Forecast evaluation and combination

Nikolaos Kourentzes (nikolaos@kourentes.com)

Contents

1 Load data and relevant packages

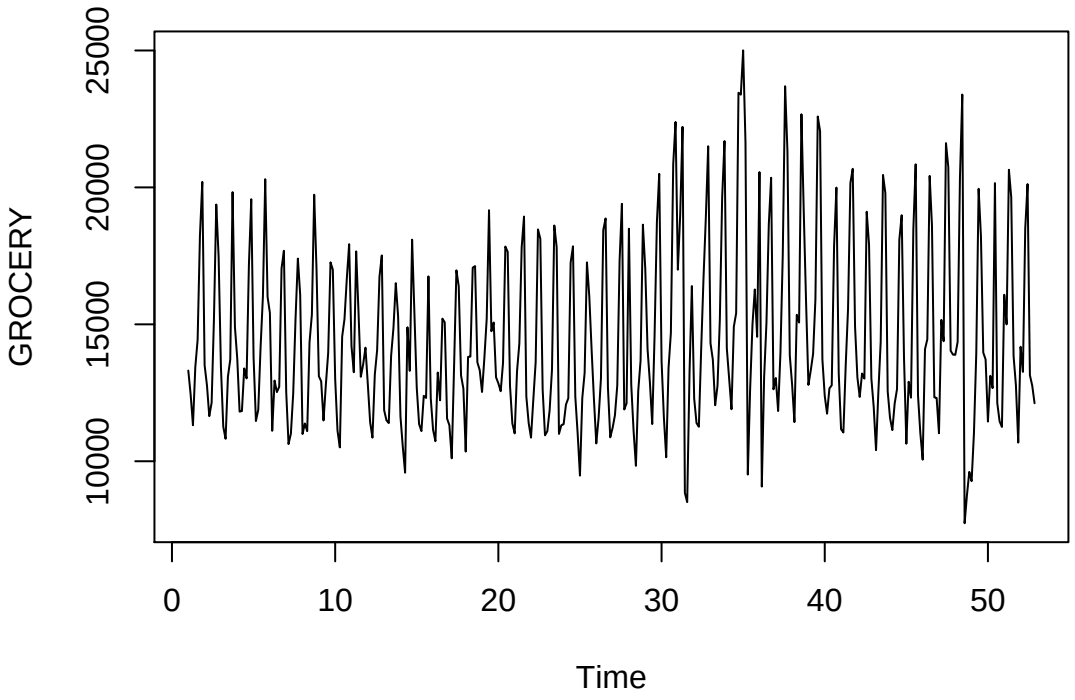
Our overall task in this workshop will be to identify a good forecasting approach for a given time series. To do this we will generate various forecasts and compare them.

We will be using grocery sales from a supermarket store in the US. These are provided in `grocery.Rdata` file. Note the file extension. The data is saved in the native R format. In the previous workshop, we had the data saved in `.csv` and had to import it to R. Now, we simply load the data into the working memory.

```
load("./grocery.Rdata")
```

The data is already a time series. Let us get some basic feel for it. First we plot the sales.

```
plot(y)
```



It looks fairly seasonal, but relatively flat, i.e. there is no apparent trend. We will check this in more detail with our time series exploration. Let us see how many observations we have, using the function `length()` that counts the length of a vector, i.e. how many data points it has.

```
n <- length(y)
n
```

```
## [1] 364
```

So we have 52 weeks of 7 days. The series is pre-set with a seasonality of 7 days. Before we proceed with the analysis, let us load the **forecast** and the **tsutils** packages to help us with the time series exploration and the forecasting. Since we had installed them last time, we do not need to do this again. If for any reason these are not installed, please consult the notes of the previous lab to do so!

```
library(forecast)
library(tsutils)
```

2 Separate into in- and out-of-sample and data exploration

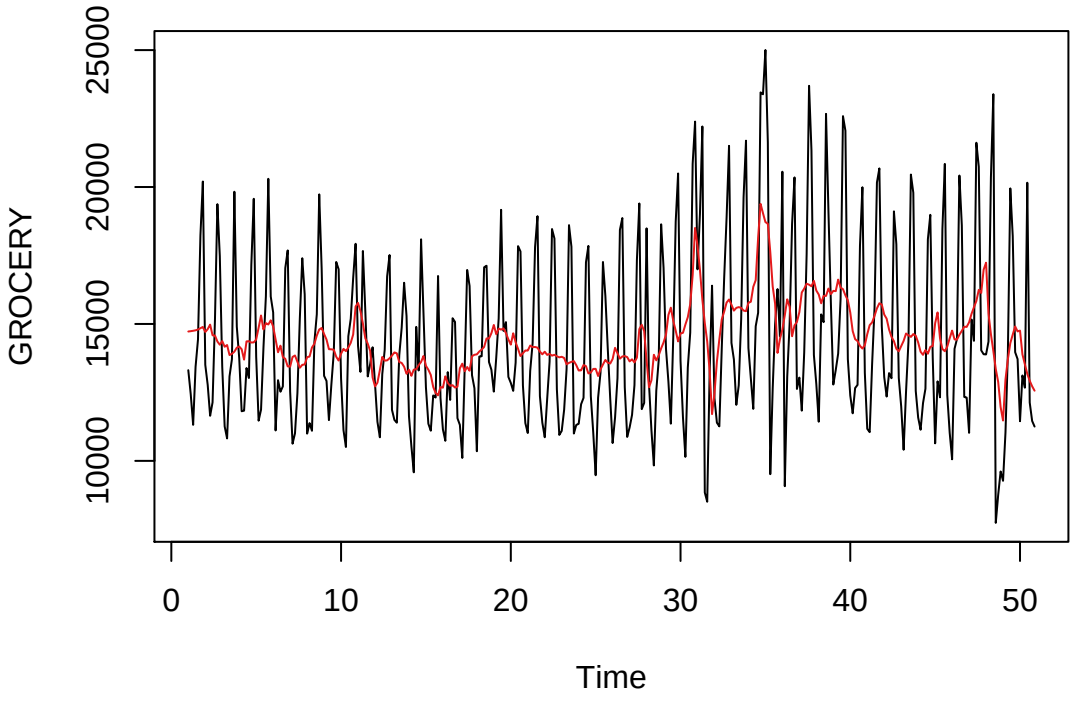
First, before we proceed with the analysis we will keep some observations aside, as a test set to evaluate how good our forecasts are. For this purpose, let us use the last 2 weeks. Given the amount of data we have available, this is rather short and in practice we would use more data. We restrict here to only 2 weeks, for computational speed. If you were to do this on your own, for an exercise/real life you should keep more observations, so that you can see more seasons and events in the test set.

```
y.trn <- head(y,7*50) # That reads as take the first 50 weeks of the series.
y.tst <- tail(y,7*2) # ... and the last two weeks as test.
```

We have stored in variable `y.trn` and `y.tst` the training and test sets respectively. Let me remind you that training, fitting, in-sample are used interchangeably. Similarly, test, out-of-sample, and hold-out sets are used as the same. Also, as commented in the previous workshop, since we have the **forecast** package loaded, the functions `head()` and `tail()` copy the time series properties as well, so our new variables are ready to use.

