Simple Exponential Smoothing

FISH 550 – Applied Time Series Analysis

Eli Holmes

19 Feb 2019

Naive forecast

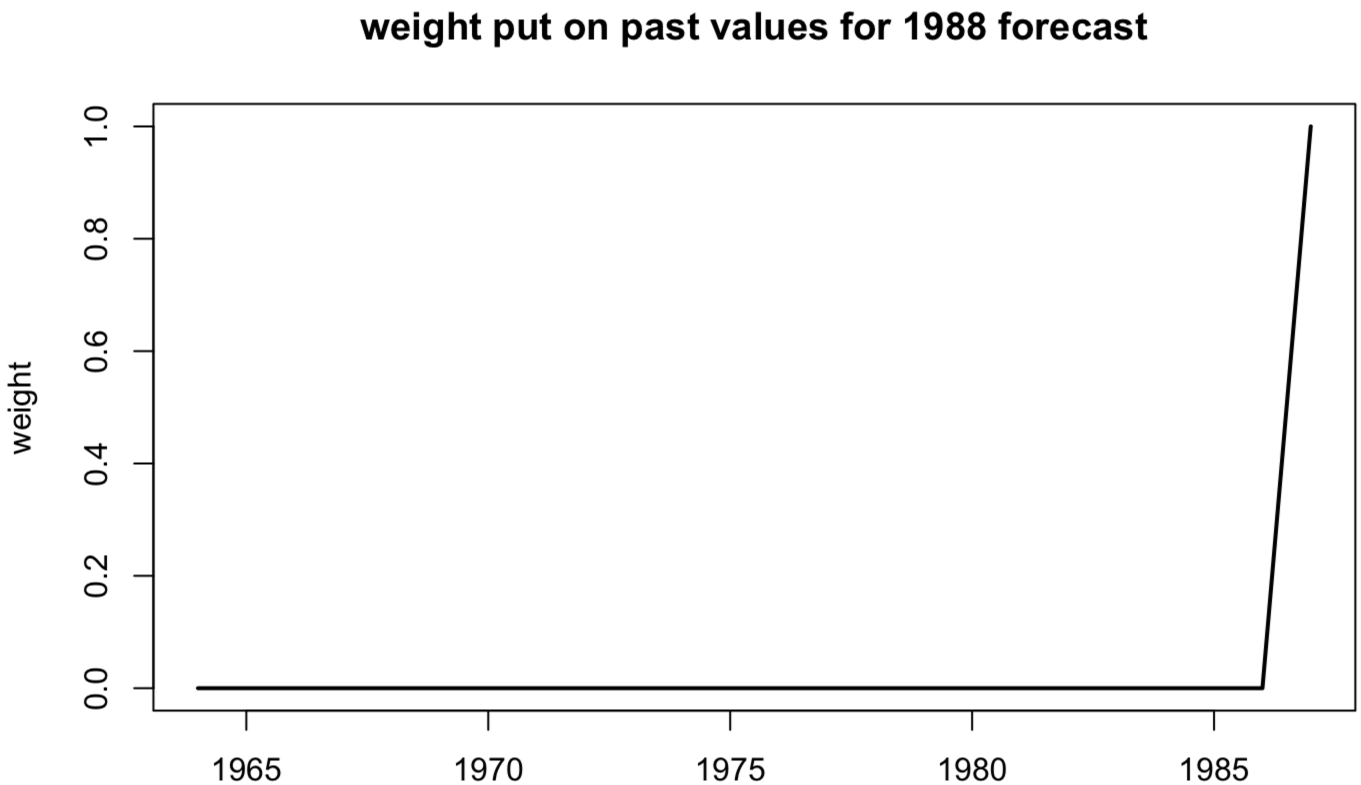
For a naive forecast of the anchovy catch in 1988, we just use the 1987 catch.

$$\hat{x}_{1988} = x_{1987}$$

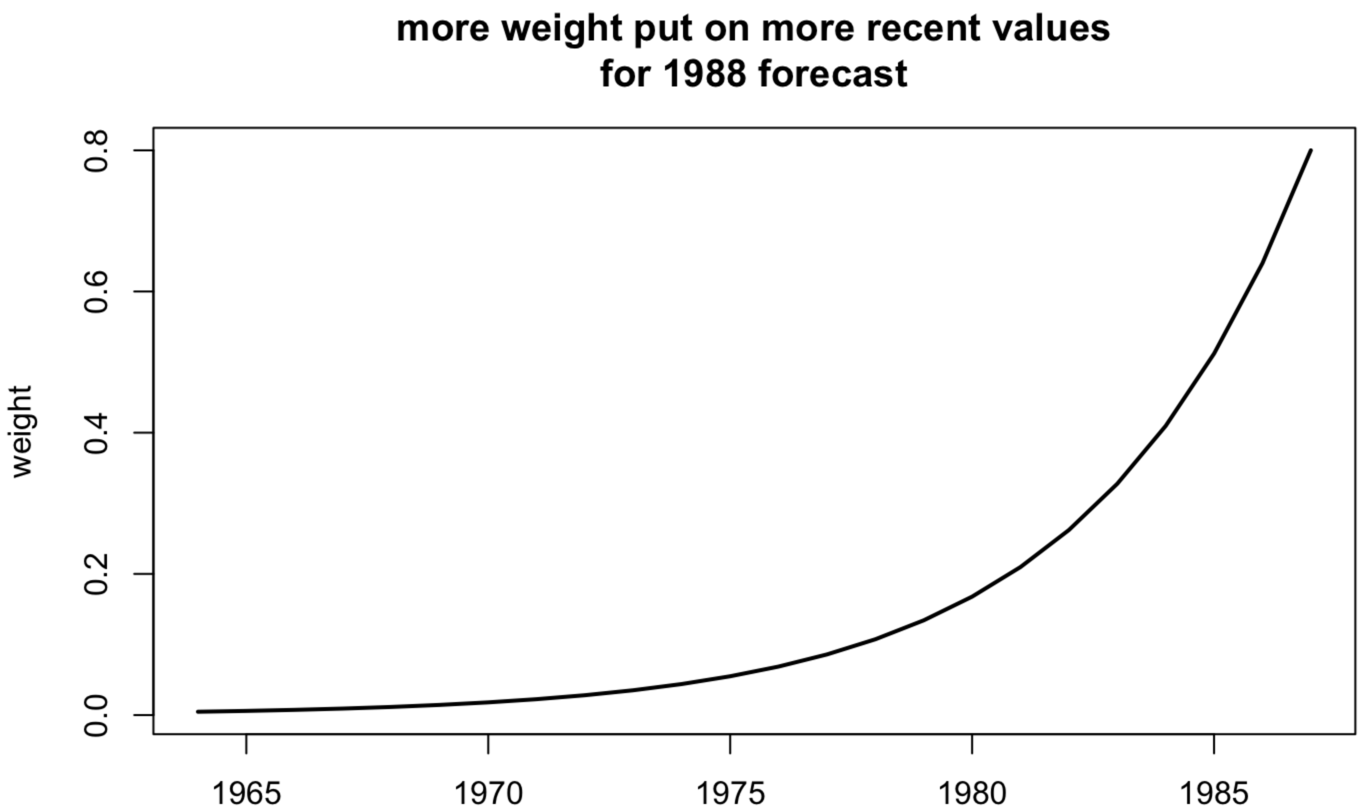
Which is the same as saying that we put 100% of the 'weight' on the most recent value and no weight on any value prior to that.

$$\hat{x}_{1988} = 1 \times x_{1987} + 0 \times x_{1986} + 0 \times x_{1985} + \dots$$

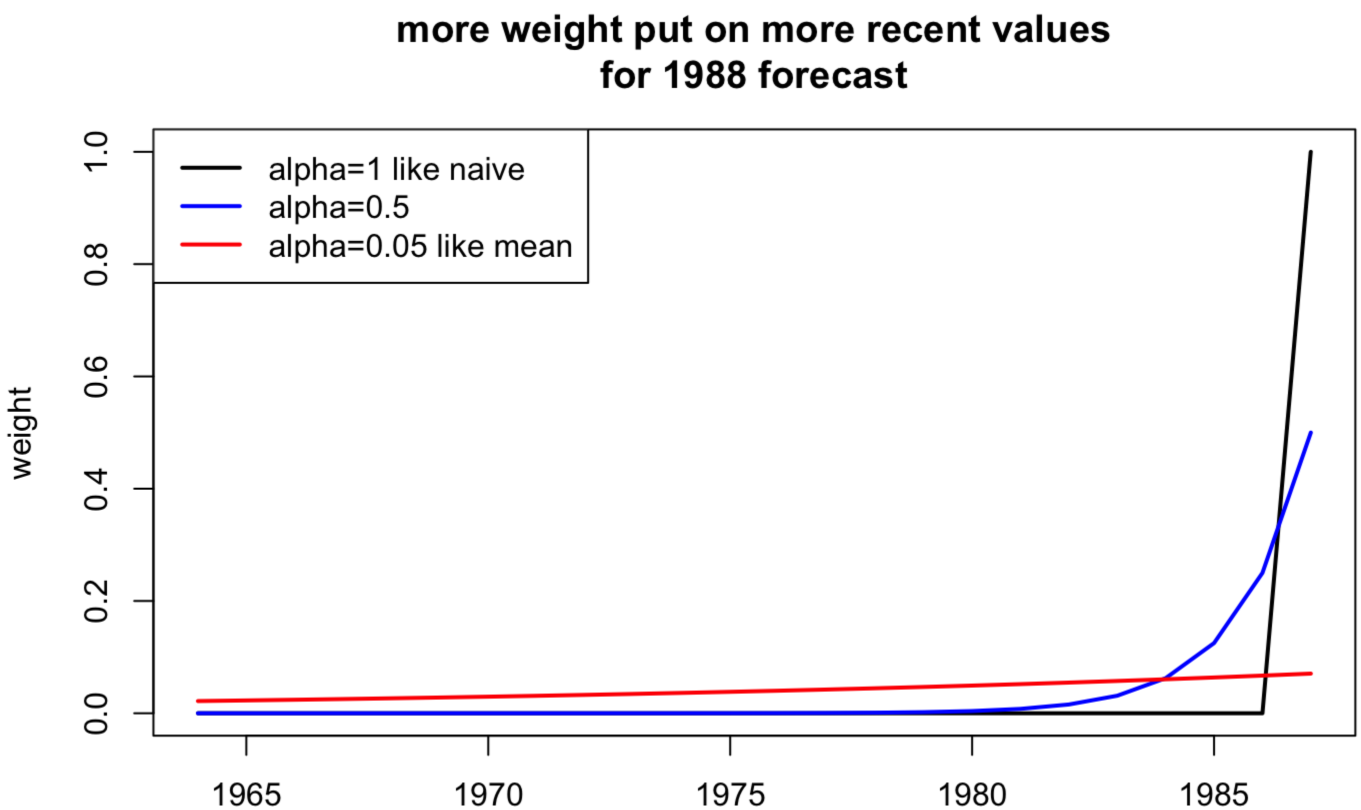
Past values in the time series have information about the current state, but only the most recent past value.



That’s a bit extreme. Often the values prior to the last value also have some information about future states. But the ‘information content’ should decrease the farther in the past that we go.



Simple exponential smoothing uses this type of weighting that falls off exponentially and the objective is to estimate the best weighting (α):



Fitting exponential smoothing (ETS) models

The forecast package will fit a wide variety of ETS models. The main fitting function is `ets()`:

```
ets(y, model = "ZZZ", < + many other arguments >)
```

- y : your data. A time series of responses.
- model: what type of ETS model.
  - Z=choose, A=additive, M=multiplicative
  - first letter is for level
  - second letter is for trend
  - third letter is for season

We are going to `ets()` to fit three simple types of ETS models:

model	"ZZZ"	alternate function
ETS no trend	"ANN"	`ses()`
ETS with trend	"AAN"	`holt()`
ETS choose trend	"AZN"	NA

The alternate function does exactly the same fitting. It is just a 'shortcut'.

Fit ETS with no trend

This is like the naive model that just uses the last value to make the forecast, but instead of only using the last value it will use values farther in the past also. The weighting fall off exponentially.

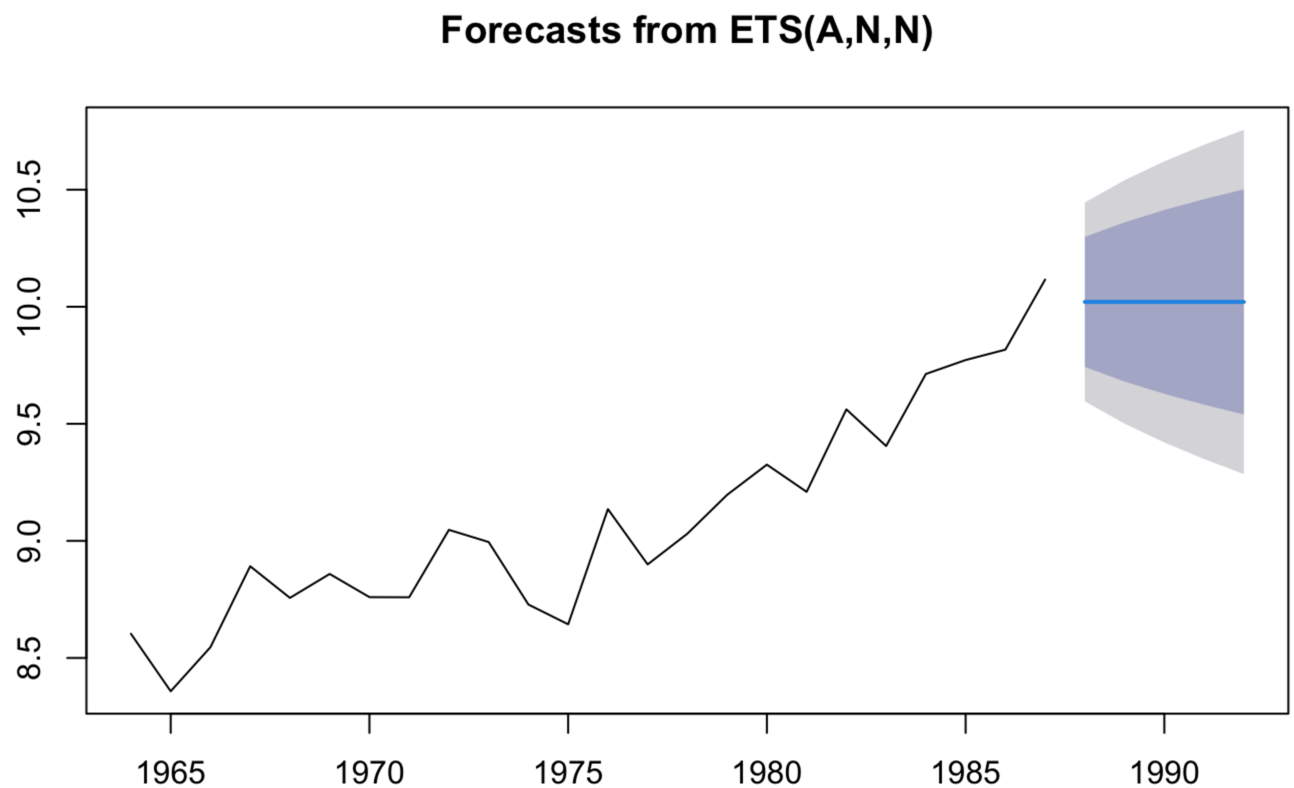
Load the data and **forecast** package.

```
data(greeklandings, package="atsalibrary")
anchovy <- subset(greeklandings,
                  Species=="Anchovy" & Year<=1987)$log.metric.tons
anchovy <- ts(anchovy, start=1964)
library(forecast)
```

