Forecasting II

DS 6030 | Fall 2023

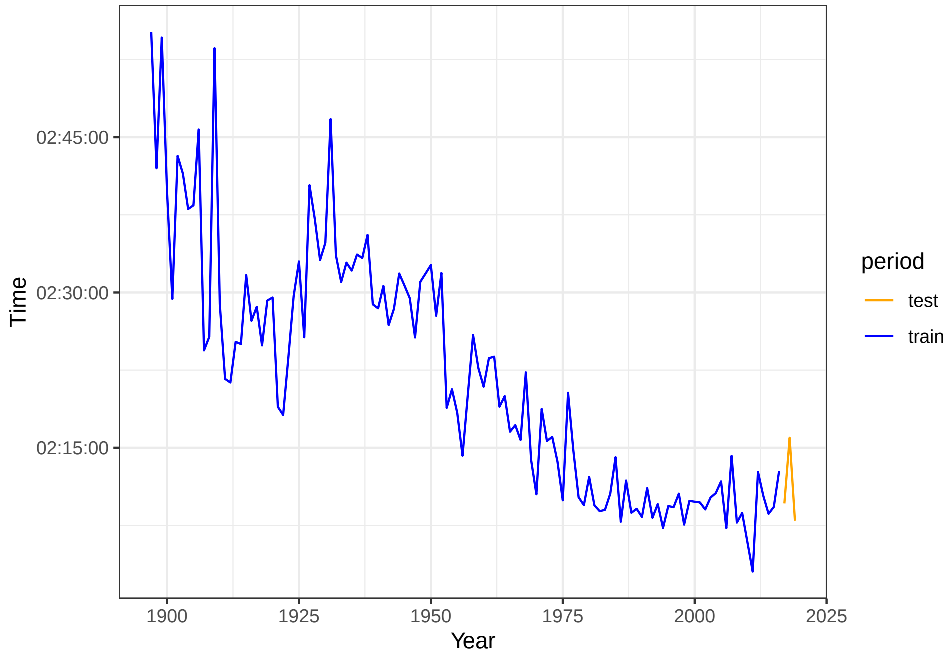
forecasting.pdf

Contents

1 Time Series Data

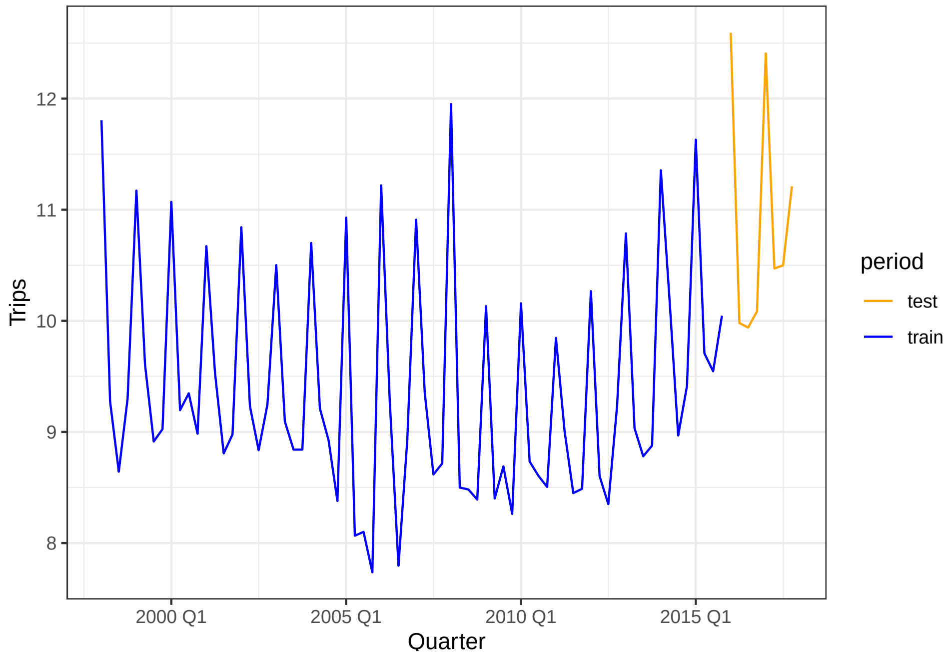
1.1 Boston Marathon Times

```
boston = boston_marathon %>% filter(Event == "Men's open division") %>%
  update_tsibble(index = Year, key = NULL) %>%
  mutate(period = ifelse(Year <= 2016, "train", "test"))

# boston %>% autoplot(Time)
boston %>%
  ggplot(aes(Year, Time, color = period)) + geom_line() +
  scale_color_manual(values = c(train = "blue", test = "orange"))
```



1.2 Australian domestic overnight trips

```
aus_holidays = tourism %>%
  filter(Purpose == "Holiday")  %>%
  summarise(Trips = sum(Trips)/1e3) %>%
  mutate(period = ifelse(Quarter < yearquarter("2016-01-01"), "train", "test" ))

# aus_holidays %>% autoplot(Trips)
aus_holidays %>%
  ggplot(aes(Quarter, Trips, color = period)) + geom_line() +
  scale_color_manual(values = c(train = "blue", test = "orange"))
```



2 Exponential Smoothing

Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight. This framework generates reliable forecasts quickly and for a wide range of time series, which is a great advantage and of major importance to applications in industry.

- Hyndman and Athanasopoulos https://otexts.com/fpp3/expsmooth.html

2.1 Simple Exponential Smoothing (SES)

The SES model is a simple one-parameter[1] model that creates a *flat* forecast. That is,

$$\text{Forecast Equation: } \hat{y}(t+h \mid t) = l_t$$

for all $h > 0$. In this equation, l_t is the "level".

The level is estimated by exponential smoothing using model parameter α.

$$l_t = \alpha y_t + (1-\alpha) l_{t-1}$$

There are a few ways to re-write this equation that can shed some light on how it works.

1. Innovations

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha) l_{t-1} \\ &= \alpha y_t + l_{t-1} - \alpha l_{t-1} \\ &= l_{t-1} + \alpha (y_t - \hat{y}_{t|t-1}) \end{aligned}$$

[1] Technically, there are two parameters if the initial level needs to be estimated.

2. Smoothed 1-step ahead forecasts

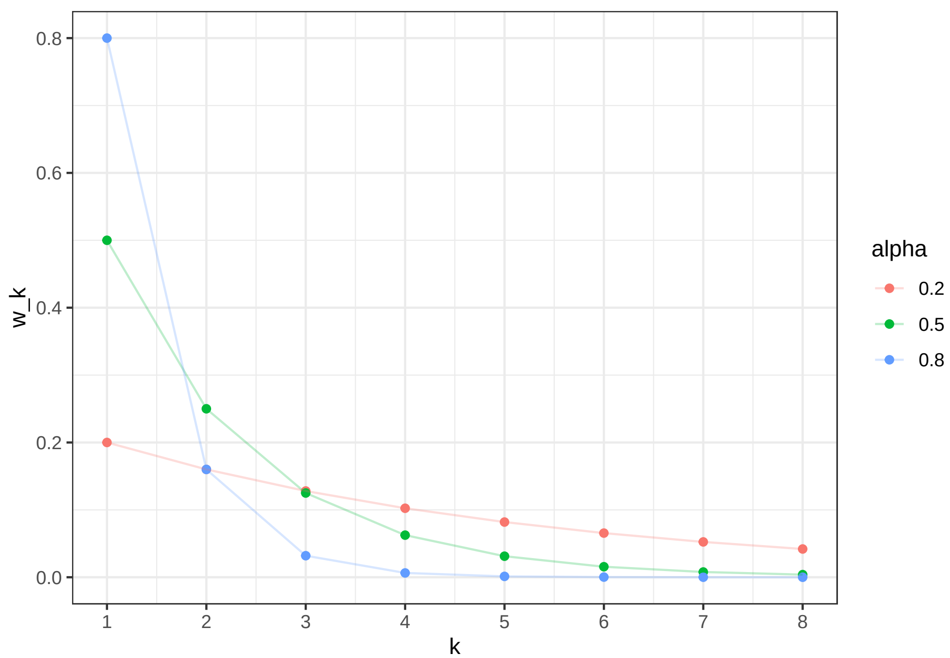
$$
\begin{aligned}
l_t &= \alpha y_t + (1-\alpha) l_{t-1} \\
&= \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}
\end{aligned}
$$

