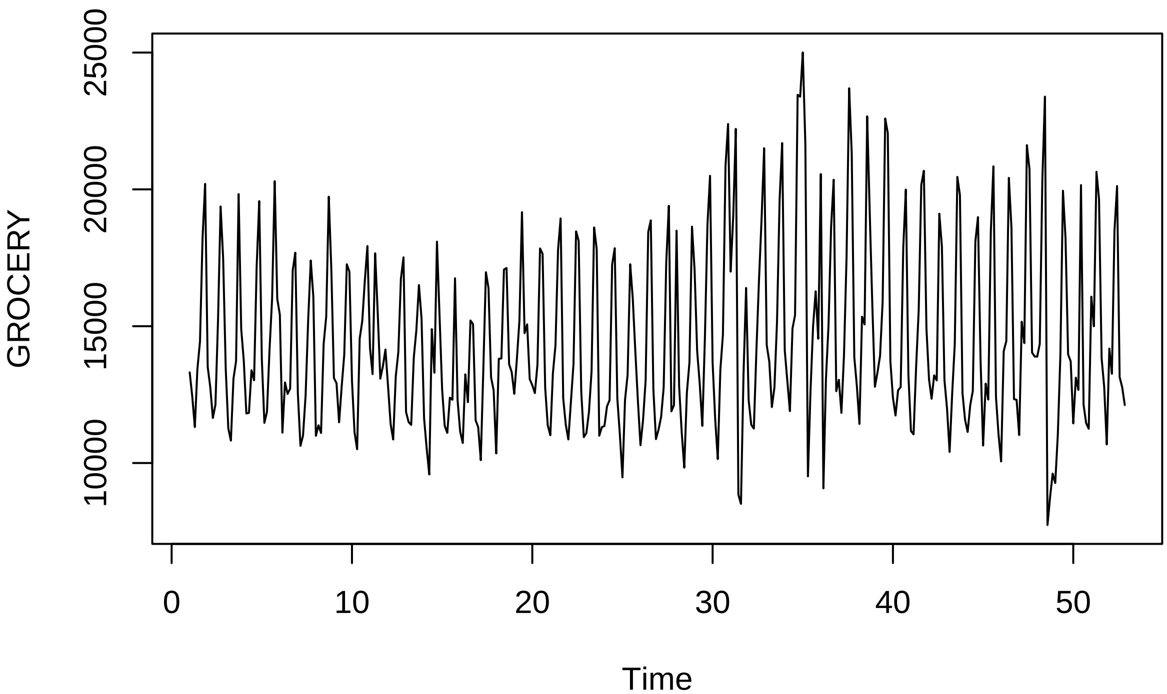
Lab3_ForecastEvaluation&Combination_a24kimwu

2025-09-16

Load data & relevant packages

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

```
plot(y)
```




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

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

```
library(forecast)
```

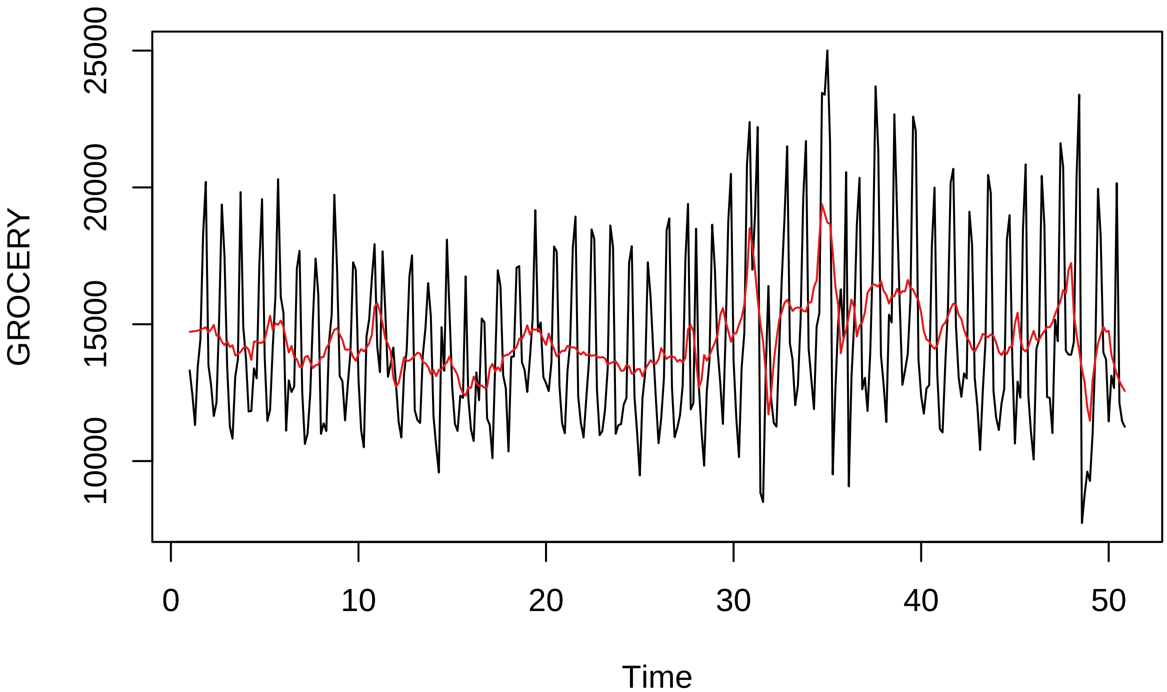
```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(tsutils)
```

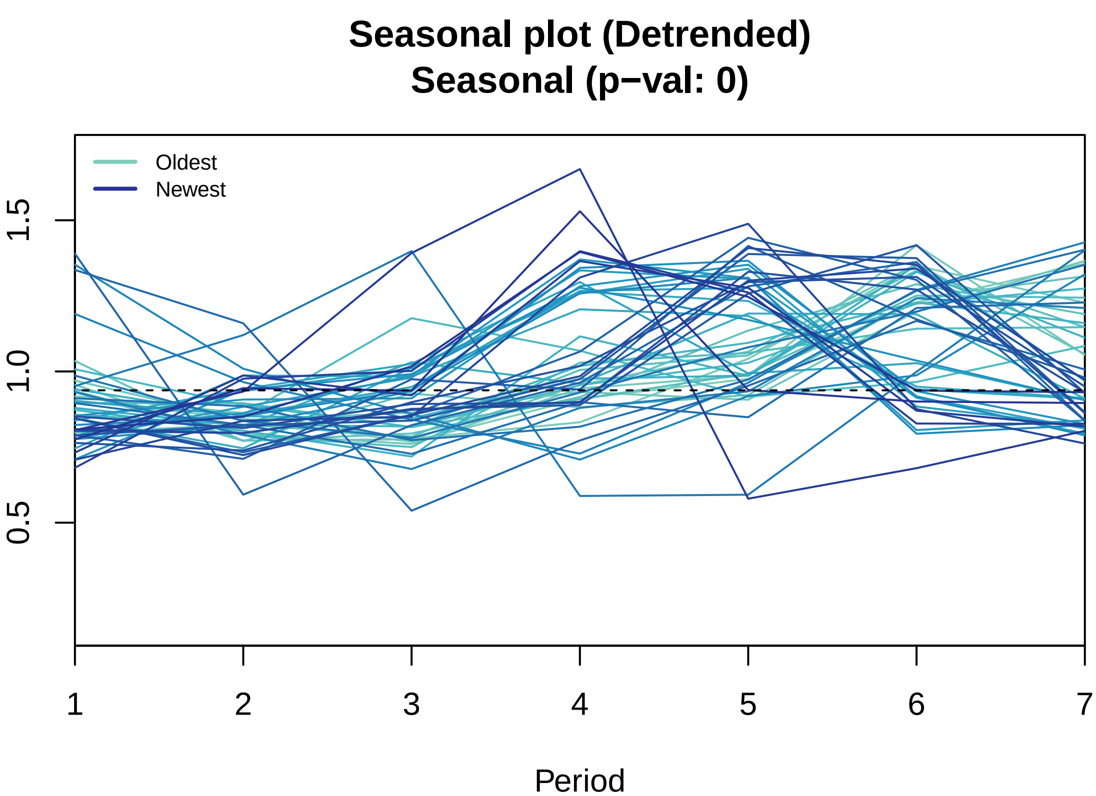
Separate into in- and out-of-sample and data exploration

```
y.train <- head(y, 7*50)
y.test <- tail(y, 7*2)
```