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

```
fit <- ets(anchovy, model="ANN")
fr <- forecast(fit, h=5)
```

```
plot(fr)
```



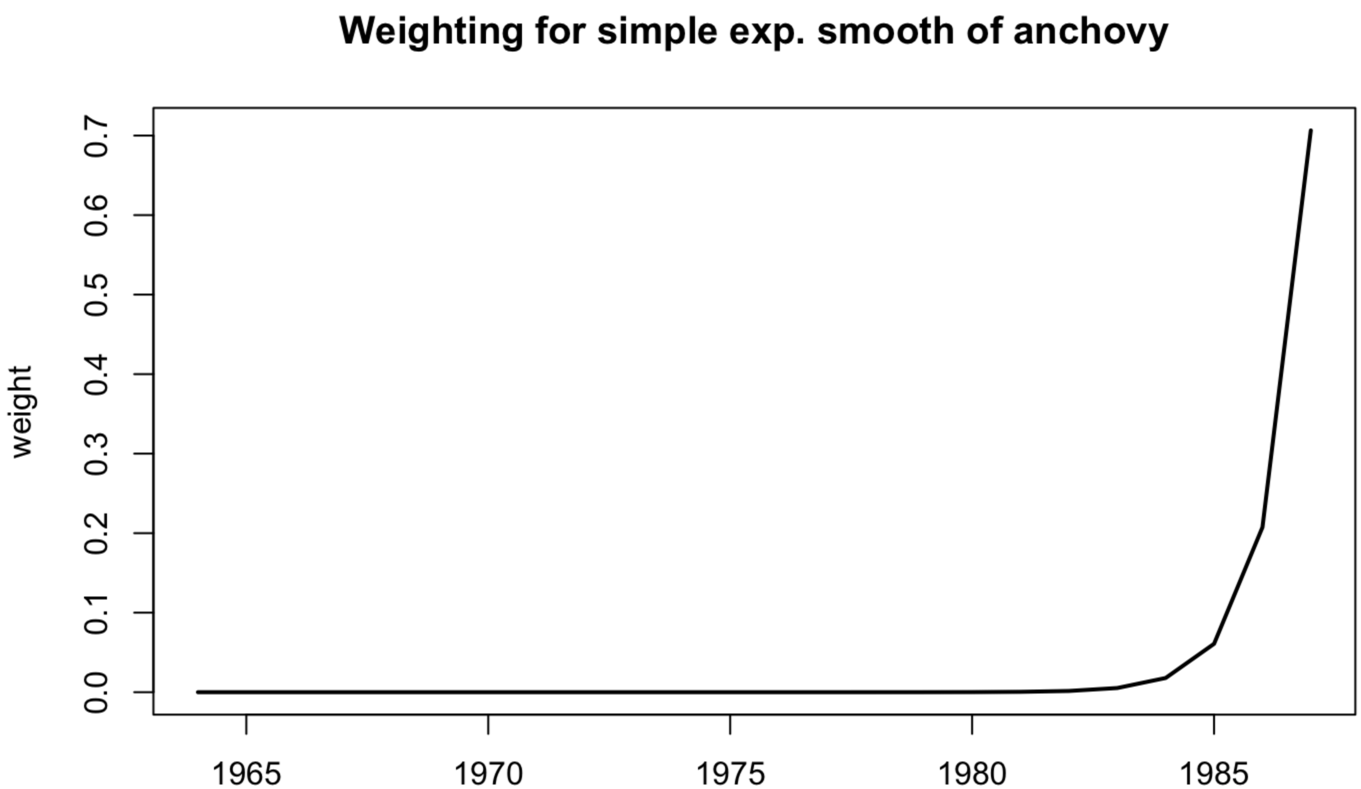
Forecasts from ETS(A,N,N)
10.5
10.0
9.5
9.0
8.5
1965
1970
1975
1980
1985
1990

Look at the estimates

```
fit
```

```
## ETS(A,N,N)
##
## Call:
##  ets(y = anchovy, model = "ANN")
##
##   Smoothing parameters:
##     alpha = 0.7065
##
##   Initial states:
##     l = 8.5553
##
##   sigma:  0.2166
##
##       AIC      AICc       BIC
##  6.764613  7.964613 10.298775
```

The weighting function



Produce forecast with ETS from a previous fit

Say you want to estimate a forecasting model from one dataset and use that model to forecast another dataset or another area. Here is how to do that.

This is the fit to the 1964-1987 data:

```
fit1 <- ets(anchovy, model="ANN")
```

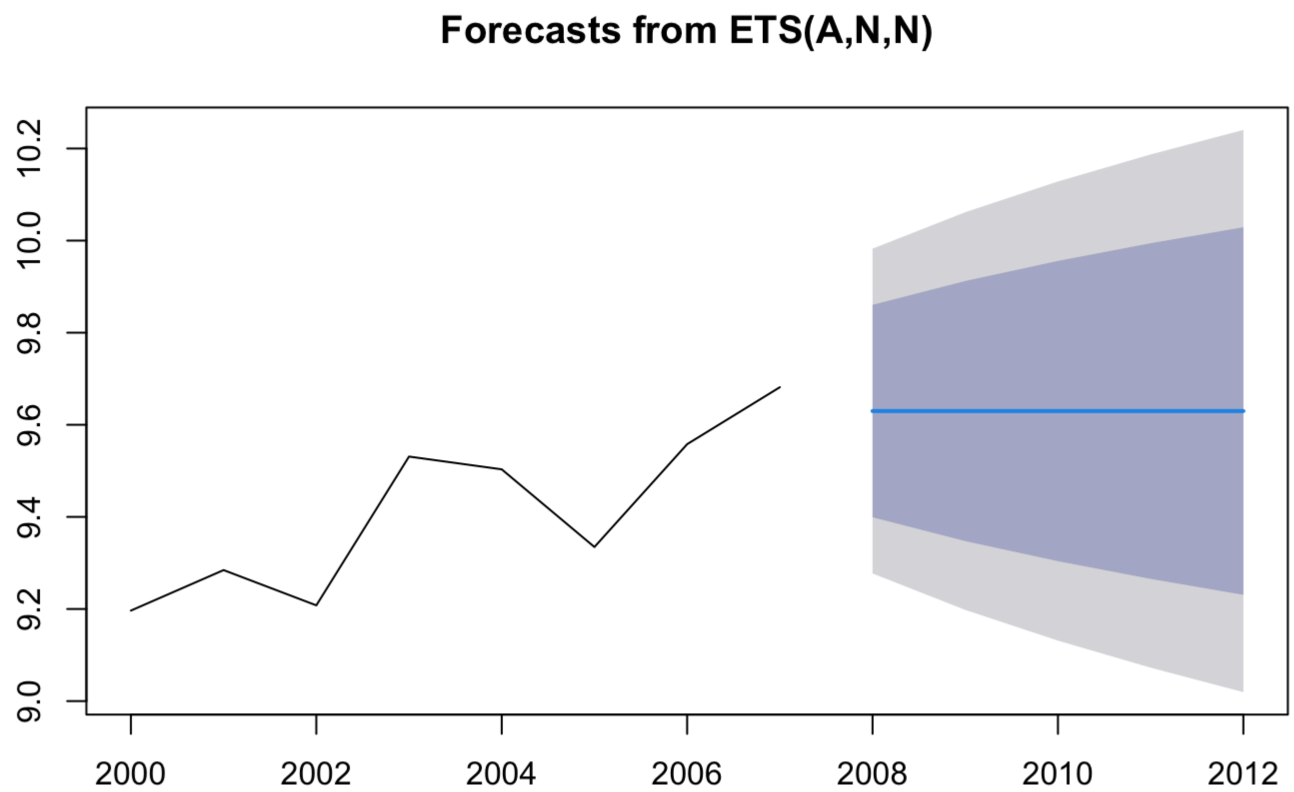
Use that model with the 2000-2007 data and produce a forecast:

```
dat <- subset(landings, Species=="Anchovy" & Year>=2000 & Year<=2007)
dat <- ts(dat$log.metric.tons, start=2000)
fit2 <- ets(dat, model=fit1)
```

```
## Model is being refit with current smoothing parameters but initial states are being re-estimated.
## Set 'use.initial.values=TRUE' if you want to re-use existing initial values.
```

```
fr2 <- forecast(fit2, h=5)
```

```
plot(fr2)
```

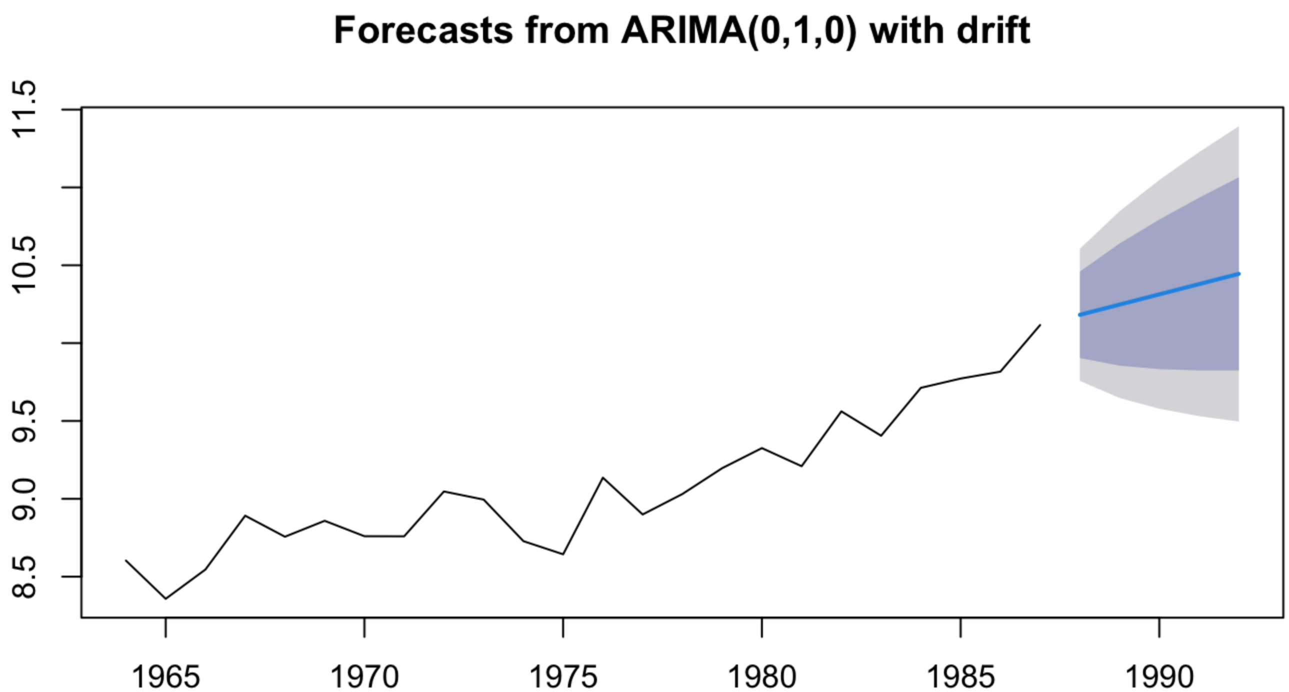

Forecasts from ETS(A,N,N)
9.0
9.2
9.4
9.6
9.8
10.0
10.2
2000
2002
2004
2006
2008
2010
2012

Naive model with trend (drift)

Fit a model that uses the last observation as the forecast but includes a trend (or drift) estimated from ALL the data. This is what the naive model with trend does.

```
fit.rwf <- Arima(anchovy, order=c(0,1,0), include.drift=TRUE)
fr.rwf <- forecast(fit.rwf, h=5)
```

```
plot(fr.rwf)
```



Forecasts from ARIMA(0,1,0) with drift
8.5
9.0
9.5
10.5
11.5
1965
1970
1975
1980
1985
1990

The trend seen in the blue line is estimated from the overall trend in ALL the data.

```
coef(fit.rwf)
```

```
##      drift
## 0.06577281
```

The trend from all the data is (last-first)/(number of steps).

```
mean(diff(anchovy))
```

```
## [1] 0.06577281
```

So we only use the latest data to choose the level for our forecast but use all the data to choose the trend? It would make more sense to weight the more recent trends more heavily.