3. Autoregressive model

$$
\begin{aligned}
l_t &= \alpha y_t + (1-\alpha) l_{t-1} \\
&= \alpha y_t + (1-\alpha)[\alpha y_{t-1} + (1-\alpha) l_{t-2}] \\
&= \alpha y_t + \alpha(1-\alpha) y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \ldots + \alpha(1-\alpha)^t y_0 \\
&= \sum_{k=1}^{t-1} w_k y_{t-k} + w_t y_0
\end{aligned}
$$

where $w_k = \alpha(1-\alpha)^{k-1}$. If $0 \leq \alpha \leq 1$, the weights are equivalent to a geometric pmf.

k	alpha = 0.2	alpha = 0.5	alpha = 0.8
1	0.200	0.500	0.800
2	0.160	0.250	0.160
3	0.128	0.125	0.032
4	0.102	0.062	0.006
5	0.082	0.031	0.001
6	0.066	0.016	0.000
7	0.052	0.008	0.000
8	0.042	0.004	0.000

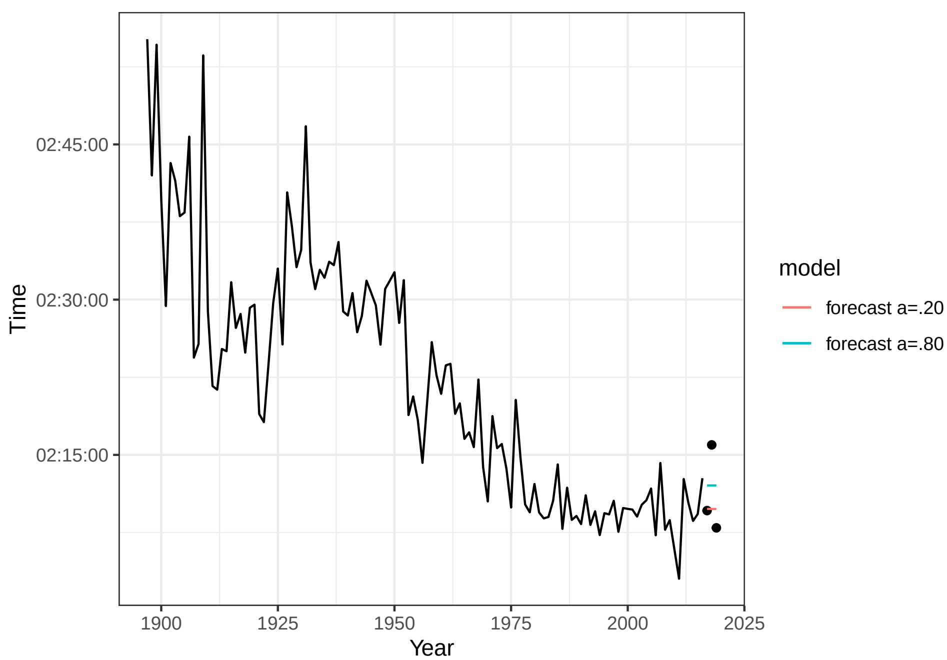


Although this reduces to an Autoregressive model, it is simple in the sense that the edf is only 1-2 (equivalent to an AR(2)).

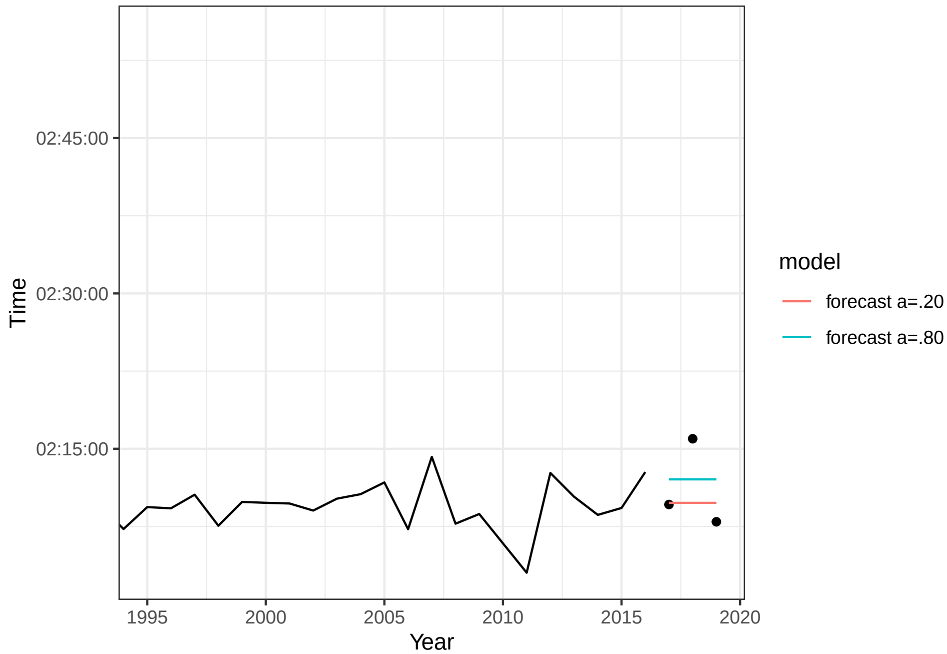
```
ANN_20 = boston %>% filter(period == "train") %>%
  model(
    ANN = ETS(Time ~ trend("N", alpha = .2) + season("N") + error("A"))
  )
fcast_ANN_20 = forecast(ANN_20, h = 3)
```

```
ANN_80 = boston %>% filter(period == "train") %>%
  model(
    ANN = ETS(Time ~ trend("N", alpha = .8) + season("N") + error("A"))
  )
fcast_ANN_80 = forecast(ANN_80, h = 3)
```

```
boston %>%
  ggplot(aes(Year, Time)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_point(data = . %>% filter(period == "test"), color = "black") +
  geom_line(data = fcast_ANN_20, aes(y = .mean, color = "forecast a=.20") ) +
  geom_line(data = fcast_ANN_80, aes(y = .mean, color = "forecast a=.80") ) +
  labs(color = "model")
```




```
last_plot() + coord_cartesian(xlim = c(1995, NA))
```



The big idea with SES is that forecasts are constant; it cannot accommodate trends or seasonality.

2.2 Double Exponential (DES) / Holt

Holt's linear trend method (or double exponential smoothing) allows linear trends.

$$\text{Forecast Equation: } \hat{y}(t+h \mid t) = l_t + hb_t$$

for all $h > 0$. In this equation, l_t is the "level" and b_t is the trend.

The level is estimated by exponential smoothing using model parameter α.

