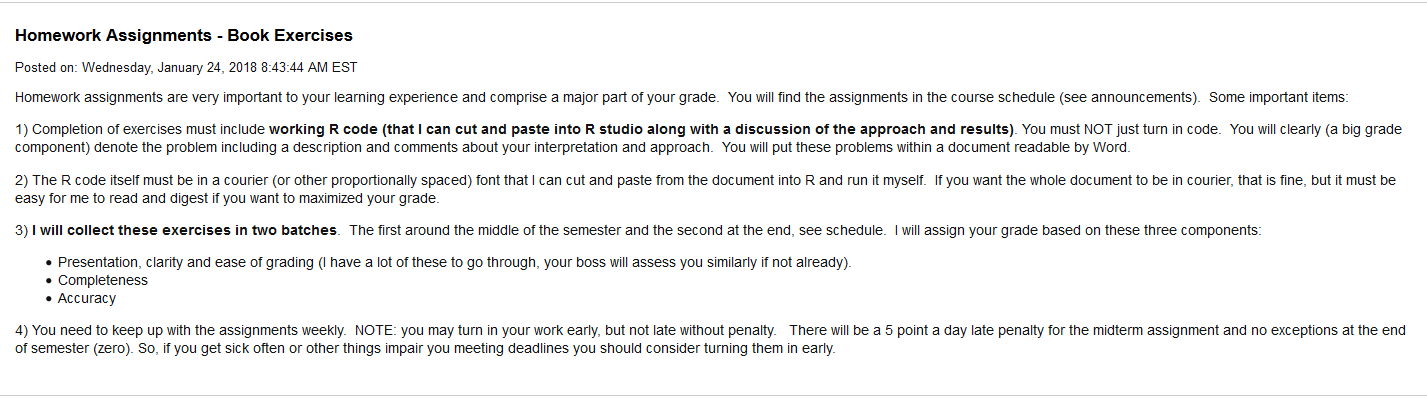
BHao\_HW1



Presentation – Could be better.

Completeness – Could have been better – see my posts.

Accuracy – Good

Grade – 86%

library(fma)  
# data(package='fma')

## 2.8.1.a. Monthly total of people on unemployed benefits in Australia (January 1956–July 1992).

Given the exponential nature of population growth in absolute terms, a log transformation makes sense. To remove the impact of population changes, we could look at the data on a per-capita basis as well.

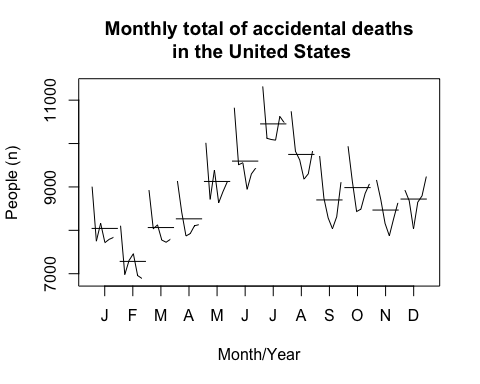
plot(log(dole), ylab = 'People (log n)', xlab = 'Year',   
 main = 'Monthly total of people on unemployed\nbenefits in Australia')



## 2.8.1.b. Monthly total of accidental deaths in the United States (January 1973–December 1978).

Given the 1) monthly seasonality of this data and 2) the general declining trend over the years, the monthplot chart seemed like an appropriate choice. To remove the impact of population changes, we could look at the data on a per-capita basis as well.

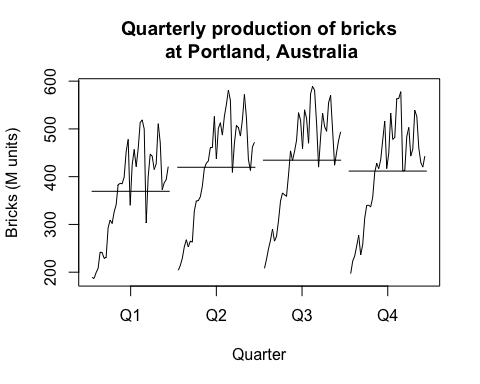
monthplot(usdeaths, ylab = 'People (n)', xlab = 'Month/Year',   
 main = 'Monthly total of accidental deaths\n in the United States')



## 2.8.1.c. Quarterly production of bricks (in millions of units) at Portland, Australia (March 1956–September 1994).

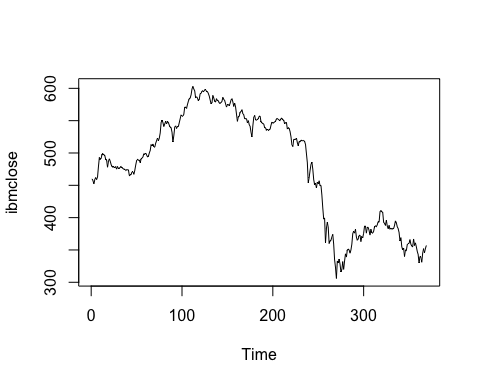
Again, there appears to be some quarterly seasonality in the data as well as a general increase over time (with a few years of sharp decline); as such, a monthplot (or quarterly in this case) seems to capture that information well.

monthplot(bricksq, ylab = 'Bricks (M units)', xlab = 'Quarter',   
 main = 'Quarterly production of bricks\n at Portland, Australia')



## 2.8.3.a. Consider the daily closing IBM stock prices (data set ibmclose). Produce some plots of the data in order to become familiar with it.