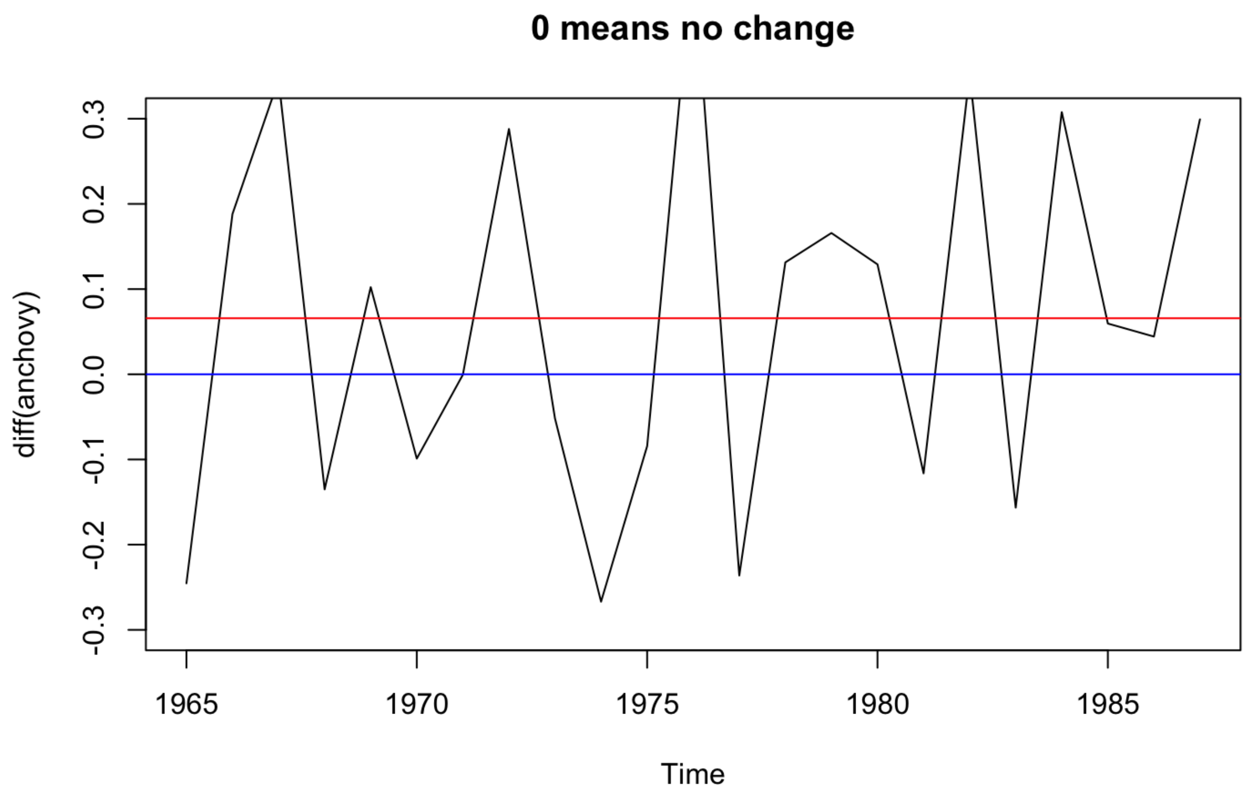
ETS model with trend

The ETS model with trend does this. The one-year trend is

$$x_t - x_{t-1}$$

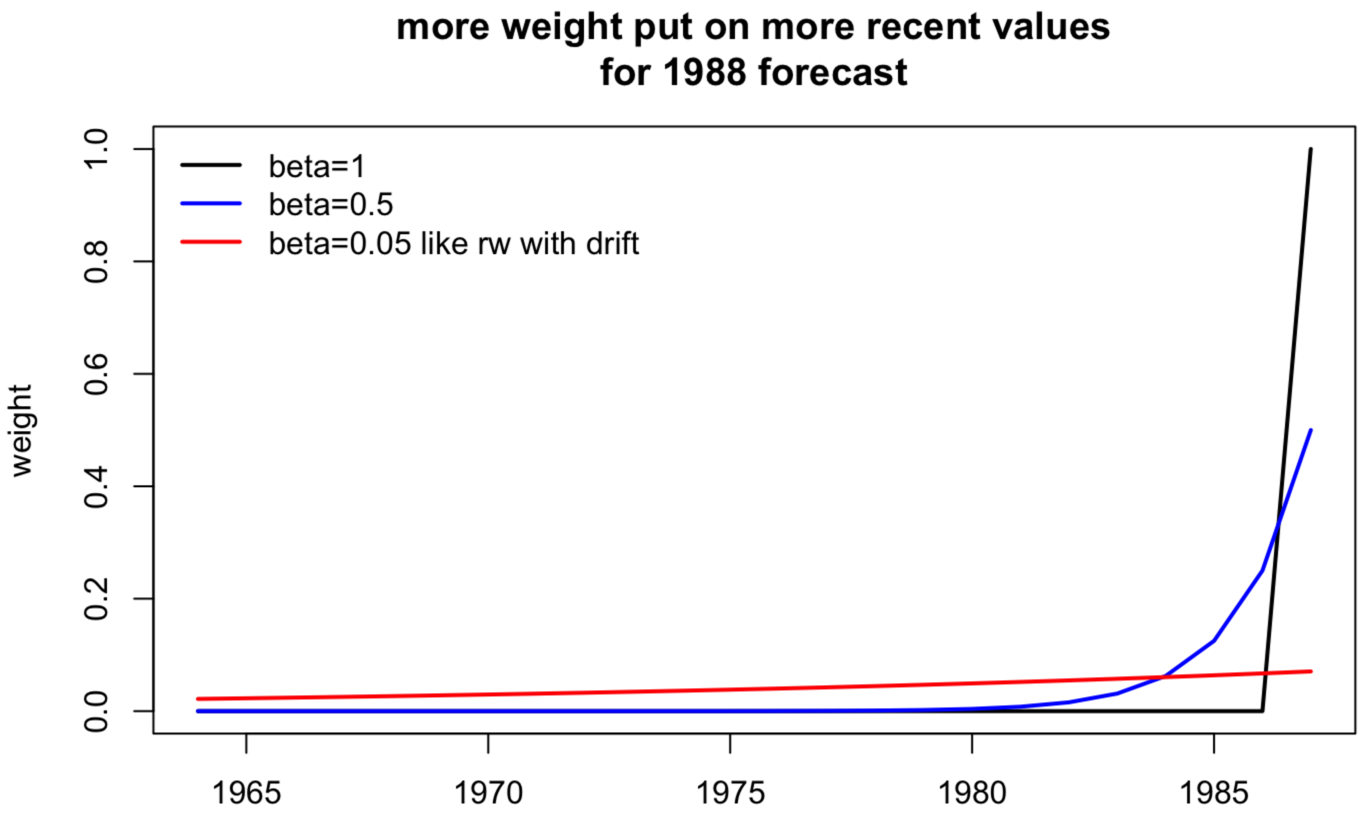
That is how much the data increased or decreased.

```r
plot(diff(anchovy),ylim=c(-0.3,.3))
abline(h=0, col="blue")
abline(h=mean(diff(anchovy)),col="red")
title("0 means no change")
```



If we take the average of all one-step changes ($\Delta x_t = x_t - x_{t-1}$ we are using the average trend like the naive model with drift. We put an equal weighting on all the Δx_t in the data.

But we could use a weighting that falls off exponentially so that we more recent Δx_t affect the forecast more than Δx_t in the distant past. That is what an ETS model with trend does.


more weight put on more recent values
for 1988 forecast
beta=1
beta=0.5
beta=0.05 like rw with drift
weight
0.0
0.2
0.4
0.6
0.8
1.0
1965
1970
1975
1980
1985

Naive model with level and trend

If your training data are length T, then a forecast for $T + h$ is

$$\hat{x}_{T+h} = l_T + h\hat{b}$$

where $\hat{b}$ is the mean of the the one-step changes in x, so the mean of $\Delta x_t = x_t - x_{t-1}$.

$$\hat{b} = \sum_{t=2}^{T}(x_t - x_{t-1})$$

ETS model with level and trend

The ETS model puts more weight on the recent Δx_t (one-step trends in the data).

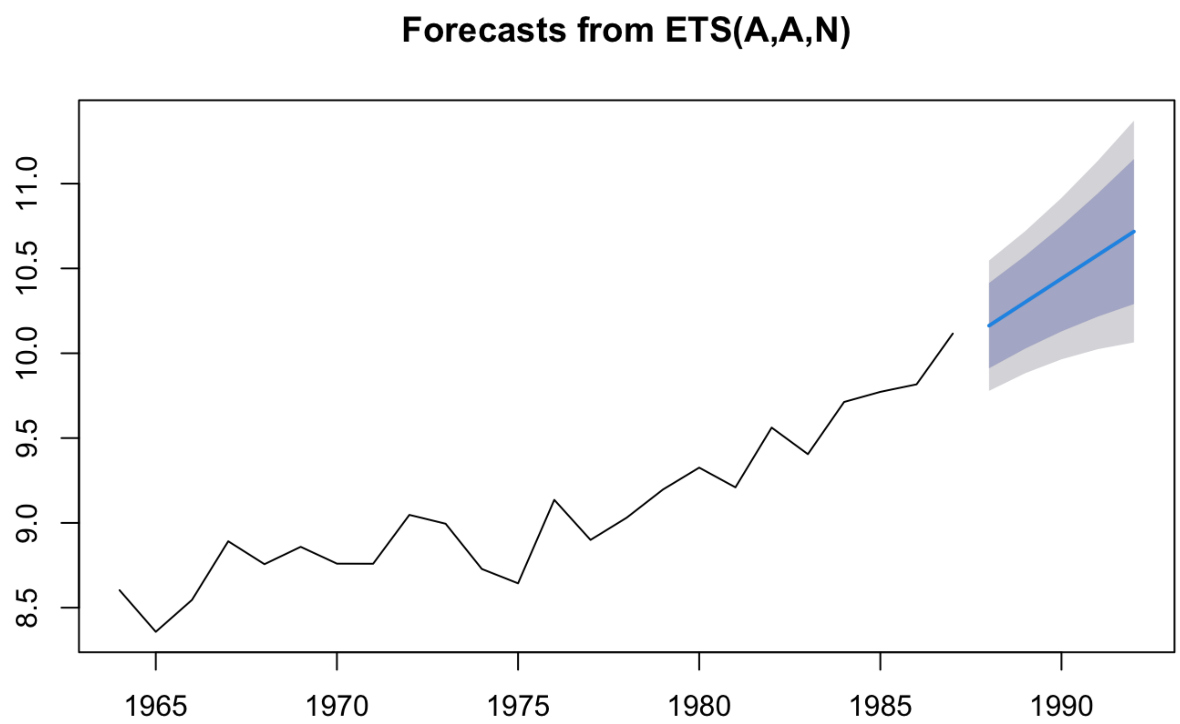
$$\hat{x}_{T+h} = l_T + hb_T$$

where b_T is a weighted average with the more recent Δx_t (trends) given more weight.

$$b_t = \sum_{t=2}^{T} \beta(1-\beta)^{t-2}(x_t - x_{t-1})$$

Fit using `ets()`

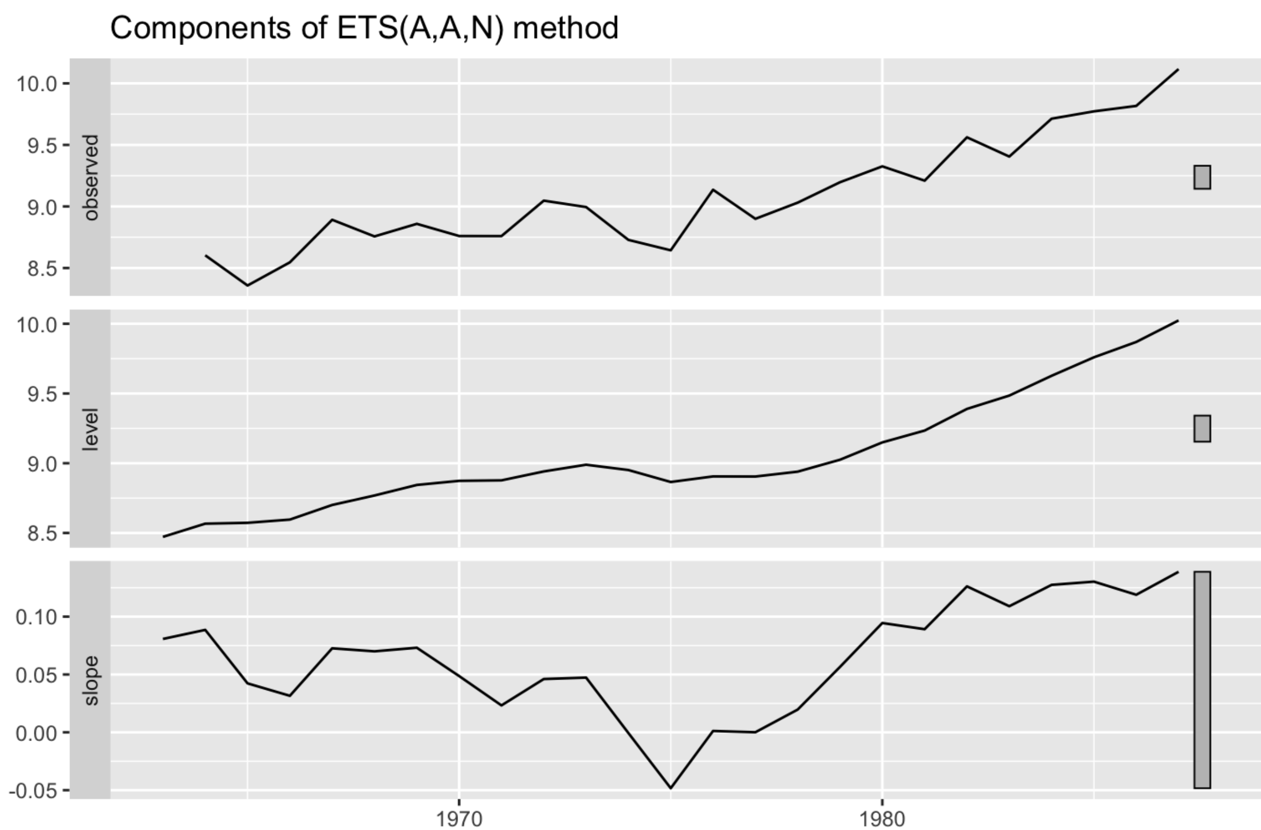
```
fit <- ets(anchovy, model="AAN")
fr <- forecast(fit, h=5)
plot(fr)
```



Decomposing your model fit

Sometimes you would like to see the smoothed level and smoothed trend that the model estimated. You can see that with `plot(fit)` or `autoplot(fit)`.

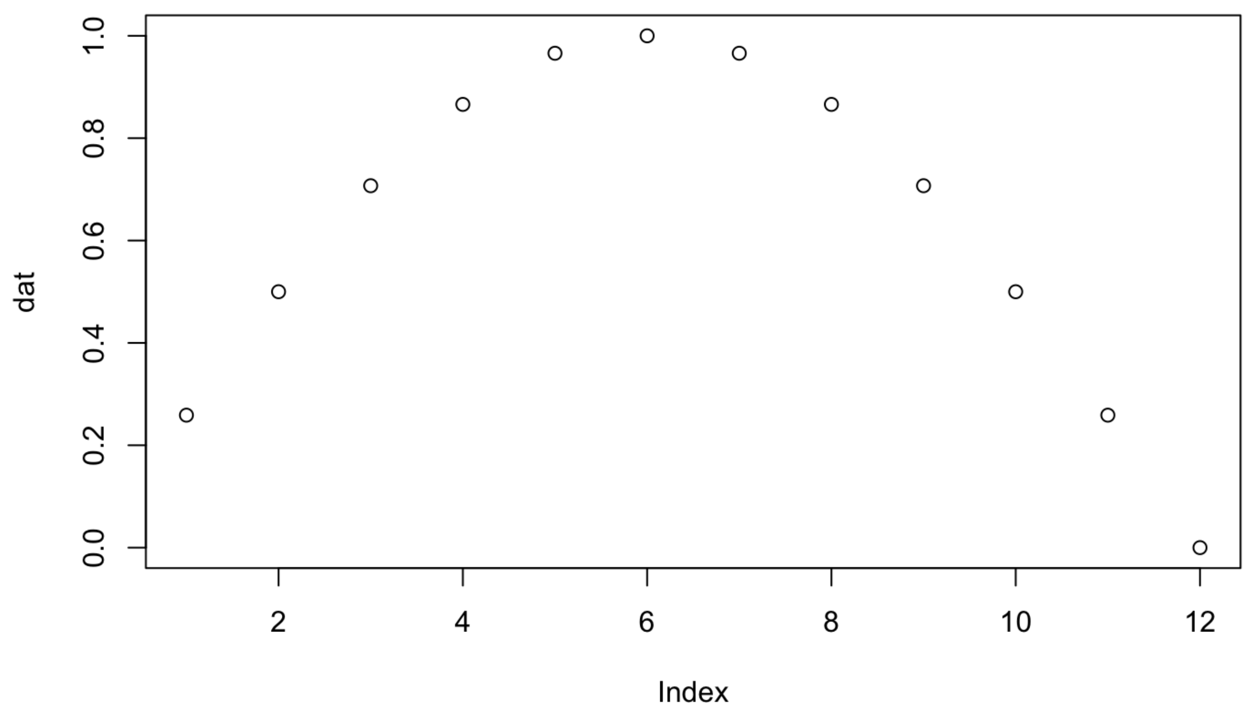
```
autoplot(fit)
```



Simulated data with level and trend

Let's imagine that our data look like so:

```
dat <- sin((1:12)*pi/12)
plot(dat)
```



It is quite smooth (no noise). Let's fit an ETS with level and trend.

