exercise_part03

Forecasting: Exponential Smoothing

Original Source

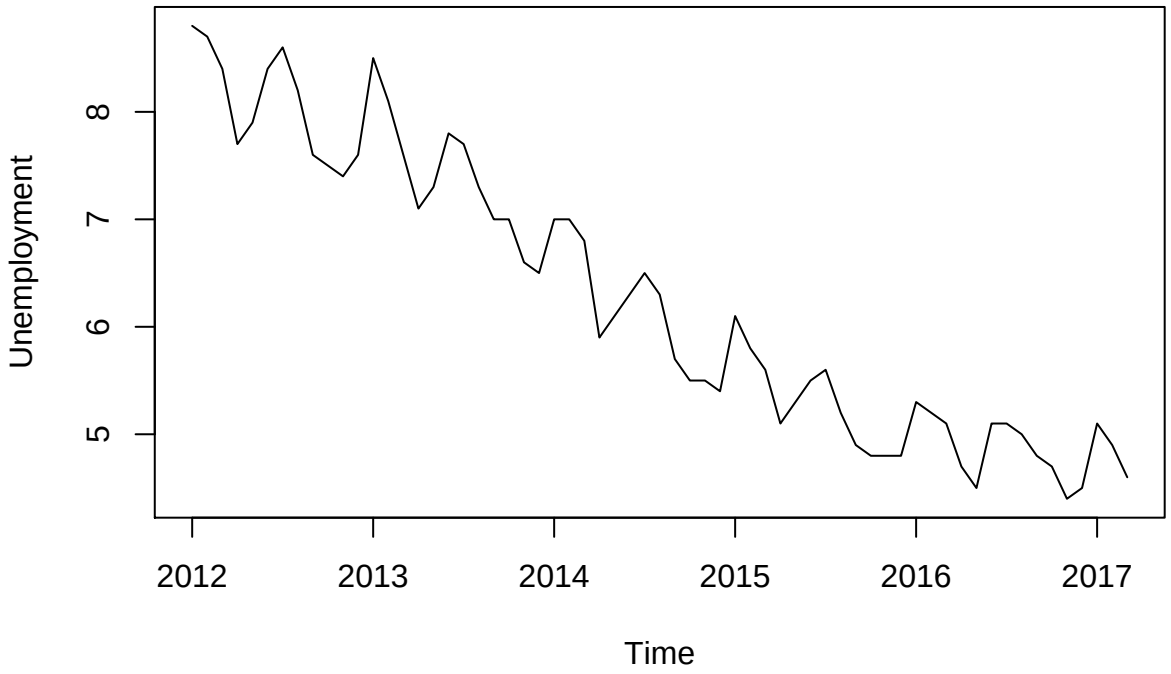
Exponetial smoothing is a method of finding patterns in time seires, which can be used to make forecasts. In its simple form, exponential smoothing is a weighted moving average: each smoothed value is a weighted average fo all past time series values (with weights decreassing exponetially from the most recent to the oldest values). In more complicated forms, exponential smoothing is applied to a time series recursively to allow for a trend and seasonality. In that case, the model is said to consist of three components - error, trend, and seasonality, from which another notation for exponential smoothing ("ETS") is derived.

This set of exercises focuses primarily on the ets function from the forecast package. The function can be used to apply various exponential smoothing methods (including Holt's and Holt-Winters' methods), and allows for both automatic and manual selection of the model structure (for example, whether the model includes trend and seasonal components).