For now, we ignore `y.tst` and focus only on the training set, `y.trn`. We explore the data in the usual way. First, we calculate a centred moving average and we plot the result, using the function `cmav()` that is part of the **tsutils** package.

```
cma <- cmav(y.trn, outplot=TRUE)
```

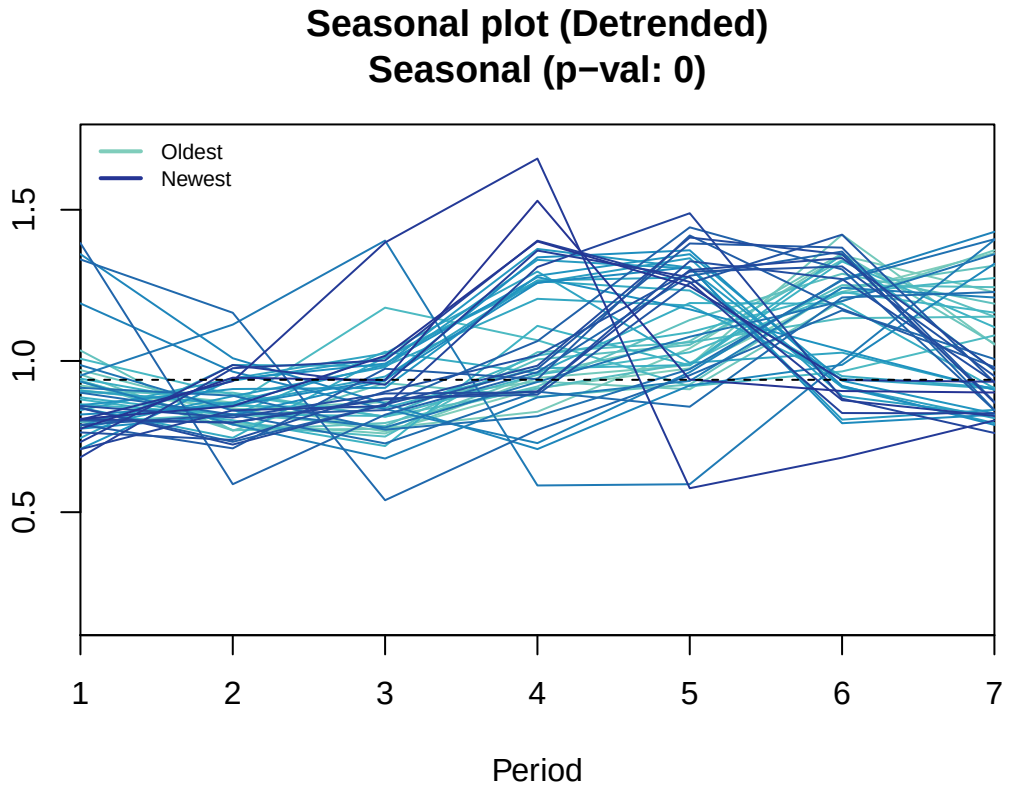



```
# I also save the result in the variable cma
```

There is a local change in the level across the 50 weeks. This would be due to the annual seasonality that we are not modelling, due to lack of data. If we had several years we would bother with that as well. Therefore, if we attribute the slow change in level in the annual seasonality, we can conclude that most probably there is no trend in the data.

The presence of seasonality is evident, but let us construct a seasonal diagram (we use the `seasplot()` function from the **tsutils** package) anyways, so see if we can pry away more information about our time series.

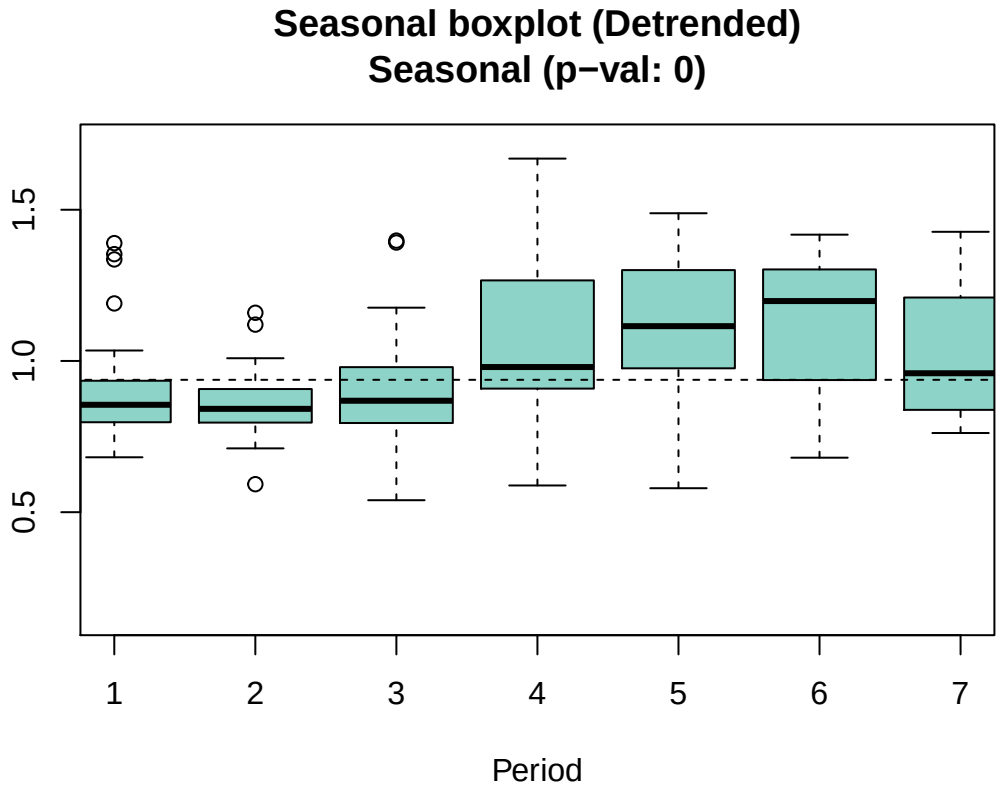
```
seasplot(y.trn)
```




```
## Results of statistical testing
## Evidence of trend: TRUE  (pval: 0)
## Evidence of seasonality: TRUE  (pval: 0)
```

The result is not that easy to read. It seems that the seasonal shape is evolving over the year. Perhaps other views of the seasonal diagram are more helpful

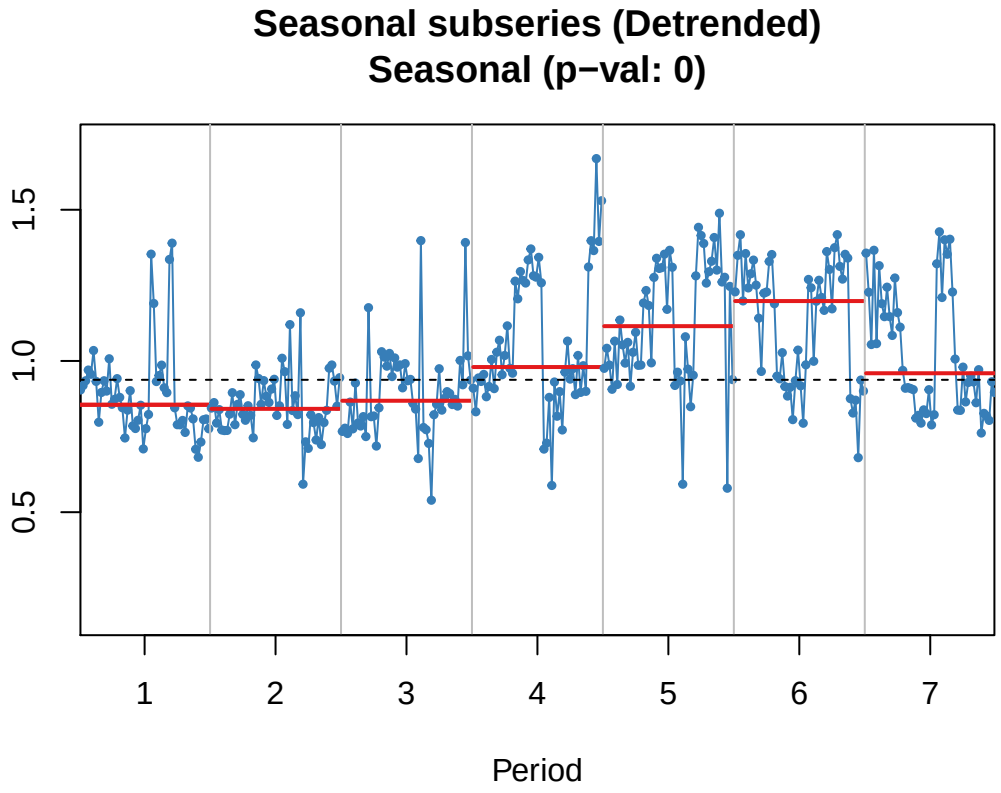
```
seasplot(y.trn,outplot=2)
```




```
## Results of statistical testing
## Evidence of trend: TRUE  (pval: 0)
## Evidence of seasonality: TRUE  (pval: 0)
```

When we look at the seasonal boxplots we can see a clearer signal. The seasonal subseries plot may allow us to understand the evolution of the seasonal pattern over the year.

