Exponential Smoothing

Seun Odeyemi

February 19, 2018

```r
devtools::session_info()
```

```
##  setting  value
##  version  R version 3.4.3 (2017-11-30)
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Zulu
##  date     2018-02-24
##
##  package   * version date       source
##  backports   1.1.2   2017-12-13 CRAN (R 3.4.3)
##  base      * 3.4.3   2017-12-01 local
##  compiler    3.4.3   2017-12-01 local
##  datasets  * 3.4.3   2017-12-01 local
##  devtools    1.13.5  2018-02-18 CRAN (R 3.4.3)
##  digest      0.6.15  2018-01-28 CRAN (R 3.4.3)
##  evaluate    0.10.1  2017-06-24 CRAN (R 3.4.3)
##  graphics  * 3.4.3   2017-12-01 local
##  grDevices * 3.4.3   2017-12-01 local
##  htmltools   0.3.6   2017-04-28 CRAN (R 3.4.3)
##  knitr       1.20    2018-02-20 CRAN (R 3.4.3)
##  magrittr    1.5     2014-11-22 CRAN (R 3.4.3)
##  memoise     1.1.0   2017-04-21 CRAN (R 3.4.3)
##  methods   * 3.4.3   2017-12-01 local
##  Rcpp        0.12.15 2018-01-20 CRAN (R 3.4.3)
##  rmarkdown   1.8     2017-11-17 CRAN (R 3.4.3)
##  rprojroot   1.3-2   2018-01-03 CRAN (R 3.4.3)
##  stats     * 3.4.3   2017-12-01 local
##  stringi     1.1.6   2017-11-17 CRAN (R 3.4.3)
##  stringr     1.3.0   2018-02-19 CRAN (R 3.4.3)
##  tools       3.4.3   2017-12-01 local
##  utils     * 3.4.3   2017-12-01 local
##  withr       2.1.1   2017-12-19 CRAN (R 3.4.3)
##  yaml        2.1.16  2017-12-12 CRAN (R 3.4.3)
```

```r
# runif(1, 0, 10^8)
set.seed(77159275) #for reproducibility of results
```

Loading some useful libraries

```r
#library(XLConnect)
library(dplyr)
library(ggplot2)
#library(forecast)
```

```r
library(fpp2)
library(readxl)
library(data.table)
```

Set Working Directory

```r
setwd("/home/sdotserver1/projects/")
```

Recap on the naive and the mean method of forecasting

Two very simple forecasting methods are the naive method and the mean method. The **naive** method uses *only* the most recent observation as the forecast for all future periods. While the **mean** method uses the average of all observations as the forecast for all future periods. Something between these two *extremes* will be useful. A forecast based on all observations, but where the most recent observations are heavily weighted. This is the idea behind *exponentially-weighted forecasts*, which is commonly known as **simple exponential smoothing (SES)**.

Forecasting Equation

$$\hat{y}_{t+h|t} = point\ forecast\ of\ y_{t+h}\ given\ data\ y_1, ..., y_t$$

Here we use $\hat{y}$ to denote a point forecast where the subscript $_{t+h|t}$ tell us the period we are forecasting and how far ahead we are forecasting. This means we are forecasting h *steps* ahead given data up to time, t.

Exponentially-Weighted Forecast Equation

$$\hat{y}_{t+h|t} = \alpha y_t \ + \ \alpha(1-\alpha)y_{t-1} \ + \ \alpha(1-\alpha)^2 y_{t-2} + ...$$

where $0 \leq \alpha \leq 1$

This is a weighted average of all the data up to time, t with the weights decreasing exponentially as you go back in time. The α indicates how much weight is placed on the most recent observation and how quickly the weights decay away. A large α indicates that more weight is placed on the most recent observation and weights decay very quickly. A small α indicates that a small weight is placed on the most recent observation and the weights decay away more slowly. $\alpha \to 0$ indicates a lot randomness in the data; $\alpha \to 1$ indicates not much randomness in the data.

A Component Form of the Forecast Equation

Forecast equation: $\hat{y}_{t+h|t} = \ell_t$ Smoothing equation: $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$

where:

- ℓ_t is the level (or the smoothed value) of the series at time t
- We choose α and ℓ_0 by minimizing the SSE:

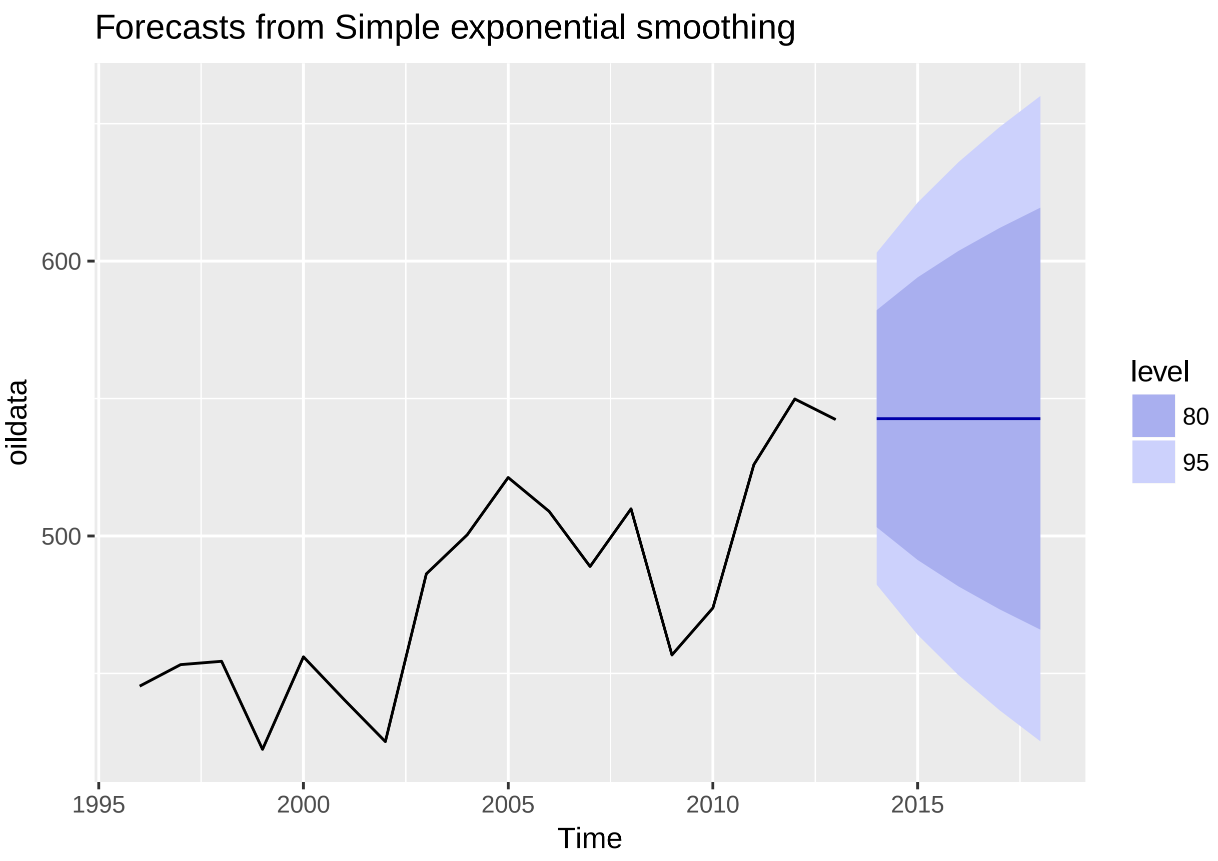
$$\text{SSE} = \sum_{t=1}^{T} \left(y_t - \hat{y}_{t|t-1} \right)^2$$

In regression, parameters are estimated by minimizing the sum of squared errors. You can do exactly the same here. However, unlike regression there is no nice formula that gives the optimal parameters. Instead we use a non-linear optimization routine. You can do all this in R with the `ses()` function.

```
# The oil dataset in the fpp2 package is ts object containing the annual oil production in Saudi Arabia
oildata <- window(oil, start = 1996)
fc<-ses(oildata, h = 5)
summary(fc)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = oildata, h = 5)
##
##   Smoothing parameters:
##     alpha = 0.8339
##
##   Initial states:
##     l = 446.5868
##
##   sigma:  30.8065
##
##      AIC     AICc      BIC
## 178.1430 179.8573 180.8141
##
## Error measures:
##                    ME     RMSE     MAE      MPE     MAPE      MASE
## Training set 6.401975 28.12234 22.2587 1.097574 4.610635 0.9256774
##                     ACF1
## Training set -0.03377748
##
## Forecasts:
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2014       542.6806 503.2005 582.1607 482.3010 603.0602
## 2015       542.6806 491.2749 594.0862 464.0624 621.2987
## 2016       542.6806 481.6363 603.7249 449.3214 636.0397
## 2017       542.6806 473.3245 612.0367 436.6096 648.7515
## 2018       542.6806 465.9074 619.4538 425.2661 660.0951
```

```
autoplot(fc)
```



This function has estimated an alpha value of 0.83 (quite high), which means that 83% of the forecast is based on the most recent observation. 14% is based on the observation before that and the remaining 3% from the earlier observations. The initial level, ℓ_0 is estimated to be about 447. We set h = 5, so we get a forecast for the next five years.

The plot of the forecast is shown above. Please note that the `ses()` function returns the same value for all forecasts. It is the estimated mean of the future possible sample paths. Because the value of α is quite high, the forecasts is closer to the most recent observation.

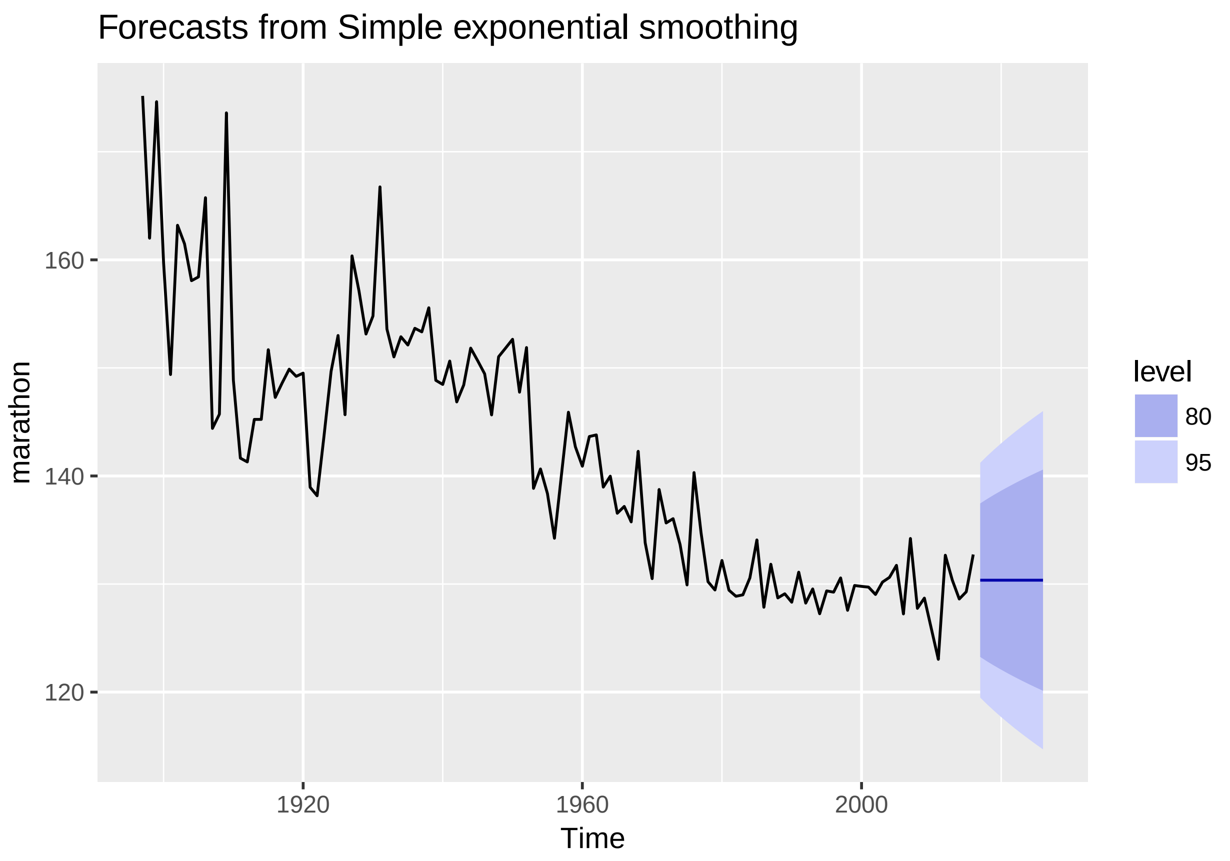
The SES is a very simple method, but it forms the starting point for complex methods in the exponential smoothing family. Methods that will handle trends or seasonality.

```
# Using SES to forecast the next 10 years of winning times in the marathon ts object
# The marathon ts object contains the annual winning times in the Boston Marathon from 1887 - 2016.

# Use ses() to forecast the next 10 years of winning times
fcmarathon <- ses(marathon, h = 10)

# Use summary() to see the model parameters
summary(fcmarathon)
```