Exercise 1

Load the data, transformm it to the ts type (indicating that the data is monthly and the first period is January 2012), and plot it.

```
df <- read.csv("unemployment.csv")
unempl <- ts(df, start=c(2012, 1), frequency = 12)
plot(unempl)
```



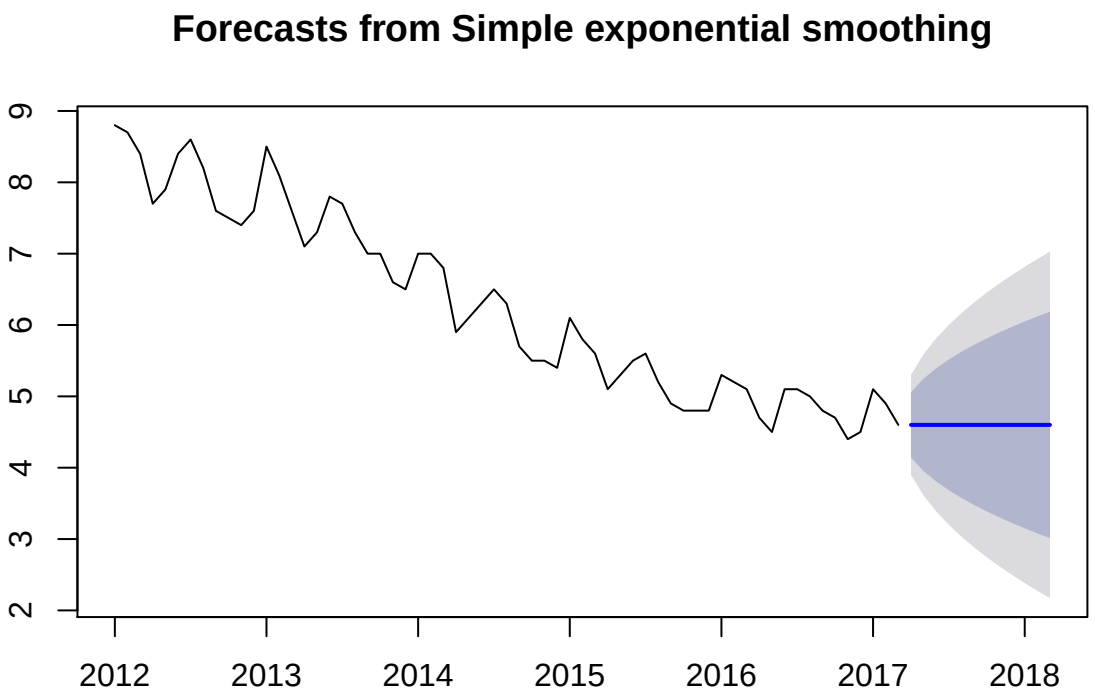
Exercise 2

Use the ses function from the forecast package to get a forecast based on simple exponential smoothing for the next 12 months, and plot the forecast.

```
require(forecast)
```

```
## Loading required package: forecast
```

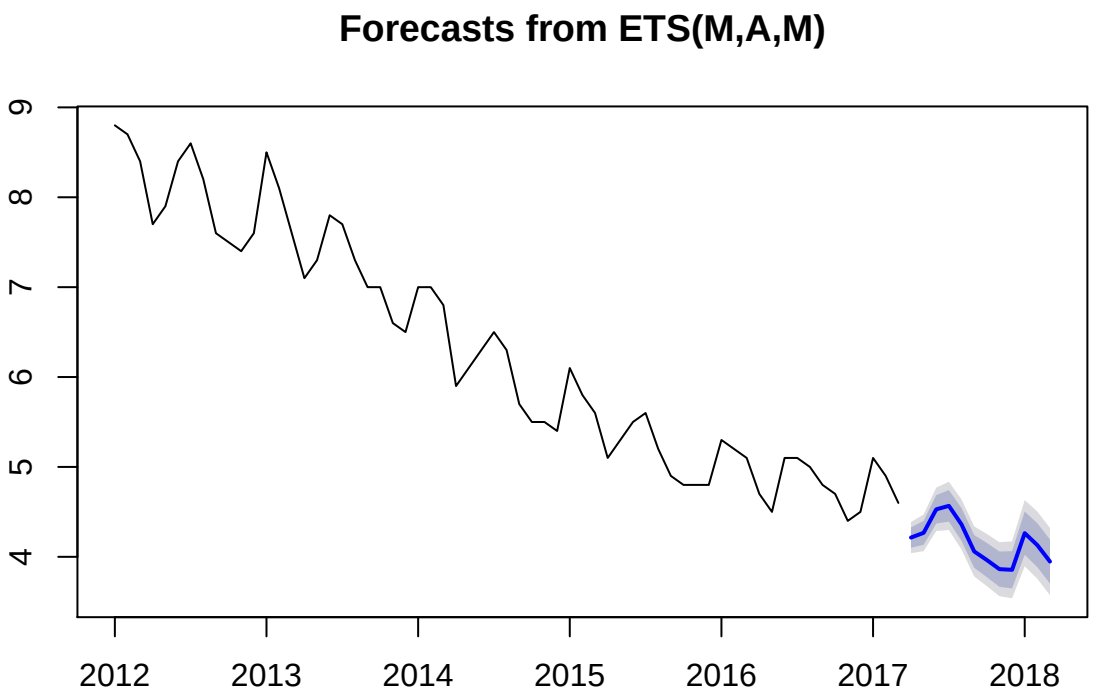
```
fcast_ses <- ses(unempl, h = 12)
plot(fcast_ses)
```