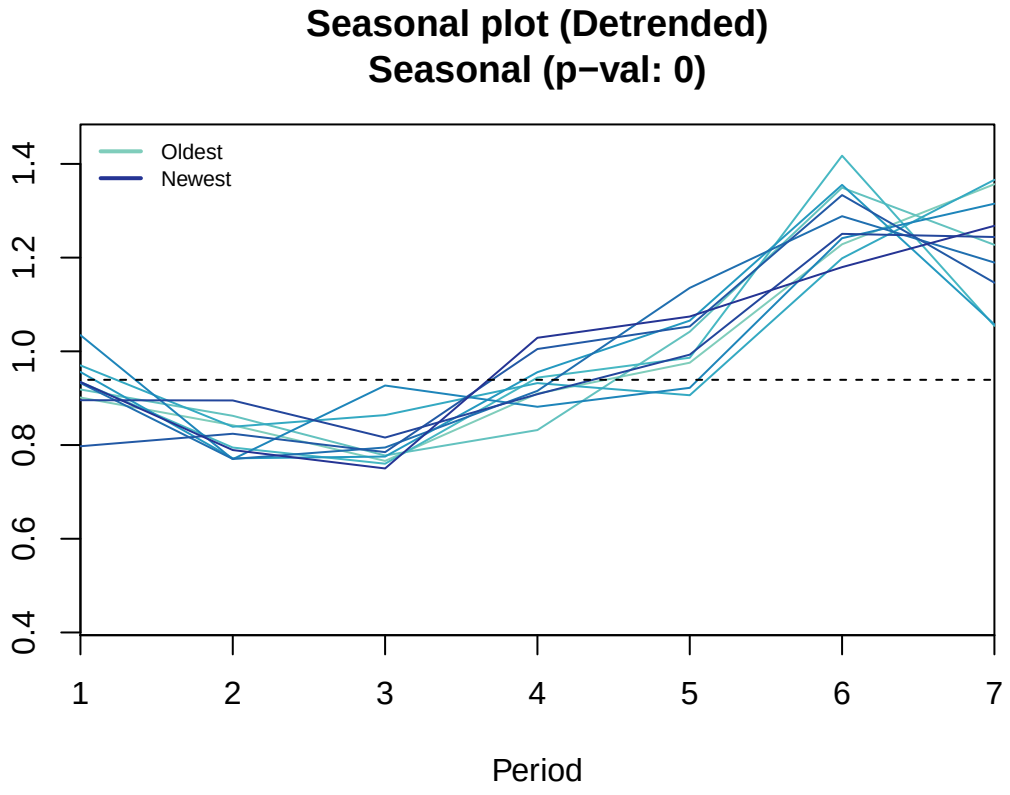
```
seasplot(y.trn,outplot=3)
```




```
## Results of statistical testing
## Evidence of trend: TRUE  (pval: 0)
## Evidence of seasonality: TRUE  (pval: 0)
```

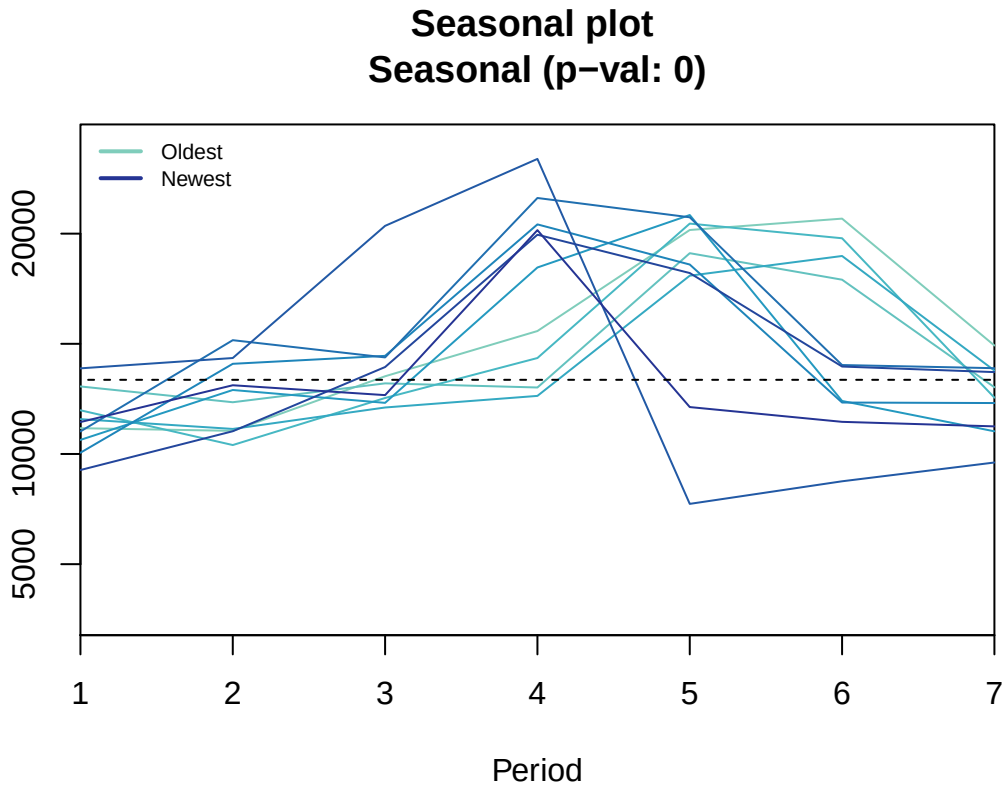
Each part of the plot (separated by vertical lines) plots the average value for that day of the week in red and the different values across the year with blue dots. We can see that there are some distinct behaviours. In the early part there is some stability, while in the middle part, for all days of the week, the values change substantially. In other words, the seasonal shape changes throughout the year quite a bit! Let us illustrate that. We will construct a few seasonal plots for the various parts of the year.

```
seasplot(head(y.trn,10*7)) # First 10 weeks only
```




```
## Results of statistical testing
## Evidence of trend: TRUE  (pval: 0.001)
## Evidence of seasonality: TRUE  (pval: 0)
```

```
seasplot(tail(y.trn,10*7)) # Last 10 weeks only
```