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

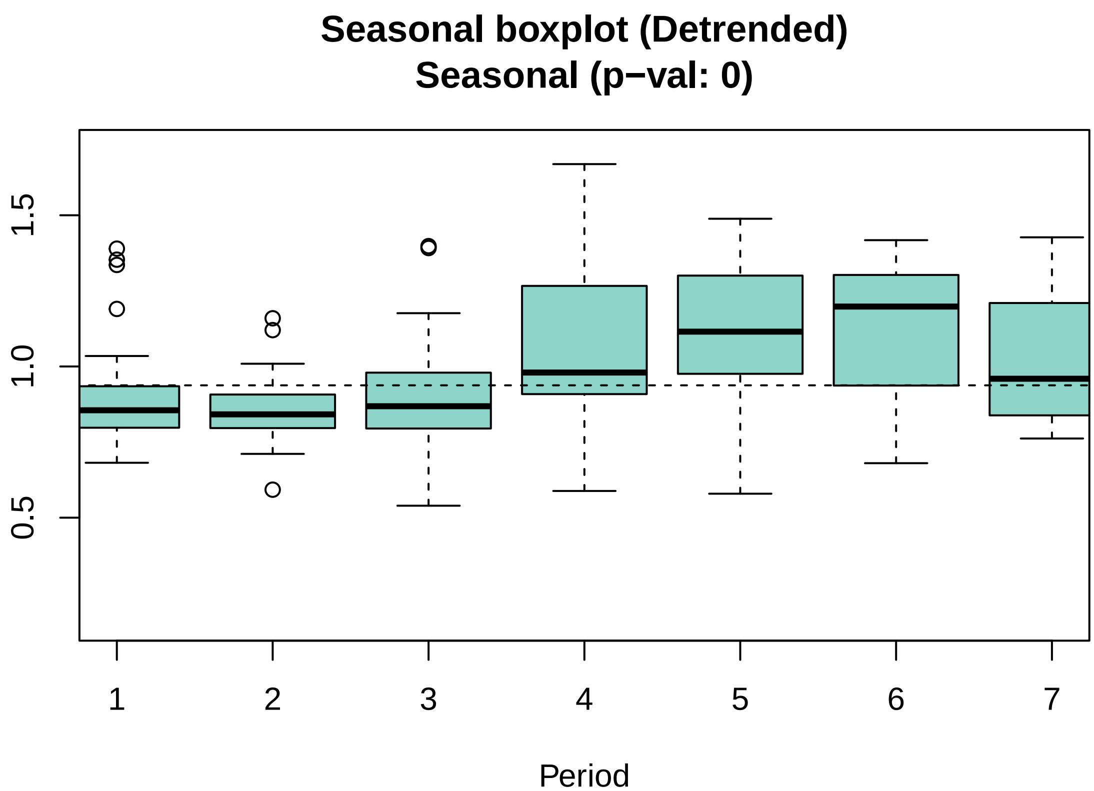



```r
seasplot(y.train)
```




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

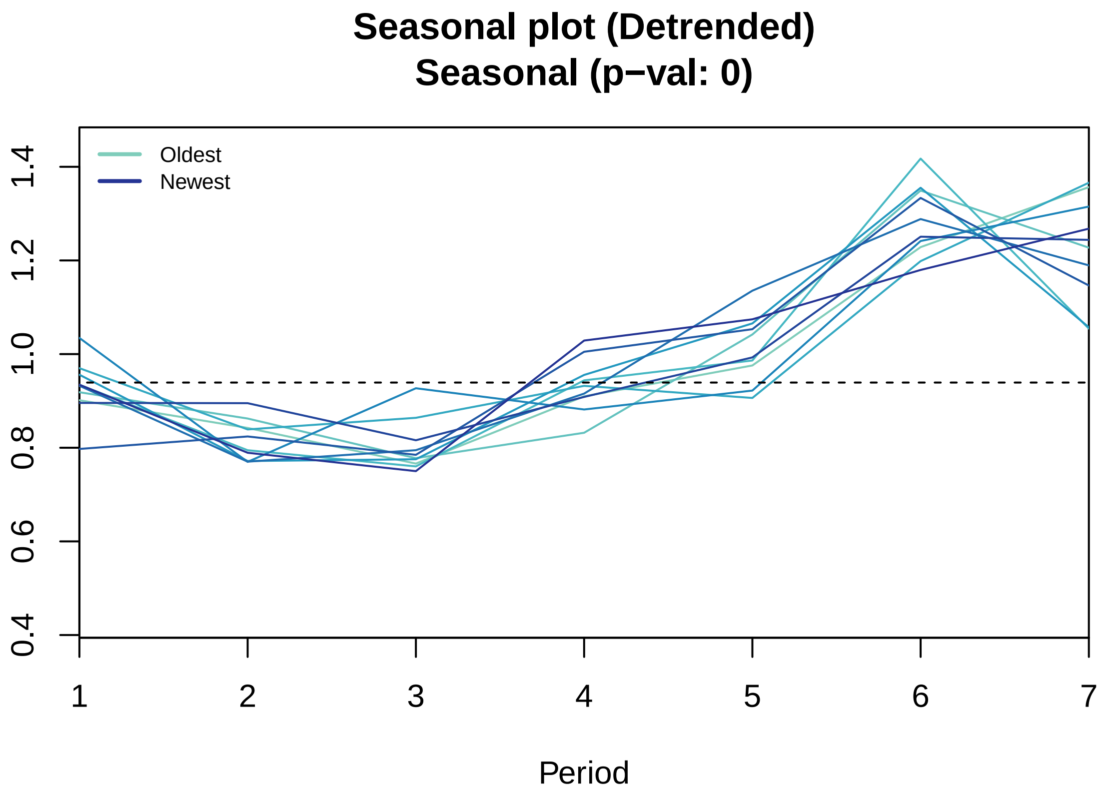
```r
seasplot(y.train, outplot = 2)
```




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