Exercise 3

Estimate an exponential smoothing model using the ets function with default parameters. Then pass the model as input to the forecast function to get a forecast for the next 12 months, and plot the forecast (both functions are from the forecast package).

```
fit_ets_default <- ets(unempl)
fcast_ets_default <- forecast(fit_ets_default, h = 12)
plot(fcast_ets_default)
```



Exercise 4

Print a summary of the model estimated in the previous exercise, and find the automatically estimated structure of the model. Does it include trend and seasonal components? If those components are present are they additive or multiplicative?

```
summary(fit_ets_default)
```

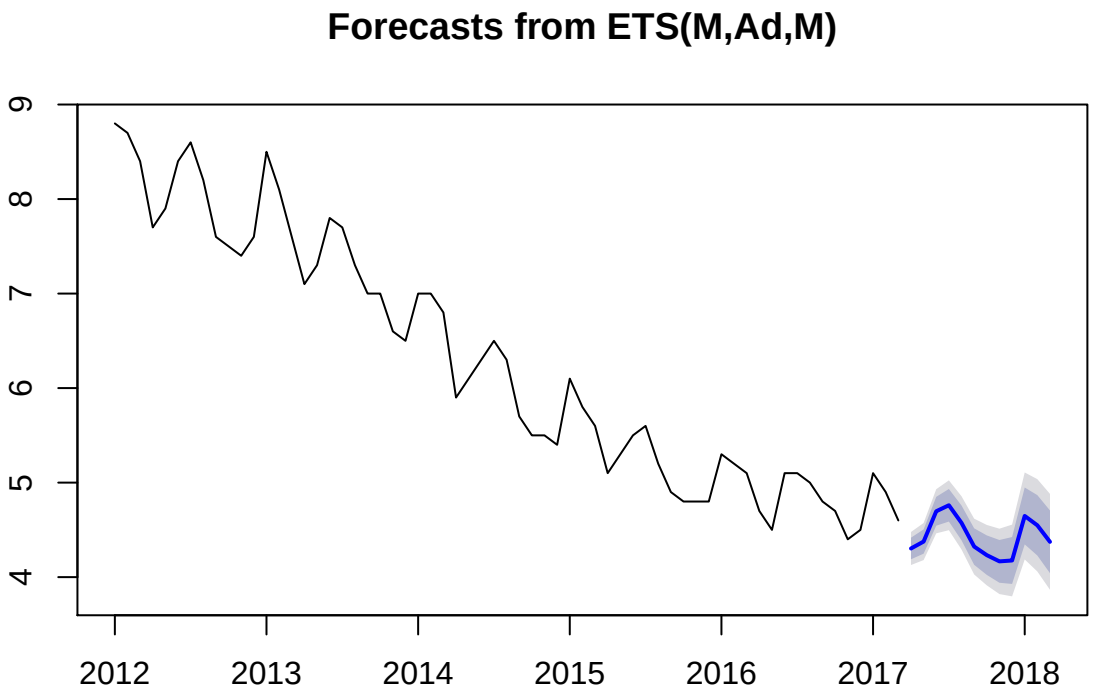
```
## ETS(M,A,M)
##
## Call:
##  ets(y = unempl)
##
##   Smoothing parameters:
##     alpha = 0.5717
##     beta  = 1e-04
##     gamma = 1e-04
##
##   Initial states:
##     l = 8.4353
##     b = -0.0564
##     s=0.9567 0.9453 0.957 0.9669 1.0245 1.0591
##            1.0365 0.9642 0.9403 1.0224 1.0542 1.0729
##
##   sigma:  0.021
##
##      AIC     AICc      BIC
## 37.38299 50.98299 73.81628
##
## Training set error measures:
##                       ME      RMSE       MAE        MPE     MAPE      MASE
```

```
## Training set -0.009791689 0.1271141 0.102154 -0.1209349 1.687472 0.1322298
##                     ACF1
## Training set 0.08649403
```