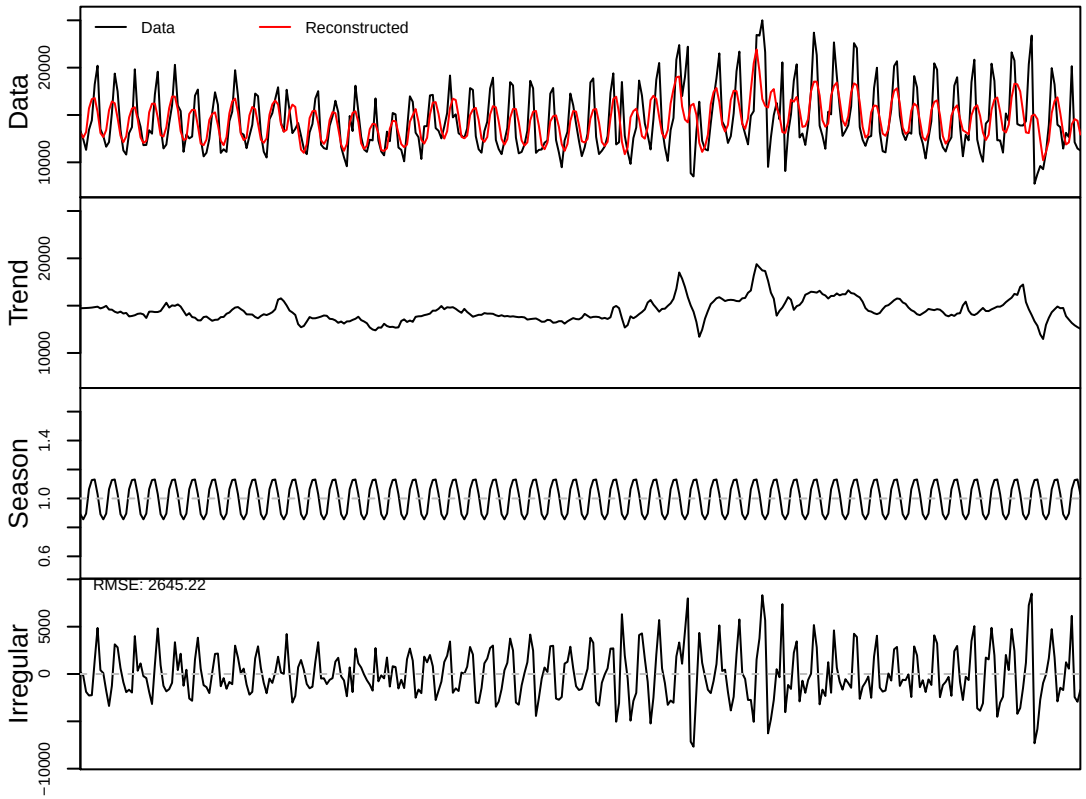



```
## Results of statistical testing
## Evidence of trend: FALSE  (pval: 0.045)
## Evidence of seasonality: TRUE  (pval: 0)
```

We get dramatically different seasonal shapes. We could do this exploration to some detail and find all the included shapes. This changing pattern within the year is very common in daily data and is driven by the changing daylight hours, temperature, and how customers arrange their daily lives differently in different seasons of the year. In technical terms, this is a *stochastic seasonality*, which evolves over time, in contrast to *deterministic seasonality* that stays fixed over time. Exponential smoothing is capable of modelling both, but excels on stochastic seasonality.

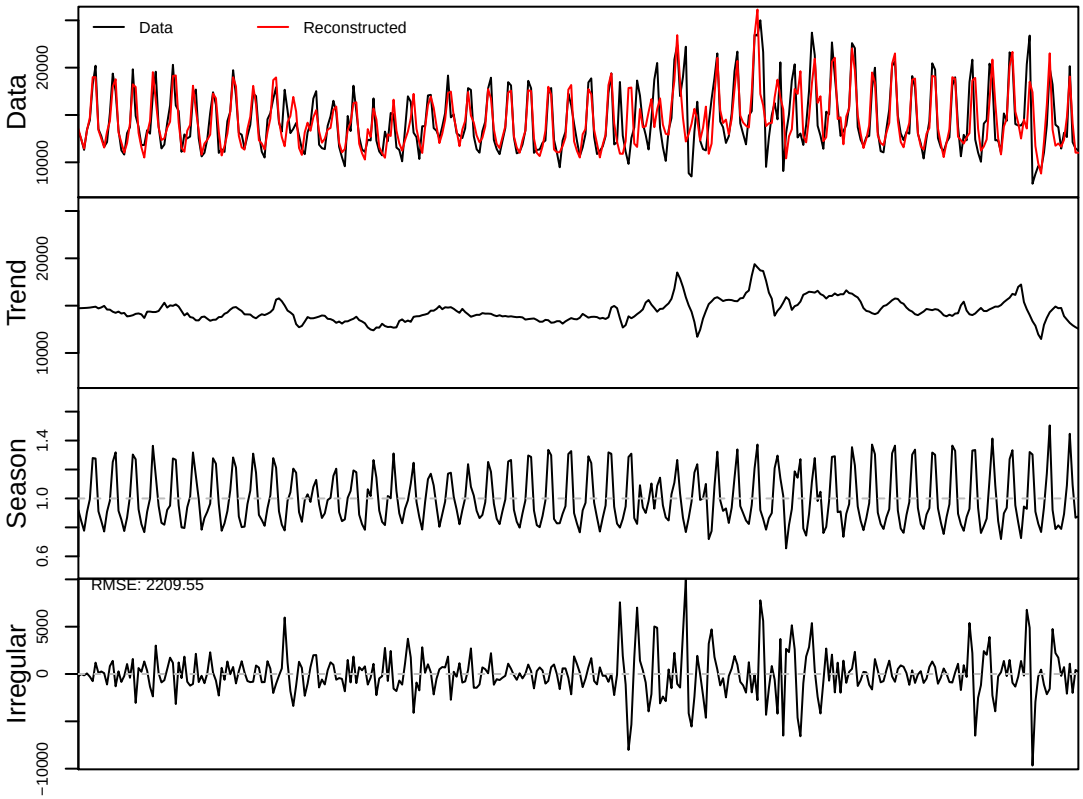
Let us proceed with the full decomposition, using the `decomp()` function from the **tsutils** package.

```
dc <- decomp(y.trn, outplot=TRUE)
```



This did not work very well. Observe the poor fit of the reconstructed signal to the series and that a lot of the seasonal part remains in the noise part. This is due to the function trying to fit a single seasonal shape throughout the series, which as we established is a bad idea. Let us tell the function to use a stochastic encoding for the seasonality, based on exponential smoothing.

```
dc <- decomp(y.trn, outplot=TRUE, type="pure.seasonal")
```



Much better! Observe how the seasonal shape changes over the year, but also how the errors increase for a short period at every change of the seasonal pattern. The model does not know when the change happens, but it quickly adapts. This is a limitation of extrapolative models, such as exponential smoothing: they can adapt to changes quickly, but not anticipate them. For this, we would need to use some regression style model. Note, however, this is far from trivial and in practice we often find that extrapolative models still perform best!

3 Forecasting

We have done some basic forecasting already in the previous workshop. The only difference here is that we will use rolling origin evaluation, instead of producing a single forecast. Evaluating multiple forecasts will give us a better feel of which forecasting method is better.

The test set can only be used to evaluate forecasts. It should not be used to fit models or select between models. You can think of it as: we do all the analysis and decisions we want, and then test if it all made sense. No further choices should be based on the results of the evaluation on the test set.

In choosing between forecasts, there are two different approaches:

1. Using information criteria to compare between models of a single family. For example, between exponential smoothing models only.
2. Using a validation set, with which we need to invest some observations to construct the new subset, but on the other hand, we can compare any forecast.

3.1 Selection of forecasts using information criteria

When we fit a statistical model (assuming the implementation of the model in the software you are using is of good quality) then it should also provide you with information criteria.

For example, when we use the function `ets()` it provides us with AIC, AICc, and BIC.

```
fit <- ets(y.trn)
fit
```

```
## ETS(M,N,M)
##
## Call:
##  ets(y = y.trn)
##
##   Smoothing parameters:
##     alpha = 0.1024
##     gamma = 0.3636
##
##   Initial states:
##     l = 14657.0119
##     s = 1.2821 1.2869 0.993 0.8979 0.7854 0.8323
##           0.9224
##
##   sigma:  0.1747
##
##      AIC     AICc      BIC
## 7537.494 7538.142 7576.073
```

Observe the last two lines of the output. We focus on AICc, which is similar to AIC, but somewhat adjusted to account for small sample sizes. When the sample is large AIC and AICc are almost equivalent. The `ets()` function tries several exponential smoothing models and compares them using the AICc to find the best.