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha)[l_{t-1} + b_{t-1}] \\ &= \alpha y_t + (1-\alpha)\hat{y}_{t|t-1} \end{aligned}$$

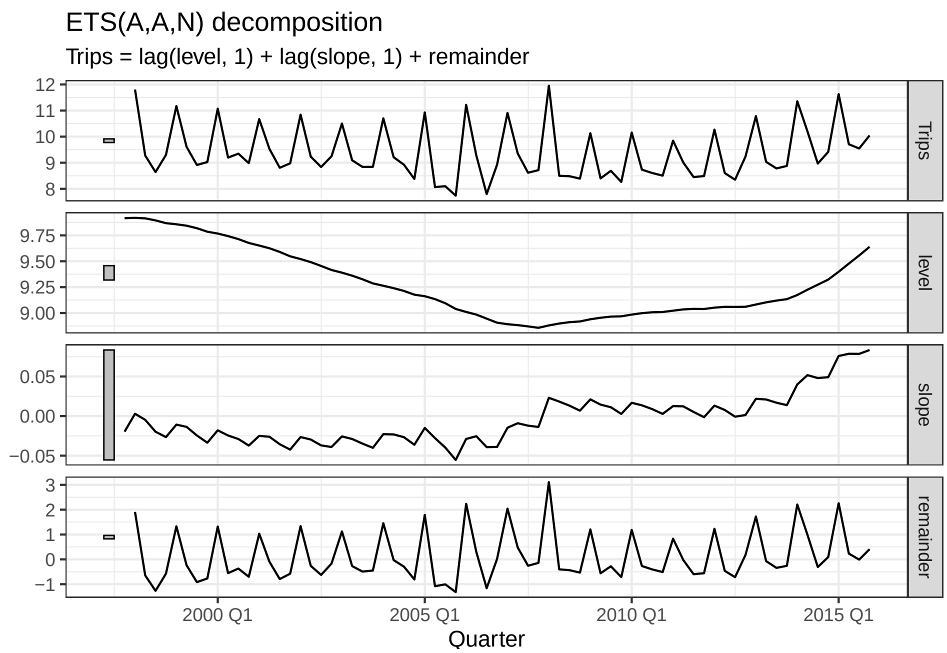
The slope is estimated by exponential smoothing using model parameter β.

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

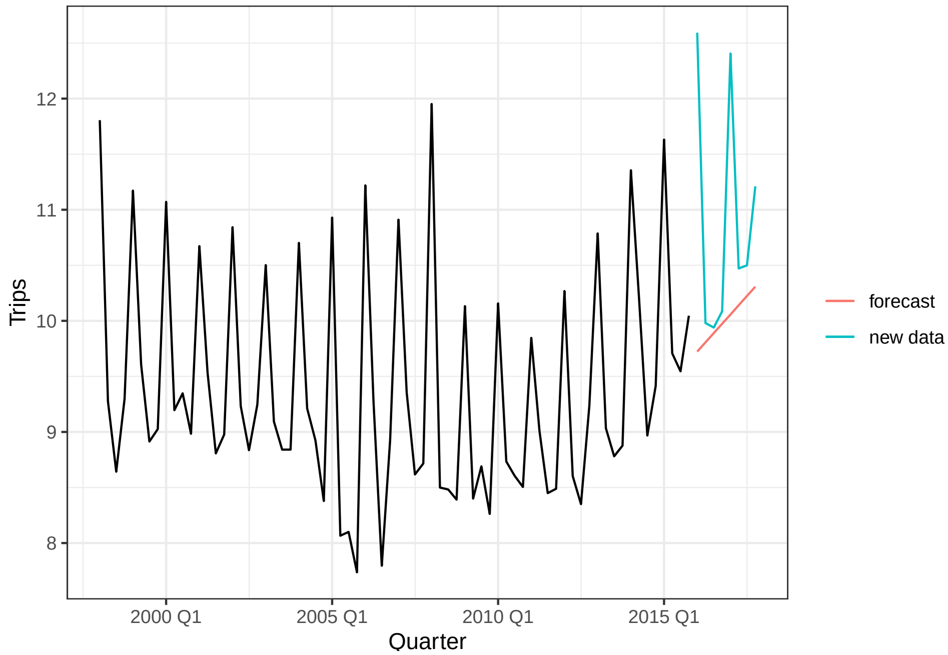
```
AAN = aus_holidays %>% filter(period == "train") %>%
  model(
    AAN = ETS(Trips ~ trend("A") + season("N") + error("A"))
  )

report(AAN)
#> Series: Trips
#> Model: ETS(A,A,N)
#>   Smoothing parameters:
#>     alpha = 0.01187
#>     beta  = 0.01187
#>
#>   Initial states:
```

```
#>    l[0]      b[0]
#>   9.916 -0.01971
#>
#>    sigma^2:  1.052
#>
#>    AIC   AICc    BIC
#> 317.5 318.4 328.9
```

```
components(AAN) %>% autoplot()
```

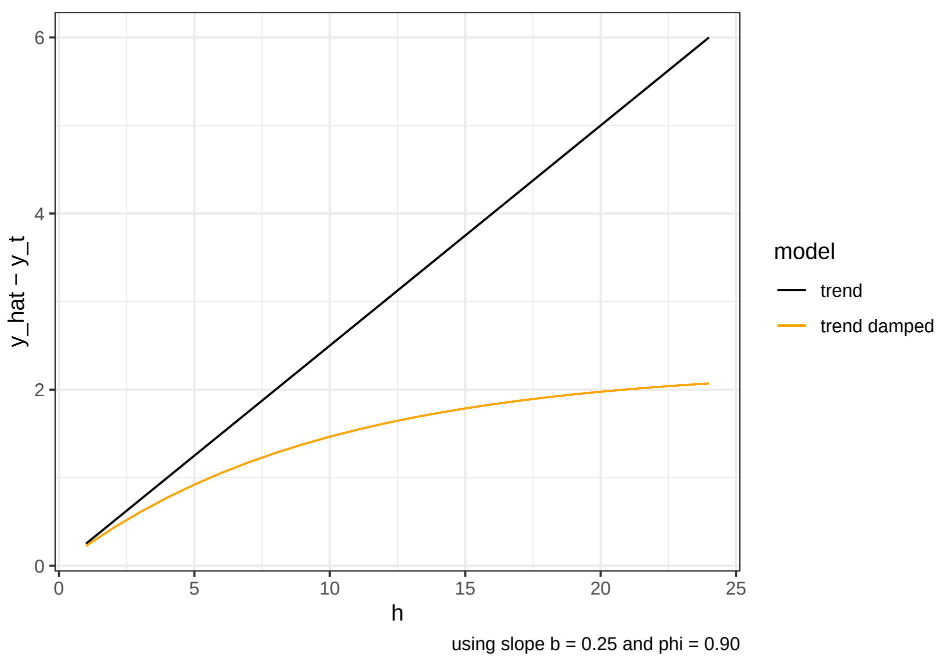



```
fcast_AAN = forecast(AAN, h = 8)
aus_holidays %>%
  ggplot(aes(Quarter, Trips)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_line(data = . %>% filter(period == "test"), aes(color = "new data")) +
  geom_line(data = fcast_AAN, aes(y = .mean, color = "forecast") )  +
  labs(color = "")
```



2.2.1 Dampening

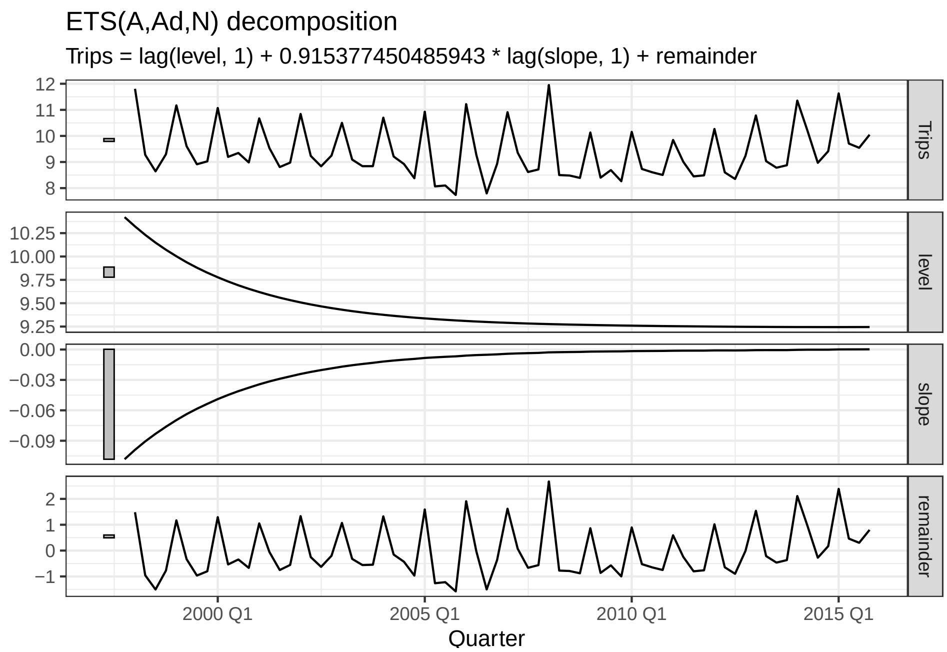
Forecasting with trends for long horizons can lead to very unreasonable predictions. Instead of using a forecasted trend of hb_t, the Holt's damped method uses a trend of $(\phi + \phi^2 + \ldots + \phi^h)b_t$ where $\phi \leq 1$ ensuring that $(\phi + \phi^2 + \ldots + \phi^h) < h$.



