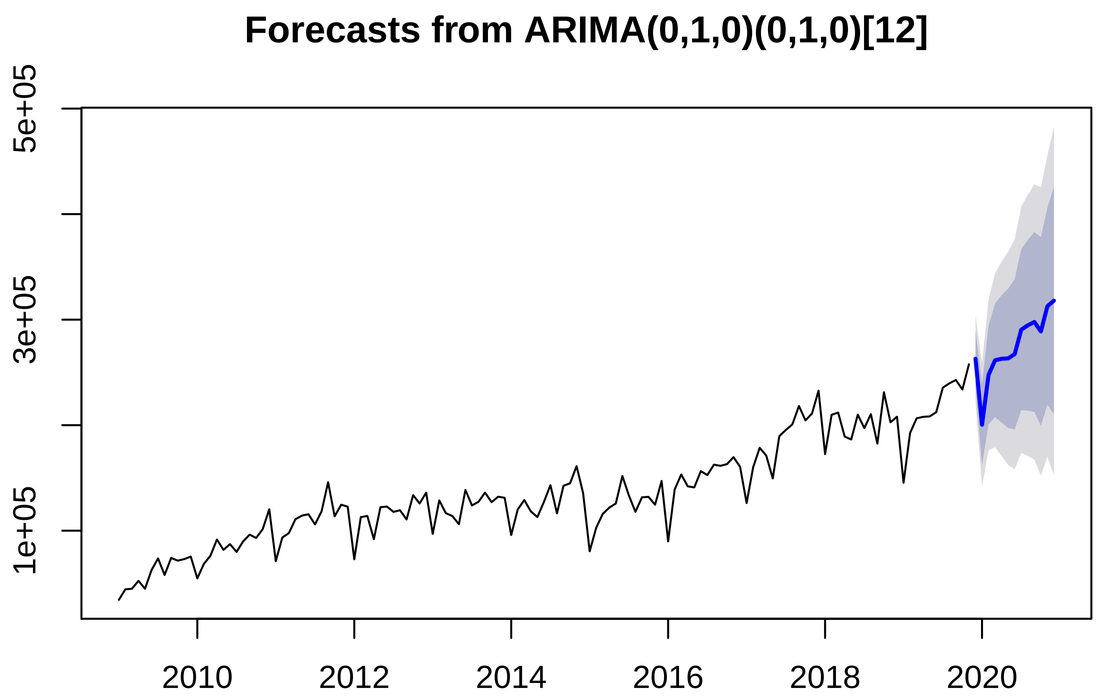
```r
ts_arima = Arima(ts, order=c(0,1,0),seasonal = list(order = c(0,1,0)))
ts_arima
```

```
## Series: ts
## ARIMA(0,1,0)(0,1,0)[12]
##
## sigma^2 estimated as 442379690:  log likelihood=-1341.99
## AIC=2685.98   AICc=2686.01   BIC=2688.75
```

```r
ts_arima_forecast = forecast(ts_arima,h = 13)
ts_arima_forecast
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Dec 2019       263021.4 236066.8 289976.1 221797.9 304245.0
## Jan 2020       200410.5 162290.9 238530.2 142111.6 258709.5
## Feb 2020       247610.5 200923.7 294297.4 176209.2 319011.9
## Mar 2020       261510.5 207601.2 315419.8 179063.4 343957.7
## Apr 2020       262910.5 202638.1 323183.0 170731.8 355089.3
## May 2020       263310.5 197285.4 329335.7 162333.8 364287.3
## Jun 2020       267310.5 195995.2 338625.8 158243.2 376377.9
## Jul 2020       290510.5 214271.3 366749.8 173912.6 407108.4
## Aug 2020       294610.5 213746.6 375474.5 170939.8 418281.3
## Sep 2020       297810.5 212572.4 383048.6 167450.1 428170.9
## Oct 2020       288810.5 199412.1 378209.0 152087.4 425533.7
## Nov 2020       312710.5 219336.9 406084.2 169907.9 455513.2
## Dec 2020       318032.0 210213.4 425850.6 153137.7 482926.3
```