Exercise 5

Use the ets function to estimate an exponential smoothing model with a damped trend. Make a forecast based on the model for the next 12 months, and plot it.

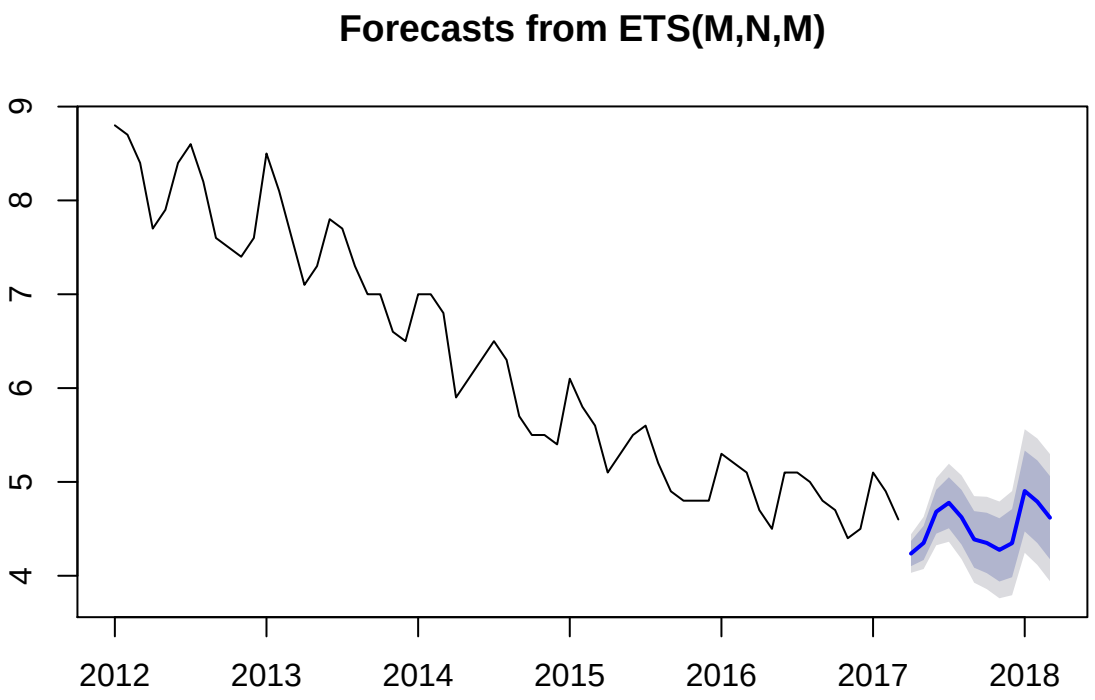
```
fit_ets_damped_trend <- ets(unempl, damped = TRUE)
fcast_ets_dampled_trend <- forecast(fit_ets_damped_trend, h = 12)
plot(fcast_ets_dampled_trend)
```



Exercise 6

Use the ets function to estimate another model that does not include a trend component. Make a forecast based on the model for the next 12 months, and plot it.