For now, let us manually construct a number of forecasts, where we preselect the exponential smoothing type. Remember the models are codified as (Error, Trend, Season) and we use the letters N for none, A for additive, and M for multiplicative. For example, ANN, means Additive errors, No trend, and No seasonality. For the damped trend, we use the argument `damped=TRUE`.

```
# Level model
fit1 <- ets(y.trn,model="ANN")
# Seasonal model
fit2 <- ets(y.trn,model="ANA")
# Linear trend model
fit3 <- ets(y.trn,model="AAN",damped=FALSE)
# Damped trend model
fit4 <- ets(y.trn,model="AAN",damped=TRUE)
# Trend seasonal model
fit5 <- ets(y.trn,model="AAA",damped=FALSE)
# Damped trend seasonal model
fit6 <- ets(y.trn,model="AAA",damped=TRUE)
```

We can access the AICc of each model by adding to the variable that contains the model the suffix `$aicc`, as in the following example:

```
fit1$aicc
```

```
## [1] 7727.092
```

Let us save all AICc's in a vector, so we can easily compare them. We will use the function `c()` to concatenate

all AICc's together in a single vector.

```
aicc <- c(fit1$aicc,fit2$aicc,fit3$aicc,fit4$aicc,fit5$aicc,fit6$aicc)
# We name the aicc vectr to easily identify which is which
names(aicc) <- c("ANN","ANA","AAN","AAdN","AAA","AAdA")
aicc
```

```
##      ANN      ANA      AAN     AAdN      AAA     AAdA
## 7727.092 7573.495 7732.440 7767.566 7578.661 7580.221
```

The lowest AICc is best. We use the function `which.min()` to directly get the position of the lowest value.

```
which.min(aicc)
```

```
## ANA
##   2
```

So it seems that the seasonal model (`fit2`) is the best. We can compare that with the result in `fit`, the automatically identified model.

```
fit$aicc
```

```
## [1] 7538.142
```

```
fit2$aicc
```

```
## [1] 7573.495
```

The automatic specification is slightly better. It is assuming multiplicative errors and seasonality, which gives a slightly lower fit error, and therefore better AICc. The `ets()` function greatly simplifies the modelling process, as it does this comparison of AICc's automatically, between all possible exponential smoothing models.

Note that our time series exploration is in agreement with the results from AICc. Arguably, identifying additive or multiplicative errors and seasonality would require more careful exploration.

Finally, in using information criteria to select the best model, we did not have to produce a single forecast, speeding up the selection process.

3.2 Selection of forecasts using a validation set

We repeat the analysis using a validation set. To do this, we split `y.trn` again into a smaller training set and a validation set. For this example, let us retain the last 2 weeks of `y.trn` as validation.

```
y.ins <- head(y.trn,48*7)
y.val <- tail(y.trn,2*7)
```

For this exercise, let us assume that we need to forecast 7-periods ahead. We will save that into a variable:

```
h <- 7
```

The validation set is larger than the forecast horizon, which allows us to perform a rolling origin evaluation. We can do that using a for-loop. Before we introduce that, let us consider how would a single iteration, i.e. the forecasts from a single origin look like, in terms of code. We consider two additional forecasts: ETS(M,N,M) that was identified as best, by using the AICc, and the seasonal naive, so that we have a forecast for which calculating the AICc would be impossible.

```
# Note that now I will be using y.ins, instead of y.trn
fit1v <- ets(y.ins,model="ANN")
fit2v <- ets(y.ins,model="ANA")
fit3v <- ets(y.ins,model="AAN",damped=FALSE)
fit4v <- ets(y.ins,model="AAN",damped=TRUE)
```

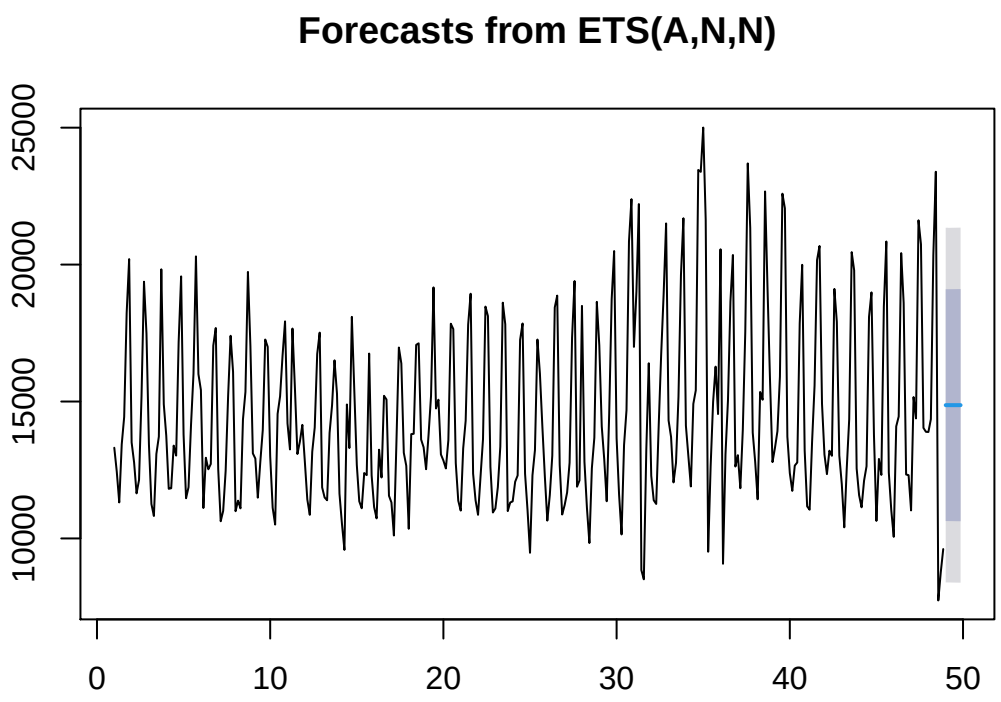
```
fit5v <- ets(y.ins,model="AAA",damped=FALSE)
fit6v <- ets(y.ins,model="AAA",damped=TRUE)
fit7v <- ets(y.ins,model="MNM")
```

There is no model to fit for the seasonal naive, so we proceed directly to produce the forecasts.

```
frc1v <- forecast(fit1v,h=h)
frc2v <- forecast(fit2v,h=h)
frc3v <- forecast(fit3v,h=h)
frc4v <- forecast(fit4v,h=h)
frc5v <- forecast(fit5v,h=h)
frc6v <- forecast(fit6v,h=h)
frc7v <- forecast(fit7v,h=h)
# And the naive:
frc8v <- tail(y.ins,frequency(y.ins))[1:h] # that is copy the last season\
# using the function frequency() to find how long is a season and
# copy the last h of those observations.
```

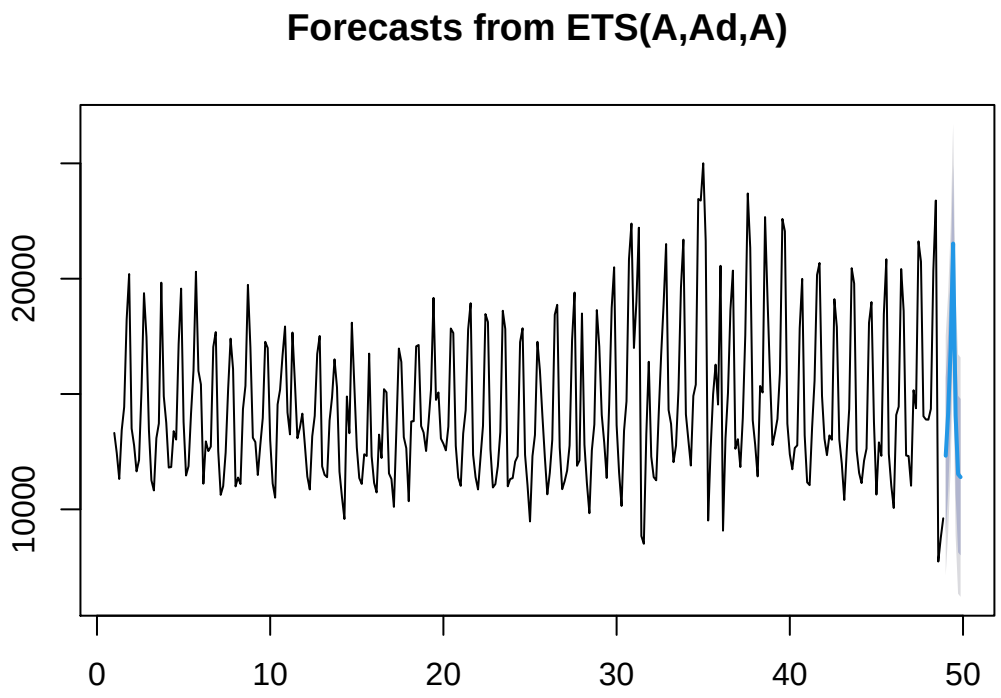
As an example, we plot a few of the forecasts to see what we have:

```
plot(frc1v)
```