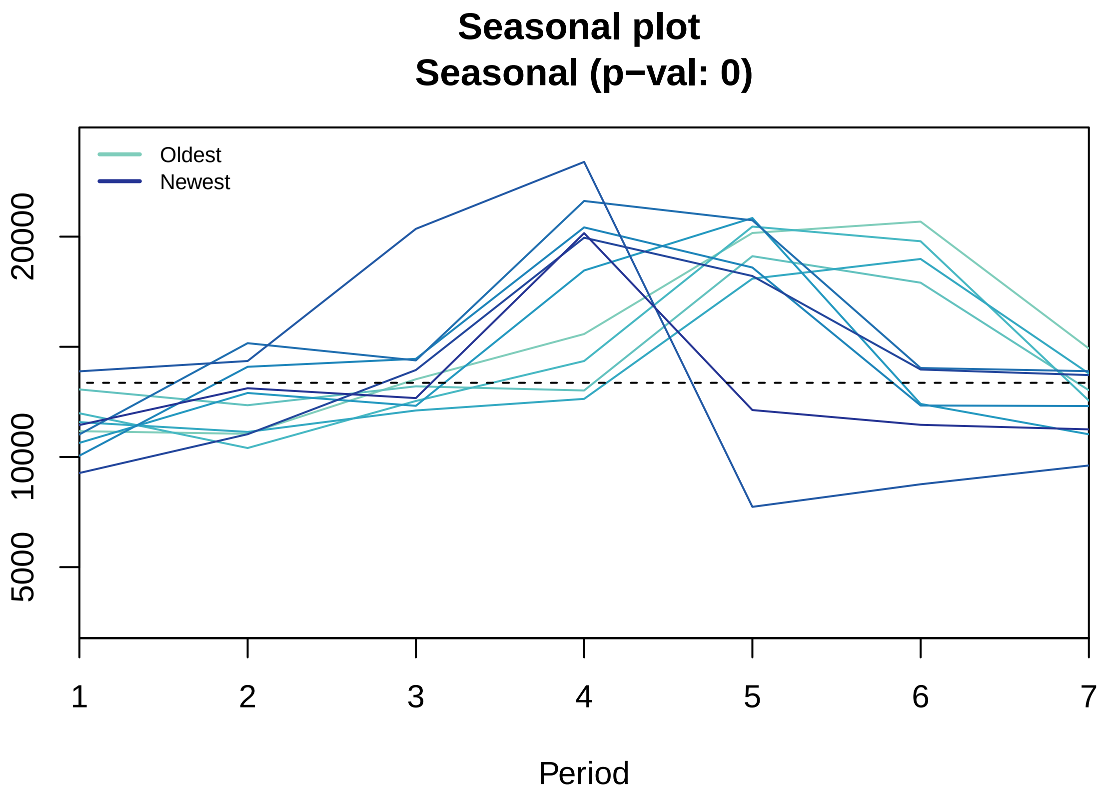
{seasplot(y.train, outplot = 3)}

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



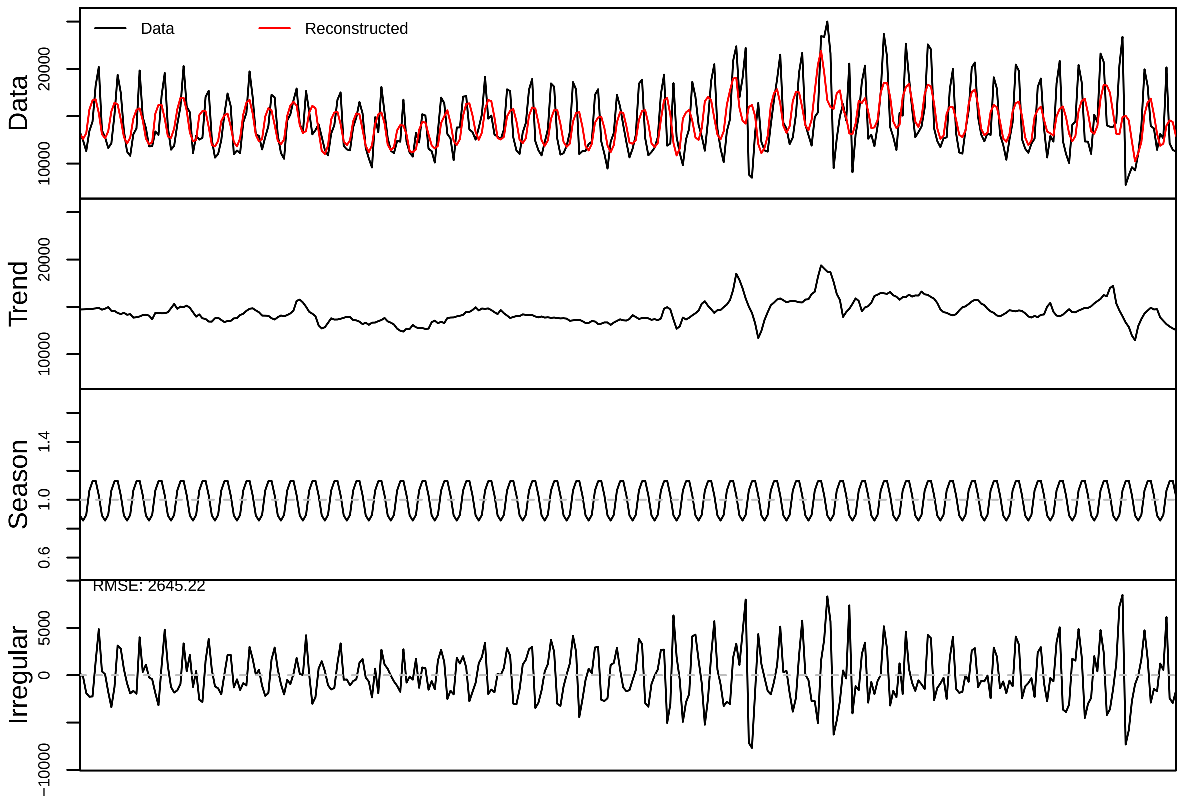

```
## Results of statistical testing
## Evidence of trend: TRUE  (pval: 0.001)
## Evidence of seasonality: TRUE  (pval: 0)
```