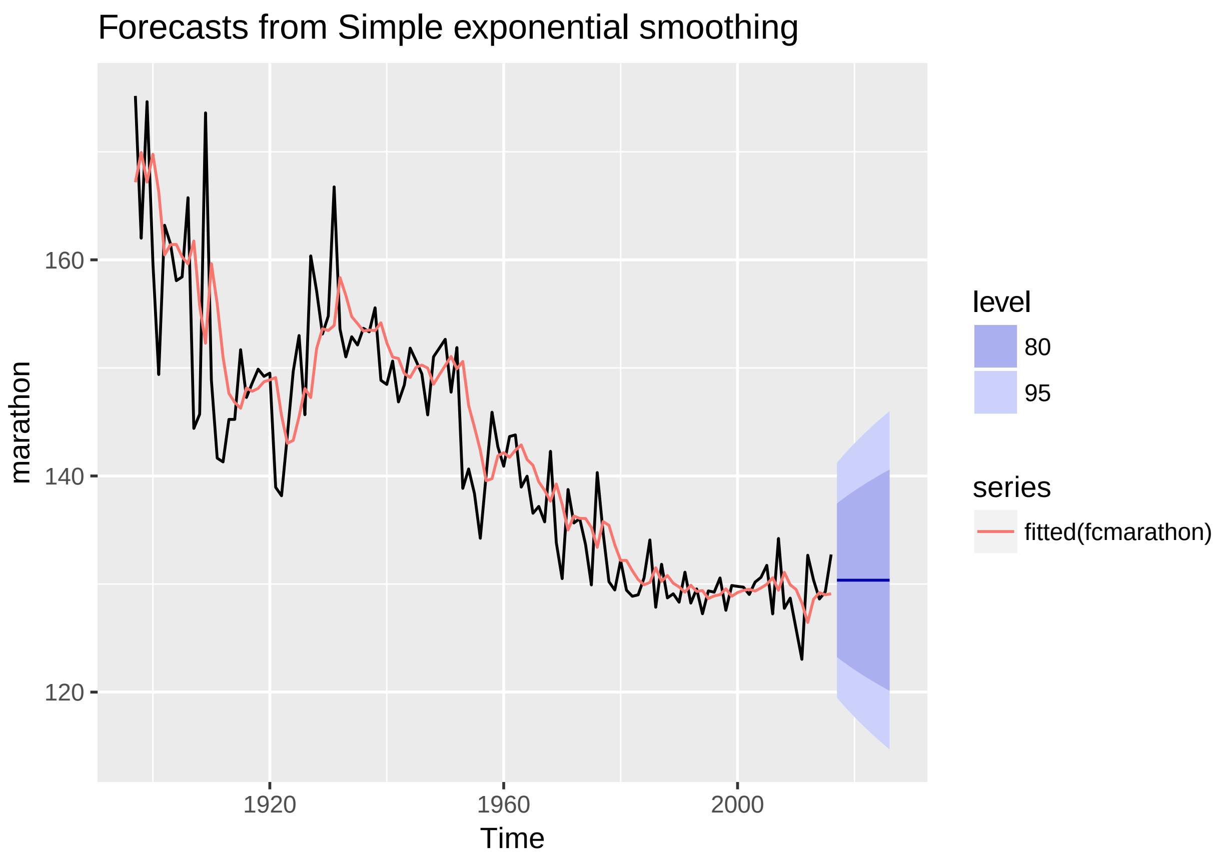
```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = marathon, h = 10)
```

```
##
##   Smoothing parameters:
##     alpha = 0.3457
##
##   Initial states:
##     l = 167.1741
##
##   sigma:  5.5425
##
##      AIC     AICc      BIC
## 988.4474 988.6543 996.8099
##
## Error measures:
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -0.8874349 5.472771 3.826294 -0.7097395 2.637644 0.8925685
##                     ACF1
## Training set -0.01211236
##
## Forecasts:
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2017       130.3563 123.2533 137.4593 119.4932 141.2194
## 2018       130.3563 122.8408 137.8719 118.8623 141.8503
## 2019       130.3563 122.4498 138.2629 118.2643 142.4484
## 2020       130.3563 122.0772 138.6355 117.6945 143.0182
## 2021       130.3563 121.7207 138.9920 117.1492 143.5635
## 2022       130.3563 121.3783 139.3344 116.6256 144.0871
## 2023       130.3563 121.0485 139.6642 116.1212 144.5915
## 2024       130.3563 120.7300 139.9827 115.6341 145.0786
## 2025       130.3563 120.4217 140.2910 115.1626 145.5501
## 2026       130.3563 120.1227 140.5900 114.7053 146.0074
```

```
# Use autoplot() to plot the forecasts
autoplot(fcmarathon)
```



```
# Add the one-step forecasts for the training data to the plot
autoplot(fcmarathon) + autolayer(fitted(fcmarathon))
```



fitted(fc) will create a time series of the one-step forecasts, so, autolayer(fitted(fc)) will add the fitted values to a plot.

SES v. naive

```
# Create a training set using subset.ts()
train <- subset(marathon, start = 1, end = 99)
# train <- subset.ts(marathon, start = 1, end = 99)

# Compute SES and naive forecasts, save to fcses and fcnaive
fcses <- ses(train, h = 20)
fcnaive <- naive(train, h = 20)

# Calculate forecast accuracy measures
accuracy(fcses, marathon)
```

```
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -1.0997202 5.893222 4.194329 -0.8718679 2.853774 0.8984573
## Test set      0.4090967 2.370638 1.757812  0.2835383 1.360834 0.3765369
##                     ACF1 Theil's U
## Training set -0.01681247        NA
## Test set     -0.14007447 0.6731108
```

```
accuracy(fcnaive, marathon)
```

```
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -0.46734694 6.939871 4.668367 -0.4118803 3.154511 1.0000000
## Test set     -0.06416669 2.335954 1.669167 -0.0825949 1.296984 0.3575483
##                    ACF1 Theil's U
## Training set -0.3591249        NA
## Test set     -0.1400745 0.6632255
```

```
# Save the best forecasts as fcbest
fcbest <- fcnaive
```

The results above shows that more complex models aren't always better.

Exponential smoothing methods with trend

Simple exponential smoothing works fine provided that your data has no trend and no seasonality. To handle trend and seasonalit, we will need to add more features to the forecasting equation. Remember the forecast equation for SES, $\hat{y}_{t+h|t} = \ell_t$, which gives the forecast of the last value of the estimated level. The second equation, the smoothing equation $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$, describes how the level changes over time as a function of the most recent observation and the previous estimate of the level. To deal with trend, we will need to add a trend component to the equation. The Holt's linear trend equations:

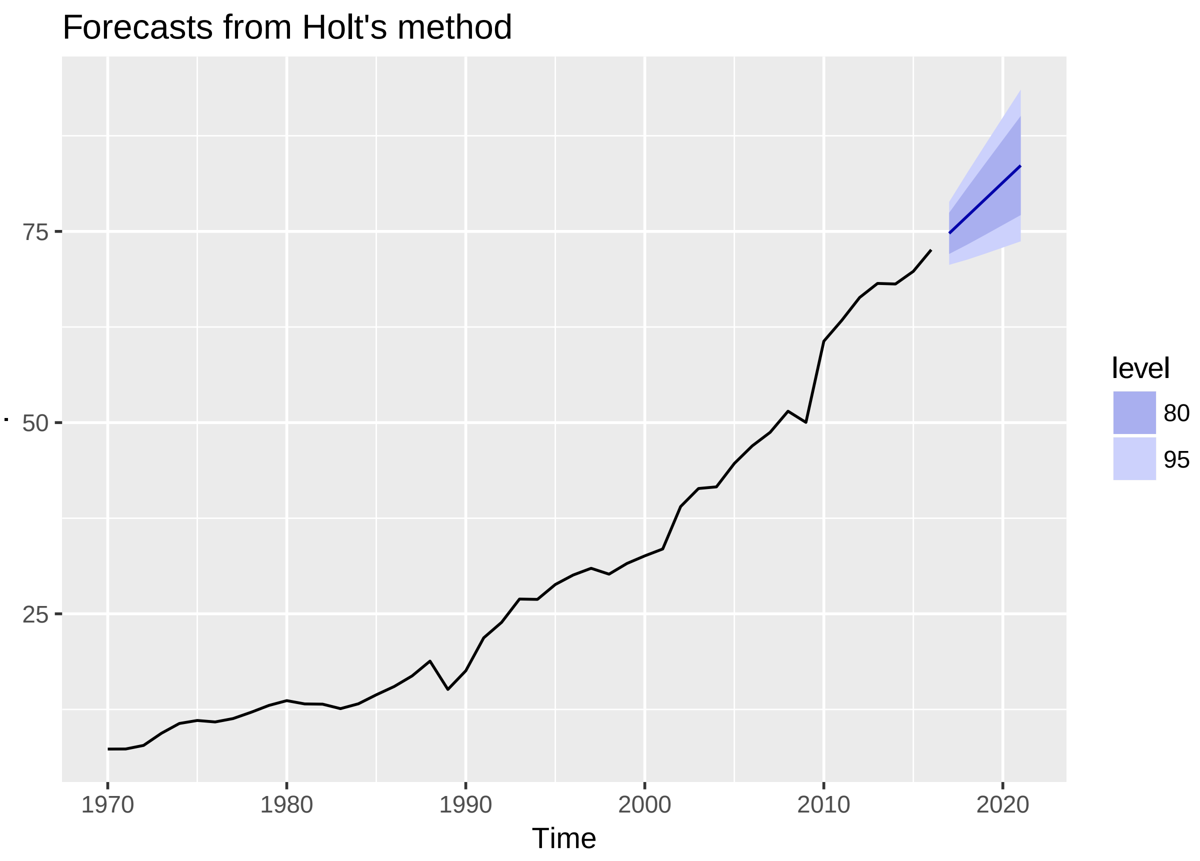
- Forecast $= \hat{y}_{t+h|t} = \ell_t + hb_t$
- Level $= \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$
- Trend $= b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$

The forecast equation is now a linear function of the forecast horizon so it gives a *trended forecast* with slope $= b_t$. The level is similar to what it was before, but we should it adjust slightly to allow for the fact that the data are now trended. The trend equation describes how the slope changes over time. Because the slope is allowed to change over time, this is often called a **local linear trend**. The β^* controls how quickly the slope can change: a small β^* value means the slope hardly changes. So the data will have a trend that is close to linear throughout the series. A larger β^* value means the slope changes rapidly, allowing for highly non-linear trend. (We use β^* here rather than β because we will use β later on).