What should the level be? The one-step ahead ($h = 1$) prediction model for an ETS is

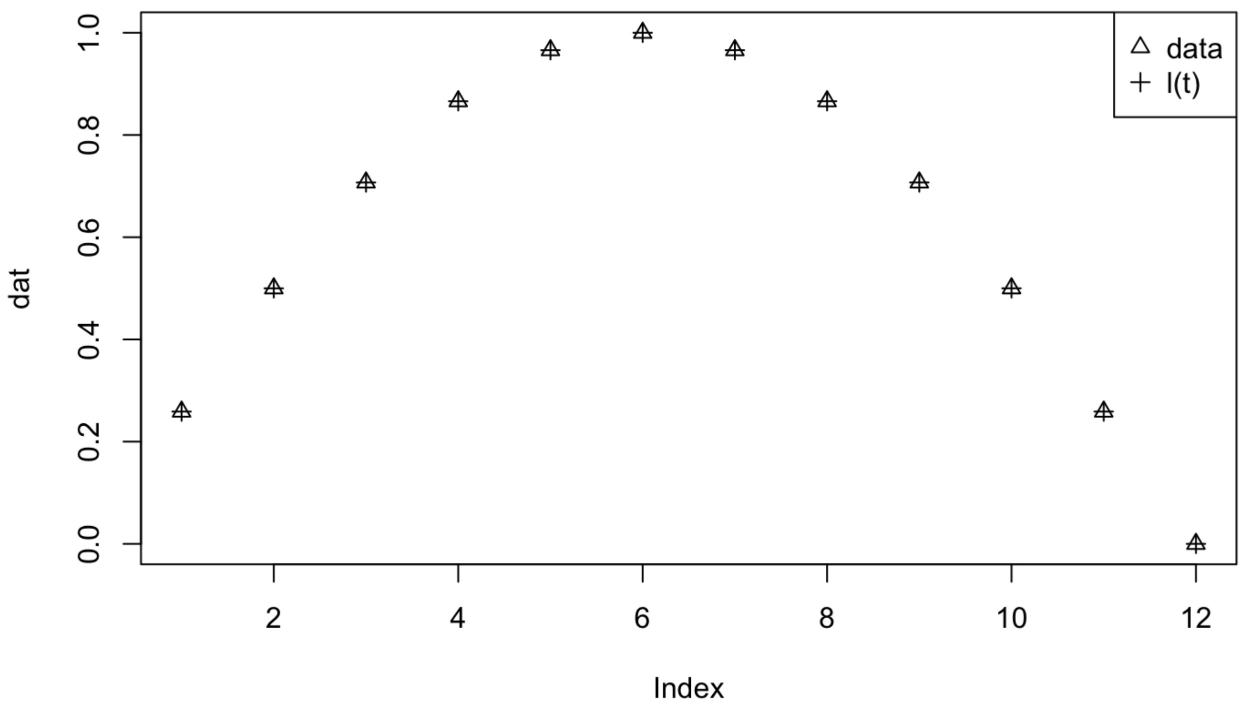
$$\hat{y}_{t+1|1:t} = l_t + b_t$$

where l_t is the level to use in our prediction. l_t is where to "start" our prediction and then we'll add our estimated trend b_t.

Since there is no error in the data and it is smooth, l_t should be y_t!

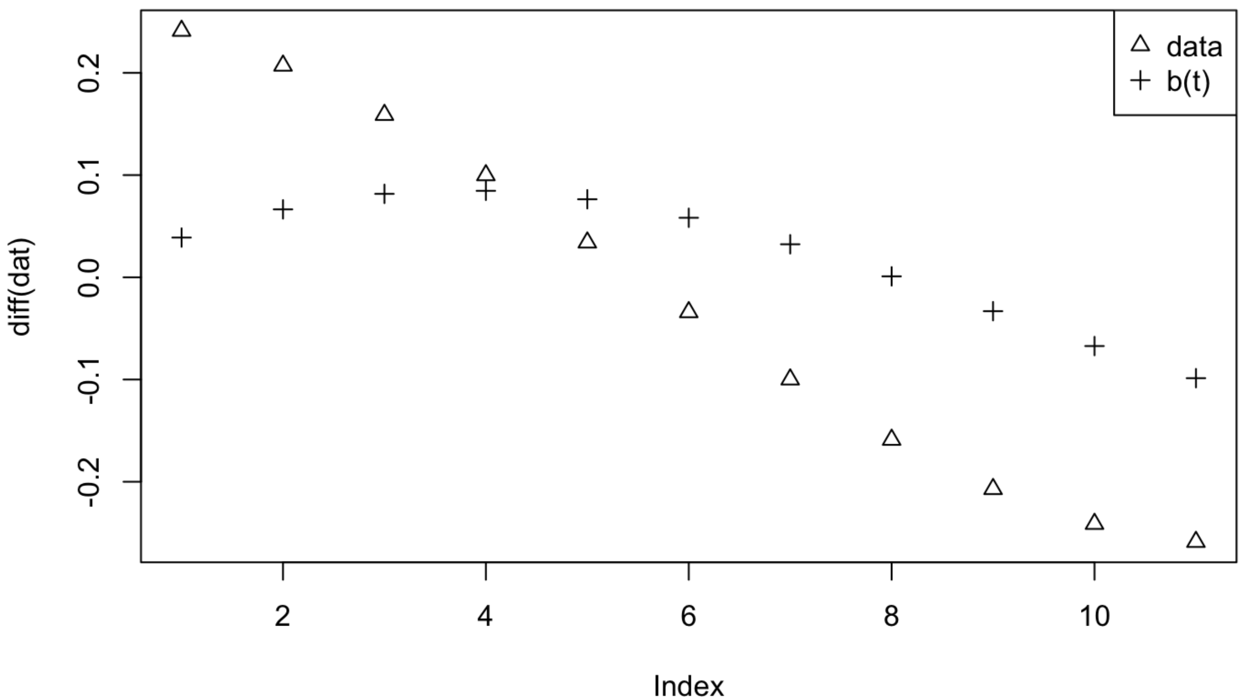
The estimated level for the ETS is in `fit$states`:

```
library(forecast)
fit <- ets(dat, model="AAN", damped=FALSE)
plot(dat, type="p",pch=2)
points(fit$states[2:13,1],pch=3)
legend("topright",c("data","l(t)"),pch=c(2,3))
```



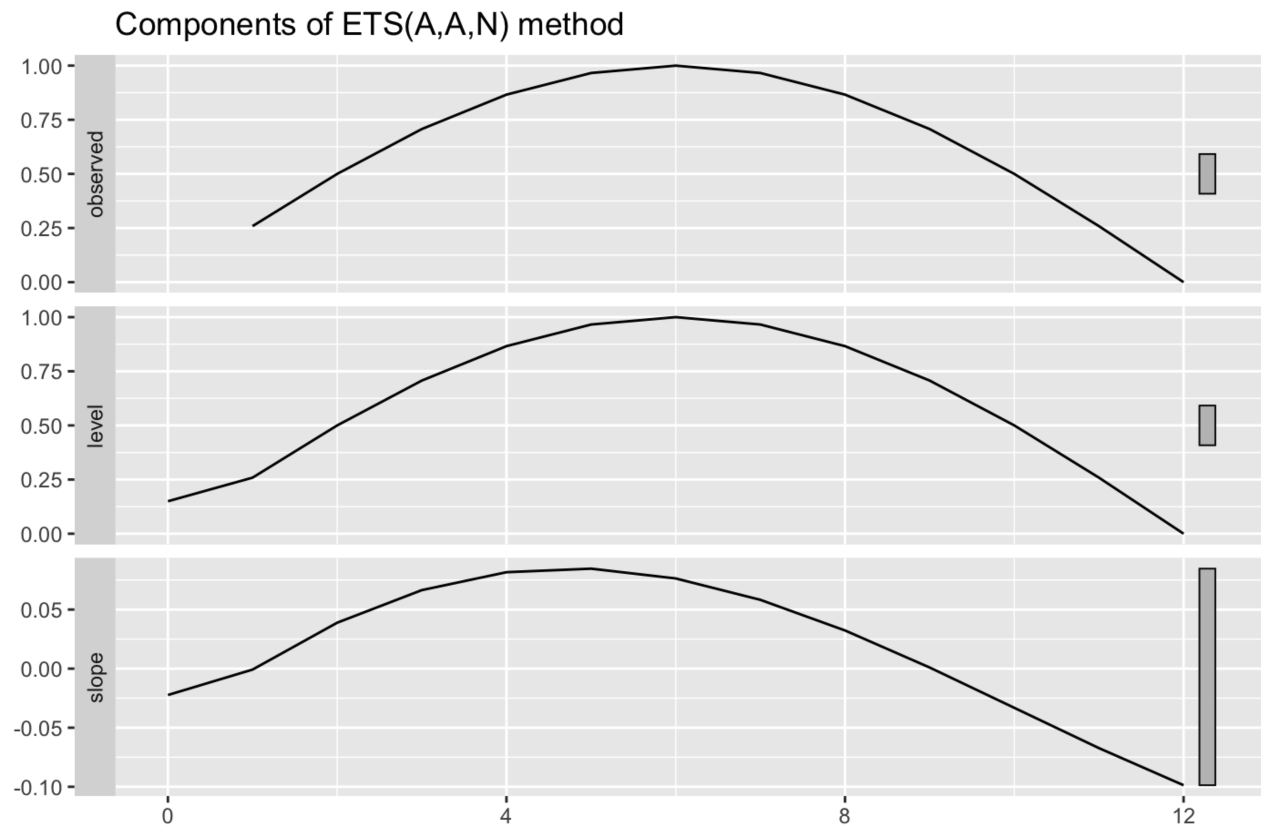
The estimated trend to use depends on the β (weighting). In this case, it is a weighted average of past trends (`diff(dat)`) which make sense as the trend keeps changing.

```
plot(diff(dat), type="p",pch=2)
points(fit$states[3:13,2],pch=3)
legend("topright",c("data","b(t)"),pch=c(2,3))
```



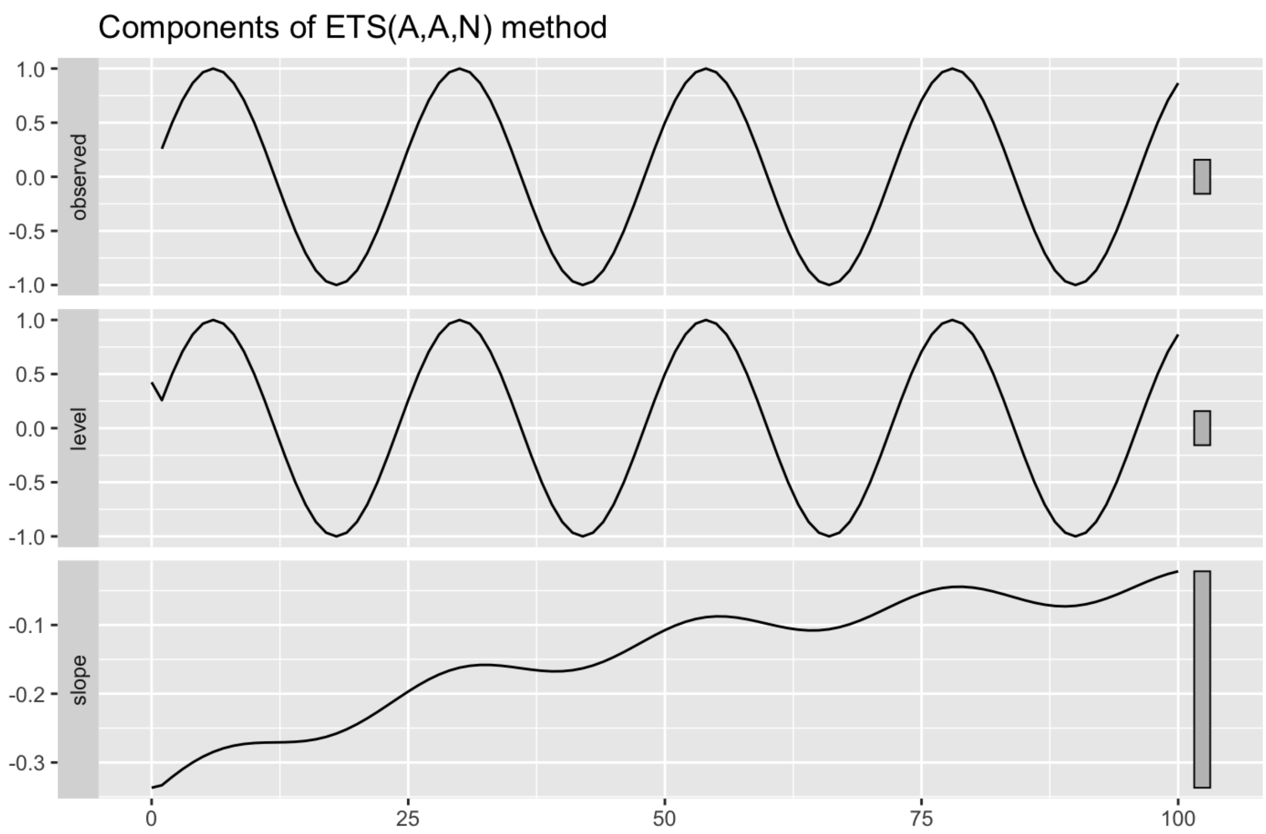
Estimated ETS with level and trend

```
autoplot(fit)
```