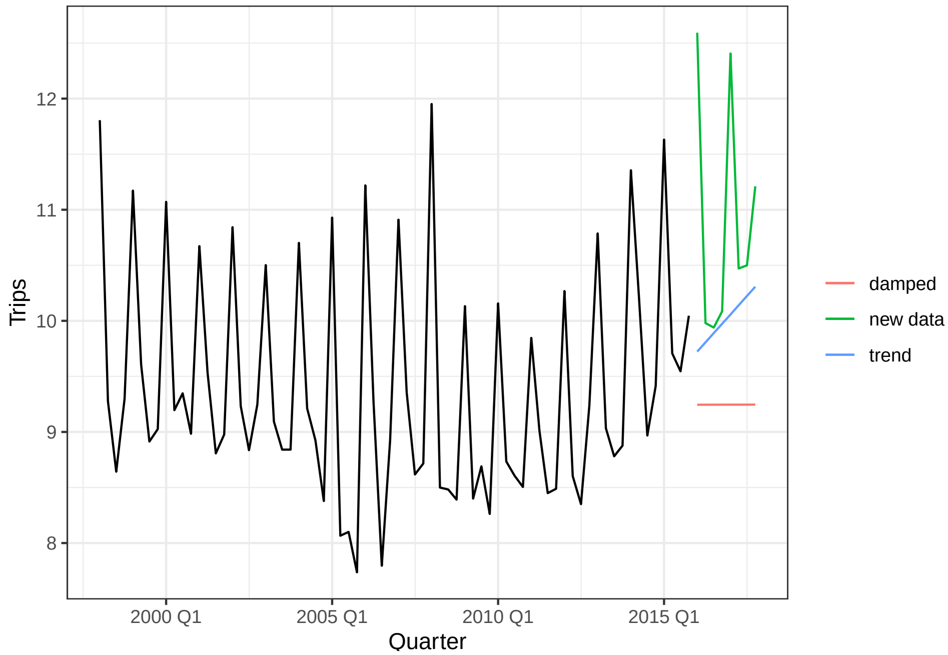

```
AAdN = aus_holidays %>% filter(period == "train") %>%
  model(
    AAdN = ETS(Trips ~ trend("Ad") + season("N") + error("A"))
  )

report(AAdN)
```

```
#> Series: Trips
#> Model: ETS(A,Ad,N)
#>   Smoothing parameters:
#>     alpha = 1e-04
#>     beta  = 1e-04
#>     phi   = 0.9154
#>
#>   Initial states:
#>   l[0]    b[0]
#>  10.42 -0.1083
#>
#>   sigma^2:  1.059
#>
#>   AIC  AICc   BIC
#> 318.9 320.2 332.5
```

```
components(AAdN) %>% autoplot()
```




```
fcast_AAdN = forecast(AAdN, h = 8)
aus_holidays %>%
  ggplot(aes(Quarter, Trips)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_line(data = . %>% filter(period == "test"), aes(color = "new data")) +
  geom_line(data = fcast_AAN, aes(y = .mean, color = "trend") )  +
  geom_line(data = fcast_AAdN, aes(y = .mean, color = "damped") )  +
  labs(color = "")
```



2.3 Triple Exponential Smoothing / Holt-Winters

Holt-Winters method allows both a trend and seasonality.

$$\text{Forecast Equation: } \hat{y}(t+h \mid t) = l_t + hb_t + s_t(h)$$

for all $h > 0$. In this equation, l_t is the "level" and b_t is the trend, and $s_t(h)$ is the seasonality at forecast horizon h. If m is the period (e.g., $m = 12$ for monthly data, $m = 4$ for quarterly data), then

$$s_t(h) = \begin{cases} s_{t+h-m} & h \leq m \\ s_{t+h-2m} & m < h \leq 2m \\ s_{t+h-3m} & 2m < h \leq 3m \\ \dots & \dots \end{cases}$$

which ensures that forecasts only use the data at the time the forecast is made.

The level equation averages the *deseasoned* level with the previous level and trend.

$$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)[l_{t-1} + b_{t-1}]$$

The slope is estimated by exponential smoothing using model parameter β (same as was done in DES).

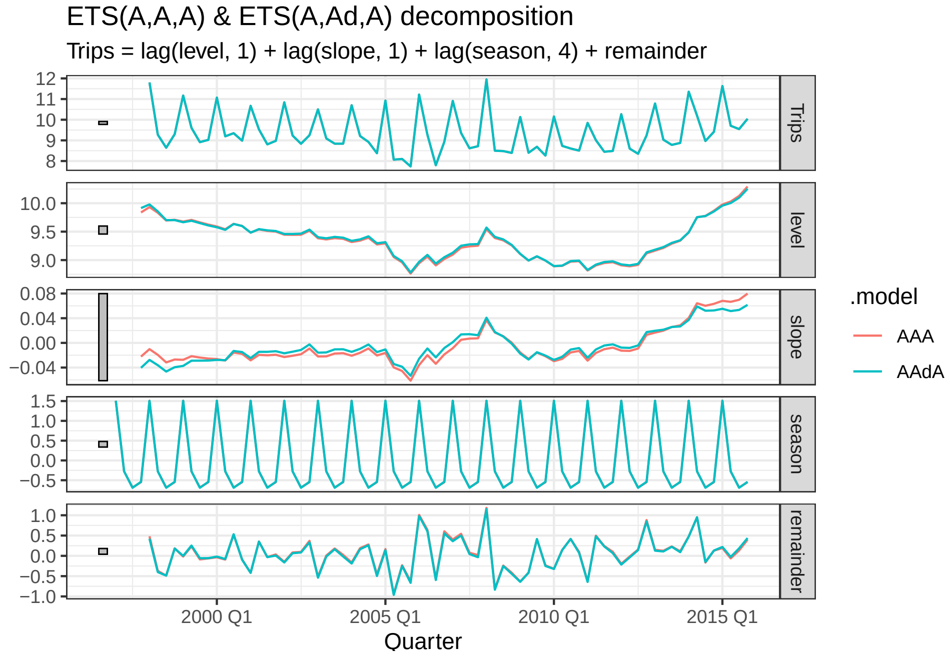
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

The seasonality term uses parameter γ to mix the *detrended* residuals with the previous seasonality at lag m.