- The smoothing parameters α and β^* $(0 \leq \alpha, \beta^* \leq 1)$.

There are now four parameters to estimate – $\alpha, \beta^*, \ell_0, b_0$ – to minimize SSE. The smoothing parameters α and β^* and the state parameters ℓ_0, b_0. The `holt()`function in R will handle these for us. This method is named after Charles Holt who developed it in the 1950s while working on forecasting for the US Navy. An example of applying the method to the `ts()` object containing total air passenger traffic in Australia is shown below:

```
ausair %>% holt(h=5) %>% autoplot()
```

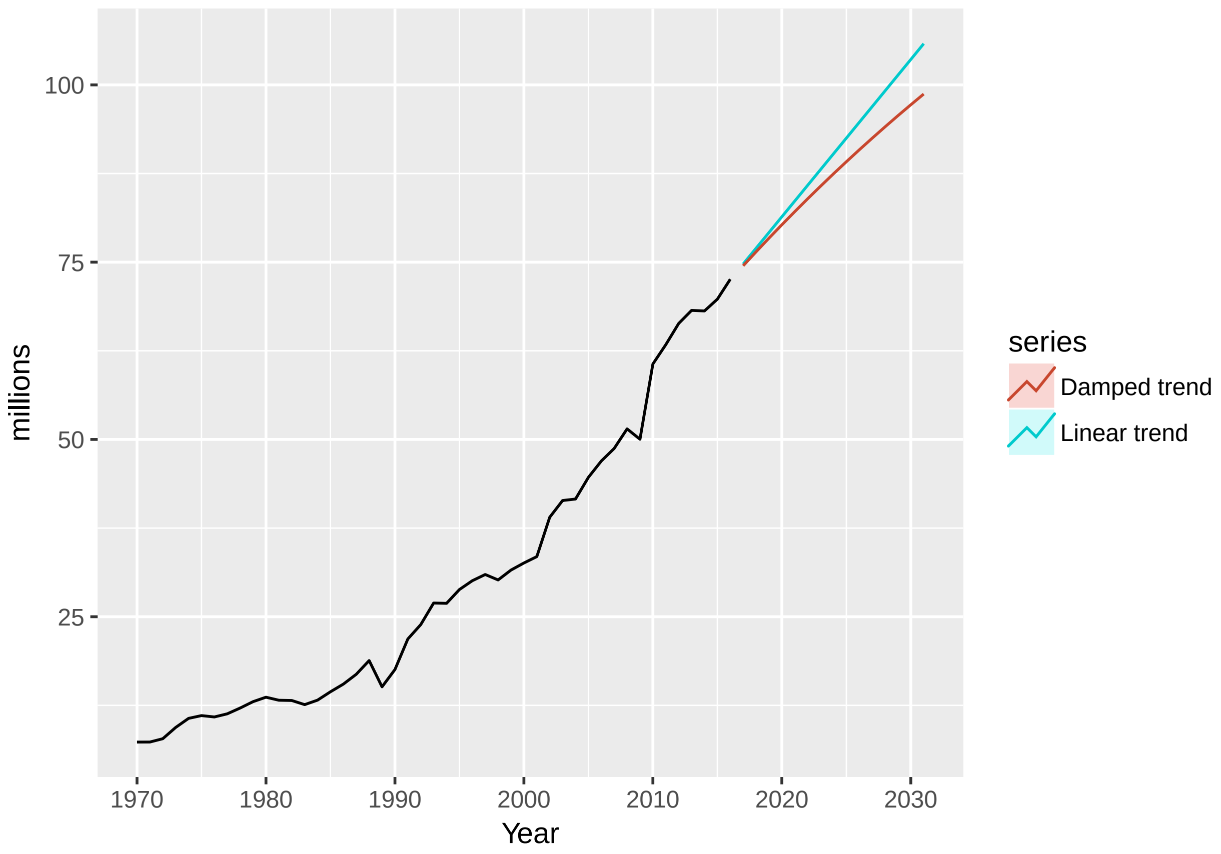


Like the `ses()` function, the `holt()` function will estimate the parameters and compute the forecast. The returned object contains information about parameters, forecasts and the predicted intervals. The `holt()` method will produce a forecast where the trend will continue at the same slope indefinitely into the future. A variation of this method is to allow the trend to **dampen** over time. This is called the **damped trend method**. The method was introducted by F. Gardner and Eddie McKenzie. They proposed a variation of the Holt's method with one extra parameter ϕ to control the dampen. The larger the value of ϕ the less dampen there is. $\phi = 1$ is equivalent to the Holt's method. Under this forecast, the short-run forecast are trended, and the long-run forecast are constant.

- Forecast $= \hat{y}_{t+h|t} = \ell_t + (\phi + \phi^2 + ... + \phi^h)b_t$
- Level $= \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$
- Trend $= b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$
- Damping parameter is $0 < \phi < 1$
- if $\phi = 1$, identical to Holt's linear method
- Short-run forecasts trended, long-run forecasts constant

An example is provided below:

```r
fc1 <- holt(ausair, h = 15, PI = FALSE)
fc2 <- holt(ausair, damped = TRUE, h = 15, PI = FALSE)
autoplot(ausair) + xlab("Year") + ylab("millions") +
autolayer(fc1, series="Linear trend") +
autolayer(fc2, series="Damped trend")
```



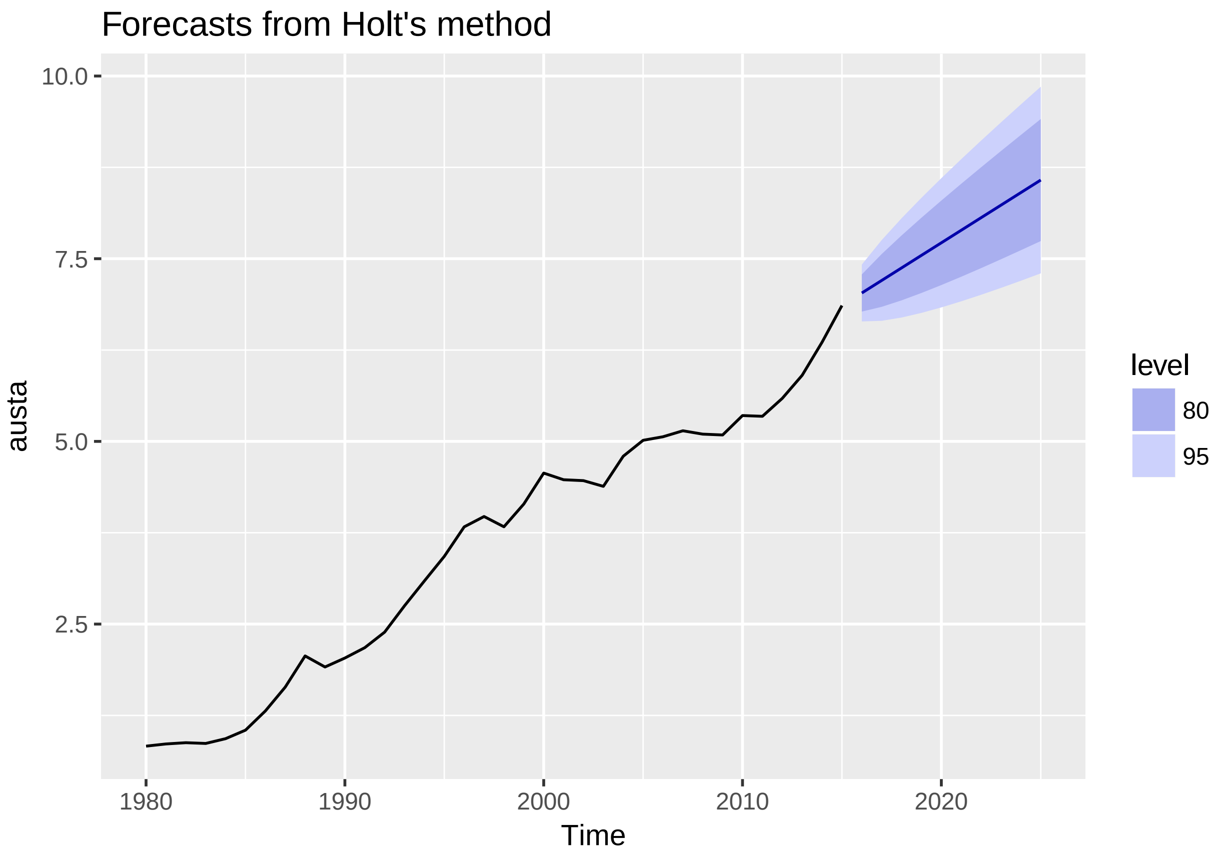
Notice that the damped trend method levels off while the linear trend method continues at the same slope for all future periods. The parameter ϕ which controls the dampen is estimated alongside the other parameters by the `holt()` function.

```
# Produce 10 year forecasts of austa using holt()
fcholt <- holt(austa, h = 10)

# Look at fitted model using summary()
summary(fcholt)
```