Let's look at longer data

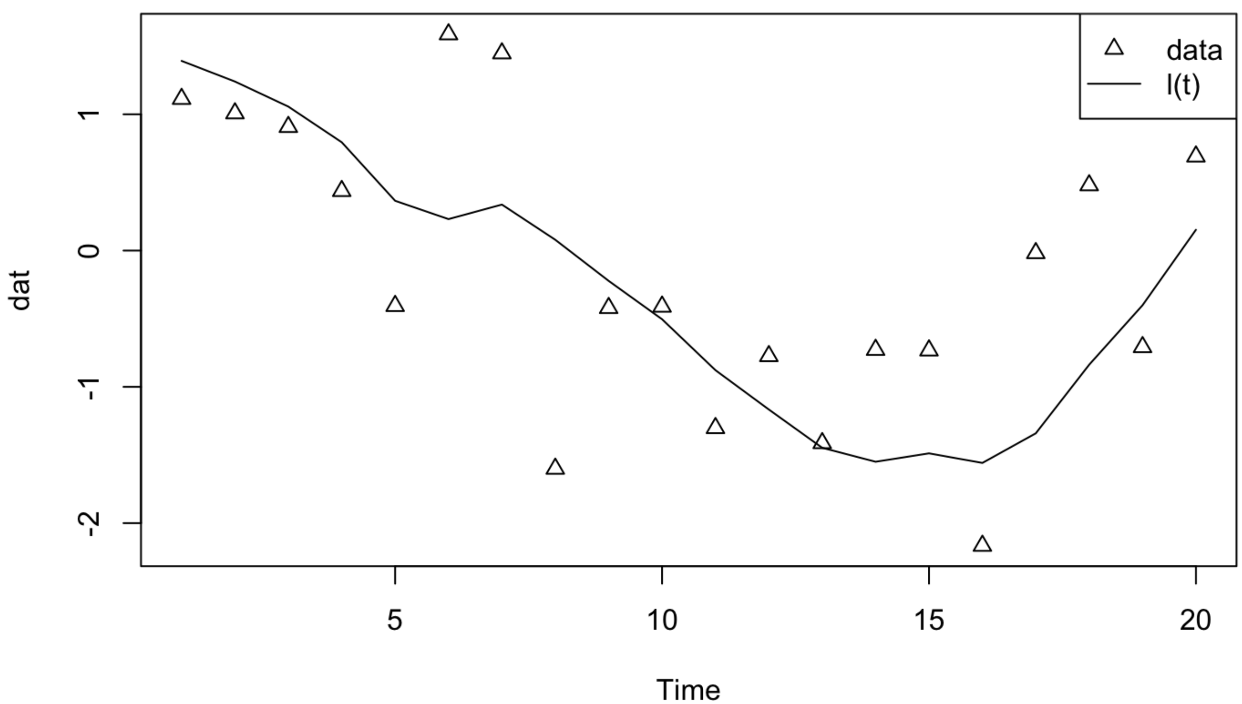
```
dat <- sin((1:100)*pi/12)
fit <- ets(dat, model="AAN", damped=FALSE)
autoplot(fit)
```



Add error

Now the best level (line through the data) is smoothed.

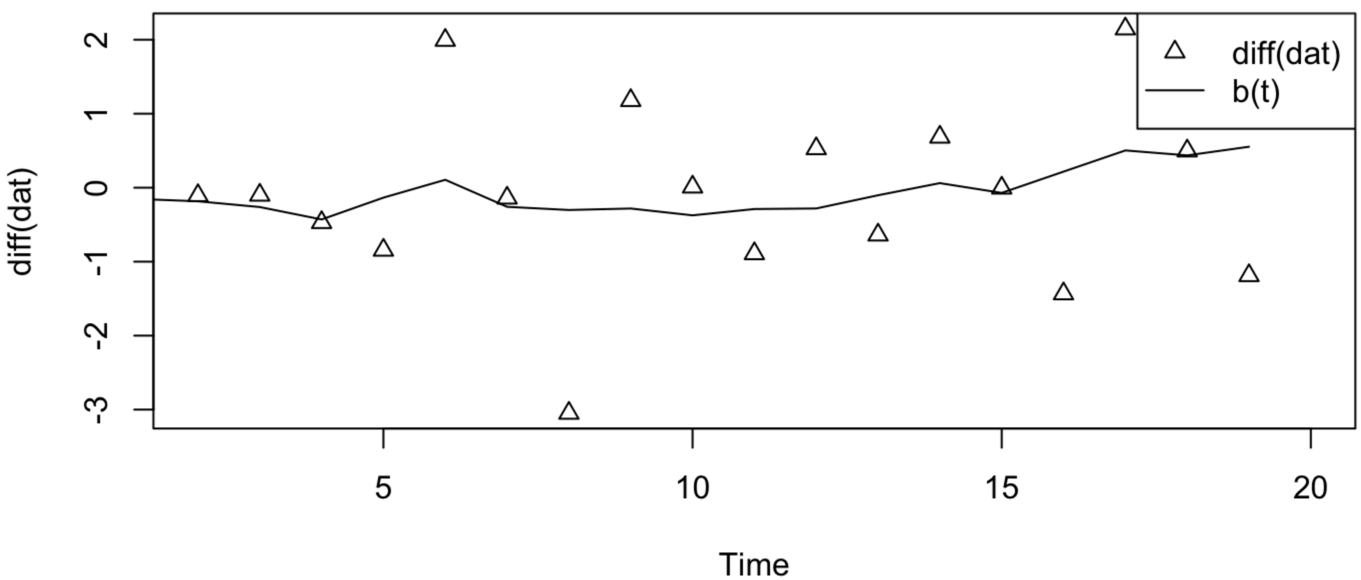
Let's imagine that our data look like so:



Estimated trend

$$\hat{y}_{t+1|1:t} = l_t + b_t$$

The best trend b_t will take into account the estimated level and will be a smoothed value of the `diff(dat)`.



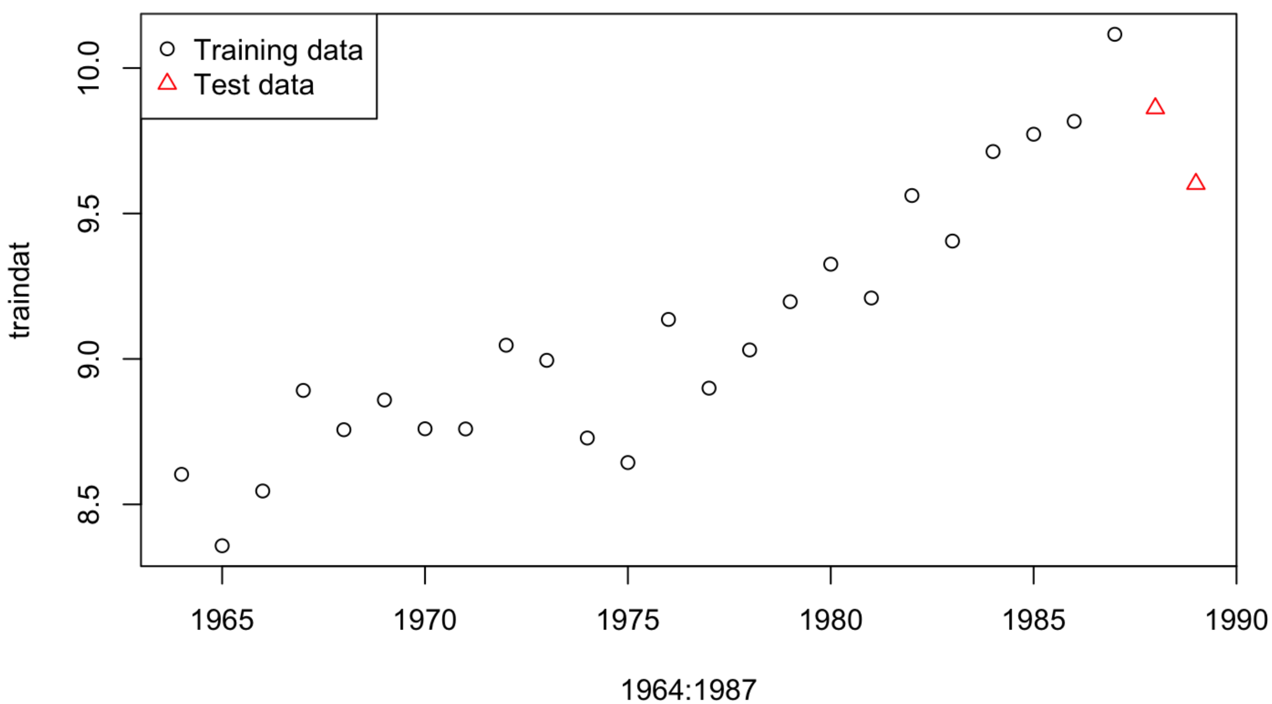
Validation

Once we have a model(s), we will want to evaluate the performance of the model. We'll see examples using the anchovy landings.

```
load("landings.RData")
spp <- "Anchovy"
training = subset(landings, Year <= 1987)
test = subset(landings, Year >= 1988 & Year <= 1989)
traindat <- subset(training, Species==spp)$log.metric.tons
testdat <- subset(test, Species==spp)$log.metric.tons
```

Measures of forecast fit

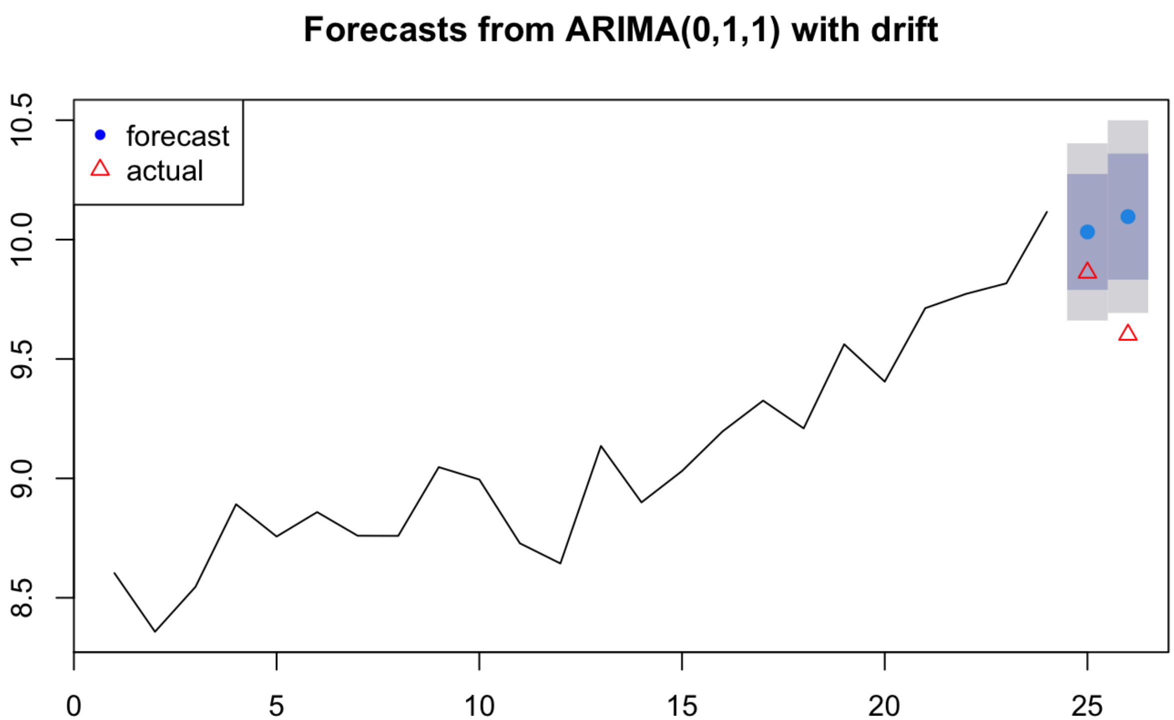
To measure the forecast fit, we fit a model to training data and test a forecast against data in a test set. We 'held out' the test data and did not use it at all in our fitting.



We will fit to the training data and make a forecast.

```
fr <- forecast(auto.arima(traindat), h=2)
fr
```

```
##    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 25       10.03216 9.789577 10.27475 9.661160 10.40317
## 26       10.09625 9.832489 10.36001 9.692861 10.49964
```



How to we quantify the difference between the forecast and the actual values?

```
fr.err <- testdat - fr$mean
fr.err
```

```
## Time Series:
## Start = 25
## End = 26
## Frequency = 1
## [1] -0.1704302 -0.4944778
```

There are many metrics. The `accuracy()` function in forecast provides many different metrics: mean error, root mean square error, mean absolute error, mean percentage error, mean absolute percentage error.

ME Mean error

```
me <- mean(fr.err); me
```

```
## [1] -0.332454
```

RMSE Root mean squared error

```
rmse <- sqrt(mean(fr.err^2)); rmse
```

```
## [1] 0.3698342
```

MAE Mean absolute error

```
mae <- mean(abs(fr.err)); mae
```

```
## [1] 0.332454
```

MPE Mean percentage error

```
fr.pe <- 100*fr.err/testdat
mpe <- mean(fr.pe); mpe
```

```
## [1] -3.439028
```

MAPE Mean absolute percentage error

```
mape <- mean(abs(fr.pe)); mape
```

```
## [1] 3.439028
```

```r
accuracy(fr, testdat)[,1:5]
```

```
##                       ME      RMSE       MAE        MPE     MAPE
## Training set -0.00473511 0.1770653 0.1438523 -0.1102259 1.588409
## Test set     -0.33245398 0.3698342 0.3324540 -3.4390277 3.439028
```

```r
c(me, rmse, mae, mpe, mape)
```

```
## [1] -0.3324540  0.3698342  0.3324540 -3.4390277  3.4390277
```

Test all the models in your candidate set

Compute metrics for all the models in your candidate set.