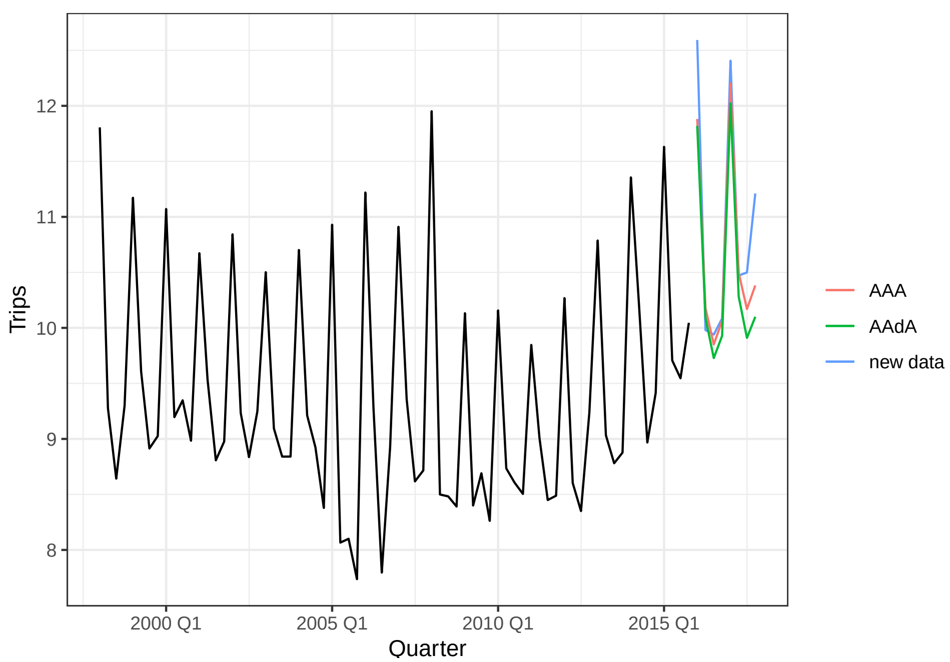
$$s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$$

```
fits <- aus_holidays %>% filter(period == "train") %>%
  model(
    AAA = ETS(Trips ~ trend("A") + season("A") + error("A")),
    AAdA = ETS(Trips ~ trend("Ad") + season("A") + error("A")),
  )
fits %>% select(AAA) %>% report()
#> Series: Trips
#> Model: ETS(A,A,A)
#>   Smoothing parameters:
#>     alpha = 0.2401
#>     beta  = 0.0251
#>     gamma = 0.0001001
#>
#>   Initial states:
#>   l[0]     b[0]    s[0]   s[-1]   s[-2] s[-3]
#>  9.838 -0.02223 -0.5503 -0.6826 -0.2751 1.508
#>
#>   sigma^2:  0.195
#>
#>   AIC  AICc   BIC
#> 199.7 202.6 220.2
fits %>% select(AAdA) %>% report()
#> Series: Trips
#> Model: ETS(A,Ad,A)
#>   Smoothing parameters:
#>     alpha = 0.2424
#>     beta  = 0.02519
#>     gamma = 1e-04
#>     phi   = 0.9457
#>
#>   Initial states:
#>   l[0]     b[0]    s[0]  s[-1]   s[-2] s[-3]
#>  9.914 -0.04045 -0.5401 -0.691 -0.2763 1.507
#>
#>   sigma^2:  0.1964
#>
#>   AIC  AICc   BIC
#> 201.1 204.7 223.9
```

```
components(fits) %>% autoplot()
```




```
fcast_fits = forecast(fits, h = 8)
aus_holidays %>%
  ggplot(aes(Quarter, Trips)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_line(data = . %>% filter(period == "test"), aes(color = "new data")) +
  geom_line(data = fcast_fits, aes(y = .mean, color = .model) ) +
  labs(color = "")
```



2.4 Multiplicative Seasonality

Instead of the seasonality component entering the model additively, it can be a multiplicative factor.

The Holt-Winters multiplicative model using this form

$$\text{Forecast Equation: } \hat{y}(t+h \mid t) = (l_t + hb_t)s_t(h)$$

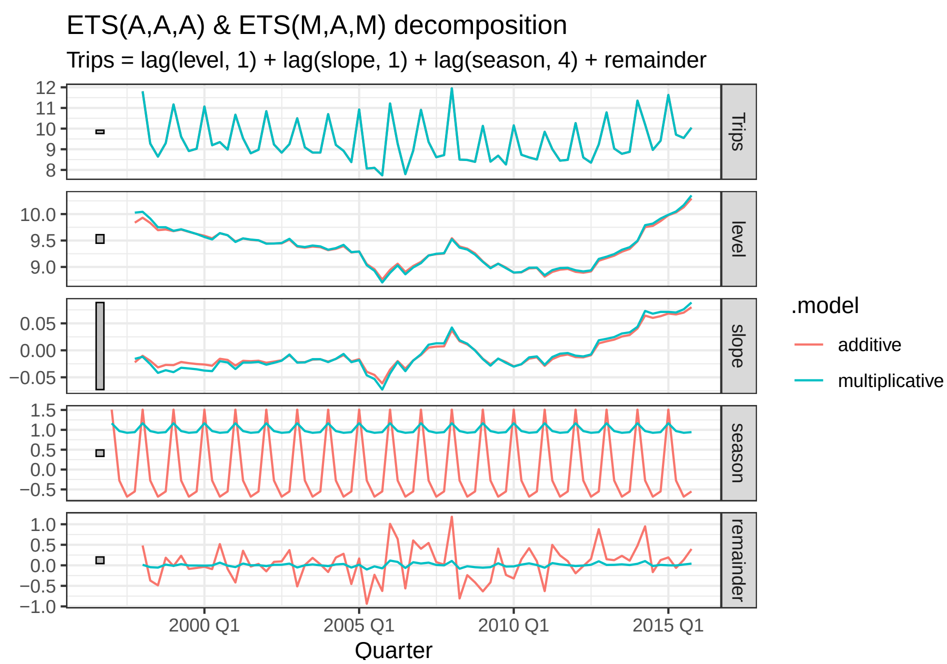
for all $h > 0$. In this equation, l_t is the "level" and b_t is the trend, and $s_t(h)$ is the seasonality at forecast horizon h.

The level, trend, and seasonality updates now account for the multiplicative term:

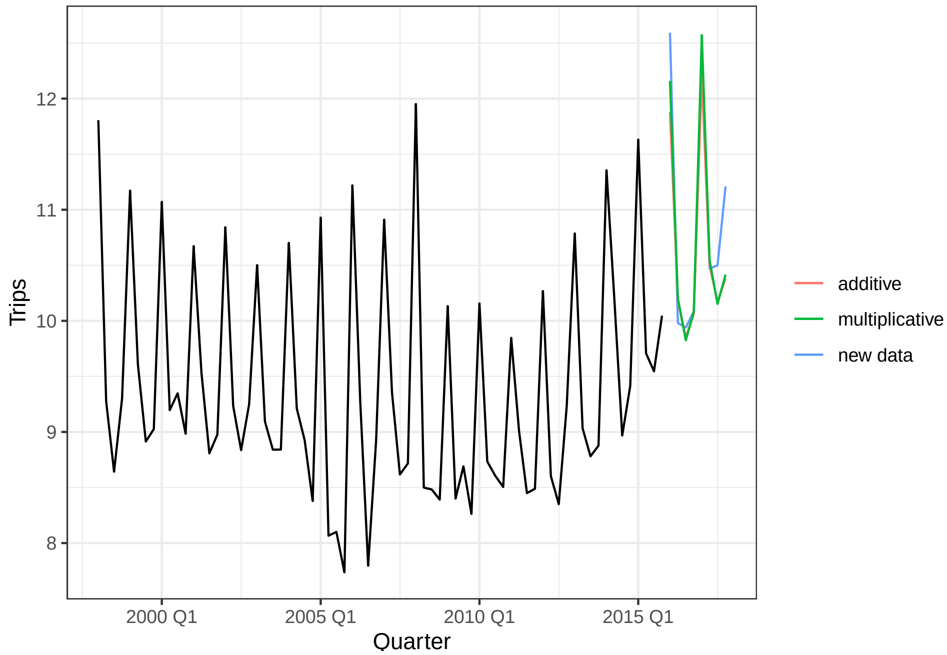
$$l_t = \alpha \frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1})$$
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$
$$s_t = \gamma \left(\frac{y_t}{l_{t-1} + b_{t-1}} \right) + (1-\gamma)s_{t-m}$$

```
fit_AM = aus_holidays  %>% filter(period == "train") %>%
  model(
    additive = ETS(Trips ~ error("A") + trend("A") + season("A")),
    multiplicative = ETS(Trips ~ error("M") + trend("A") + season("M"))
  )
```

```
components(fit_AM) %>% autoplot()
```




```
fcast_fit_AM = forecast(fit_AM, h = 8)
aus_holidays %>%
  ggplot(aes(Quarter, Trips)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_line(data = . %>% filter(period == "test"), aes(color = "new data")) +
  geom_line(data = fcast_fit_AM, aes(y = .mean, color = .model) ) +
  labs(color = "")
```