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

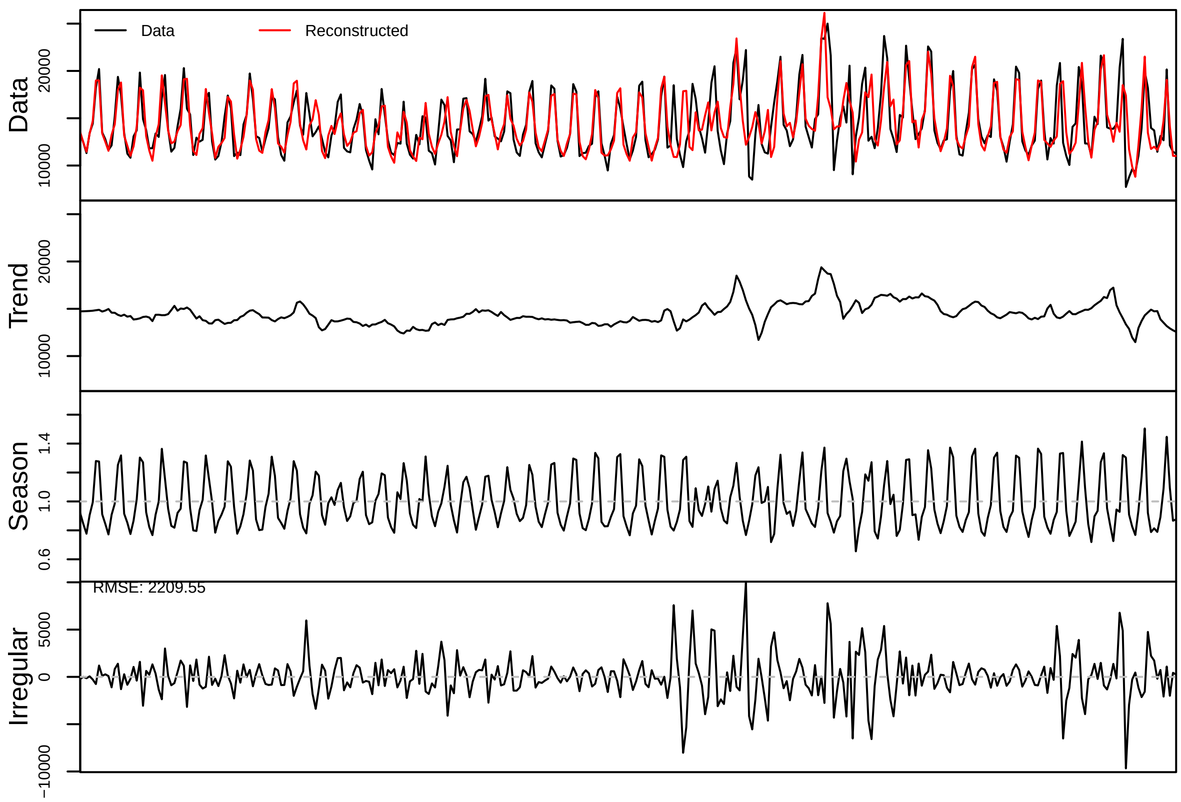



```
## Results of statistical testing
## Evidence of trend: FALSE  (pval: 0.045)
## Evidence of seasonality: TRUE  (pval: 0)
```

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




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



Forecasting

Selection of forecasts using information criteria

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

```
## ETS(M,N,M)
##
## Call:
## ets(y = y.train)
##
##   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
```

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

```
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
```

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

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

```
 fit$aicc
```

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

```
fit2$aicc
```

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

Selection of forecasts using a validation set

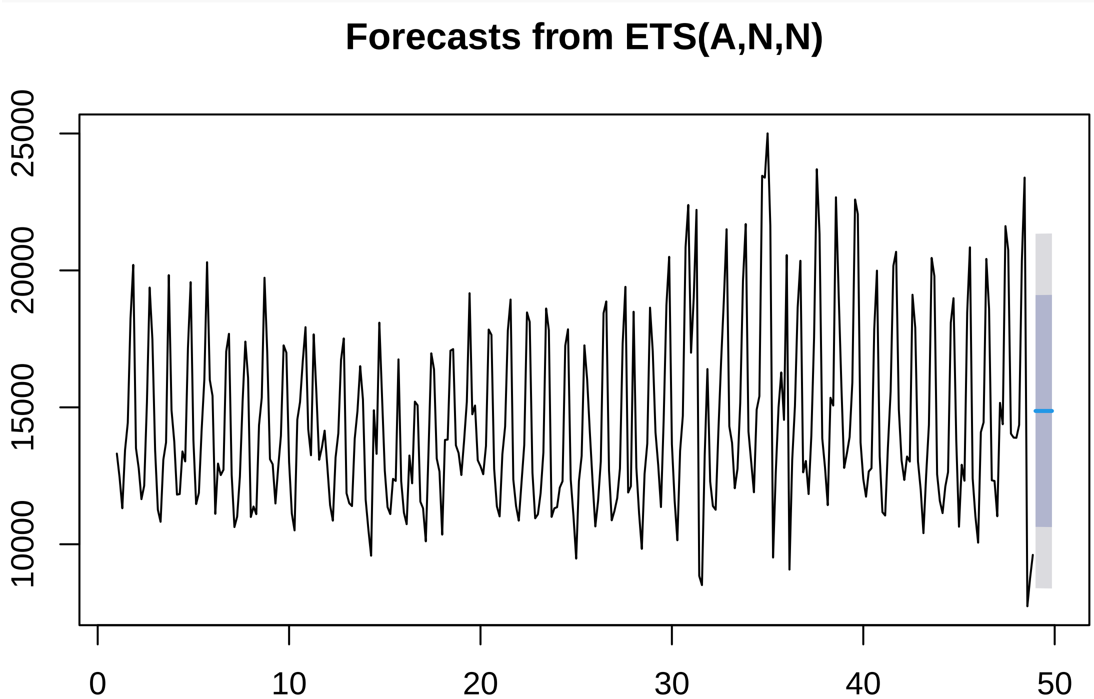
```r
y.ins <- head(y.train,48*7)
y.val <- tail(y.train,2*7)
```

```r
h <- 7 #forecast 7 periods ahead
```

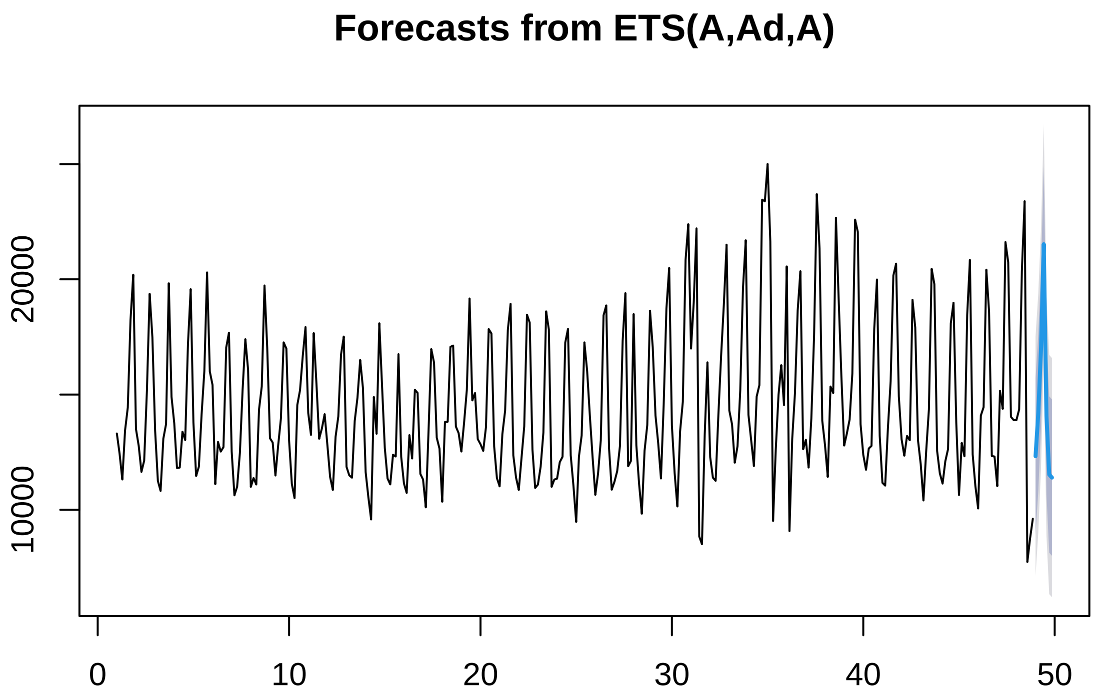
```r
fit1v <- ets(y.ins,model="ANN")
fit2v <- ets(y.ins,model="ANA")
fit3v <- ets(y.ins,model="AAN",damped=FALSE)
fit4v <- ets(y.ins,model="AAN",damped=TRUE)
fit5v <- ets(y.ins,model="AAA",damped=FALSE)
fit6v <- ets(y.ins,model="AAA",damped=TRUE)
fit7v <- ets(y.ins,model="MNM")
```

```r
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.
```

```r
plot(frc1v)
```



```
plot(frc6v)
```




```
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))
```

```
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
```