and

```
plot(frc6v)
```



Now we calculate forecast errors for these. We will be using Mean Absolute Error (MAE) to measure the forecast accuracy. As the horizon is 7 periods long, we will use only the first 7 observations of the validation set. The function `abs()` calculates the absolute values and `mean()` the average.

```
err1v <- mean(abs(y.val[1:h] - frc1v$mean))
# y.val[1:h] gives us the first 7 observations of y.val
# frc1v$mean gives us the point forecasts of frc1v
# mean(abs( )) gives us the MAE.
err2v <- mean(abs(y.val[1:h] - frc2v$mean))
err3v <- mean(abs(y.val[1:h] - frc3v$mean))
err4v <- mean(abs(y.val[1:h] - frc4v$mean))
err5v <- mean(abs(y.val[1:h] - frc5v$mean))
err6v <- mean(abs(y.val[1:h] - frc6v$mean))
err7v <- mean(abs(y.val[1:h] - frc7v$mean))
# For the naive we just have the numeric values in frc8v, so we do
# not need the suffix $mean
err8v <- mean(abs(y.val[1:h] - frc8v))
```

We collect all errors in a vector and name it. Then we find the minimum error.

```
errv <- c(err1v, err2v, err3v, err4v, err5v, err6v, err7v, err8v)
names(errv) <- c("ANN","ANA","AAN","AAdN","AAA","AAdA","MNM","Naive")
errv
```

```
##      ANN      ANA      AAN     AAdN      AAA     AAdA      MNM    Naive
## 2975.734 2822.253 3040.983 4786.245 2826.875 2822.616 2270.250 5367.263
```

```
which.min(errv)
```

```
## MNM
##   7
```

The MAE suggests that MNM is our best forecast, in agreement with the previous analysis. However, this is based on a single forecast error. Ideally, we want to base our decision on multiple forecasts, in case we were very lucky/unlucky on this set of forecasts. For this we will ask R to repeat the same experiment multiple times.

First, we need to find how many forecasts we can fit into the validation set. The maximum number of forecast origins is calculated as: the size of the validation set - forecast horizon + 1.

```
omax <- length(y.val) - h + 1
omax
```

```
## [1] 8
```

So with the current data we can do 8 forecasts in our validation set and consider the average performance across all forecast origins. In the rolling origin we will keep expanding the in-sample, until we have used as much of the validation set as we can.

Note that in our previous evaluation of forecast errors there were many repetitions in the code. We can try to be a bit smarter in how we do things, so we type very few lines. First, we need to setup some variables, listing what we will be running, and where we will be storing the results.

```
# This is what we will be running
models <- c("ANN", "ANA", "AAN", "AAN", "AAA", "AAA", "MNM",  "Naive")
damped <- c(FALSE, FALSE, FALSE, TRUE,  FALSE, TRUE,   FALSE, FALSE)
# And this is where we will store things
# Forecast errors across forecast origins
err <- array(NA,c(omax,8)) # This has omax rows and 8 columns,
# one for each different forecasting method.
# Forecasts for a given origin
frcs <- array(NA,c(h,8))
```

At this point we need to introduce a `for loop`. This tells R to run a particular set of instructions several times. The syntax is as follows:

```
for (o in 1:omax){
  print(o)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
```

We use a counter `o` (it could be any variable name) that will run from 1 to `omax` and execute all commands contained within the curly brackets `{ }`. The range of values could be anything, continuous, such as `1:omax`, or not, such as `c(1,5,9,10)`. The example above just prints `o` at every iteration.

Now that we have a basic understanding of for-loops, let us construct the evaluation loop.

```
# For each forecast origin
for (o in 1:omax){
```

```
  # Split training set
  y.ins <- head(y.trn,48*7-1+o) # As o increases, so will the in-sample.
  y.val <- tail(y.trn,2*7-o+1) # As o increases, the validation will decrease.

  # Fit and forecast with all exponential smoothing models
  for (m in 1:7){
    fitTemp <- ets(y.ins,model=models[m],damped=damped[m])
    frcs[,m] <- forecast(fitTemp,h=h)$mean
    err[o,m] <- mean(abs(y.val[1:h] - frcs[,m]))
  }

  # Forecast using the seasonal naive
  # Remember we do not have a model for this
  frcs[,8] <- tail(y.ins,frequency(y.ins))[1:h]
  err[o,8] <- mean(abs(y.val[1:h] - frcs[,8]))

}
```

We name the columns of the errors matix `err` for convenience, using the function `colnames()`:

```
colnames(err) <- c("ANN", "ANA", "AAN", "AAdN", "AAA", "AAdA", "MNM",  "Naive")
err
```

```
##           ANN      ANA      AAN     AAdN      AAA     AAdA      MNM    Naive
## [1,] 2975.734 2822.253 3040.983 4786.245 2826.875 2822.616 2270.250 5367.263
## [2,] 2603.560 2459.703 2647.831 5340.174 2461.724 2452.621 2172.020 5019.013
## [3,] 2262.265 2055.915 1945.982 3877.711 2061.953 2050.249 2248.446 4842.480
## [4,] 2434.633 1988.818 2449.286 2160.198 1997.578 1988.870 2488.761 4109.314
## [5,] 2530.054 1843.592 2574.834 5254.436 1856.715 1847.040 2251.791 3647.554
## [6,] 2472.729 1813.034 2568.853 4881.620 1815.201 1811.906 1781.343 3019.866
## [7,] 2816.176 1684.053 2903.166 2208.285 1684.137 1684.079 1606.699 2634.483
## [8,] 3148.847 1546.037 3233.429 2379.490 1546.174 1545.162 1568.200 2400.111
```

Each row (from [1,] to [8,]) contains the errors for the different forecasts for a single forecast origin. As we have 8 origins, we have 8 sets of errors. We calculate the average error per column (per forecasting method), using the function `colMeans()` and find the minimum error.

```
errMean <- colMeans(err)
errMean
```