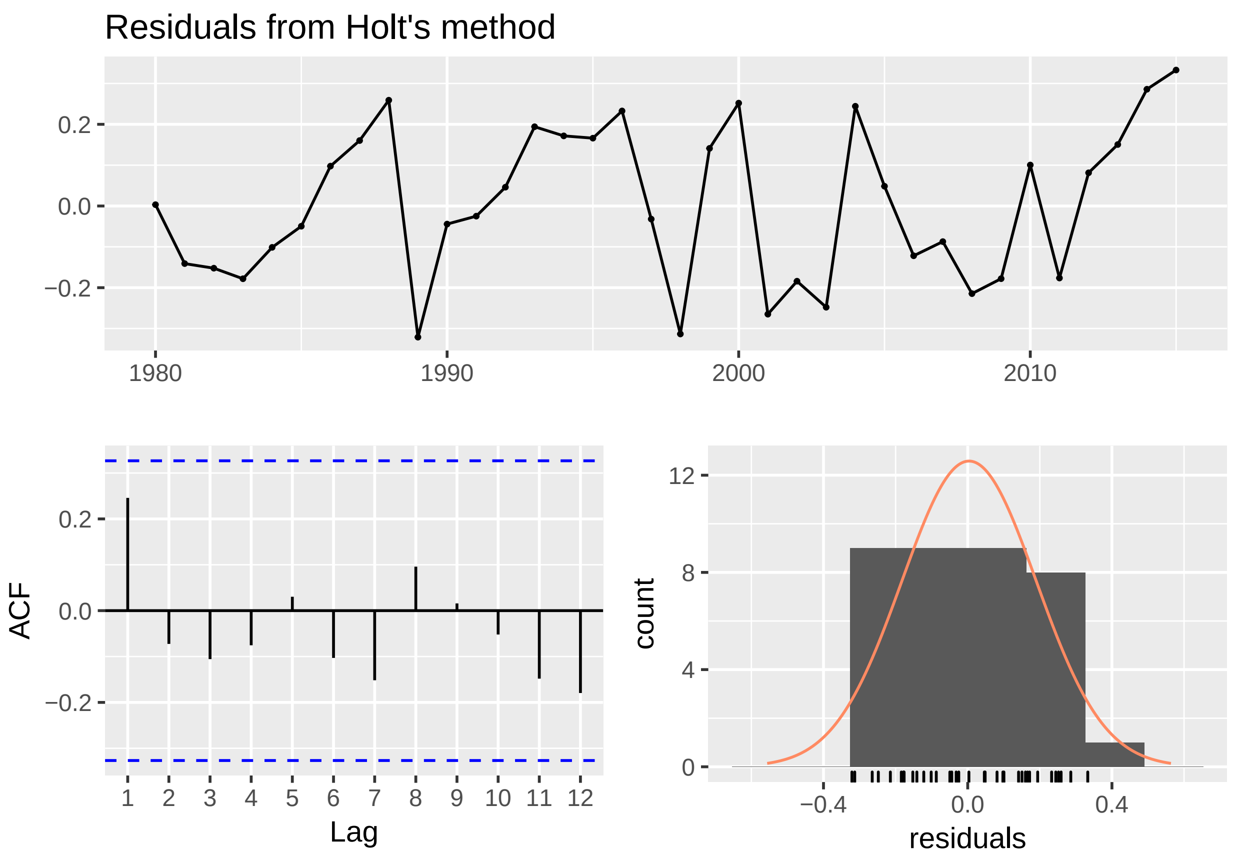
```
##
## Forecast method: Holt's method
##
## Model Information:
## Holt's method
##
## Call:
##  holt(y = austa, h = 10)
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.0085
##
##   Initial states:
##     l = 0.656
##     b = 0.1706
##
##   sigma:  0.1984
```

```
##
##      AIC     AICc      BIC
## 17.14959 19.14959 25.06719
##
## Error measures:
##                      ME      RMSE       MAE       MPE     MAPE      MASE
## Training set 0.00372838 0.1840662 0.1611085 -1.222083 5.990319 0.7907078
##                   ACF1
## Training set 0.2457733
##
## Forecasts:
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2016       7.030683 6.776480 7.284886 6.641913 7.419453
## 2017       7.202446 6.841429 7.563463 6.650318 7.754574
## 2018       7.374209 6.930176 7.818242 6.695119 8.053299
## 2019       7.545972 7.031070 8.060875 6.758497 8.333447
## 2020       7.717736 7.139618 8.295853 6.833582 8.601890
## 2021       7.889499 7.253527 8.525471 6.916863 8.862134
## 2022       8.061262 7.371438 8.751086 7.006267 9.116257
## 2023       8.233025 7.492473 8.973578 7.100448 9.365603
## 2024       8.404788 7.616021 9.193555 7.198473 9.611103
## 2025       8.576552 7.741643 9.411460 7.299669 9.853434
```

```r
# Plot the forecasts
autoplot(fcholt)
```



```r
# Check that the residuals look like white noise
checkresiduals(fcholt)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from Holt's method
## Q* = 5.493, df = 6, p-value = 0.4823
##
## Model df: 4.   Total lags used: 10
```

Exponential smoothing methods with trend and seasonality

Charles Holt also introduced a method that accounts for seasonality as well as trend. It has since been known as the **Holt-Winters method** – as Holt's student, Peter Winters, showed how to do the calculation efficiently. There are two versions of the Holt-Winters method: the **additive version** and **multiplicative version**. Here are the equations for the additive method:

- Forecast = $\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t-m+h_m^+}$
- Level = $\ell_t = \alpha(y_t - s_t - m) + (1-\alpha)(\ell_{t-1} + b_{t-1})$
- Trend = $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$
- Seasonality = $\gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$

They are similar to the equation of Holt's trend method but they include an additional term for the seasonal component, and an additional equation showing how the seasonal component evolves over time. There is one more smoothing parameter to estimate, γ and several more state parameters to estimate to account for the initial seasonal pattern. In this additive version the seasonal component averages to **zero**.

- $s_{t-m+h_m^+}$ = seasonal component from final year of data
- Smoothing paramaters: $0 \leq \alpha \leq 1$, $0 \leq \beta^* \leq 1$, $0 \leq \gamma \leq 1-\alpha$
- m = period of seasonality (e.g. m= 4 for quarterly data)

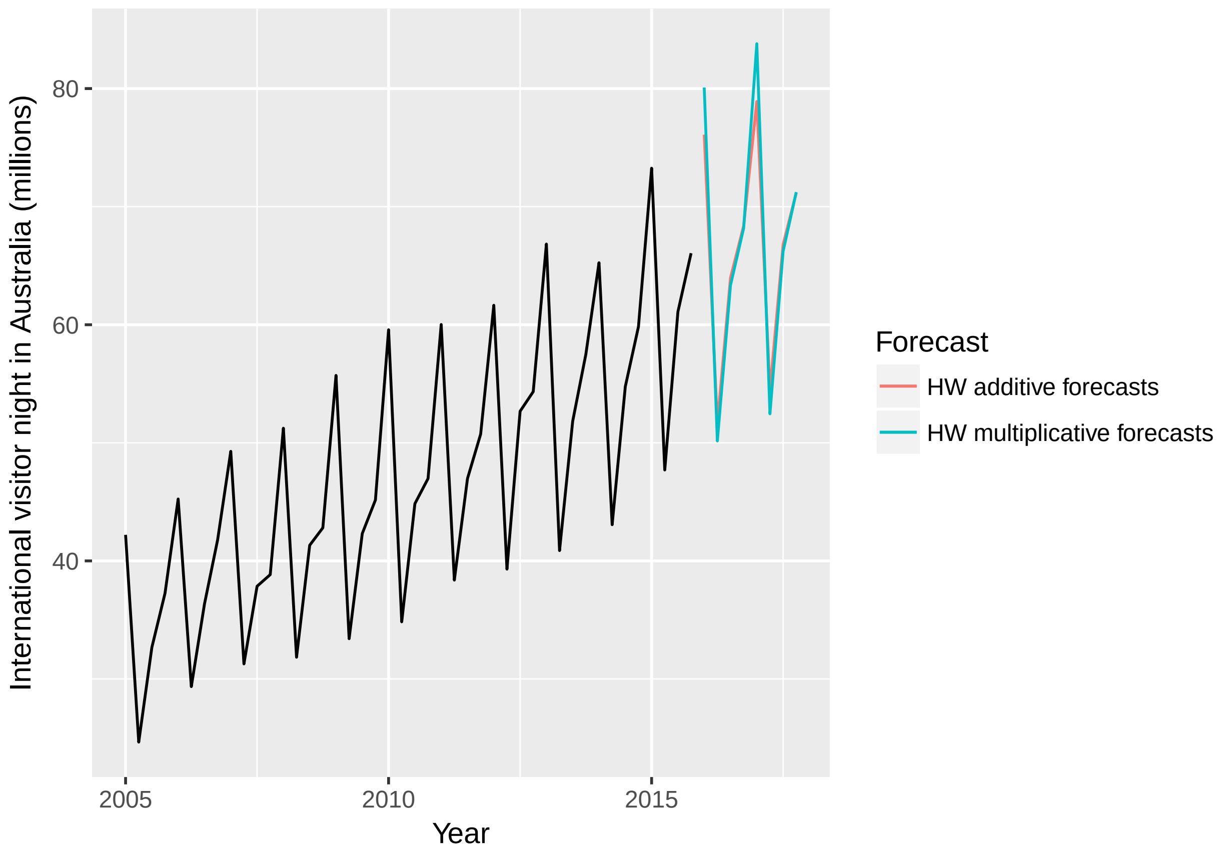
- seasonal component averages **zero**

The multiplicative version is very similar but instead of adding or substracting seasonality we use multiplication and division.

- Forecast $= \hat{y}_{t+h|t} = \ell_t + hb_t + s_{t-m+h_m^+}$
- Level $= \ell_t = \alpha \frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1})$
- Trend $= b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$
- Seasonality $= \gamma \frac{y_t}{(\ell_{t-1} - b_{t-1})} + (1-\gamma)s_{t-m}$
- $s_{t-m+h_m^+}$ = seasonal component from final year of data
- Smoothing paramaters: $0 \leq \alpha \leq 1$, $0 \leq \beta^* \leq 1$, $0 \leq \gamma \leq 1 - \alpha$
- m = period of seasonality (e.g. m= 4 for quarterly data)
- seasonal component averages **one**

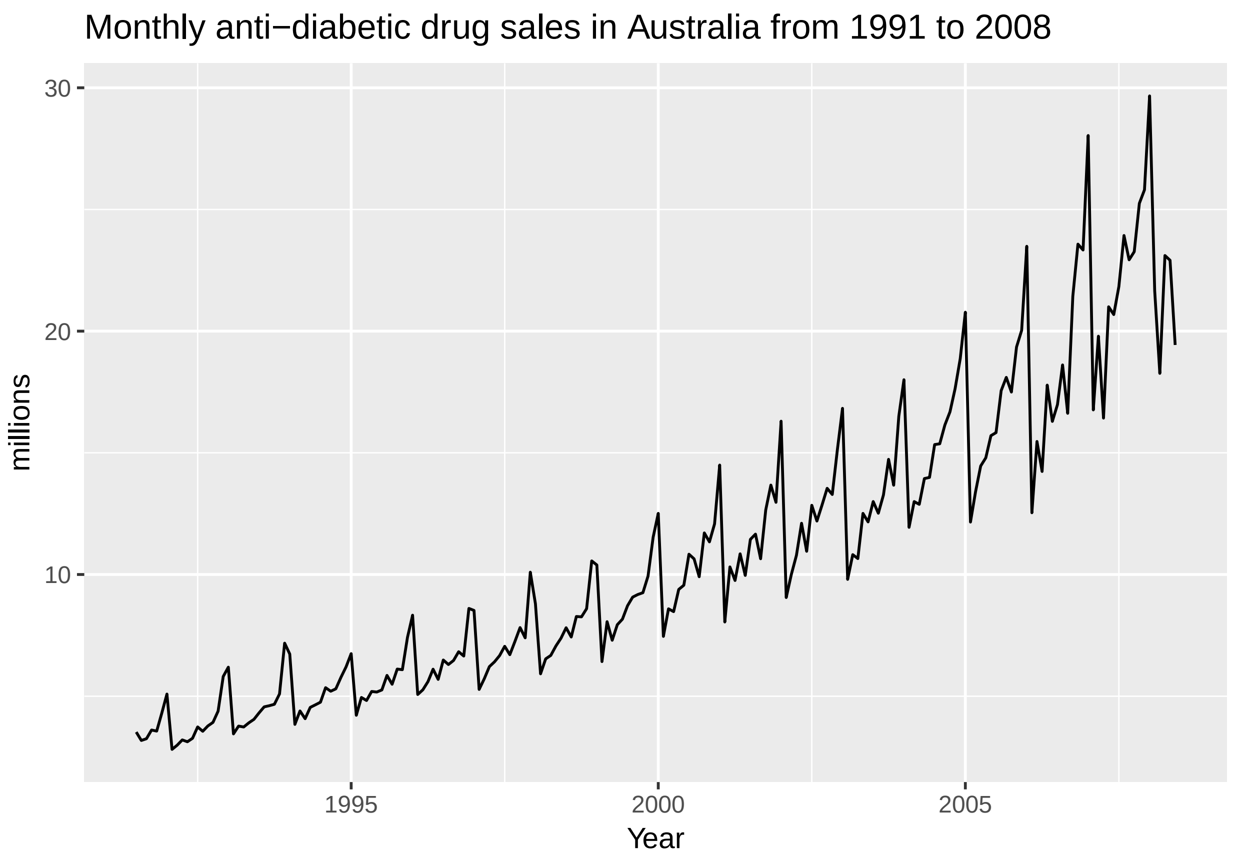
Notice that in the multiplicative version, the trend is still linear, but the seasonality is multiplicative. An example of a plot showing the number of nights spent by visitors in Australian accommodations such as hotels, motels, and guesthouses is provided below:

```r
aust <- window(austourists,start=2005)
fit1 <- hw(aust,seasonal="additive")
fit2 <- hw(aust,seasonal="multiplicative")
autoplot(aust) +
  autolayer(fit1$mean, series="HW additive forecasts") +
  autolayer(fit2$mean, series="HW multiplicative forecasts") +
  xlab("Year") + ylab("International visitor night in Australia (millions)") +
  guides(colour=guide_legend(title="Forecast"))
```