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

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

```r
# 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 metho
frcs <- array(NA,c(h,8))
```

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

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

```r
# For each forecast origin
for (o in 1:omax){
  # Split training set
  y.ins <- head(y.train,48*7-1+o) # As o increases, so will the in-sample.
  y.val <- tail(y.train,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]))
}
```

```r
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
```

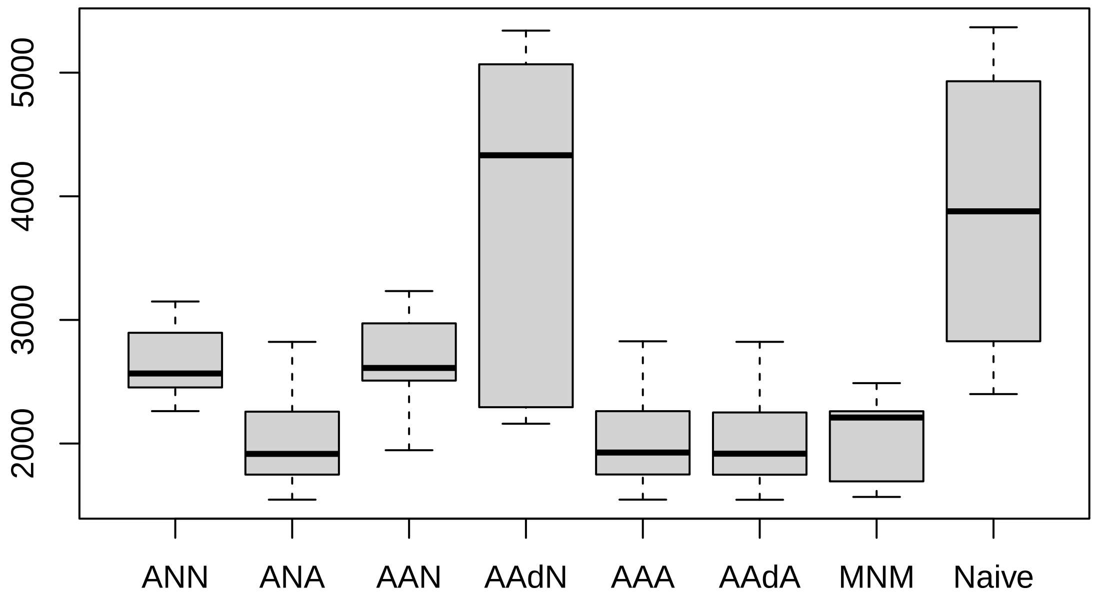
```r
errMean <- colMeans(err)
errMean
```