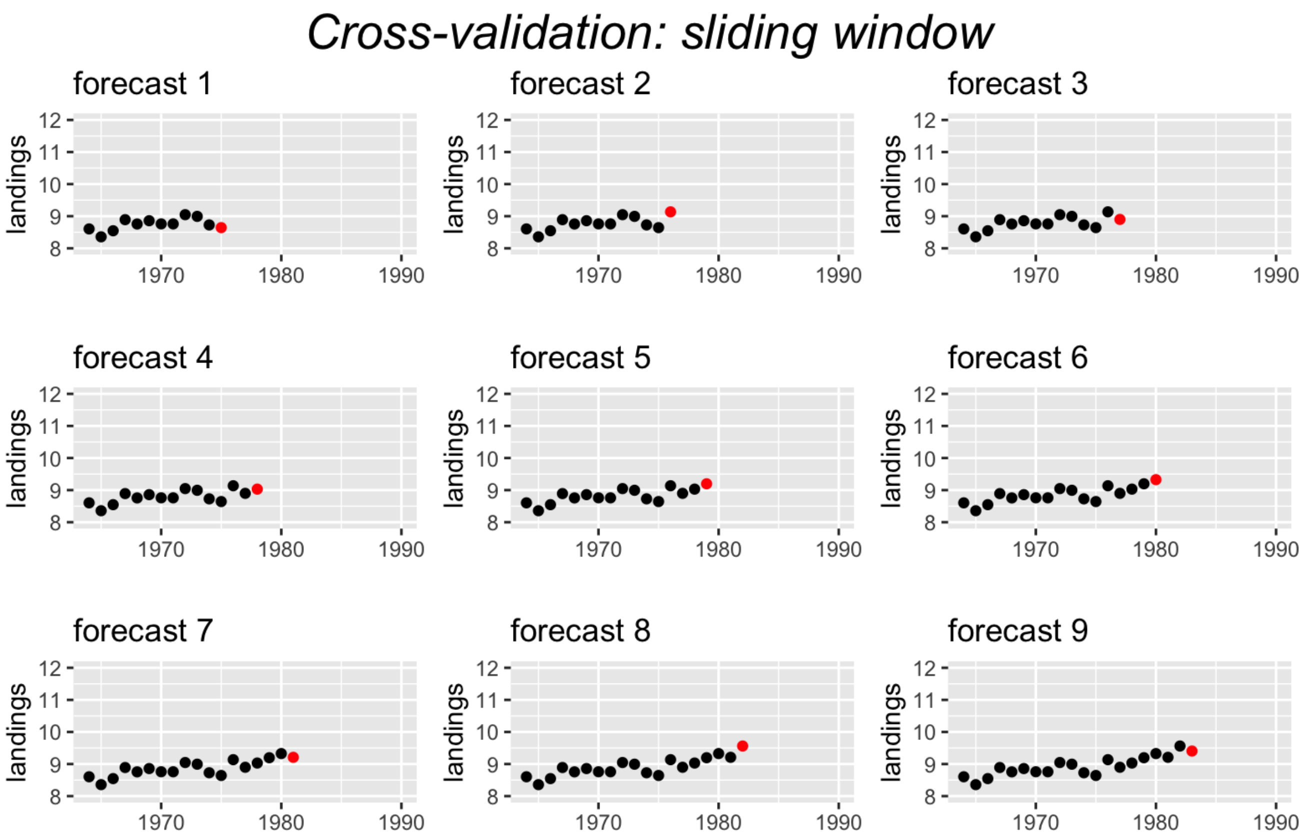
```
# The model picked by auto.arima
fit1 <- Arima(traindat, order=c(0,1,1))
fr1 <- forecast(fit1, h=2)
test1 <- accuracy(fr1, testdat)[2,1:5]

# AR-1
fit2 <- Arima(traindat, order=c(1,1,0))
fr2 <- forecast(fit2, h=2)
test2 <- accuracy(fr2, testdat)[2,1:5]

# Naive model with drift
fit3 <- rwf(traindat, drift=TRUE)
fr3 <- forecast(fit3, h=2)
test3 <- accuracy(fr3, testdat)[2,1:5]
```

Show a summary

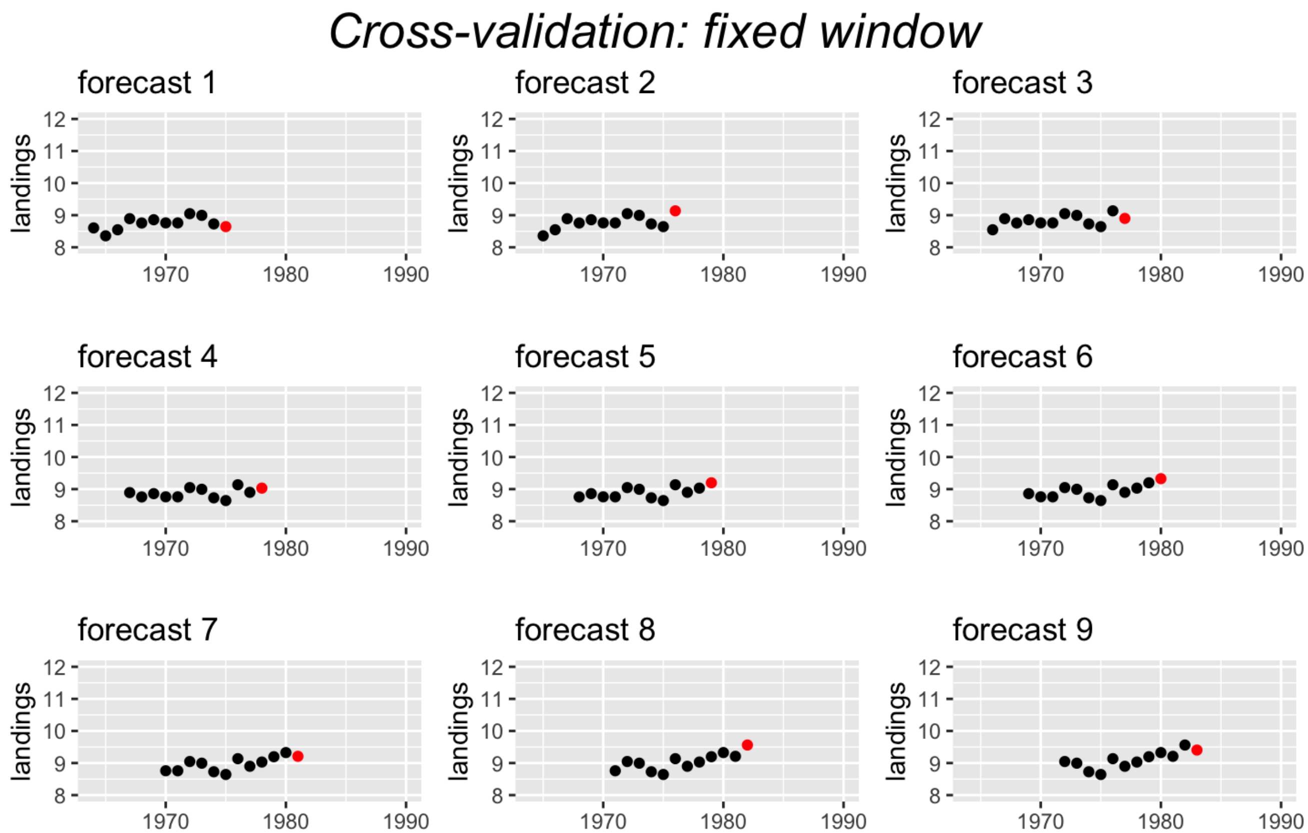
	ME	RMSE	MAE	MPE	MAPE
(0,1,1)	-0.293	0.320	0.293	-3.024	3.024
(1,1,0)	-0.309	0.341	0.309	-3.200	3.200
Naive	-0.483	0.510	0.483	-4.985	4.985

Cross-validation

An alternate approach to testing a model's forecast accuracy is to use windows or shorter segments of the whole time series to make a series of single forecasts.



Another approach uses a fixed window. For example, a 10-year window.



Time-series cross-validation with the forecast package

```
far2 <- function(x, h, order){
  forecast(Arima(x, order=order), h=h)
  }
e <- tsCV(traindat, far2, h=1, order=c(0,1,1))
tscv1 <- c(ME=mean(e, na.rm=TRUE), RMSE=sqrt(mean(e^2, na.rm=TRUE)),
           MAE=mean(abs(e), na.rm=TRUE))
tscv1
```

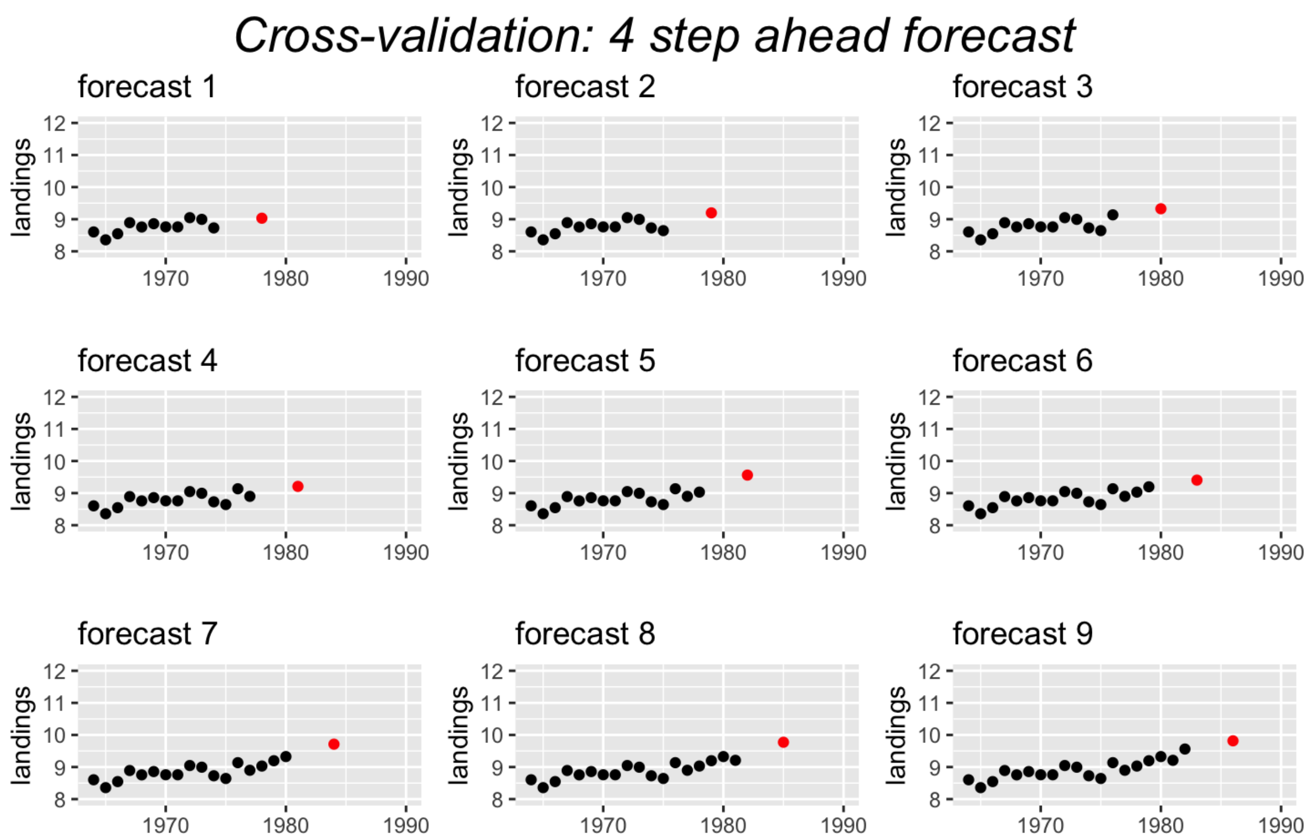
```
##        ME      RMSE       MAE
## 0.1128788 0.2261706 0.1880392
```

Compare to RMSE from just the 2 test data points.

```
test1[c("ME","RMSE","MAE")]
```

```
##         ME       RMSE        MAE
## -0.2925326  0.3201093  0.2925326
```

Cross-validation farther in future



Compare accuracy of forecasts 1 year out versus 4 years out. If `h` is greater than 1, then the errors are returned as a matrix with each `h` in a column. Column 4 is the forecast, 4 years out.

```
e <- tsCV(traindat, far2, h=4, order=c(0,1,1))[,4]
#RMSE
tscv4 <- c(ME=mean(e, na.rm=TRUE), RMSE=sqrt(mean(e^2, na.rm=TRUE)),
           MAE=mean(abs(e), na.rm=TRUE))
rbind(tscv1, tscv4)
```

```
##              ME      RMSE       MAE
## tscv1 0.1128788 0.2261706 0.1880392
## tscv4 0.2839064 0.3812815 0.3359689
```

Compare accuracy of forecasts with a fixed 10-year window and 1-year out forecasts.

```
e <- tsCV(traindat, far2, h=1, order=c(0,1,1), window=10)
#RMSE
tscvf1 <- c(ME=mean(e, na.rm=TRUE), RMSE=sqrt(mean(e^2, na.rm=TRUE)),
            MAE=mean(abs(e), na.rm=TRUE))
tscvf1
```

```
##        ME      RMSE       MAE
## 0.1387670 0.2286572 0.1942840
```

```
comp.tab <- rbind(test1=test1[c("ME","RMSE","MAE")],
      slide1=tscv1,
      slide4=tscv4,
      fixed1=tscvf1)
knitr::kable(comp.tab, format="html")
```

	ME	RMSE	MAE
test1	-0.2925326	0.3201093	0.2925326
slide1	0.1128788	0.2261706	0.1880392
slide4	0.2839064	0.3812815	0.3359689
fixed1	0.1387670	0.2286572	0.1942840

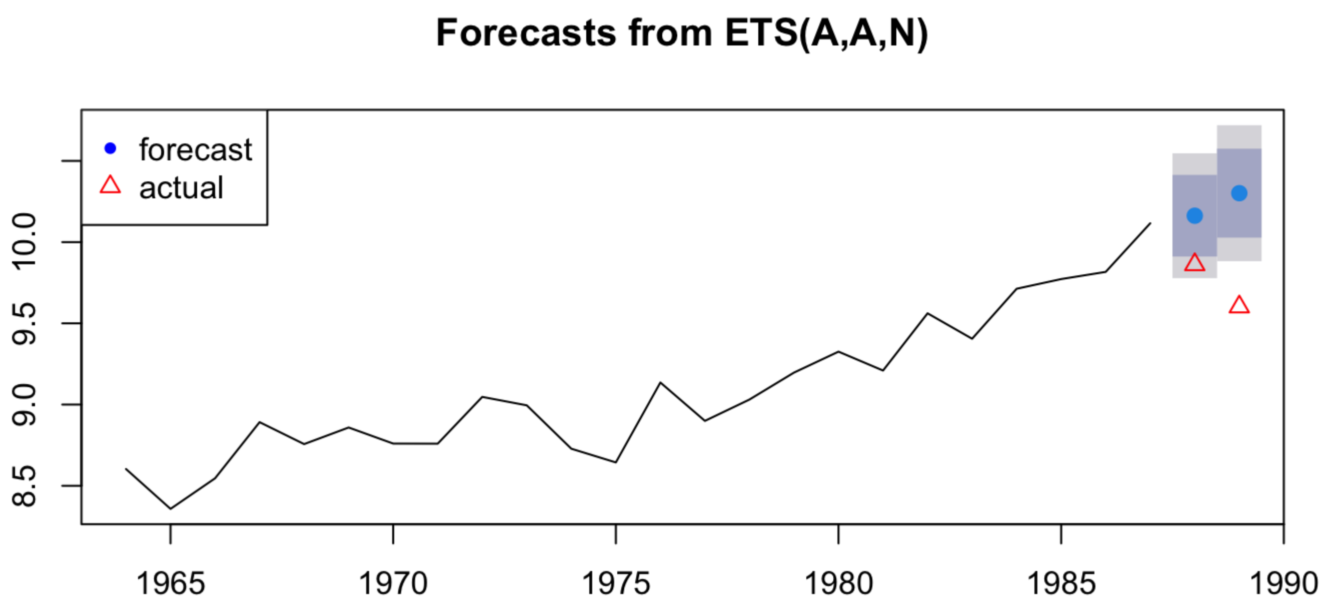
Forecast performance for ETS models

We can evaluate the forecast performance with forecasts of our test data or we can use all the data and use time-series cross-validation.