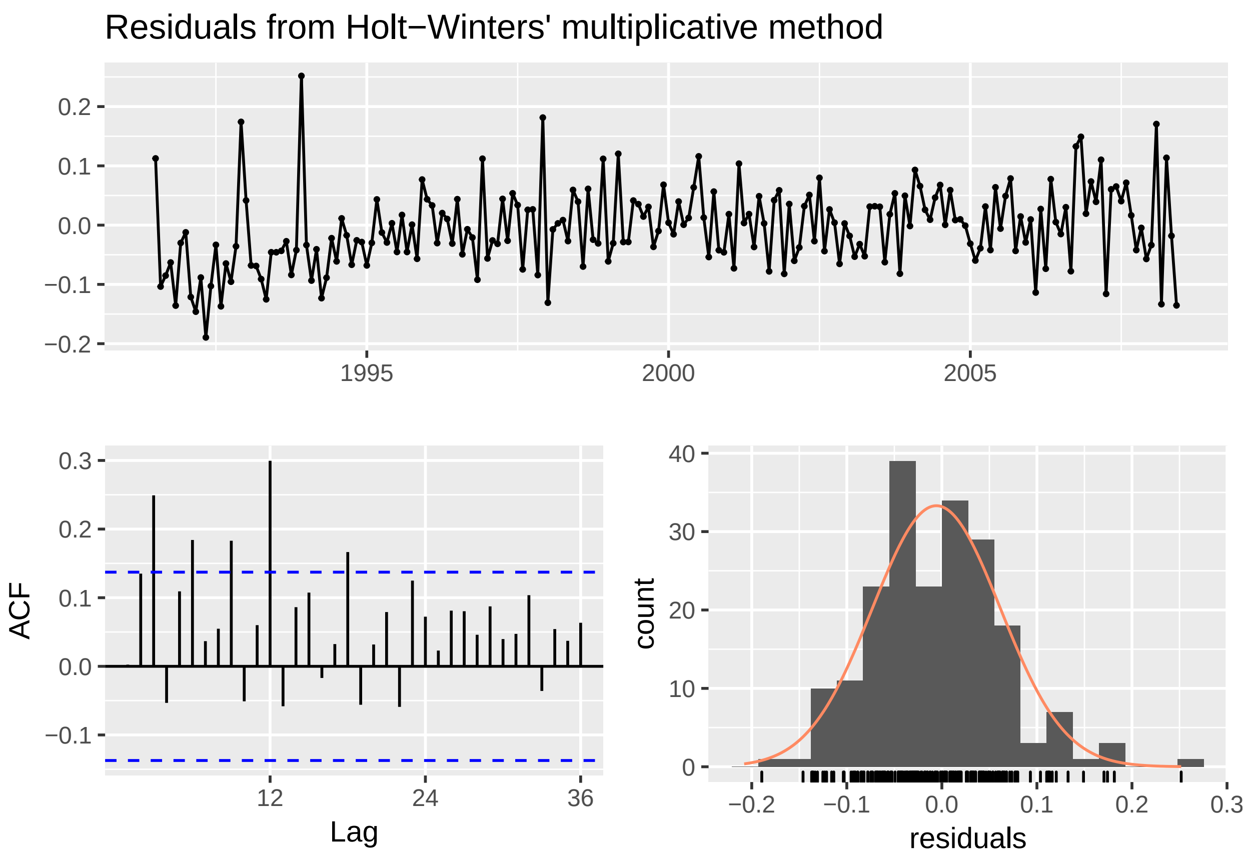


The data is quaterly and has strong seasonal pattern as you would expect. It is nicer to spend holidays in Australia during the summer. The `hw()` function produces forecasts using the Holt's-Winters method. The `seasonal` argument controls whether you want the additive or multiplicative forecast. In this particular example, it does not make much difference which version we use because the variation is much the same over the whole series. In cases when the *seasonal variation increases with the level of the series*, we will use the **multiplicative** method.

```
# Plot the data
autoplot(a10) +  xlab("Year") + ylab("millions") +
  ggtitle("Monthly anti-diabetic drug sales in Australia from 1991 to 2008")
```

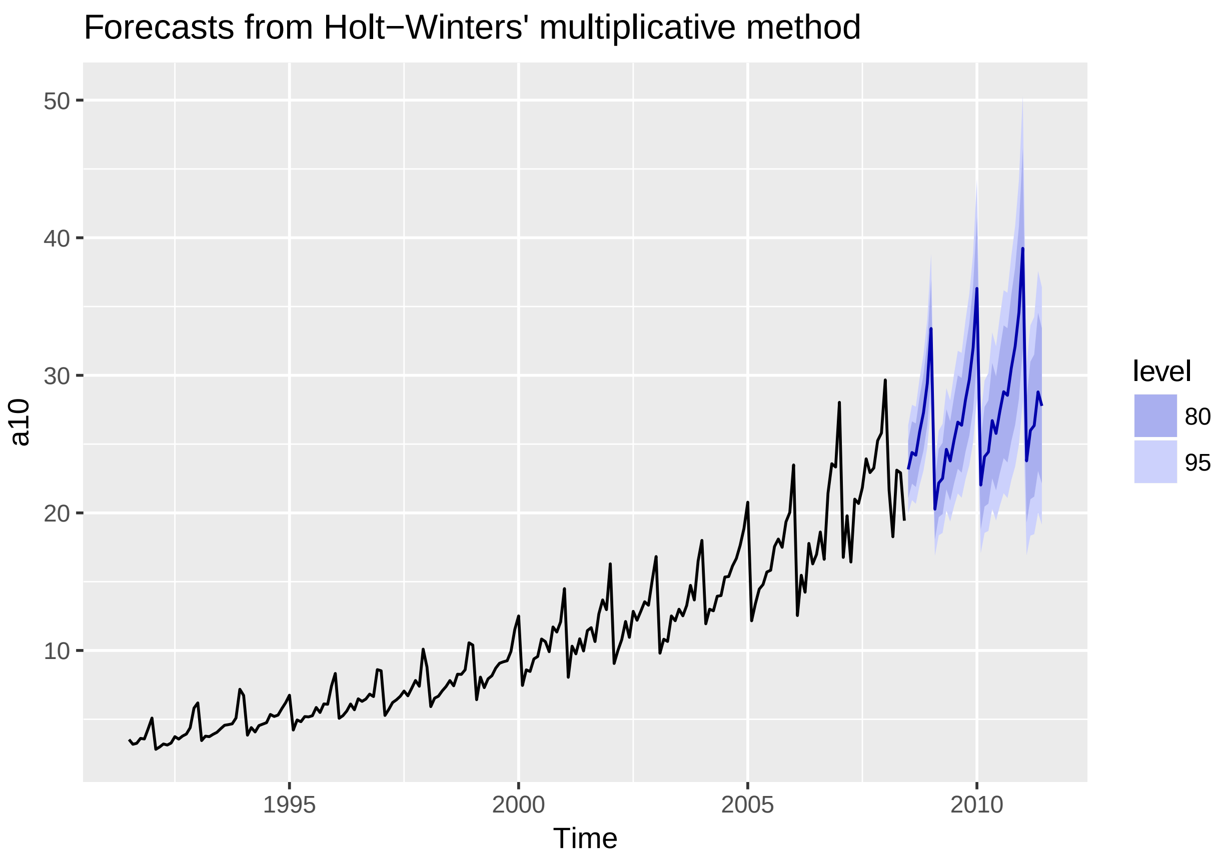


```r
# Produce 3 year forecasts
fc <- hw(a10, seasonal = "multiplicative", h = 36)

# Check if residuals look like white noise
checkresiduals(fc)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from Holt-Winters' multiplicative method
## Q* = 75.764, df = 8, p-value = 3.467e-13
##
## Model df: 16.   Total lags used: 24
```

```
whitenoise <- FALSE

# Plot forecasts
autoplot(fc)
```



The forecasts might still provide useful information even with residuals that fail the white noise test.

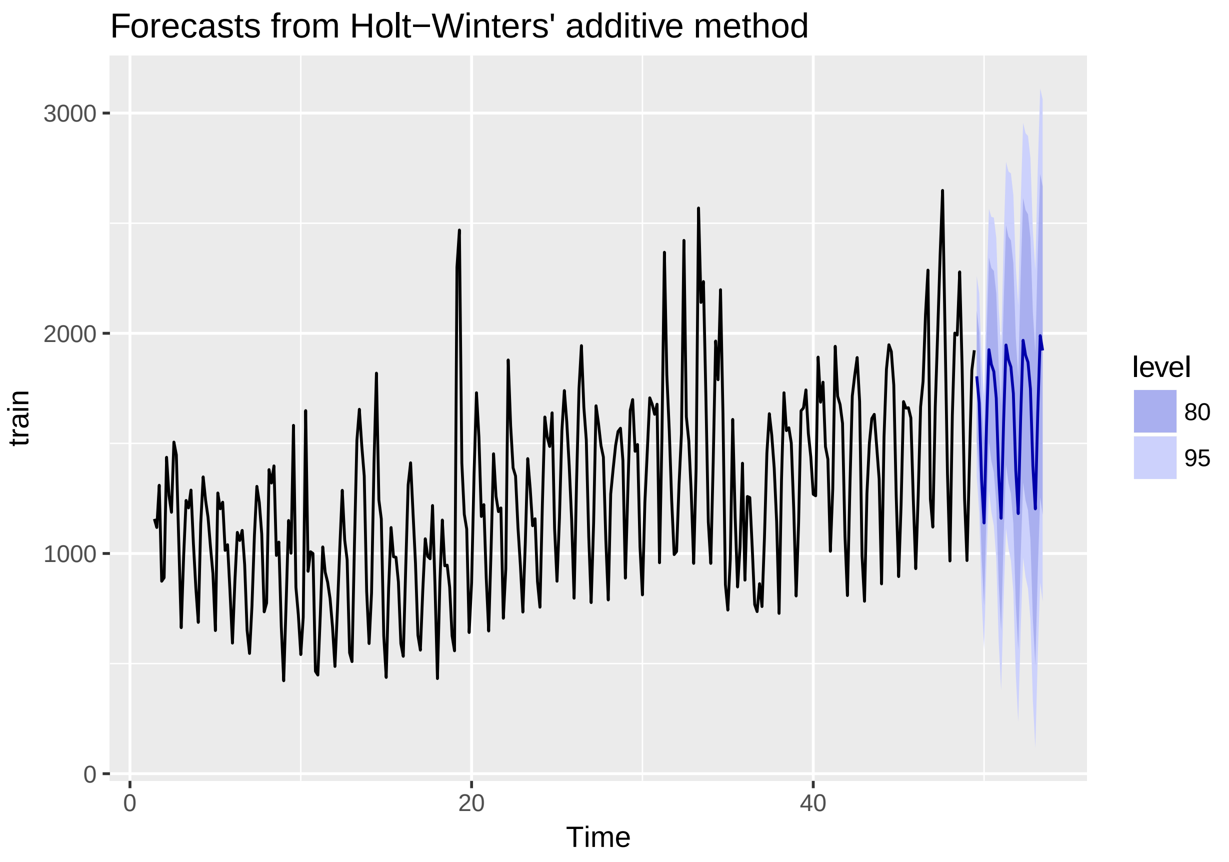
```r
# Create training data with subset()
train <- subset(hyndsight, end = length(hyndsight) - 28)
# subset.ts(hyndsight, end = length(hyndsight) - 28)
# train <- subset(hyndsight, end = 361)

# Holt-Winters additive forecasts as fchw
fchw <- hw(train, seasonal = "additive", h = 28)

# Seasonal naive forecasts as fcsn
fcsn <- snaive(train, h = 28)

# Find better forecasts with accuracy()
accuracy(fchw, hyndsight)
```

```
##                     ME     RMSE      MAE       MPE    MAPE      MASE
## Training set -3.976241 228.2440 165.0244 -2.407211 13.9955 0.7492131
## Test set     -3.999460 201.7656 152.9584 -3.218292 10.5558 0.6944332
##                   ACF1 Theil's U
## Training set 0.1900853        NA
## Test set     0.3013328 0.4868701
```

```r
accuracy(fcsn, hyndsight)
```

```
##                 ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 10.50 310.3282 220.2636 -2.1239387 18.01077 1.0000000
## Test set      0.25 202.7610 160.4643 -0.6888732 10.25880 0.7285101
##                     ACF1 Theil's U
## Training set 0.4255730        NA
## Test set     0.3089795  0.450266
```

```
# Plot the better forecasts
autoplot(fchw)
```



The **trend component** of a time series can be None (N), Additive (A), or Additive Damped (A_d). The **seasonal component** can be None (N), Additive (A), or Multiplicative (M).