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

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

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

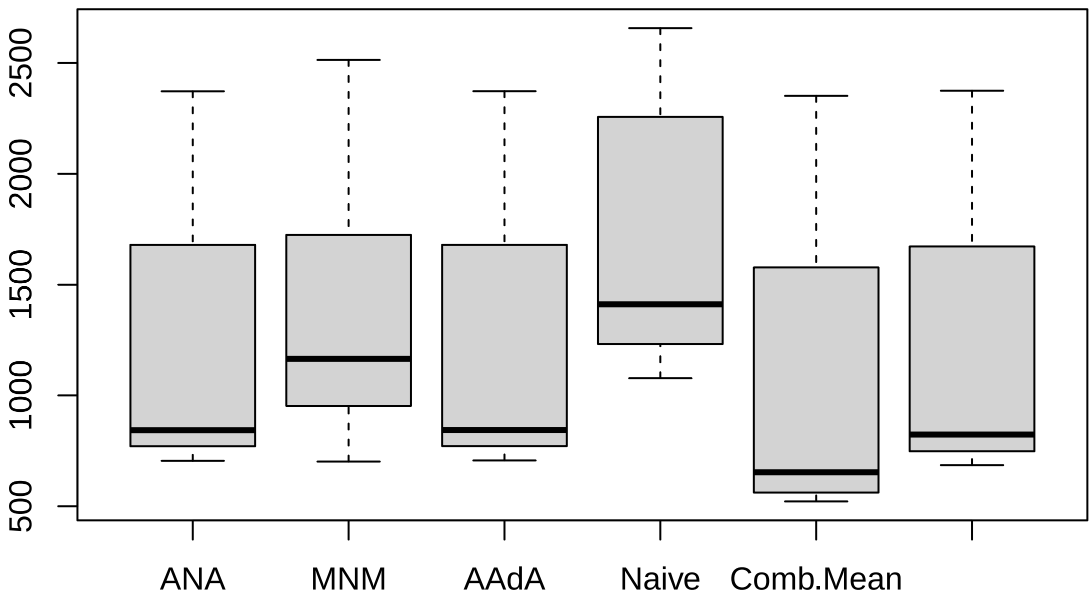
```r
boxplot(err)
```



Out-of-sample evaluation

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

# Pre-allocate memory
omaxTest <- length(y.test)- 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 a
  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
```

Forecast combination with AIC weights

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

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

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

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

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

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

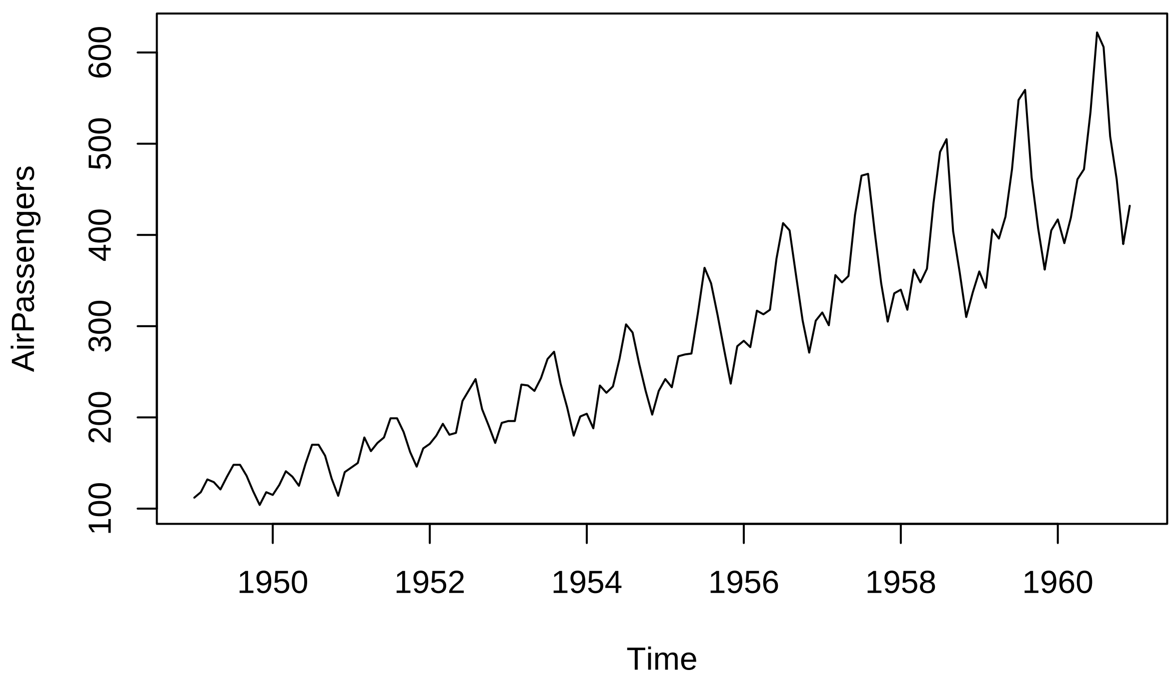
```r
 dAIC <- AIC- min(AIC)
 dAIC <- exp(-0.5*dAIC)
 waic <- dAIC/sum(dAIC)
 waic
```

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

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

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

```r
 plot(AirPassengers)
```




```r
 #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))
```

```r
#Fit models and generate forecasts
for(i in 1:6){
fit2[[i]] <-ets(y.train,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
```

```r
# 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")
```

```r
err <- matrix(rep(y.test,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
```

Exercises

1. Grocery time series with increased validation & test set

```r
y.train_1 <- head(y, 7*45)
y.test_1 <- tail(y, 7*7)
```

```r
h_1 <- 7
y.val_1 <- tail(y.train_1,5*7)
```

```r
omax_1 <- length(y.val_1)- h_1 + 1
omax_1
```

```
## [1] 29
```

```r
models_1 <- c("ANN", "ANA", "AAN", "AAN", "AAA", "AAA", "MNM", "Naive")
damped_1 <- c(FALSE, FALSE, FALSE, TRUE, FALSE, TRUE, FALSE, FALSE)
# And this is where we will store things
# Forecast errors across forecast origins
err_1 <- array(NA,c(omax_1,8)) # This has omax rows and 8 columns, one for each different forecasting m
frcs_1 <- array(NA,c(h_1,8))
```

```r
for (o in 1:omax_1){
  # Split training set
  y.ins_1 <- head(y.train_1,40*7-1+o) # As o increases, so will the in-sample.
  y.val_1 <- tail(y.train_1,5*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_1,model=models_1[m],damped=damped_1[m])
  frcs_1[,m] <- forecast(fitTemp,h=h_1)$mean
  err_1[o,m] <- mean(abs(y.val_1[1:h_1]- frcs_1[,m]))
 }
 # Forecast using the seasonal naive
 # Remember we do not have a model for this
 frcs_1[,8] <- tail(y.ins_1,frequency(y.ins_1))[1:h_1]
 err_1[o,8] <- mean(abs(y.val_1[1:h_1]- frcs[,8]))
}
```

```r
colnames(err_1) <- c("ANN", "ANA", "AAN", "AAdN", "AAA", "AAdA", "MNM", "Naive")
err_1
```

```
##            ANN       ANA      AAN     AAdN       AAA      AAdA       MNM
##  [1,] 3076.380  866.2621 3170.189 3287.858  834.6769  867.8297 1171.3082
##  [2,] 2765.747  647.9016 2845.267 4435.224  600.8809  649.1710 1172.2796
##  [3,] 2560.926  559.0736 2593.698 4707.484  490.1267  560.8097 1297.2553
##  [4,] 2599.492  572.5887 2646.674 2743.252  530.7351  573.8516 1102.0152
##  [5,] 2892.196  666.3296 3008.532 3003.780  637.0696  666.0890  908.5401
##  [6,] 5119.186  782.7705 2908.734 5120.772  772.4585  780.1582  831.9916
##  [7,] 5883.344 1119.6824 2565.299 5885.173 1127.6279 1114.3946 1002.2140
##  [8,] 2447.036 1191.3955 2832.984 2446.975 1218.2989 1189.6227 1077.3430
##  [9,] 1844.779 1200.8504 2960.803 1844.662 1245.5068 1199.4369 1160.3067
## [10,] 2405.196 1345.2460 3204.149 2405.311 1410.2037 1343.5889 1382.2688
## [11,] 2237.229 1425.2719 3277.843 2236.743 1493.0938 1424.1347 1422.0604
## [12,] 2350.886 1195.6472 2492.960 2350.831 1239.6615 1194.2941 1030.4943
## [13,] 5113.489 1170.3009 3289.784 5114.129 1196.8910 1169.4577 1054.1952
## [14,] 4523.109  850.5385 3577.045 4523.199  859.1260  850.0237  824.9574
## [15,] 3516.249  832.6461 3621.869 2877.673  839.7686  832.3124  840.3402
## [16,] 3108.702  846.3900 3643.322 3109.232  852.1939  846.0125  878.1179
## [17,] 4224.938  643.8622 3482.535 4226.189  648.0600  643.8061  742.8407
## [18,] 2826.906  656.3456 3510.362 2826.685  659.6553  656.2349  787.2232
## [19,] 3327.163  841.9668 3748.059 3327.249  854.7893  841.4500  849.4168
## [20,] 6469.714 1002.2144 3474.833 6471.660  779.7502 1000.8742  897.5182
## [21,] 5918.778 1031.0938 3403.668 5920.413 1060.5062 1029.6500  871.2393
## [22,] 3092.661  921.1452 3199.755 2302.603  948.0616  919.2962  712.5812
## [23,] 2726.961 1005.4271 3295.102 2727.541 1025.2691 1003.7794  769.4352
## [24,] 3169.724 1217.1184 2991.479 3170.551 1225.4160 1215.7441 1018.8763
## [25,] 2504.924 1132.4904 2919.029 2505.595 1139.7473 1131.2056  926.1661
## [26,] 3051.948 1614.3301 2800.099 3052.180 1609.9510 1613.5378 1488.2227
## [27,] 3819.308 1578.6880 3407.908 3819.329 1578.4906 1578.5699 1676.3688
## [28,] 5035.723 2482.9313 3327.827 5036.575 2481.6473 2482.7784 2535.6730
## [29,] 3052.491 2762.3666 3706.984 3052.334 2763.8088 2761.9646 2714.8382
##          Naive
##  [1,] 2368.146
##  [2,] 1456.426
##  [3,] 3044.619
##  [4,] 4926.071
```