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



Growth

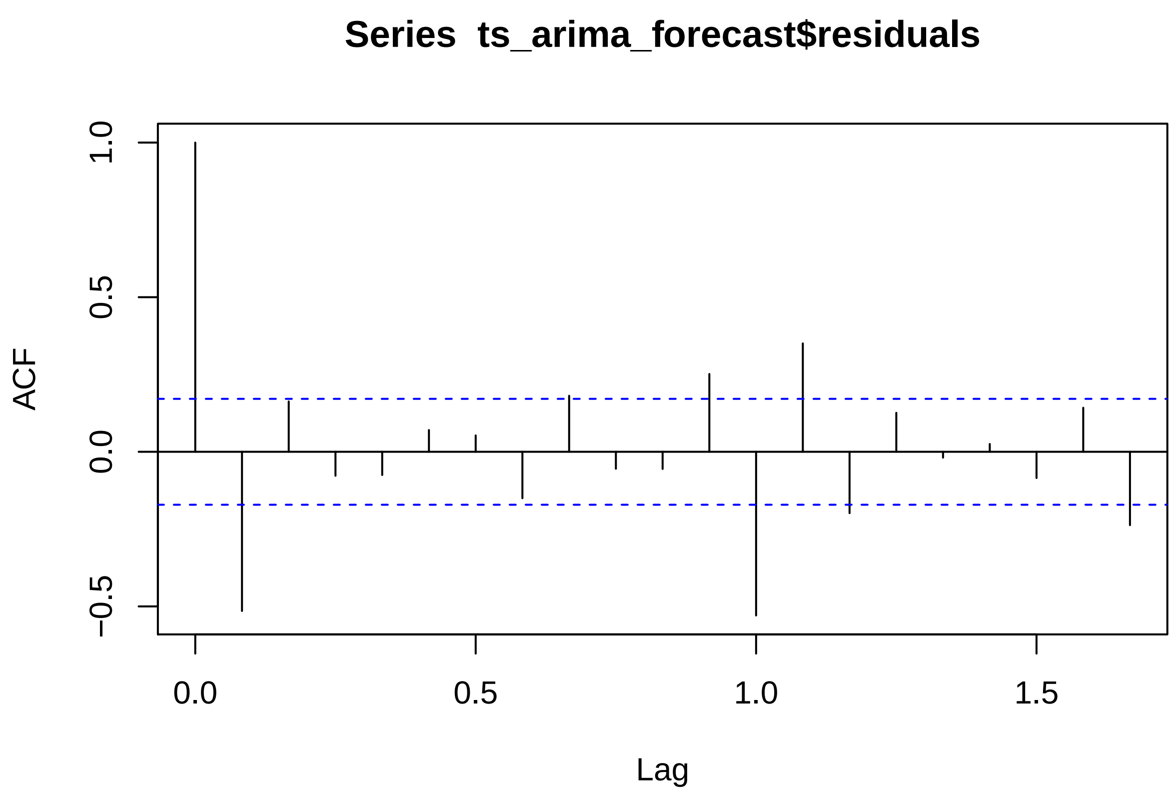
```r
this_year_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1]

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


year_2019_predict_ARIMA <- (as.data.frame(ts_arima_forecast))[1][c(1),]
sum_year_2019 = sum(c(year_2019,year_2019_predict_ARIMA))
year_2020 = (as.data.frame(ts_arima_forecast))[1][c(2:13),]
growth_ARIMA <- growth(sum_year_2019, sum(year_2020))
-growth_ARIMA
```

```
## [1] 0.1997026
```