2.5 ETS Taxonomy

Trend	Seasonal N	Seasonal A	Seasonal M
N	$\hat{y}_{t+h\|t} = \ell_t$ $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = \ell_t s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t / \ell_{t-1}) + (1-\gamma)s_{t-m}$
A	$\hat{y}_{t+h\|t} = \ell_t + hb_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + b_{t-1})) + (1-\gamma)s_{t-m}$
$\mathbf{A_d}$	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - \phi b_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + \phi_h b_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + \phi b_{t-1})) + (1-\gamma)s_{t-m}$

```r
fit_add = aus_holidays %>% filter(period == "train") %>%
  model(
    ANN = ETS(Trips ~ trend("N") + season("N") + error("A")),
    AAN = ETS(Trips ~ trend("A") + season("N") + error("A")),
    AAdN = ETS(Trips ~ trend("Ad") + season("N") + error("A")),
    ANA = ETS(Trips ~ trend("N") + season("A") + error("A")),
    AAA = ETS(Trips ~ trend("A") + season("A") + error("A")),
    AAdA = ETS(Trips ~ trend("Ad") + season("A") + error("A")),
    ANM = ETS(Trips ~ trend("N") + season("M") + error("A")),
    AAM = ETS(Trips ~ trend("A") + season("M") + error("A")),
    AAdM = ETS(Trips ~ trend("Ad") + season("M") + error("A")),
  )
```

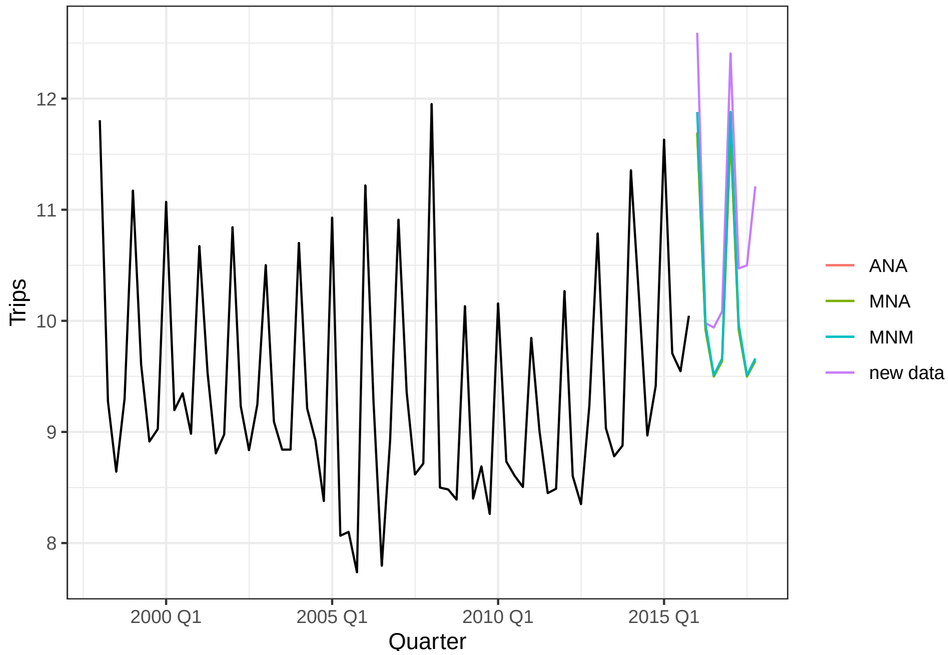
```r
fit_mult = aus_holidays %>% filter(period == "train") %>%
  model(
    MNN = ETS(Trips ~ trend("N") + season("N") + error("M")),
    MAN = ETS(Trips ~ trend("A") + season("N") + error("M")),
    MAdN = ETS(Trips ~ trend("Ad") + season("N") + error("M")),
```

```
    MNA = ETS(Trips ~ trend("N") + season("A") + error("M")),
    MAA = ETS(Trips ~ trend("A") + season("A") + error("M")),
    MAdA = ETS(Trips ~ trend("Ad") + season("A") + error("M")),
    MNM = ETS(Trips ~ trend("N") + season("M") + error("M")),
    MAM = ETS(Trips ~ trend("A") + season("M") + error("M")),
    MAdM = ETS(Trips ~ trend("Ad") + season("M") + error("M")),
  )
```

```
bind_cols(
  fit_add,
  fit_mult
) %>%
  glance %>% arrange(BIC)
#> # A tibble: 18 x 9
#>   .model  sigma2 log_lik   AIC  AICc   BIC   MSE  AMSE    MAE
#>   <chr>    <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
#> 1 MNA    0.00214   -90.3  195.  196.  211. 0.175 0.182 0.0342
#> 2 MNM    0.00215   -90.4  195.  197.  211. 0.176 0.182 0.0337
#> 3 ANA    0.191     -91.2  196.  198.  212. 0.175 0.183 0.322
#> 4 ANM    0.191     -91.2  196.  198.  212. 0.175 0.183 0.319
#> 5 MAA    0.00225   -90.8  200.  202.  220. 0.175 0.184 0.0343
#> 6 MAM    0.00225   -90.8  200.  202.  220. 0.175 0.186 0.0337
#> # i 12 more rows
```

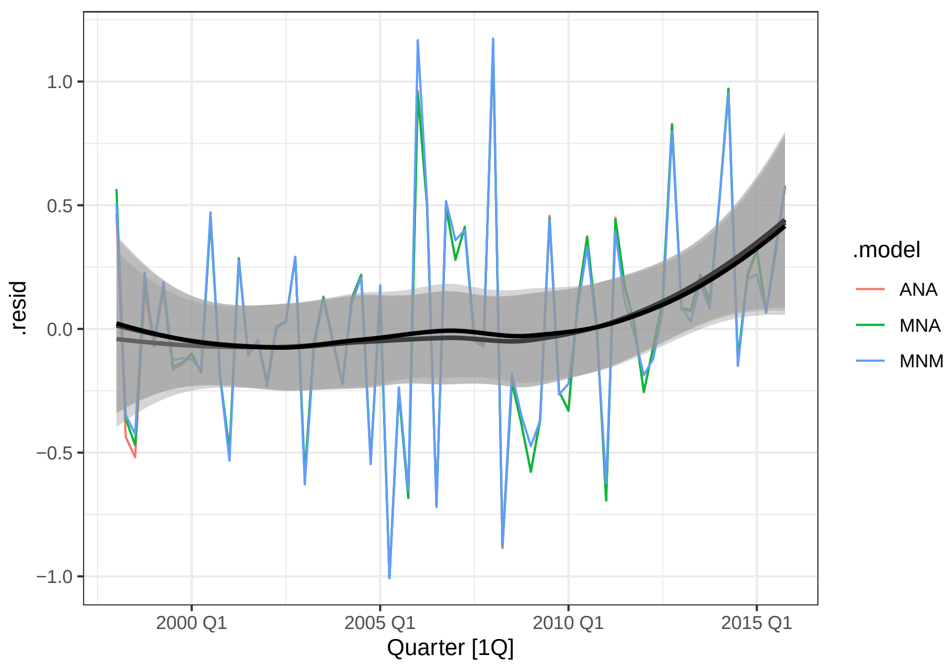
Looks like the Multiplicative with no-trend models fit best to the training data. Do you think the no-trend models are really best at forecast time?

```
best = aus_holidays %>% filter(period == "train") %>%
  model(
    MNA = ETS(Trips ~ trend("N") + season("A") + error("M")),
    MNM = ETS(Trips ~ trend("N") + season("M") + error("M")),
    ANA = ETS(Trips ~ trend("N") + season("A") + error("A")),
  )
fcast_best = forecast(best, h = 8)
aus_holidays %>%
  ggplot(aes(Quarter, Trips)) +
  geom_line(data = . %>% filter(period == "train"), color = "black" ) +
  geom_line(data = . %>% filter(period == "test"), aes(color = "new data")) +
  geom_line(data = fcast_best, aes(y = .mean, color = .model) ) +
  labs(color = "")
```