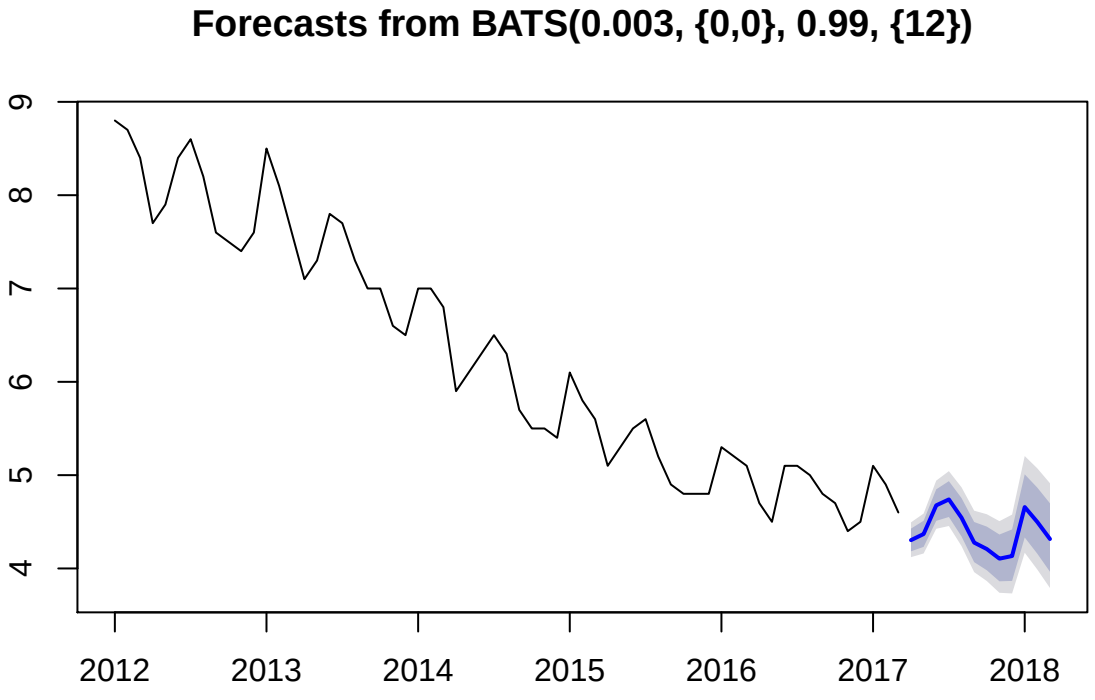
```
fit_ets_no_trend <- ets(unempl, model="ZNZ")
fcast_ets_no_trend <- forecast(fit_ets_no_trend, h=12)
plot(fcast_ets_no_trend)
```



Exercise 7

Find a function in the forecast package that estimates the BATS model (exponential smoothing state space model with Box-Cox transformation, ARMA errors, trend sand seasonal components). Use it to estimate the model with a damped trend, and make a forecast. Plot the forecast.

```
fit_bats <- bats(unempl, use.damped.trend = TRUE)
fcast_bats <- forecast(fit_bats, h=12)
plot(fcast_bats)
```



Exercise 8

Use the accuracy function from the forecast package to get a matrix of accuracy measures for the forecast obtained in the previous exercise. Explore the structure of the matrix, and save a measure of the mean absolute error (MAE) in a variable.

```
acc_bats <- accuracy(fcast_bats)
print(acc_bats)
```

```
##                       ME      RMSE       MAE        MPE     MAPE      MASE
## Training set -0.02284822 0.1503005 0.1073459 -0.2853875 1.707524 0.1389502
##                    ACF1
## Training set 0.1717618
```

```
mae_bats <- acc_bats[1, "MAE"]
```

Exercise 9

Write a funciton that inputs a time series and a list f model estimation functions, calculates forecasts for the next 12 periods using each of the functions (with default parameters), and outputs the forecast with the smallest mean absolute error.

Run the function using the **unemployment** time series and a list of functions that includes ets, bats, and auto.arima. Plot the obtained result.