Let's start with the former.

Test forecast performance against a test data set

We will fit an an ETS model with trend to the training data and make a forecast for the years that we 'held out'.



We can calculate a variety of forecast error metrics with

```
accuracy(fr, testdat)
```

```
##                       ME      RMSE       MAE        MPE     MAPE      MASE
## Training set  0.0155561 0.1788989 0.1442712  0.1272938 1.600532 0.7720807
## Test set     -0.5001701 0.5384355 0.5001701 -5.1678506 5.167851 2.6767060
##                       ACF1 Theil's U
## Training set -0.008371542        NA
## Test set     -0.500000000  2.690911
```

We would now repeat this for all the models in our candidate set and choose the model with the best forecast performance.

Forecast performance with time-series cross-validation

We can use `tsCV()` as we did for ARIMA models. We will redefine `traindat` as all our Anchovy data.

tsCV() function

We will use the `tsCV()` function. We need to define a function that returns a forecast.

```
far2 <- function(x, h, model){
  fit <- ets(x, model=model)
  forecast(fit, h=h)
  }
```

Now we can use `tsCV()` to run our `far2()` function to a series of training data sets. We will specify that a NEW ets model be estimated for each training set. We are not using the weighting estimated for the whole data set but estimating the weighting new for each set.

The `e` are our forecast errors for all the forecasts that we did with the data.

```
e <- tsCV(traindat, far2, h=1, model="AAN")
e
```

```
## Time Series:
## Start = 1
## End = 26
## Frequency = 1
##  [1] -0.245378390  0.366852341  0.419678595 -0.414861770 -0.152727933
##  [6] -0.183775208 -0.013799590  0.308433377 -0.017680471 -0.329690537
## [11] -0.353441463  0.266143346 -0.110848618 -0.005227309  0.157821831
## [16]  0.196184446  0.008135667  0.326024067  0.085160559  0.312668447
## [21]  0.246437781  0.117274740  0.292601670 -0.300814605 -0.406118961
## [26]           NA
```

Let's look at the first few `e` so we see exactly with `tsCV()` is doing.

```
e[2]
```

```
## [1] 0.3668523
```

This uses training data from $t = 1$ to $t = 2$ so fits an ets to the first two data points alone. Then it creates a forecast for $t = 3$ and compares that forecast to the actual value observed for $t = 3$.

```
TT <- 2 # end of the temp training data
temp <- traindat[1:TT]
fit.temp <- ets(temp, model="AAN")
fr.temp <- forecast(fit.temp, h=1)
traindat[TT+1] - fr.temp$mean
```

```
## Time Series:
## Start = 3
## End = 3
## Frequency = 1
## [1] 0.3668523
```

```
e[3]
```

```
## [1] 0.4196786
```

This uses training data from $t = 1$ to $t = 2$ so fits an ets to the first two data points alone. Then it creates a forecast for $t = 3$ and compares that forecast to the actual value observed for $t = 3$.

```
TT <- 3 # end of the temp training data
temp <- traindat[1:TT]
fit.temp <- ets(temp, model="AAN")
fr.temp <- forecast(fit.temp, h=1)
traindat[TT+1] - fr.temp$mean
```

```
## Time Series:
## Start = 4
## End = 4
## Frequency = 1
## [1] 0.4196786
```

Testing a specific ETS model

By specifying `model="AAN"`, we estimated a new ets model (meaning new weighting) for each training set used. We might want to specify that we use only the weighting we estimated for the full data set.

We do this by passing in a fit to `model`.

The `e` are our forecast errors for all the forecasts that we did with the data. `fit1` below is the ets estimated from all the data 1964 to 1989. Note, the code will produce a warning that it is estimating the initial value and just using the weighting. That is what we want.

```r
fit1 <- ets(traindat, model="AAN")
e <- tsCV(traindat, far2, h=1, model=fit1)
e
```

```
## Time Series:
## Start = 1
## End = 26
## Frequency = 1
##  [1]           NA  0.576663901  1.031385937  0.897828249  1.033164616
##  [6]  0.935274283  0.958914499  1.265427119 -0.017241938 -0.332751184
## [11] -0.330473144  0.255886314 -0.103926617  0.031206730  0.154727479
## [16]  0.198328366 -0.020605522  0.297475742  0.005297401  0.264939892
## [21]  0.196256334  0.129798648  0.335887872 -0.074017535 -0.373267163
## [26]           NA
```

Forecast accuracy metrics

Now we can compute forecast accuracy metrics from the forecast errors (`e`).

RMSE: root mean squared error

```
rmse <- sqrt(mean(e^2, na.rm=TRUE))
```

MAE: mean absolute error

```
mae <- mean(abs(e), na.rm=TRUE)
```

We would repeat this process for all models in our candidate set and compare forecast fits to choose our forecast model.

Comparing performance in a candidate set

Now you are ready to compare forecasts from a variety of models and choose a forecast model. Note when you compare models, you can use both ‘training data/test data’ and use time-series cross-validation, but report the metrics in separate columns. Example, ‘RMSE from tsCV’ and ‘RMSE from test data’.

Example candidate set for anchovy

- ETS model with trend

```
fit <- ets(traindat, model="AAN")
fr <- forecast(fit, h=1)
```

- ETS model no trend

```
fit <- ets(traindat, model="ANN")
fr <- forecast(fit, h=1)
```

- ARIMA(0,1,1) with drift (best)

```
fit <- Arima(traindat, order(0,1,1), include.drift=TRUE)
fr <- forecast(fit, h=1)
```

- ARIMA(2,1,0) with drift (within 2 AIC of best)

```
fit <- Arima(traindat, order(2,1,0), include.drift=TRUE)
fr <- forecast(fr)
```

Candidate models continued

- Time-varying regression with linear time

```
traindat$t <- 1:24
fit <- lm(log.metric.tons ~ t, data=traindat)
fr <- forecast(fit, newdata=data.frame(t=25))
```

- Naive no trend

```
fit <- Arima(traindat, order(0,1,0))
fr <- forecast(fit, h=1)
# or simply
fr <- rwf(traindat)
```

- Naive with trend

```
fit <- Arima(traindat, order(0,1,0), include.drift=TRUE)
fr <- forecast(fit)
# or simply
fr <- rwf(traindat, drift=TRUE)
```

Candidate models continued

- Average or mean

```
fit <- Arima(traindat, order(0,0,0))
fr <- forecast(fit)
```

Seasonal ETS models

```r
data("chinook", package="atsalibrary")
head(chinook.month)
```

Year	Month	Species	State	log.metric.tons	metric.tons	value.usd
1990	Jan	Chinook	WA	3.26	26.1	108685
1990	Feb	Chinook	WA	3.78	43.7	309196
1990	Mar	Chinook	WA	3.47	32.1	201279
1990	Apr	Chinook	WA	4.23	69	433238
1990	May	Chinook	WA	5.09	162	876329
1990	Jun	Chinook	WA	4.28	71.9	421885

The data are monthly and start in January 1990. To make this into a ts object do

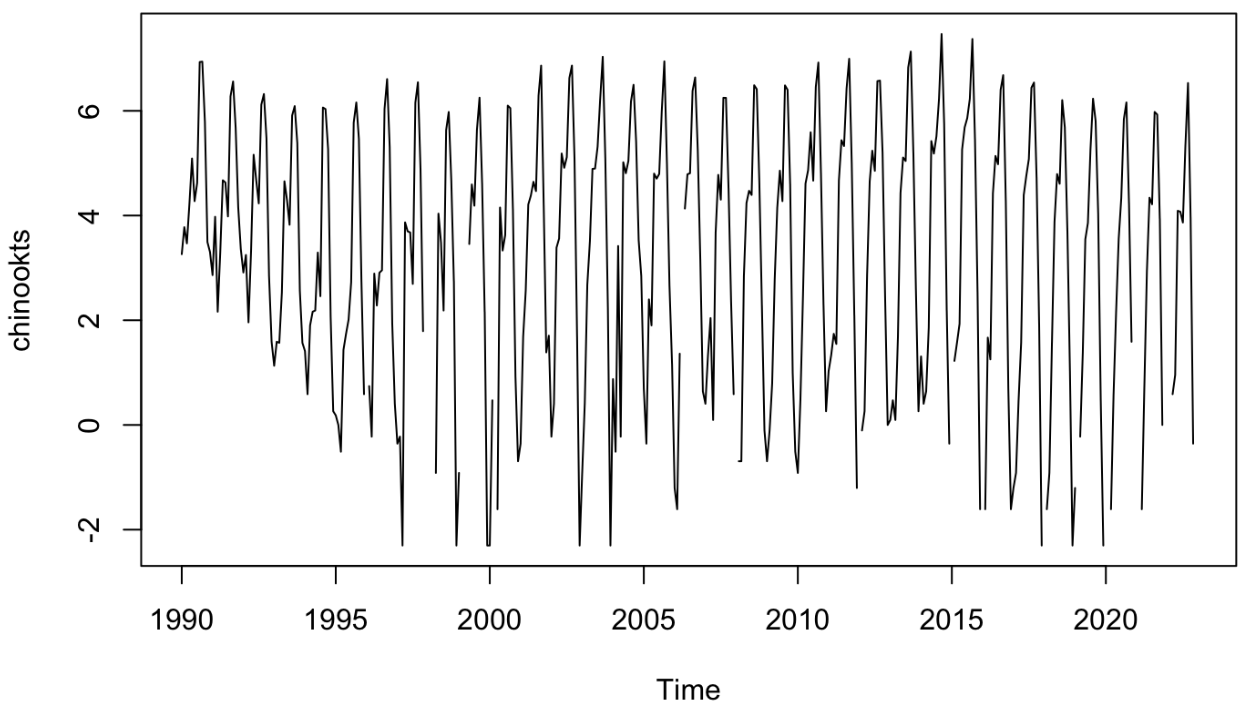
```
df <- chinook.month %>% subset(State == "WA")
chinookts <- ts(df$log.metric.tons, start=c(1990,1), frequency=12)
```

`start` is the year and month and frequency is the number of months in the year.

Plot seasonal data

Now that we have specified our seasonal data as a ts object, it is easy to plot because R knows what the season is.

```
plot(chinookts)
```



Seasonal ETS model

Now we add a few more lines to our ETS table of models:

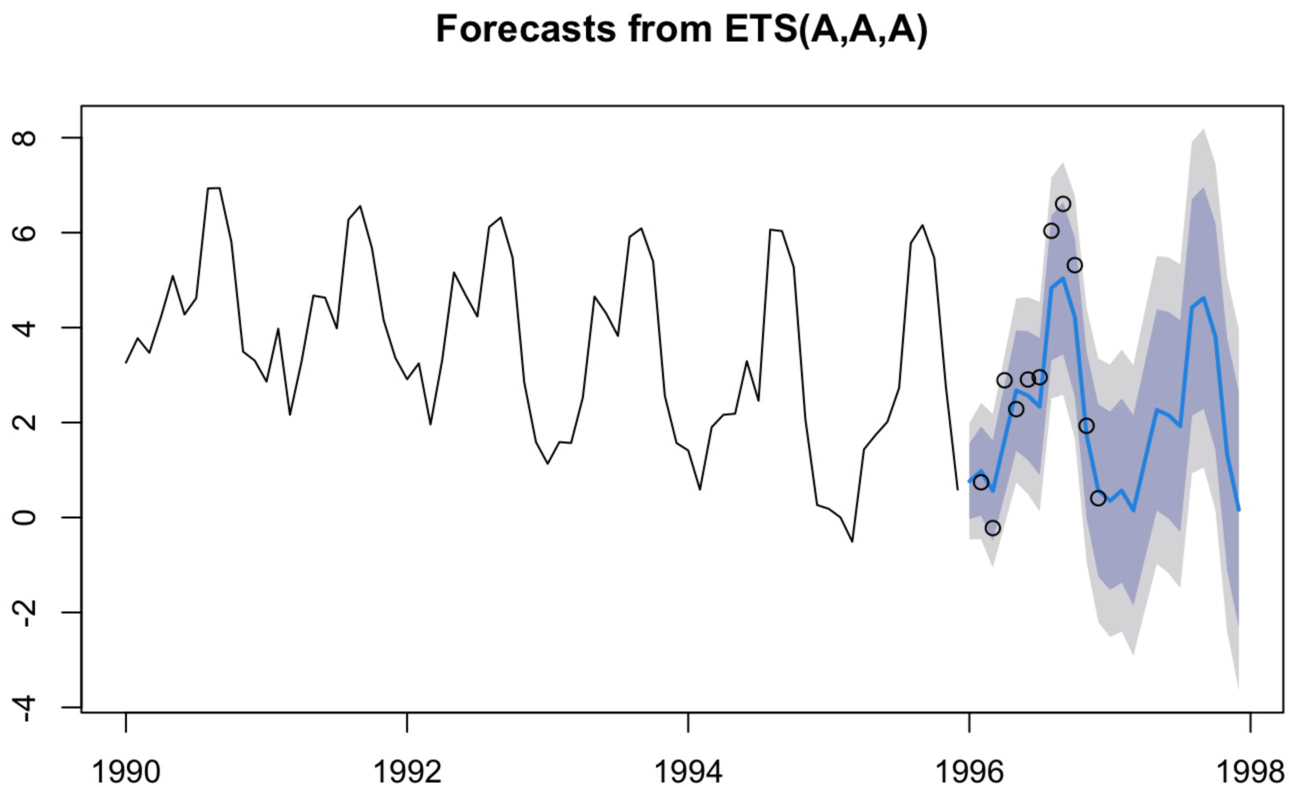
model	"ZZZ"	alternate function
ETS no trend	"ANN"	`ses()`
ETS with trend	"AAN"	`holt()`
ETS with season no trend	"ANA"	NA
ETS with season and trend	"AAA"	NA
estimate best trend and season model	"ZZZ"	NA

Unfortunately `ets()` will not handle missing values and will find the longest continuous piece of our data and use that.

```r
library(forecast)
traindat <- window(chinookts, c(1990,1), c(1999,12))
fit <- ets(traindat, model="AAA")
```

```
## Warning in ets(traindat, model = "AAA"): Missing values encountered. Using
## longest contiguous portion of time series
```