```
##  [5,] 5104.420
##  [6,] 5019.570
##  [7,] 4365.256
##  [8,] 2615.930
##  [9,] 1427.506
## [10,] 3372.479
## [11,] 4811.429
## [12,] 5067.996
## [13,] 4991.063
## [14,] 4582.120
## [15,] 2794.916
## [16,] 1240.113
## [17,] 3412.226
## [18,] 5544.926
## [19,] 5673.607
## [20,] 5004.361
## [21,] 4207.309
## [22,] 2395.834
## [23,] 1449.031
## [24,] 2588.604
## [25,] 4522.967
## [26,] 5421.170
## [27,] 5762.084
## [28,] 3145.049
## [29,] 1890.356
```

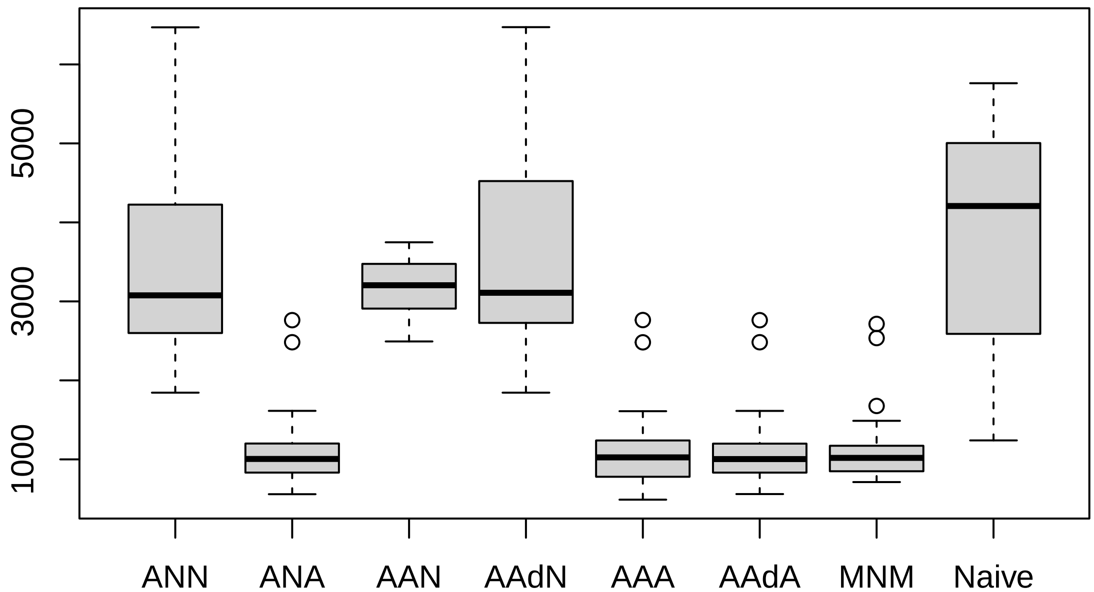
```r
errMean_1 <- colMeans(err_1)
errMean_1
```

```
##      ANN      ANA      AAN     AAdN      AAA     AAdA      MNM    Naive
## 3505.696 1109.065 3169.200 3604.524 1107.706 1108.279 1142.969 3731.227
```

```r
which.min(errMean_1)
```

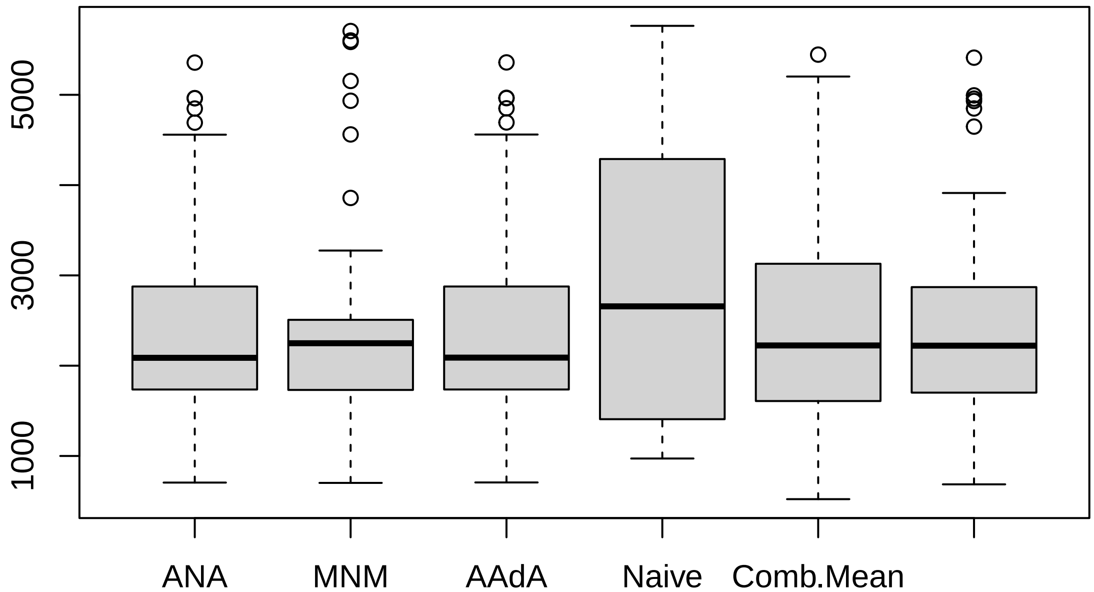
```
## AAA
##   5
```

```r
boxplot(err_1)
```




```
# What to run
modelsTest_1 <- c("ANA", "MNM", "AAA", "Naive", "CombMean", "CombMedian")
dampedTest_1 <- c(FALSE, FALSE, TRUE)

# Pre-allocate memory
omaxTest_1 <- length(y.test_1)- h_1 + 1
errTest_1 <- array(NA,c(omaxTest_1,6))
frcsTest_1 <- array(NA,c(h_1,6))

# For each forecast origin
for (o in 1:omaxTest_1){

 # Split training set
 y.trnTest_1 <- head(y,45*7-1+o) # As o increases, so will the in-sample.
 y.tstTest_1 <- tail(y,7*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_1,model=modelsTest_1[m],damped=dampedTest_1[m])
  frcsTest_1[,m] <- forecast(fitTemp,h=h_1)$mean
  errTest_1[o,m] <- mean(abs(y.tstTest_1[1:h_1]- frcsTest_1[,m]))
 }

 # Forecast using the seasonal naive
 frcsTest_1[,4] <- tail(y.trnTest_1,frequency(y.trnTest_1))[1:h_1]
 errTest_1[o,4] <- mean(abs(y.tstTest_1[1:h_1]- frcsTest_1[,4]))