Checking out the residuals can reveal the trend that starts at the end of training!

```
augment(best) %>%
  autoplot(.resid) +
  geom_smooth(color = "black", aes(group = .model))
```



3 Cross Validation and Evaluation

Let's fit a few models to the `aus_holidays` *training* data.

```
fit_train = aus_holidays %>%
  filter(period == "train") %>% # Only use training data!
    model(
```

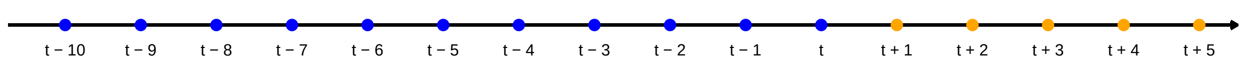
```
    MNA = ETS(Trips ~ trend("N") + season("A") + error("M")),
    MNM = ETS(Trips ~ trend("N") + season("M") + error("M")),
    AAA = ETS(Trips ~ trend("A") + season("A") + error("A"))
  )
```

Models can be compared using in-sample/training data.

```
#: compare models using training data
fit_train %>% glance()
#> # A tibble: 3 x 9
#>   .model  sigma2 log_lik   AIC  AICc   BIC   MSE  AMSE    MAE
#>   <chr>    <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
#> 1 MNA    0.00214   -90.3  195.  196.  211. 0.175 0.182 0.0342
#> 2 MNM    0.00215   -90.4  195.  197.  211. 0.176 0.182 0.0337
#> 3 AAA    0.195     -90.9  200.  203.  220. 0.173 0.183 0.319
```

```
#: training data residual-based metrics
fit_train %>% accuracy()
#> # A tibble: 3 x 10
#>   .model .type        ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
#>   <chr>  <chr>     <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 MNA    Training 0.0210 0.419 0.322 0.0446  3.42 0.797 0.789 -0.0690
#> 2 MNM    Training 0.0222 0.419 0.317 0.0485  3.36 0.784 0.791 -0.0999
#> 3 AAA    Training 0.0565 0.416 0.319 0.443   3.40 0.790 0.785 -0.0489
```

While using the mathematical adjustments like AIC/BIC are often good for model selection (i.e., choosing the best model), they do not give a good indicate of forecast accuracy. We should consider using test data. However, due to the sequential nature of time series data we can't use the normal resampling methods (e.g., cross-validation, bootstrap) as we did when the data are independent.

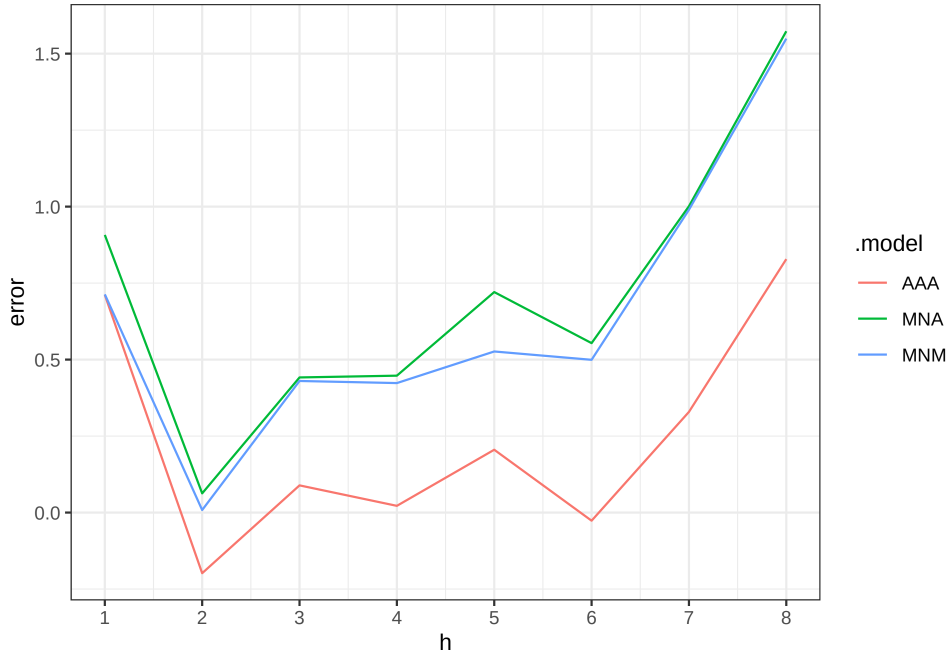


The `forecast()` function provides a way to estimate the outcome in future time periods. We can either enter h, the forecast horizon, or better to pass in the new data at which forecasts are requested.

```
# forecast(fit_train, h = 8) # make forecasts for 1:8 Quarters

#: using the test data
fc = forecast(
  fit_train,
  new_data = aus_holidays %>% filter(period == "test")
)
```

```
#: combine forecast (fc) and new data; calculate errors
eval_data = fc %>%
  left_join(
    aus_holidays %>% select(Quarter, truth = Trips),
    by = "Quarter"
  ) %>%
  mutate(
    h = dense_rank(Quarter),   # forecast horizon
    error = truth - .mean      # forecast error
  )
```

```r
#: ploting the error as function of forecast horizon (h)
eval_data %>%
  ggplot(aes(h, error, color = .model)) +
  geom_line() +
  scale_x_continuous(breaks = 1:10)
```



The plot shows the error as a function of h. As expected the error increases as h increases. You may be tempted to add up the forecasting errors, e.g., using the `accuracy()` function:

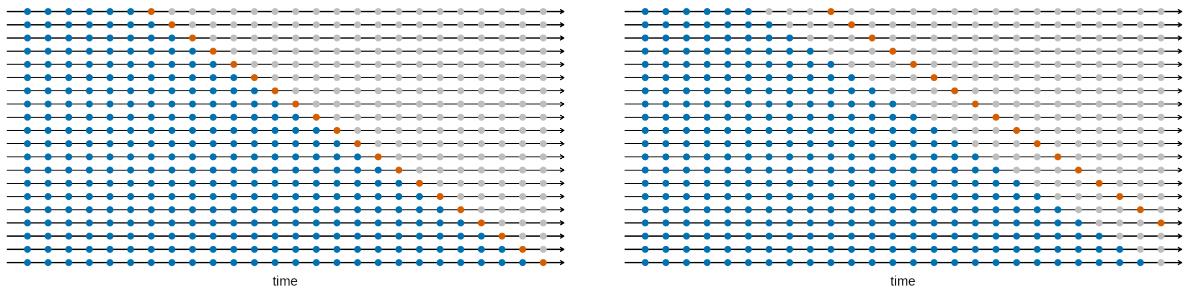
```r
#: get forecasting metrics. Average over all h.
eval_data %>% as_tibble() %>%
  group_by(.model) %>%
  summarize(
    RMSE = sqrt(mean(error^2)),
    MAE = mean(abs(error))
  )
#> # A tibble: 3 x 3
#>   .model  RMSE   MAE
#>   <chr>  <dbl> <dbl>
#> 1 AAA    0.417 0.301
#> 2 MNA    0.831 0.714
#> 3 MNM    0.773 0.642

#: the accuracy() function does this automatically for a large set of metrics
accuracy(fc, data = aus_holidays)
#> # A tibble: 3 x 10
#>   .model .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE    ACF1
#>   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 AAA    Test  0.245 0.417 0.301  2.09  2.65 0.746 0.786 -0.0624
#> 2 MNA    Test  0.714 0.831 0.714  6.42  6.42 1.77  1.57   0.221
#> 3 MNM    Test  0.642 0.773 0.642  5.82  5.82 1.59  1.46   0.300
```

But do you really want to average across all h with equal weights? In practice, you are probably most concerned with forecasts at a certain horizon (e.g., one year / 4 quarters ahead $h = 4$). This is where the idea of rolling windows comes in.

3.1 Time Series Cross-Validation

3.1.1 Growing Training Window

There are two direct ways to do time-series “cross-validation”. The first is to use a growing (i.e., *stretching*) window. This is most useful when the series is short and/or there are no *change points*.