State space models for exponential smoothing

All of the exponential smoothing methods can be written in the form of **"innovations state space models"**.

- Trend = {N, A, A_d}
- Seasonal = {N, A, M}
- Error = {A, M} = Additive or Multiplicative errors
- ETS models: Error, Trend, Seasonal

A combination of Trend and Seasonal states results in **3 X 3 = 9 possible exponential smoothing methods**. If we include the Error states, we have **9 X 2 = 18 possible state space models**. Multiplicative errors means that the noise increases with the level of the series just as multiplicative seasonality means that the seasonal fluctuation increases with the level of the series. These are known as **ETS models: Error, Trend, and Seasonal**.

ETS Models

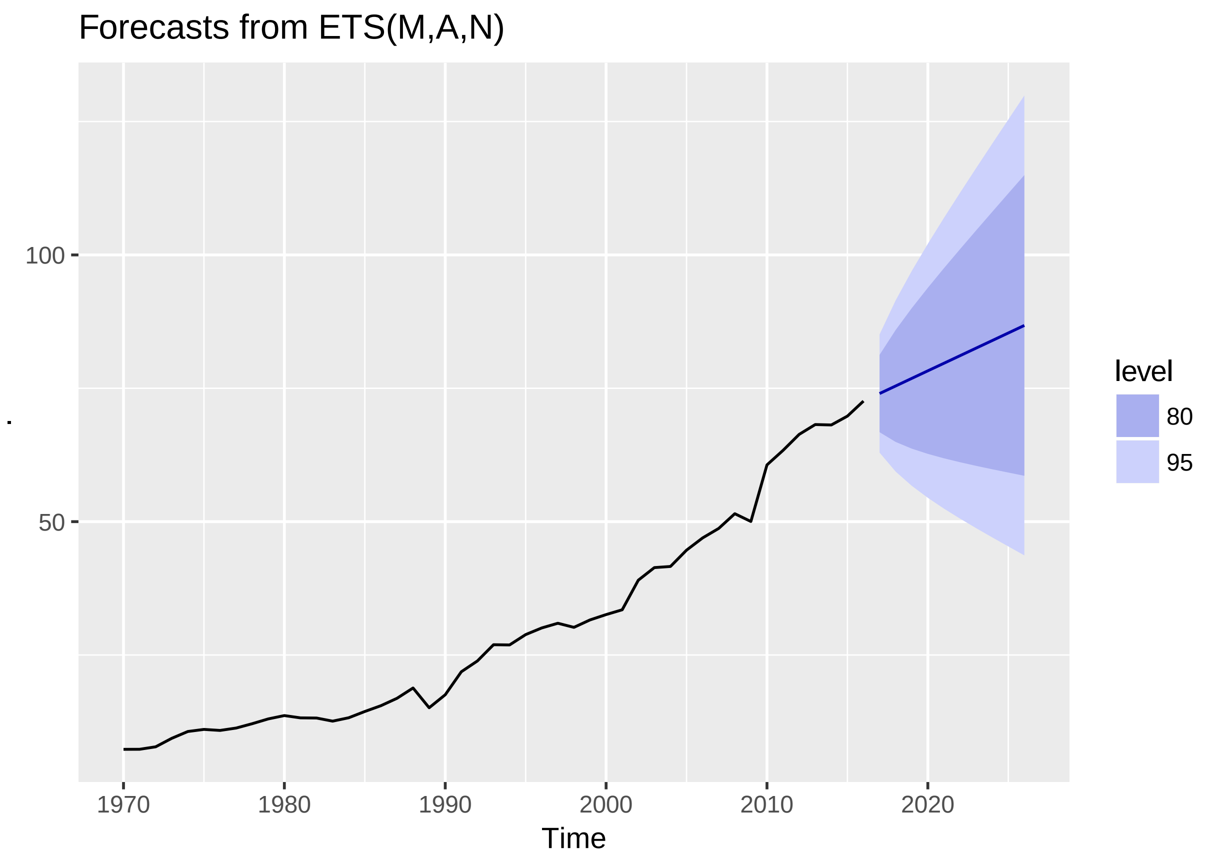
The advantage of thinking in this way is that we can then use Maximum Likelihood Estimation (MLE) to optimize the parameters and you have a way of generating prediction intervals for all models. Most importantly, we now have a way of selecting the best model for a particular time series. So rather than looking at graphs and guessing what might work in each case, we can now select an exponential smoothing state based model to use for each time series. You can do this by minimizing a *biased corrected version* of Akaike's Information Criterion (AIC_c), which is named after Japanese Statistician, **Hirotugu Akaike**. This is roughly the same as using time series cross validation especially on long time series, but it is much faster. The `ets()` function does all the work for us.

```r
ets(ausair)
```

```
## ETS(M,A,N)
##
## Call:
##  ets(y = ausair)
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.0269
##
##   Initial states:
##     l = 6.5431
##     b = 0.7393
##
##   sigma:  0.0764
##
##      AIC     AICc      BIC
## 241.6910 243.1544 250.9417
```

In the example above, our time series is a `ts` object `ausair`, which contains the total annual air passengers (in millions) in Australia from 1970 – 2015. The best model is a M, A, N model, which means the error is multiplicative, the trend is additive, and there is no seasonality. The parameters are estimated in much the same way as when we used the holt method except that it maximizes the likelihood rather than minimizing the sum of squared errors (SSE). Apart from the way the parameters are chosen, this model is equivalent to holt's linear method. What is different is that ETS does not compute the forecast for us. It returns a model. To produce forecasts, you need to pass that model to the forecast function.

```r
ausair %>% ets() %>% forecast() %>% autoplot()
```



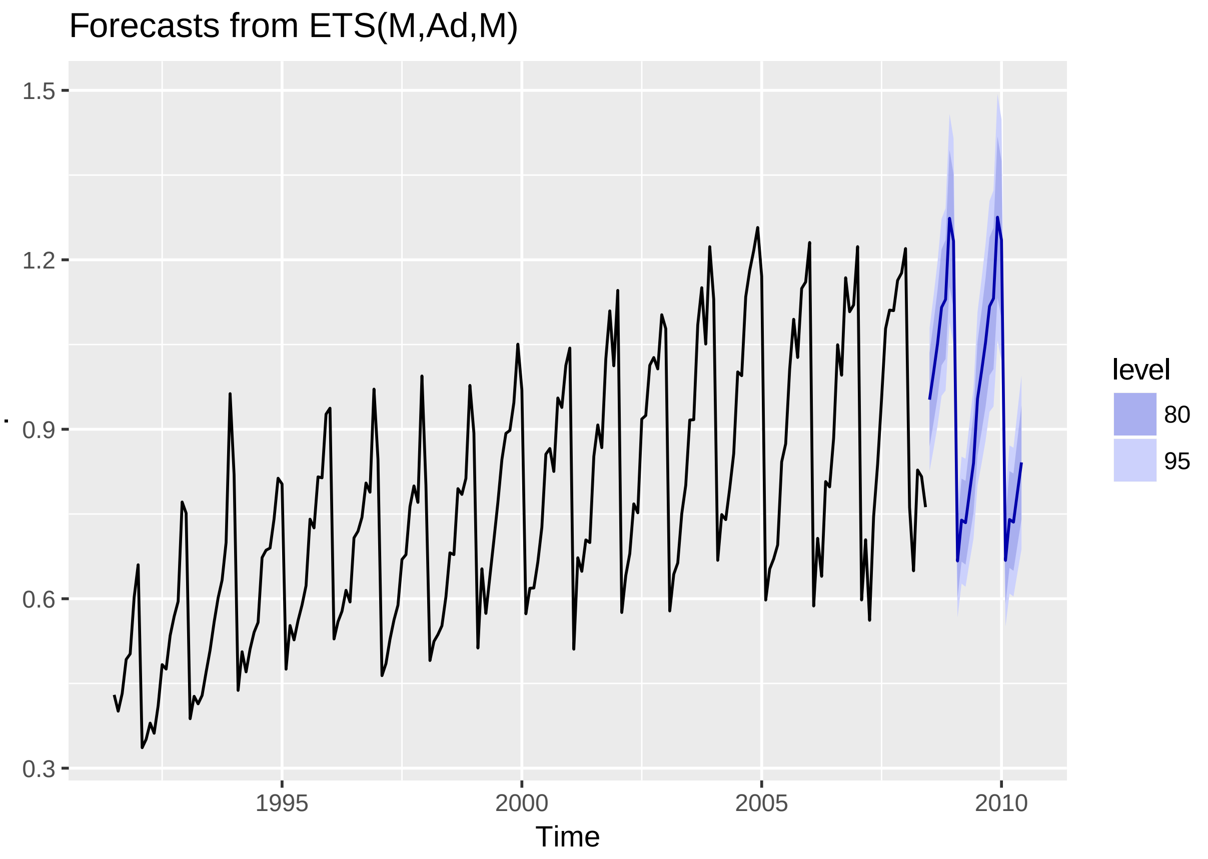
The linear trend is clearly seen. The multiplicative errors means that the width of the prediction interval grows more quickly than if an additive error had been chosen. Let us look at a seasonal example. The h02 time series contains monthly sales of cortecosteroid drug in Austrailia from July 1991 – June 2008.

```r
ets(h02)
```

```
## ETS(M,Ad,M)
##
## Call:
##  ets(y = h02)
##
##   Smoothing parameters:
##     alpha = 0.1953
##     beta  = 1e-04
##     gamma = 1e-04
##     phi   = 0.9798
##
##   Initial states:
##     l = 0.3945
##     b = 0.0085
##     s=0.874 0.8197 0.7644 0.7693 0.6941 1.2838
##            1.326 1.1765 1.1621 1.0955 1.0422 0.9924
##
##   sigma:  0.0678
##
##        AIC       AICc        BIC
## -122.90601 -119.20871  -63.17985
```