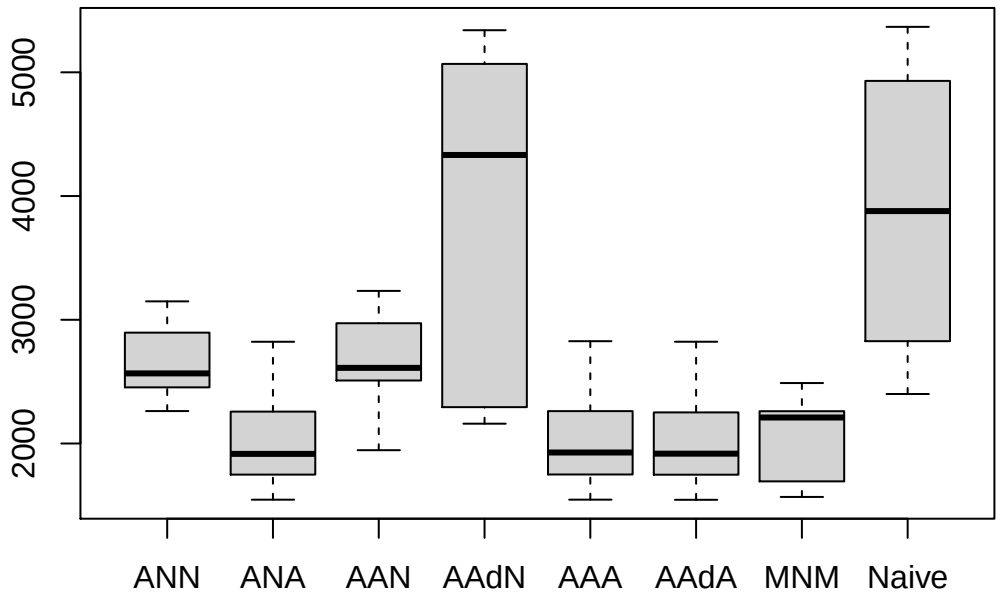
```
##      ANN      ANA      AAN     AAdN      AAA     AAdA      MNM    Naive
## 2655.500 2026.676 2670.545 3861.020 2031.295 2025.318 2048.439 3880.011
```

```
which.min(errMean)
```

```
## AAdA
##    6
```

It seems that the rolling origin evaluation of the validation set gave us model AAdA as the best result, although ANA, MNM, and AAA are very near. To get a better feel for the distributions of the errors, let us construct boxplots.

```
boxplot(err)
```



The boxplot validates our understanding, though it is interesting to note that the median of MNM is markedly different from those of ANA, AAA, and AAdA.

There are many reasons why AICc and the validation set did not give us the same result, here are the most prominent ones:

- The AICc calculation is based on 1-step ahead errors, while we set the validation exercise for 7-step ahead errors. This is in favour of the validation test evaluation scheme.
- The AICc calculation uses more data to estimate model parameters. This is in favour of the AICc.
- The validation set is using only 8 rolling forecasts, more/less forecasts may change the result. Ideally, we would like an as long as possible validation set.
- The AICc is implicitly based on squared errors. For the validation set we used absolute errors. Which errors are preferable depends on the exact objective of our forecasting.

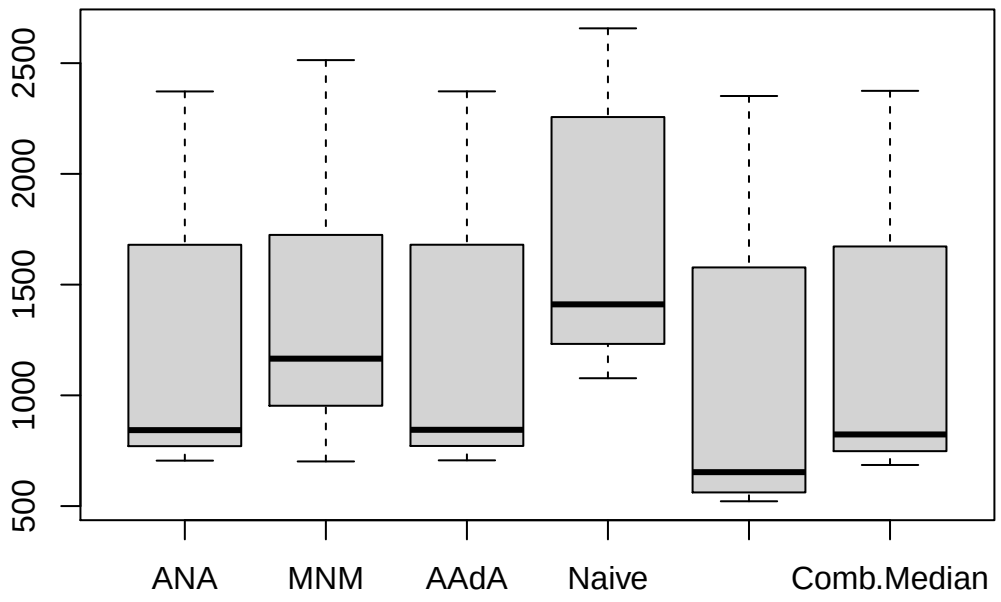
4 Out-of-sample evaluation

So far we have a few alternatives for which might be the best model. AICc suggests ETS(MNM), validation set suggests ETS(AAdA), and arguably manual exploration suggested ETS(ANA). We test all three models in the test set that we kept aside at the beginning of the workshop. We will keep on considering the seasonal naive as a benchmark. Any more complex forecast should at minimum perform better than this simplistic forecast. If not, we do not bother with additional complexities!

We will also use two simple forecast combination approaches. Taking the mean and the median forecast at each period, instead of choosing any single approach.

We have 2 weeks of data in the test set, so we can perform a rolling origin evaluation, in the same way as we did for the validation set. Copying and slightly amending the code we have:

```
# What to run
modelsTest <- c("ANA", "MNM", "AAA", "Naive", "CombMean", "CombMedian")
dampedTest <- c(FALSE, FALSE, TRUE)

# Pre-allocate memory
omaxTest <- length(y.tst) - h + 1
errTest <- array(NA,c(omaxTest,6))
frcsTest <- array(NA,c(h,6))

# For each forecast origin
for (o in 1:omaxTest){

  # Split training set
  y.trnTest <- head(y,50*7-1+o) # As o increases, so will the in-sample.
  y.tstTest <- tail(y,2*7-o+1)  # As o increases, the test will decrease.

  # Fit and forecast will all exponential smoothing models
  for (m in 1:3){
    fitTemp <- ets(y.trnTest,model=modelsTest[m],damped=dampedTest[m])
    frcsTest[,m] <- forecast(fitTemp,h=h)$mean
    errTest[o,m] <- mean(abs(y.tstTest[1:h] - frcsTest[,m]))
  }

  # Forecast using the seasonal naive
  frcsTest[,4] <- tail(y.trnTest,frequency(y.trnTest))[1:h]
  errTest[o,4] <- mean(abs(y.tstTest[1:h] - frcsTest[,4]))

  # Combinations
  # The function apply allows us to use any function we want on a matrix
  frcsTest[,5] <- apply(frcsTest[,1:4],1,mean)
  # This reads, using array frcsTest[,1:4], i.e. all rows and the first 4 columns
  # take the mean across all columns (that is what the argument 1 says).
  errTest[o,5] <- mean(abs(y.tstTest[1:h] - frcsTest[,5]))
  # And for the median:
  frcsTest[,6] <- apply(frcsTest[,1:4],1,median)
  errTest[o,6] <- mean(abs(y.tstTest[1:h] - frcsTest[,6]))

}

# Assign names to errors
colnames(errTest) <- c("ANA","MNM","AAdA","Naive","Comb.Mean","Comb.Median")

# Summarise and plot errors
boxplot(errTest)
```




```
errTestMean <- colMeans(errTest)
print(errTestMean)
```

```
##         ANA         MNM        AAdA       Naive   Comb.Mean Comb.Median
##    1208.019    1362.662    1208.888    1691.712    1057.314    1193.481
```

```
which.min(errTestMean)
```

```
## Comb.Mean
##         5
```

Eventually, the combination is best! All chosen models are better than the Naive. The validation choice performed better than the AIC choice.

5 Forecast combination with AIC weights

In the previous example we only tried simple combinations based on mean and median. Although these typically work very well in practice, using AIC weights is also very successful.

To do that we will use the `AirPassengers` data to produce some ETS models and combine them. We consider four options, ETS(ANN), ETS(AAN), ETS(MNM), and ETS(MAM), which are four basic ways to combine trend and season. Note, again for reference, the notation in ETS is (Error, Trend, Season), where A is for additive, M for multiplicative, and N stands for none. Typically, when we have multiplicative seasonality (so the magnitude of the seasonality is proportional to the level of the series) it is reasonable to work with multiplicative errors, which rely on the same proportionality. This affects the prediction intervals. The intuitive interpretation is: if we sell more during summer, we also expect larger variability during summer.