 # Combinations
 # The function apply allows us to use any function we want on a matrix
 frcsTest_1[,5] <- apply(frcsTest_1[,1:4],1,mean)
 # This reads, using array frcsTest[,1:4], i.e. all rows and the first 4 columns take the mean across a
 errTest_1[o,5] <- mean(abs(y.tstTest_1[1:h_1]- frcsTest_1[,5]))
 # And for the median:
 frcsTest_1[,6] <- apply(frcsTest_1[,1:4],1,median)
 errTest_1[o,6] <- mean(abs(y.tstTest_1[1:h_1]- frcsTest_1[,6]))
```

```
}

 # Assign names to errors
 colnames(errTest_1) <- c("ANA","MNM","AAdA","Naive","Comb.Mean","Comb.Median")

 # Summarise and plot errors
 boxplot(errTest_1)
```




```
 errTestMean_1 <-colMeans(errTest_1)
 print(errTestMean_1)
```

```
##         ANA         MNM        AAdA       Naive   Comb.Mean Comb.Median
##    2422.541    2512.475    2423.530    2948.417    2463.114    2443.858
```

```
which.min(errTestMean_1)
```

```
## ANA
##   1
```

Question: Do the results change?

Answer: Using a larger test & validation set leads to ANA being the model with the lowest error in the test data, rather than Comb.Mean with the smaller test & validation set and AAA being the model best performing in validation set rather than AAdA.

2. Air passengers with rolling origin

```
y <- AirPassengers
y.train_2 <- window(AirPassengers,end=c(1959,12))
y.test_2 <- window(AirPassengers,start=c(1960,1))
```

```
print(head(y, 12*11))
```

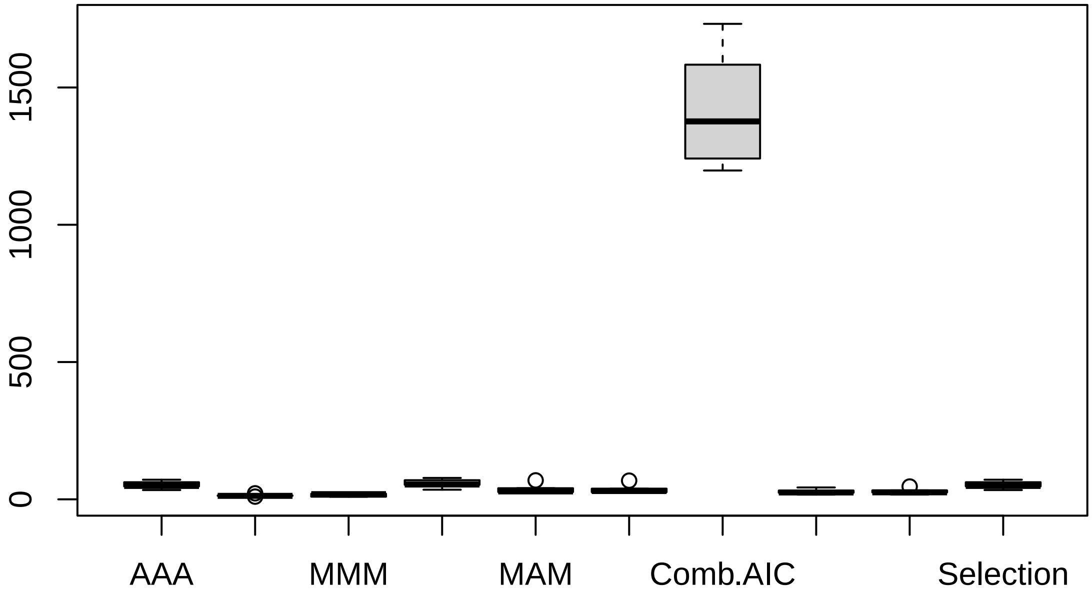
```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
```

```
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
```

```
h_2 <- 8
```

```
# What to run
modelsTest_2 <- c("AAA", "MAM", "MMM", "AAA", "MAM", "MMM", "Comb.AIC","Comb.Mean","Comb.Median","Selec
dampedTest_2 <- c(rep(FALSE, 3), rep(TRUE, 3))

# Pre-allocate memory
omaxTest_2 <- length(y.test_2)- h_2 + 1
errTest_2 <- array(NA,c(omaxTest_2,10))
frcsTest_2 <- array(NA,c(h_2,10))

# For each forecast origin
for (o in 1:omaxTest_2){

 # Split training set
 y.trnTest_2 <- head(y, 12*11 - 1+o) # As o increases, so will the in-sample
 y.tstTest_2 <- tail(y, 12 - o+1) # As o increases, the test will decrease.

 # Fit and forecast will all exponential smoothing models
 for (m in 1:6){
  fitTemp <- ets(y.trnTest_2,model=modelsTest_2[m],damped=dampedTest_2[m])
  frcsTest_2[,m] <- forecast(fitTemp,h=h_2)$mean
  errTest_2[o,m] <- mean(abs(y.tstTest_2[1:h_2]- frcsTest_2[,m]))
  AIC2 <-  AIC(fitTemp)
  waic2 [m] <- exp(-0.5 * (AIC2 - min(AIC2))) / sum(exp(-0.5 * (AIC2 - min(AIC2))))
 }

 # Forecast using AIC weights
 frcsTest_2[,7] <- cbind(y.tstTest_2[1:4] %*% cbind(waic2[1:4]))
 errTest_2[o,7] <- mean(abs(y.tstTest_2[1:h_2]- frcsTest_2[,7]))

 # Forecast using Mean
 frcsTest_2[,8] <- cbind(apply(frcsTest_2[,1:4],1,mean))
 errTest_2[o,8] <- mean(abs(y.tstTest_2[1:h_2]- frcsTest_2[,8]))

 # Forecast using median
 frcsTest_2[,9] <- cbind(apply(frcsTest_2[,1:4],1,median))
 errTest_2[o,9] <- mean(abs(y.tstTest_2[1:h_2]- frcsTest_2[,9]))

 # Forecast using Selection
 frcsTest_2[,10] <- cbind(frcsTest_2[, which.min(AIC2)])
 errTest_2[o,10] <- mean(abs(y.tstTest_2[1:h_2]- frcsTest_2[,10]))
}

 # Assign names to errors
 colnames(errTest_2) <- c("AAA", "MAM", "MMM", "AAA", "MAM", "MMM", "Comb.AIC","Comb.Mean","Comb.Median
```

```r
# Summarise and plot errors
boxplot(errTest_2)
```




```r
errTestMean_2 <-colMeans(errTest_2)
print(errTestMean_2)
```

```
##         AAA         MAM         MMM         AAA         MAM         MMM
##    52.62375    14.19167    16.47041    57.83518    37.79167    37.63183
##    Comb.AIC   Comb.Mean Comb.Median   Selection
##  1426.05000    28.20096    29.38835    52.62375
```

```r
which.min(errTestMean_2)
```

```
## MAM
##   2
```

Question: Do the results change?

Answer: Using the rolling origin for the AirPassenger dataset lead to MAM being the most accurate model, instead of Comb.Mean when using the normal evaluation.