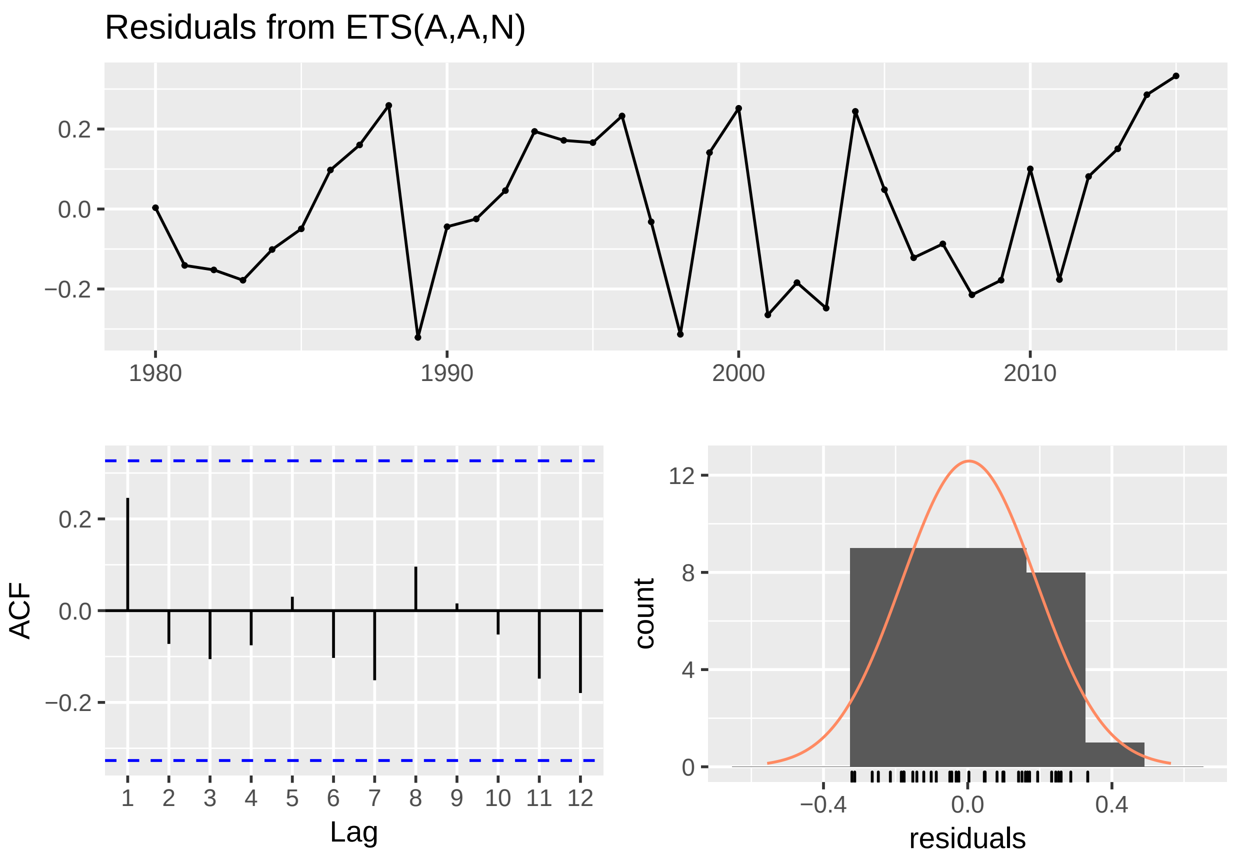
In this case, the `ets()` function has selected a model with a multiplicative error, a damped additive trend, and a multiplicative seasonality. The model returns the result of four smoothing parameters ($\alpha, \beta, \gamma, \phi$) and initial state values for level, slope and 11 seasonal values. The 12th seasonal value was computed so that the seasonal values can add up to one.

```
h02 %>% ets() %>% forecast() %>% autoplot()
```



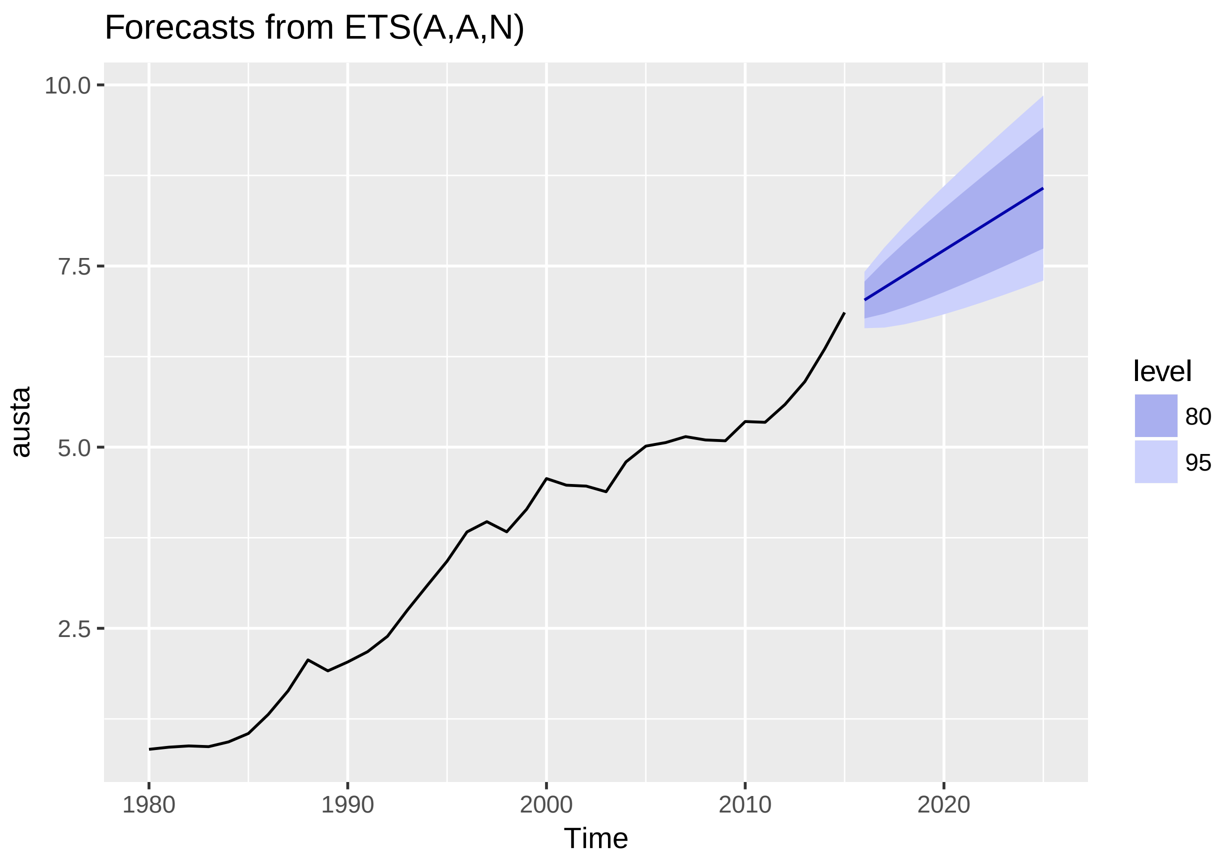
The advantage of using the `ets()` function is that the type of model is chosen for us. The `ets()` function offers a convenient way to forecast a time series that has both trend and seasonality.

```
# Fit ETS model to austa in fitaus
fitaus <- ets(austa)

# Check residuals
checkresiduals(fitaus)
```

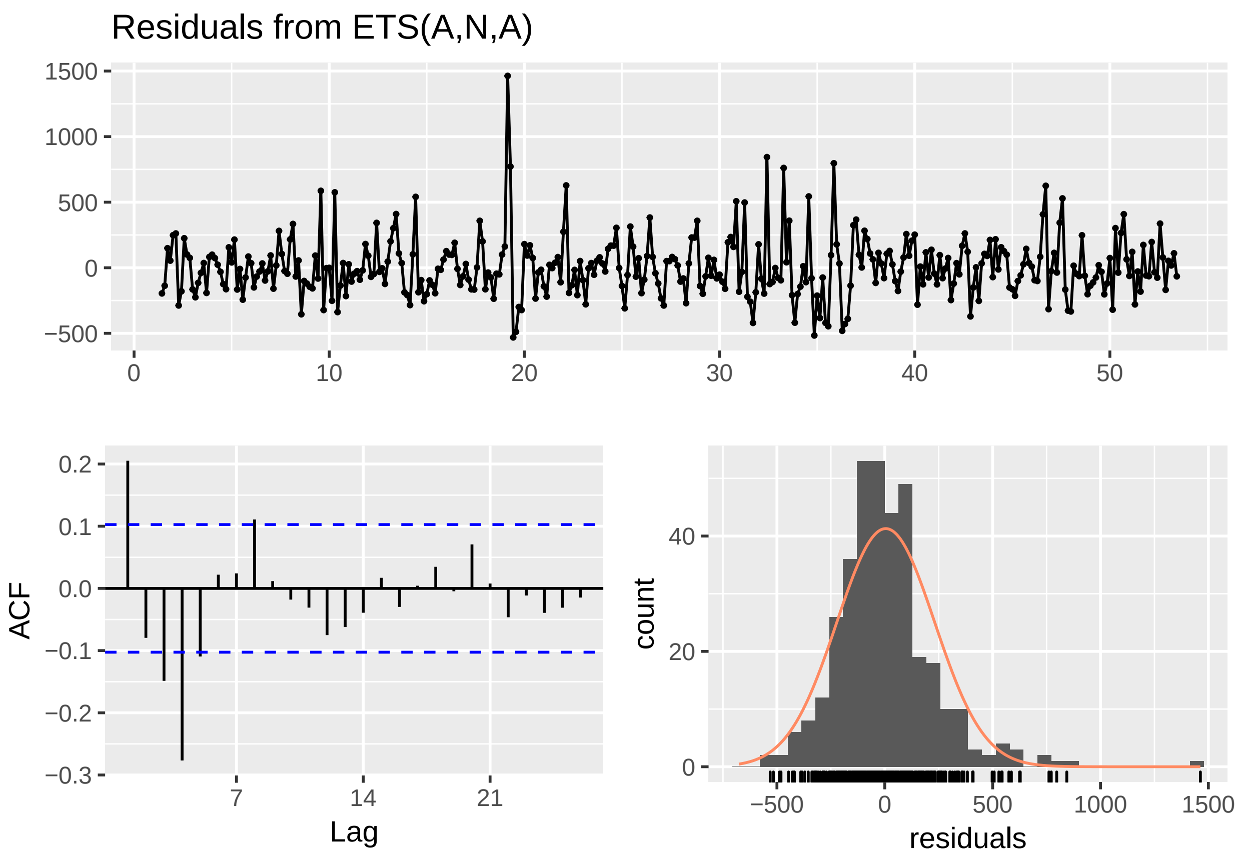


```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,A,N)
## Q* = 5.493, df = 6, p-value = 0.4823
##
## Model df: 4.   Total lags used: 10
```

```
# Plot forecasts
autoplot(forecast(fitaus))
```