First, we split the data into training and test sets.

```
y.trn <- window(AirPassengers,end=c(1959,12))
y.tst <- window(AirPassengers,start=c(1960,1))
```

I will use some more advanced coding tricks to get the AIC of the models without typing too much.

I create a vector with the model specifications:

```
models <- c("ANN","AAN","MNM","MAM")
```

First, I initialise a list to store all models that I want to build. Remember that lists are collections of variables that can be of any form. I also want to store the forecasts in an array.

```
fit <- list()    # Here I will store models
frc <- array(NA,c(12,4),dimnames=list(NULL,models))    # Here I will store forecasts
```

I build all models with a for-loop:

```
for (i in 1:4){
  # I ask ets() everytime to fit the model specified in the models variable
  fit[[i]] <- ets(y.trn,model=models[i],damped=FALSE)
  # And then give me the forecasts
  frc[,i] <- forecast(fit[[i]],h=12)$mean
}
```

Note that lists require double `[[ ]]`. Now I will use `lapply()` to extract from all the AIC. `lapply` applies the same operation to all list members and returns a list. I will reduce the list to a vector (a typical 1-dimensional collection of values in R) using the `unlist()` function.

```
AIC <- unlist(lapply(fit,function(x){x$aic}))
AIC
```

```
## [1] 1558.920 1562.628 1297.518 1257.573
```

```
# When in doubt of what a command does, remember that we can break it to smaller ones
# Try running it part by part:
# fit[[1]]$aic
# lapply(fit,function(x){x$aic})    # This will get aic from all models
# unlist(lapply(fit,function(x){x$aic}))   # Return the output as a vector
```

The last argument of `lapply()` is `function(x){x$aic}`. This tells `lapply()` use a custom function, which takes a single argument `x`, here an element from the list `fit`, and with that (this is the bit within the `{ }`) take the `$aic` value.

Now that I have the AICs I need to calculate the weights. First, we find the minimum AIC and take the differences from that one. Then I calculate the weights as proportions of the exponent of the resulting AIC differences.

```
dAIC <- AIC - min(AIC)
dAIC <- exp(-0.5*dAIC)
waic <- dAIC/sum(dAIC)
waic
```

```
## [1] 3.659296e-66 5.730754e-67 2.118497e-09 1.000000e+00
```

The weights are not *human-readable*. Let us round them to 4 decimals to see what we have.

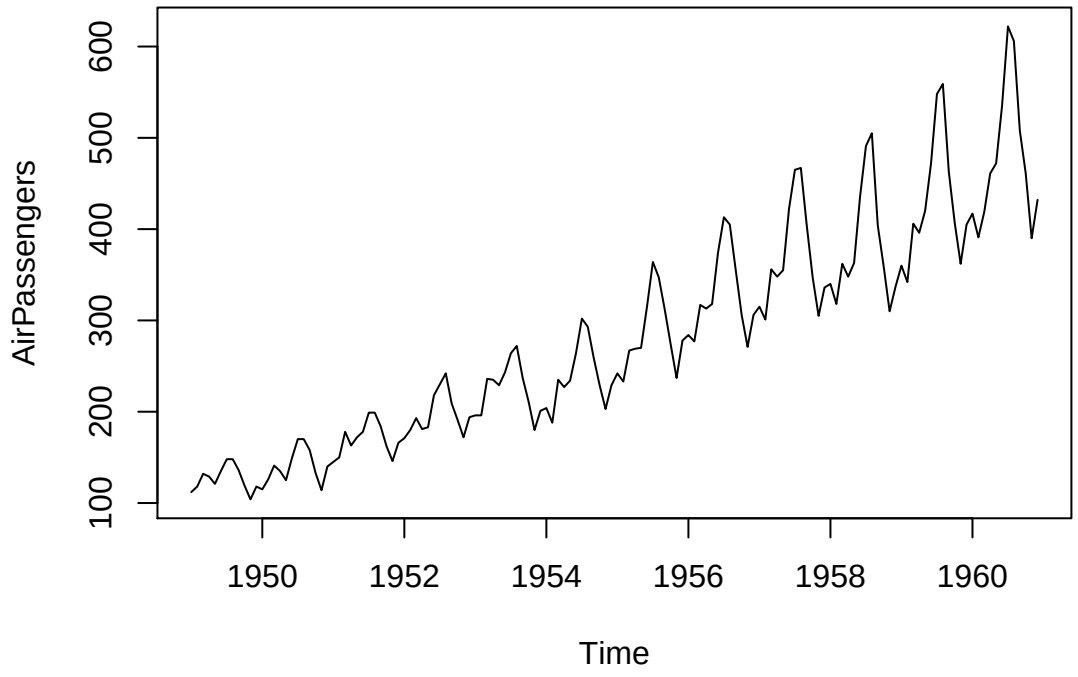
```
round(waic,4)
```

```
## [1] 0 0 0 1
```

So this combination is practically a selection of the trend-seasonal ETS (MAM model), as only this model gets non-zero weights.

If you don't remember the `AirPassengers` is a trend-seasonal time series.

```
plot(AirPassengers)
```



Therefore, the combination weights are not that unreasonable! Given the right conditions, forecast combination and selection become the same.

Let us repeat the process with a collection of trend seasonal models, with and without damped trend.

```
# Prepare variables and models
fit2 <- list()
frc2 <- array(NA,c(12,6))
models <- rep(c("AAA","MAM","MMM"),2)
damped <- c(rep(FALSE,3),rep(TRUE,3))
# Fit models and generate forecasts
for (i in 1:6){
  fit2[[i]] <- ets(y.trn,model=models[i],damped=damped[i])
  frc2[,i] <- forecast(fit2[[i]],h=12)$mean
}
# Extract AIC and calculate weights
AIC2 <- unlist(lapply(fit2,function(x){x$aic}))
dAIC2 <- AIC2 - min(AIC2)
dAIC2 <- exp(-0.5*dAIC2)
waic2 <- dAIC2/sum(dAIC2)
round(waic2,4)
```

```
## [1] 0.0000 0.0005 0.0001 0.0000 0.3481 0.6513
```

We calculate the combinations with AIC weights, mean, median and also collect the forecast with the best (minimum) AIC:

```
# AIC weights
frcComb <- frc2 %*% cbind(waic2)
# Mean
frcComb <- cbind(frcComb, rowMeans(frc2))
# To calculate the median without loops we use apply()
frcComb <- cbind(frcComb, apply(frc2,1,median))
# Selection
frcComb <- cbind(frcComb, frc2[,which.min(AIC2)])
colnames(frcComb) <- c("Comb.AIC","Comb.Mean","Comb.Median","Selection")
```

And finally, we calculate MAE to evaluate whether the combination helps in terms of accuracy.

```
err <- matrix(rep(y.tst,4),ncol=4) - frcComb
# The matrix(rep(y.tst,4),ncol=4) creates a 4 column matrix with copies
# of the test set. Run it on its own to see the result.
MAE <- colMeans(abs(err))
round(MAE,2)
```

```
##    Comb.AIC   Comb.Mean Comb.Median   Selection
##       22.03       20.74       20.99       21.64
```

In this case, the combination using the mean performs best.

Note that here we did these calculations as an example. We would prefer not to rely on a single forecast error, but we would rather perform a rolling origin evaluation, as before. In that case, we would collect multiple errors and have a much more reliable ranking of the combination approaches.

6 Exercises

1. For the grocery time series repeat the exercise with increased validation and test sets. Do the results change?
2. For the AirPassengers exercise using rolling origin in the test set evaluation. Do the results change?

Remember: R has a fantastic online community. Online you will find answers to almost anything R related!

Happy forecasting!