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)
```

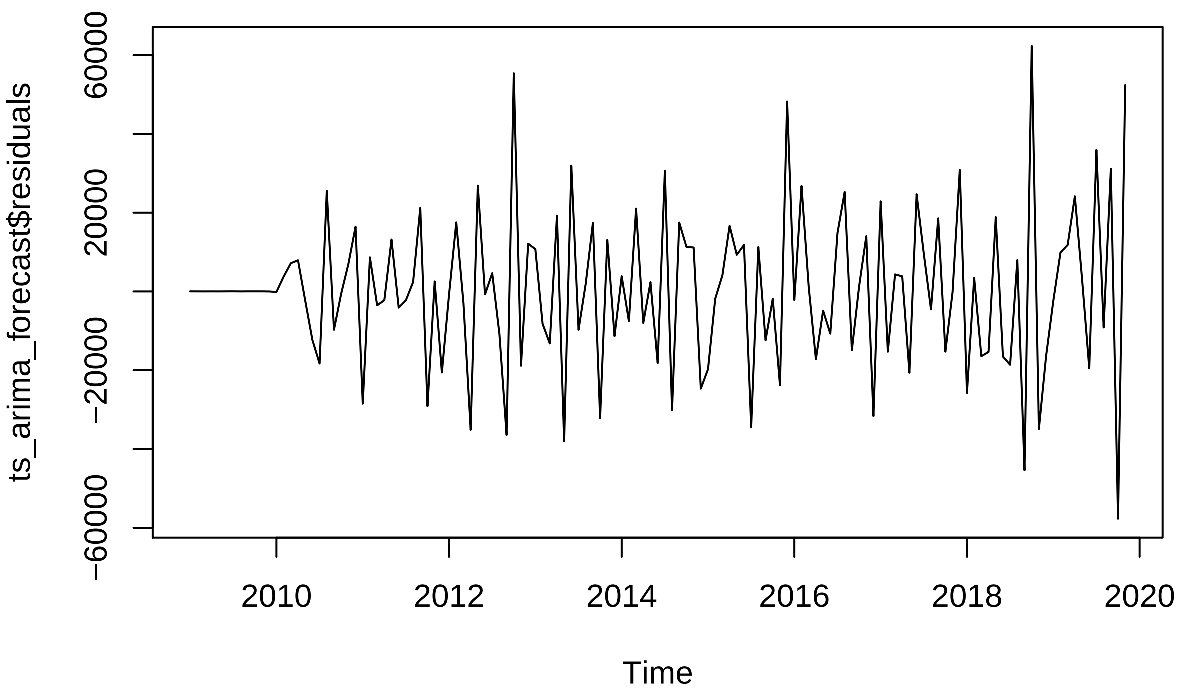