The `stretch_tsibble()` function is used for the growing window.

```
aus_holidays %>% select(-period) %>%
  stretch_tsibble(
    .step = 1,     # size increment of next set
    .init = 4      # start at t = 4
    )
#> # A tibble: 3,234 x 3
#>   Quarter Trips   .id
#>     <qtr> <dbl> <int>
#> 1 1998 Q1 11.8      1
#> 2 1998 Q2  9.28     1
#> 3 1998 Q3  8.64     1
#> 4 1998 Q4  9.30     1
#> 5 1998 Q1 11.8      2
#> 6 1998 Q2  9.28     2
#> # i 3,228 more rows
```

Notice that a new `.id` column is added. There are $n = 4$ times with `.id = 1` (1998 Q1 - 1998 Q4). The next `.id = 2` has $n = 5$ times (1998 Q1 - 1999 Q1). And the size of each `.id` set continues growing. Now for modeling, all models are fit to each `.id`. Thus the models fit to the data with `.id = 1` are only using four times (not much training data!). But as we move down the list, you can see that `.id = 60` has a training data size of $n = 63$.

The `.step =` argument controls the change in size for the next set. For example, if `.step = 3` the `.id = 2` will have $n = 7$, `.id = 3` will have $n = 10$, etc.

```
aus_holidays %>% select(-period) %>%
  stretch_tsibble(
    .step = 3,     # size increment of next set
    .init = 4      # start at t = 4
    ) %>%
  count(.id)
#> # A tibble: 26 x 2
#>     .id     n
#>   <int> <int>
#> 1     1     4
#> 2     2     7
#> 3     3    10
#> 4     4    13
#> 5     5    16
#> 6     6    19
```

```
#> # i 20 more rows
```

Here we'll stretch our data starting at `.init = 72`, which corresponds to the original `period = "train"` training data.

```
#: growing training data
aus_holidays_stretch = aus_holidays %>%
  stretch_tsibble(.step = 1, .init = 72)
```

Now, we fit our three models to each `.id`. This re-fits the models as the training data grows.

```
#: fit each model to each .id
fit_stretch = aus_holidays_stretch %>%
  model(
    MNA = ETS(Trips ~ trend("N") + season("A") + error("M")),
    MNM = ETS(Trips ~ trend("N") + season("M") + error("M")),
    AAA = ETS(Trips ~ trend("A") + season("A") + error("A"))
  )
```

```
#> # A tibble: 9 x 4
#>     .id          MNA          MNM          AAA
#>   <int>      <model>      <model>      <model>
#> 1     1 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> 2     2 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> 3     3 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> 4     4 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> 5     5 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> 6     6 <ETS(M,N,A)> <ETS(M,N,M)> <ETS(A,A,A)>
#> # i 3 more rows
```

The `forecast()` function can generate the one-step ahead predictions.

```
fc = fit_stretch %>% forecast(h = 1)
```

```
#> # A tibble: 27 x 5
#>     .id .model Quarter       Trips .mean
#>   <int> <chr>    <qtr>      <dist> <dbl>
#> 1     1 MNA    2016 Q1 N(12, 0.29)  11.7
#> 2     1 MNM    2016 Q1  N(12, 0.3)  11.9
#> 3     1 AAA    2016 Q1 N(12, 0.19)  11.9
#> 4     2 MNA    2016 Q2 N(10, 0.23)  10.2
#> 5     2 MNM    2016 Q2 N(10, 0.22)  10.1
#> 6     2 AAA    2016 Q2  N(10, 0.2)  10.4
#> # i 21 more rows
```

Notice that `.id = 1` has a $h = 1$ forecast for 2016 **Q1** (since it was trained up to 2015 Q4). But `.id = 2` has an $h = 1$ forecast for 2016 **Q2**.

We can get the overall performance using the `accuracy()` function:

```
accuracy(fc, data = aus_holidays)
#> Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missi
```

```
#> 1 observation is missing at 2018 Q1
#> # A tibble: 3 x 10
#>   .model .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE    ACF1
#>   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 AAA    Test  0.129 0.398 0.314 0.983  2.82 0.758 0.737 -0.189
#> 2 MNA    Test  0.375 0.528 0.432 3.27   3.84 1.04  0.979 -0.138
#> 3 MNM    Test  0.377 0.510 0.409 3.38   3.70 0.988 0.945  0.0554
```

3.1.2 Sliding Training Window

Notice in the *growing* window approach the training data gets larger for each set. For long time series, or series with change points, we will want to limit the window size. The `slide_tsibble()` function creates a sliding window

Sliding Window Training

```
#: growing training data
aus_holidays_slide = aus_holidays %>%
  slide_tsibble(.size = 12, .step = 1)
```

```
#> # A tibble: 828 x 4
#>   Quarter Trips period   .id
#>     <qtr> <dbl> <chr>  <int>
#> 1 1998 Q1 11.8  train      1
#> 2 1998 Q2  9.28 train      1
#> 3 1998 Q3  8.64 train      1
#> 4 1998 Q4  9.30 train      1
#> 5 1999 Q1 11.2  train      1
#> 6 1999 Q2  9.61 train      1
#> # i 822 more rows
```

Notice that the sliding windows creates all training sets to be of size $n = 12$ (three years).

Now, we fit our three models to each `.id`. This re-fits the models as the training data slides.

```
#: fit each model to each .id
fit_slide = aus_holidays_slide %>%
```

```
  model(
    MNA = ETS(Trips ~ trend("N") + season("A") + error("M")),
    MNM = ETS(Trips ~ trend("N") + season("M") + error("M")),
    AAA = ETS(Trips ~ trend("A") + season("A") + error("A"))
  )
```

The `forecast()` function can generate the one-step ahead predictions.

```
fc_slide = fit_slide %>% forecast(h = 1)

#> # A tibble: 207 x 5
#>     .id .model Quarter         Trips .mean
#>   <int> <chr>    <qtr>        <dist> <dbl>
#> 1     1 MNA    2001 Q1   N(11, 0.16) 11.3
#> 2     1 MNM    2001 Q1   N(11, 0.16) 11.3
#> 3     1 AAA    2001 Q1   N(11, 0.18) 11.1
#> 4     2 MNA    2001 Q2  N(9.4, 0.09)  9.35
#> 5     2 MNM    2001 Q2 N(9.4, 0.091)  9.36
#> 6     2 AAA    2001 Q2  N(9.3, 0.14)  9.35
#> # i 201 more rows
```

We can get the overall performance using the `accuracy()` function:

```
accuracy(fc_slide, data = aus_holidays)
#> Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missin
#> 1 observation is missing at 2018 Q1
#> # A tibble: 3 x 10
#>   .model .type      ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
#>   <chr>  <chr>   <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 AAA    Test  0.0421  0.534 0.415  0.260  4.47  1.00 0.990 -0.105
#> 2 MNA    Test  0.00756 0.538 0.433 -0.194  4.57  1.04 0.996 -0.0697
#> 3 MNM    Test  0.0422  0.559 0.445  0.144  4.67  1.07 1.04   0.0244
```

We can treat the window size (`.size`) as a tuning parameter and find the window size and model specification that gives the best rolling performance. And you may want to give a little higher weight to performance in the most recent dates, e.g.,

```
accuracy(fc_slide,
         data = aus_holidays %>% filter(year(Quarter) > 2015))
#> Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missin
#> 61 observations are missing between 2001 Q1 and 2018 Q1
#> # A tibble: 3 x 10
#>   .model .type     ME  RMSE   MAE   MPE  MAPE  MASE RMSSE   ACF1
#>   <chr>  <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
#> 1 AAA    Test  0.0937 0.366 0.289 0.725  2.64 0.490 0.537 -0.227
#> 2 MNA    Test  0.253  0.503 0.446 2.16   4.10 0.756 0.739 -0.552
#> 3 MNM    Test  0.382  0.591 0.539 3.38   4.95 0.912 0.867 -0.420
```