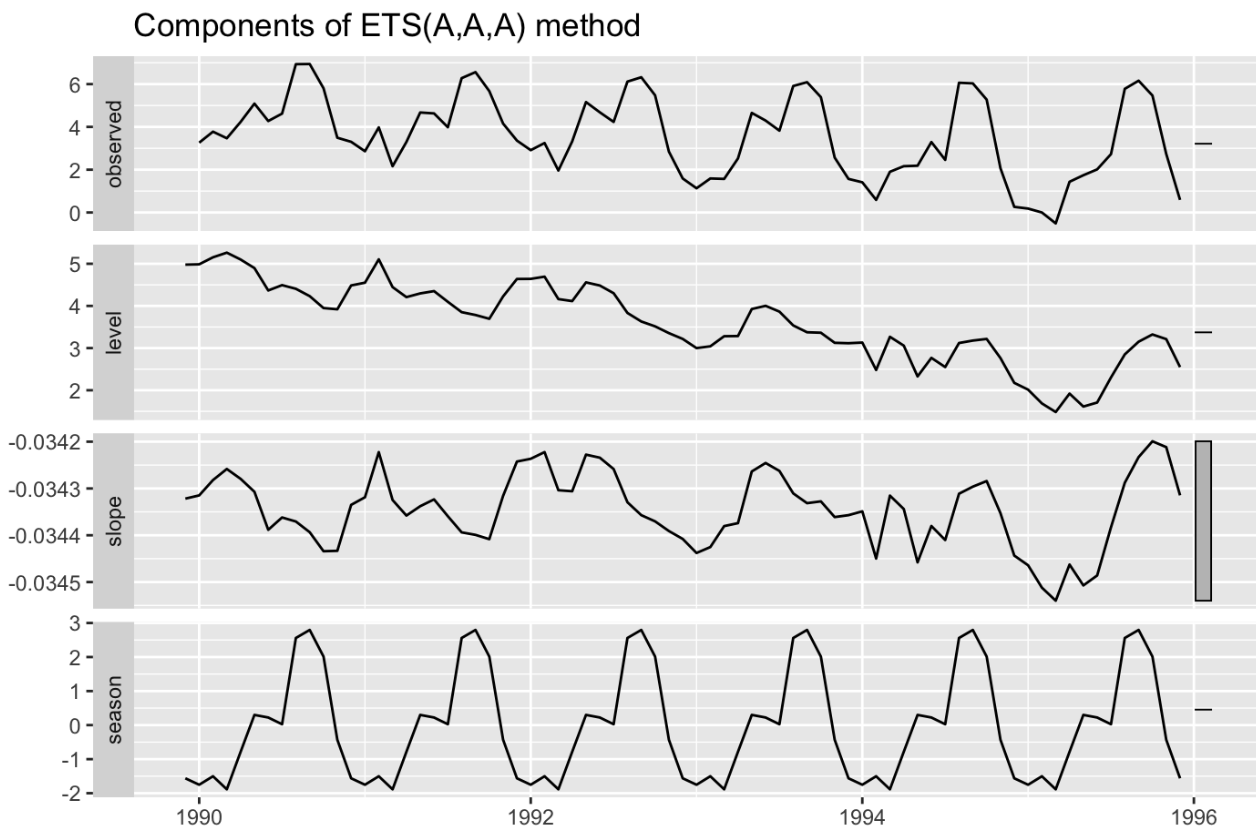
```
fr <- forecast(fit, h=24)
plot(fr)
points(window(chinookts, c(1996,1), c(1996,12)))
```



Decompose

If we plot the decomposition, we see the the seasonal component is not changing over time, unlike the actual data. The bar on the right, alerts us that the scale on the 3rd panel is much smaller.

```
autoplot(fit)
```



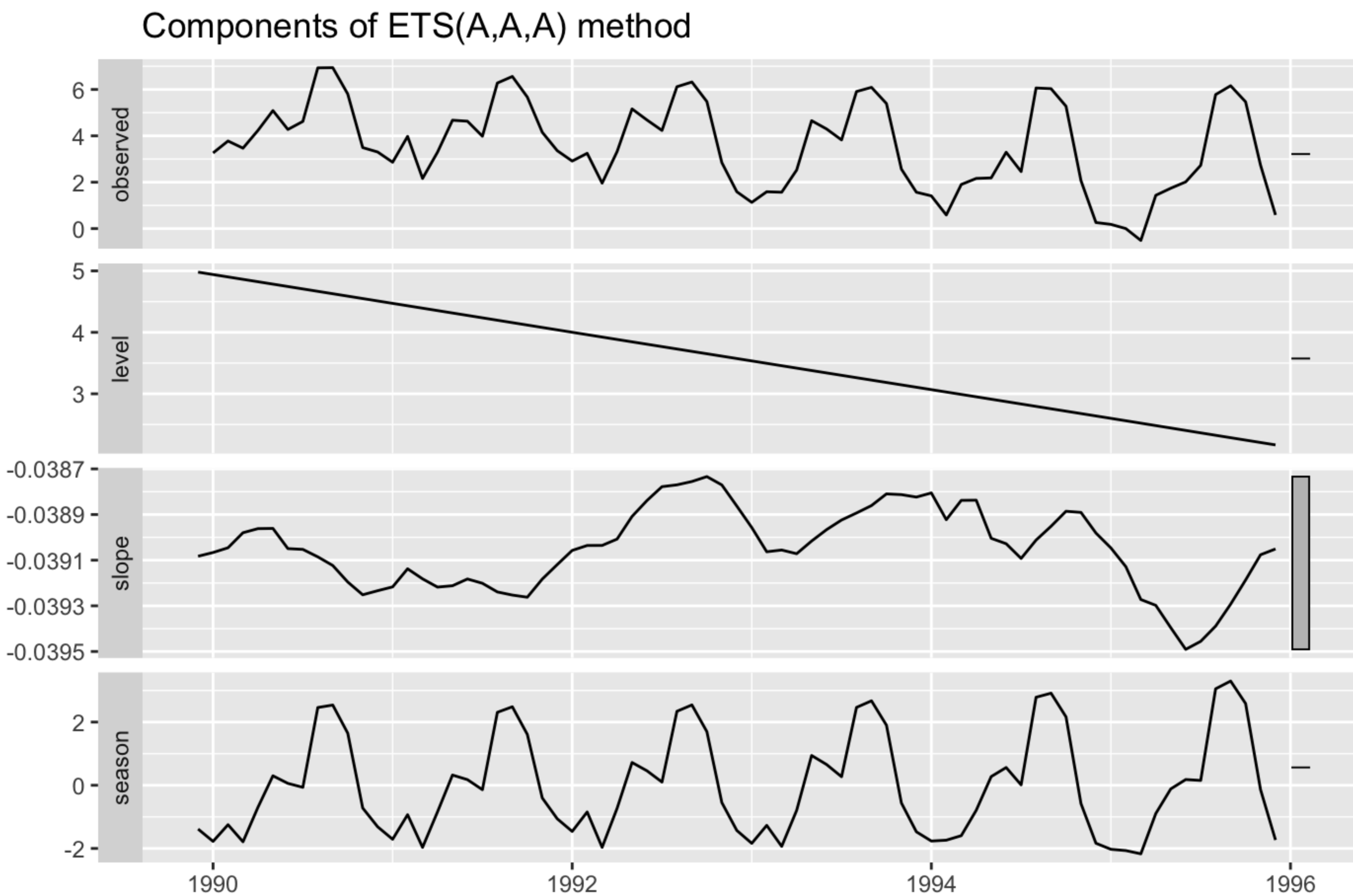
Force seasonality to evolve more

Pass in a high `gamma` (the season weighting) to force the seasonality to evolve.

```
fit <- ets(traindat, model="AAA", gamma=0.4)
```

```
## Warning in ets(traindat, model = "AAA", gamma = 0.4): Missing values
## encountered. Using longest contiguous portion of time series
```

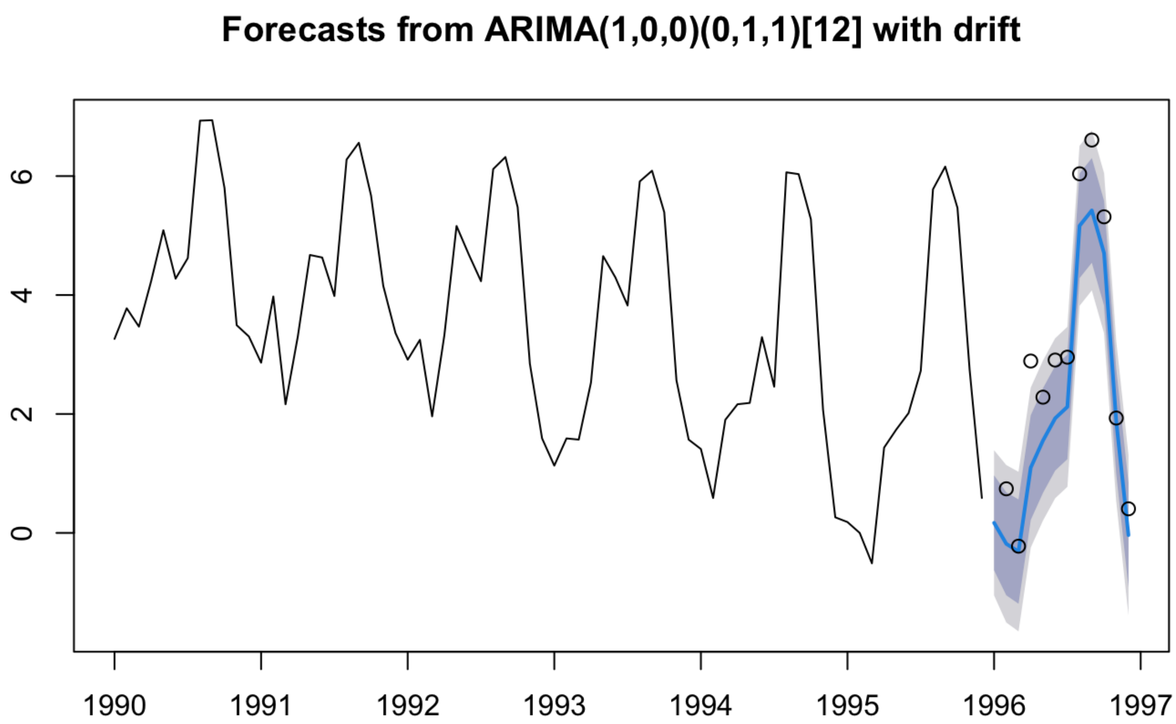
```
autoplot(fit)
```



Compare to a seasonal ARIMA model

`auto.arima()` will recognize that our data has season and fit a seasonal ARIMA model to our data. Let's use the data that `ets()` used. This is shorter than our training data. The data used by `ets()` is returned in `fit$x`.

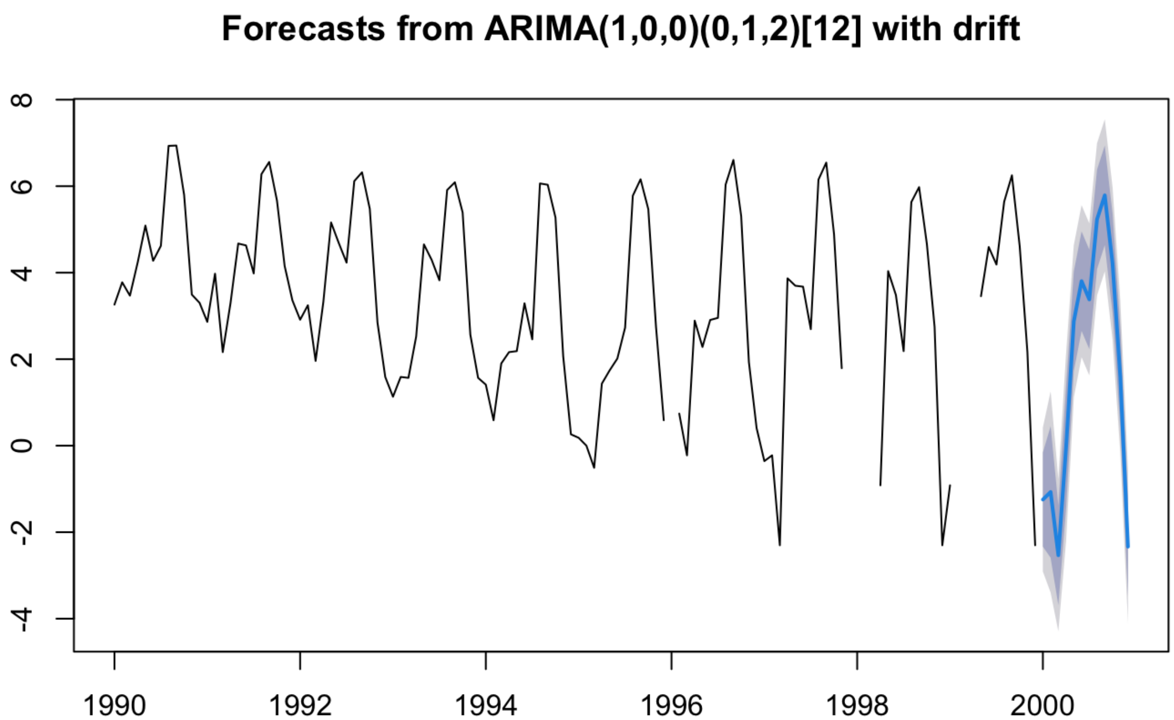
```
no_miss_dat <- fit$x
fit <- auto.arima(no_miss_dat)
fr <- forecast(fit, h=12)
plot(fr)
points(window(chinookts, c(1996,1), c(1996,12)))
```



Missing values

Missing values are ok when fitting a seasonal ARIMA model, unlike an ETS model.

```
fit <- auto.arima(traindat)
fr <- forecast(fit, h=12)
plot(fr)
```



Forecast evaluation

We can compute the forecast performance metrics as usual.

```
fit <- ets(traindat, model="AAA", gamma=0.4)
```

```
## Warning in ets(traindat, model = "AAA", gamma = 0.4): Missing values
## encountered. Using longest contiguous portion of time series
```

```
fr <- forecast(fit, h=12)
```

Look at the forecast so you know what years and months to include in your test data. Pull those 12 months out of your data using the `window()` function.

```
testdat <- window(traindat, c(1996,1), c(1996,12))
```

Use `accuracy()` to get the forecast error metrics.

```
accuracy(fr, testdat)
```

```
##                       ME      RMSE       MAE      MPE     MAPE      MASE
## Training set 0.004427699 0.6241720 0.5009298     -Inf      Inf 0.8385617
## Test set     0.804818825 0.9537111 0.8236858 42.35362 42.35362 1.3788589
##                    ACF1 Theil's U
## Training set  0.4598478        NA
## Test set     -0.2445736 0.5659964
```

We can do the same for the ARIMA model.

```
no_miss_dat <- fit$x
fit <- auto.arima(no_miss_dat)
fr <- forecast(fit, h=12)
accuracy(fr, testdat)
```

```
##                       ME      RMSE       MAE     MPE     MAPE      MASE
## Training set 0.008756236 0.5535951 0.3948974    -Inf      Inf 0.6610624
## Test set     0.774509392 0.9045662 0.7745094 35.9946 43.38318 1.2965369
##                     ACF1 Theil's U
## Training set -0.03293199        NA
## Test set     -0.21056349 0.5722577
```