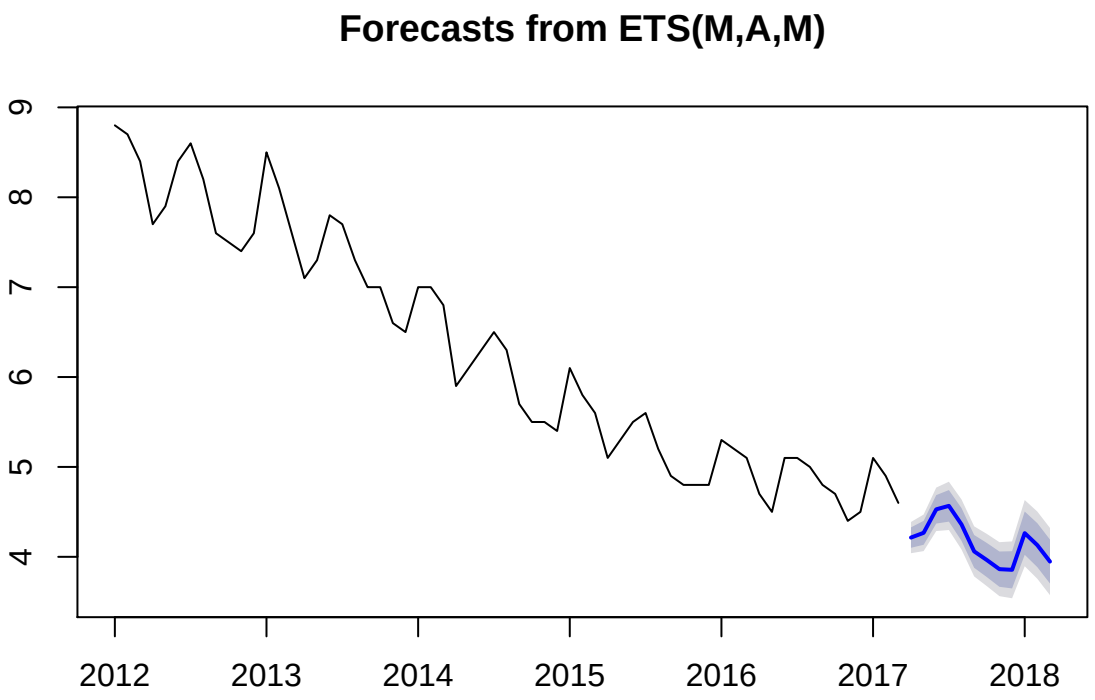
```
best_forecast <- function(series, forecasting_functions){
  smallest_mae <- Inf
  best_forecast <- NULL

  for (f in forecasting_functions){
    fit <- f(series)
    fcast <- forecast(fit, h=12)
    mae <- accuracy(fcast)[1, "MAE"]

    if (mae < smallest_mae){
      smallest_mae <- mae
      best_forecast <- fcast
    }
  }

  return(best_forecast)
}

# run the function with the unemployment time series and list of forecasting models
forecasting_functions <- c(ets, bats, auto.arima)
fcast_best <- best_forecast(unempl, forecasting_functions)
plot(fcast_best)
```



Exercise 10

Modeify the function written in the previous exercise so that it prints the mean absolute error for each forecasting model along with the name of that model (the name can be retrieved from the forecast object).

```
best_forecast <- function(series, forecasting_functions){
  smallest_mae <- Inf
  best_forecast <- NULL

  for (f in forecasting_functions){
    fit <- f(series)
    fcast <- forecast(fit, h=12)
    mae <- accuracy(fcast)[1, "MAE"]
    cat(sprintf("%-35s %f\n", fcast$method, mae))

    if (mae < smallest_mae){
      smallest_mae <- mae
      best_forecast <- fcast
    }
  }

  return(best_forecast)
}

forecasting_functions <- c(ets, bats, auto.arima)
fcast_best <- best_forecast(unempl, forecasting_functions)
```

```
## ETS(M,A,M)                          0.102154
## BATS(0.005, {0,0}, 1, {12})         0.108308
## ARIMA(2,1,2)(1,0,0)[12] with drift  0.144354
```