```r
Box.test(ts_arima_forecast$residuals, lag=20, type="Ljung-Box")
```

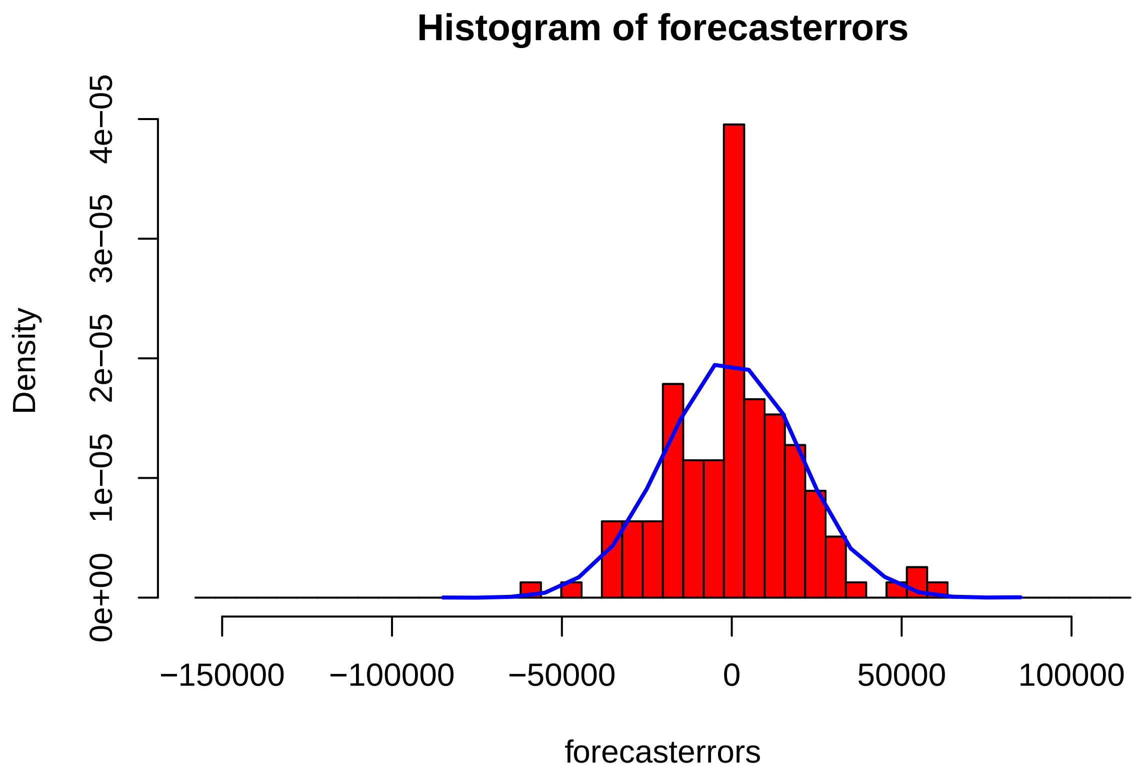
```
##
##  Box-Ljung test
##
## data:  ts_arima_forecast$residuals
## X-squared = 140.34, df = 20, p-value < 2.2e-16
```

p value too high to reject

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




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

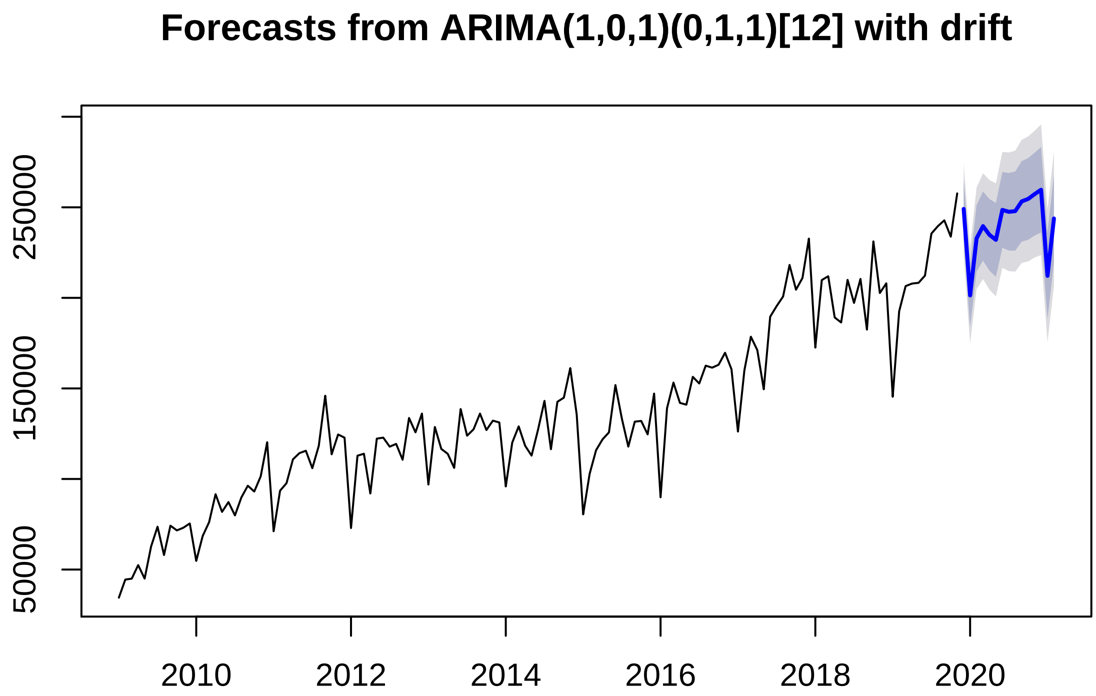


A model chosen automatically

```
fit3 <- auto.arima(ts)
fit3
```

```
## Series: ts
## ARIMA(1,0,1)(0,1,1)[12] with drift
##
## Coefficients:
##          ar1      ma1     sma1      drift
##       0.9554  -0.5899  -0.8838  1299.0445
## s.e.  0.0387   0.0854   0.1439   204.5971
##
## sigma^2 estimated as 160329918:  log likelihood=-1299.54
## AIC=2609.07   AICc=2609.6   BIC=2622.97
```

```
fit_forecast = forecast(fit3,h=15)
plot(fit_forecast)
```




```r
# str(fit)
```

Growth

```r
year_2019 <- window(ts, 2019)
year_2019_predict_HW <- (as.data.frame(ts_forcaste2))[1][c(1),]
sum_year_2019 = sum(c(year_2019,year_2019_predict_HW))
year_2020 = (as.data.frame(ts_forcaste2))[1][c(2:13),]


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

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(2:13),]
year_2020_predict_auto.arima_95_low <- (as.data.frame(fit_forecast))[4][c(2:13),]
year_2020_predict_auto.arima_95_high <- (as.data.frame(fit_forecast))[5][c(2:13),]
```

```r
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] 0.1056297
```

```r
growth_auto.arima_95_low
```

```
## [1] -0.0307078
```

```r
growth_auto.arima_95_high
```

```
## [1] 0.2393957
```