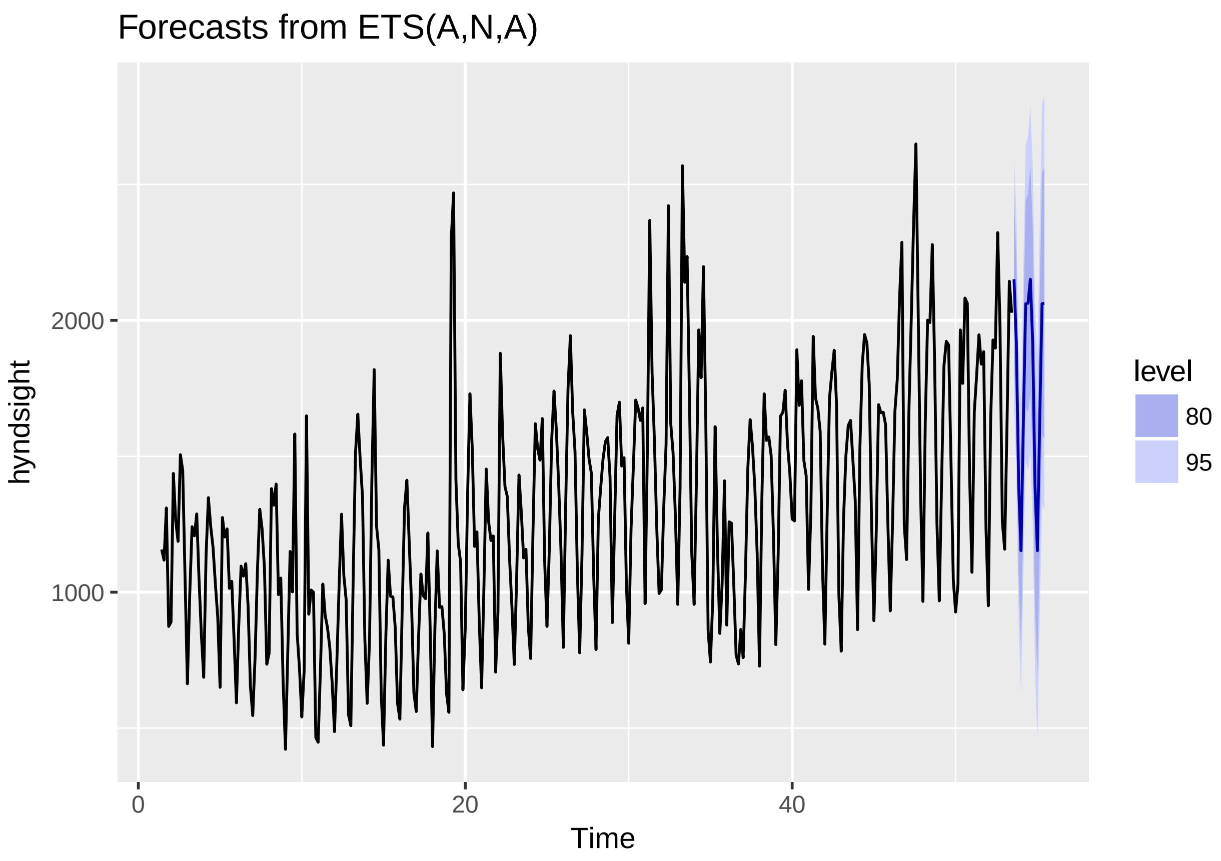


```
# fitaus %>% forecast() %>% autoplot() (can be achieved using the magrittr (%>%) function)

# Repeat for hyndsight data in fiths
fiths <- ets(hyndsight)
checkresiduals(fiths)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,N,A)
## Q* = 68.616, df = 5, p-value = 1.988e-13
##
## Model df: 9.   Total lags used: 14
```

```
autoplot(forecast(fiths))
```



```
# Which model(s) fails test? (TRUE or FALSE)
fitausfail <- FALSE
fithsfail <- TRUE
```

Remember, a model passes the Ljung-Box test when the p-value is greater than 0.05. Observe the results of this test by running the checkresiduals() function for each series in your console.

ETS v. naive

Here, you will compare ETS forecasts against seasonal naive forecasting for 20 years of cement, which contains quarterly cement production using time series cross-validation for 4 steps ahead. Because this takes a while to run, a shortened version of the cement series will be available in your workspace.

The second argument for tsCV() must return a forecast object, so you need a function to fit a model and return forecasts

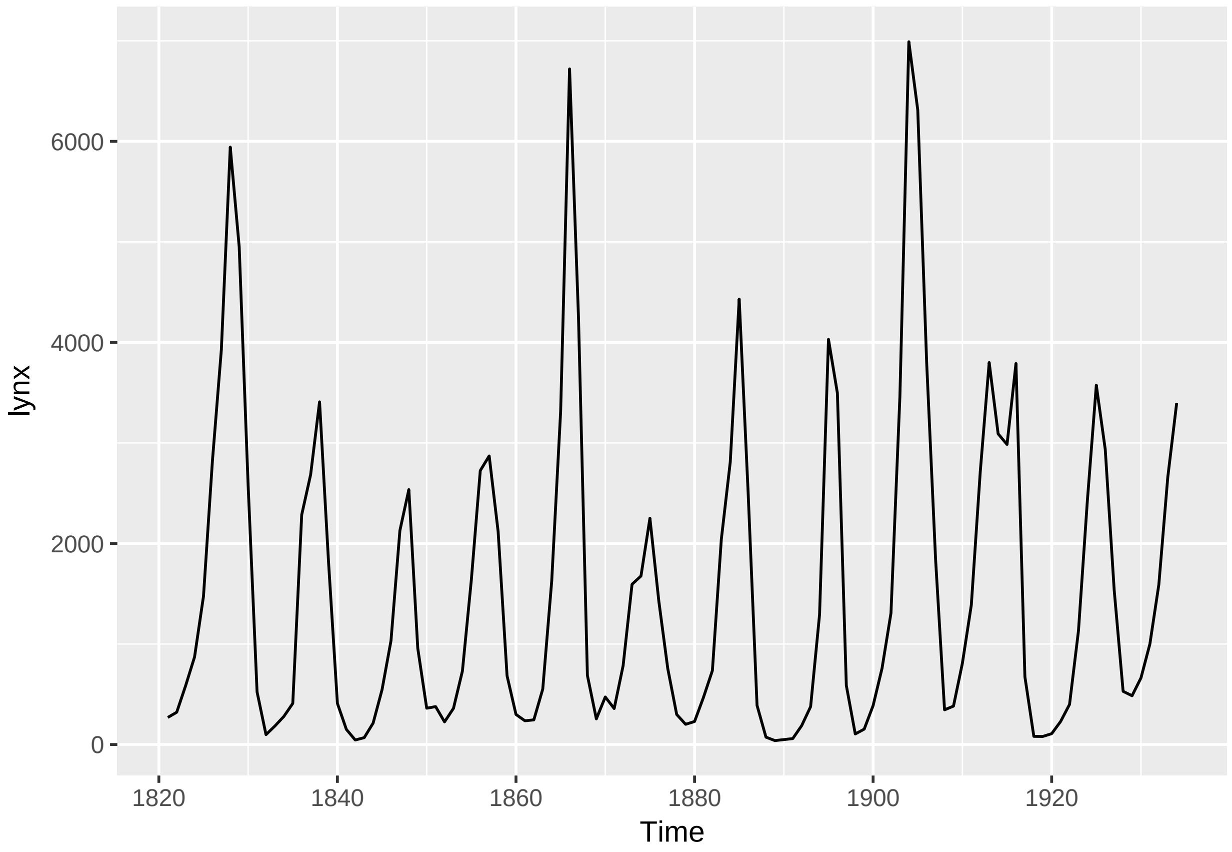
```
# # Function to return ETS forecasts
# fets <- function(y, h) {
#   forecast(ets(y), h = h)
# }
#
# # Apply tsCV() for both methods
# e1 <- tsCV(cement, forecastfunction = fets, h = 4)
# e2 <- tsCV(cement, forecastfunction = snaive, h = 4)
#
# # Compute MSE of resulting errors (watch out for missing values)
# mean(e1^2, na.rm = TRUE)
# mean(e2^2, na.rm = TRUE)
```

```
#
# # Copy the best forecast MSE
# bestmse <- mean(e2^2, na.rm = TRUE)
```

Nice! Complex isn't always better.

When does ETS fail?

```
# Plot the lynx series
autoplot(lynx)
```

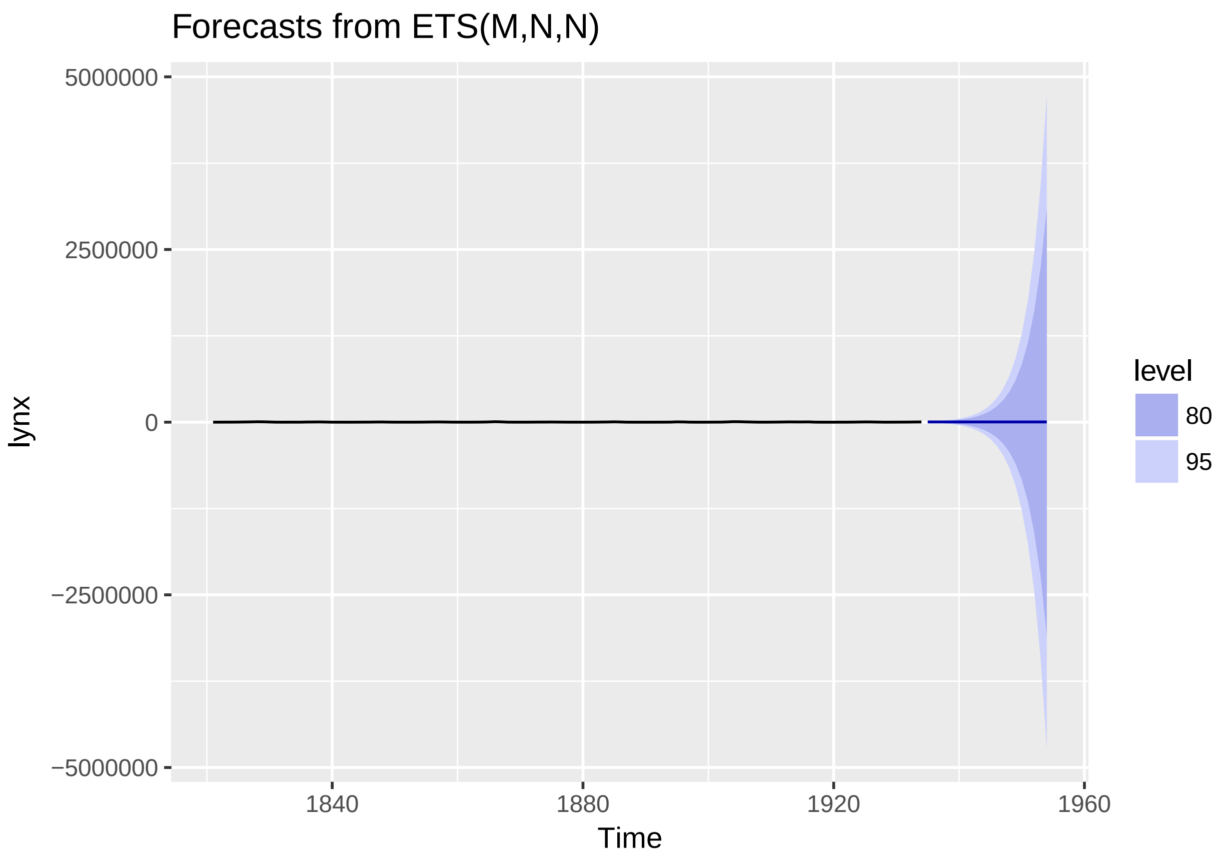


```
# Use ets() to model the lynx series
fit <- ets(lynx)

# Use summary() to look at model and parameters
summary(fit)
```

```
## ETS(M,N,N)
##
## Call:
##  ets(y = lynx)
##
##   Smoothing parameters:
##     alpha = 0.9999
##
##   Initial states:
```

```
##     l = 2372.8047
##
##   sigma:  0.9637
##
##      AIC     AICc      BIC
## 2058.138 2058.356 2066.346
##
## Training set error measures:
##                      ME     RMSE      MAE       MPE     MAPE     MASE
## Training set 8.975647 1198.452 842.0649 -52.12968 101.3686 1.013488
##                     ACF1
## Training set 0.3677583
```

```
# Plot 20-year forecasts of the lynx series
fit %>% forecast(h = 20) %>% autoplot()
```


Forecasts from ETS(M,N,N)
lynx
Time
level