plot(ibmclose)



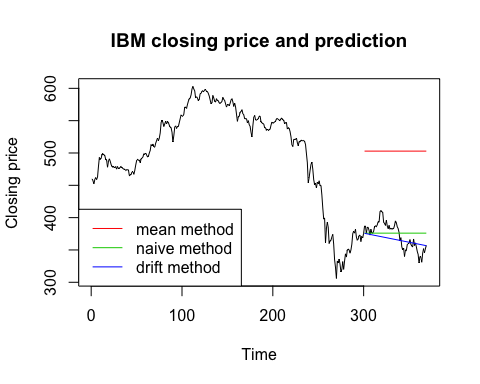
## 2.8.3.b. Split the data into a training set of 300 observations and a test set of 69 observations.

ibm\_train = window(ibmclose, start=1 , end=300)  
ibm\_test = window(ibmclose, start=301)

## 2.8.3.c. Try various benchmark methods to forecast the training set and compare the results on the test set. Which method did best?

Of the mean, naive and seasonal naive methods, the seasonal naive method produced the most accurate result.

ibm\_meanfit = meanf( ibm\_train, h = 69)  
ibm\_naivefit = rwf( ibm\_train, h = 69)  
ibm\_driftfit = rwf( ibm\_train, h = 69, drift = TRUE)  
  
plot(ibmclose, ylab = 'Closing price', xlab = 'Time',   
 main = 'IBM closing price and prediction')  
lines(ibm\_meanfit$mean , col = 2)  
lines(ibm\_naivefit$mean, col = 3)  
lines(ibm\_driftfit$mean, col = 4)  
legend('bottomleft', lty = 1, col = c(2, 3, 4),  
 legend = c('mean method', 'naive method', 'drift method'))



accuracy(ibm\_meanfit , ibm\_test)

## ME RMSE MAE MPE MAPE  
## Training set 1.660438e-14 73.61532 58.72231 -2.642058 13.03019  
## Test set -1.306180e+02 132.12557 130.61797 -35.478819 35.47882  
## MASE ACF1 Theil's U  
## Training set 11.52098 0.9895779 NA  
## Test set 25.62649 0.9314689 19.05515

accuracy(ibm\_naivefit, ibm\_test)

## ME RMSE MAE MPE MAPE MASE  
## Training set -0.2809365 7.302815 5.09699 -0.08262872 1.115844 1.000000  
## Test set -3.7246377 20.248099 17.02899 -1.29391743 4.668186 3.340989  
## ACF1 Theil's U  
## Training set 0.1351052 NA  
## Test set 0.9314689 2.973486

accuracy(ibm\_driftfit, ibm\_test)

## ME RMSE MAE MPE MAPE  
## Training set 2.870480e-14 7.297409 5.127996 -0.02530123 1.121650  
## Test set 6.108138e+00 17.066963 13.974747 1.41920066 3.707888  
## MASE ACF1 Theil's U  
## Training set 1.006083 0.1351052 NA  
## Test set 2.741765 0.9045875 2.361092

BHao\_HW2

library(fma)  
# data(package='fma')

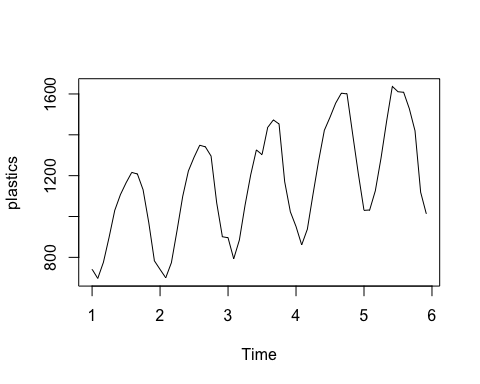
## 6.7.2 The data below represent the monthly sales (in thousands) of product A for a plastics manufacturer for years 1 through 5 (data set plastics).

plastics

## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec  
## 1 742 697 776 898 1030 1107 1165 1216 1208 1131 971 783  
## 2 741 700 774 932 1099 1223 1290 1349 1341 1296 1066 901  
## 3 896 793 885 1055 1204 1326 1303 1436 1473 1453 1170 1023  
## 4 951 861 938 1109 1274 1422 1486 1555 1604 1600 1403 1209  
## 5 1030 1032 1126 1285 1468 1637 1611 1608 1528 1420 1119 1013

## 6.7.2.a Plot the time series of sales of product A. Can you identify seasonal fluctuations and/or a trend?

plot(plastics)



## 6.7.2.b Use a classical multiplicative decomposition to calculate the trend-cycle and seasonal indices.

fit = decompose(plastics, type='multiplicative')  
  
# seasonal indices   
fit$seasonal

## Jan Feb Mar Apr May Jun Jul  
## 1 0.7670466 0.7103357 0.7765294 0.9103112 1.0447386 1.1570026 1.1636317  
## 2 0.7670466 0.7103357 0.7765294 0.9103112 1.0447386 1.1570026 1.1636317  
## 3 0.7670466 0.7103357 0.7765294 0.9103112 1.0447386 1.1570026 1.1636317  
## 4 0.7670466 0.7103357 0.7765294 0.9103112 1.0447386 1.1570026 1.1636317  
## 5 0.7670466 0.7103357 0.7765294 0.9103112 1.0447386 1.1570026 1.1636317  
## Aug Sep Oct Nov Dec  
## 1 1.2252952 1.2313635 1.1887444 0.9919176 0.8330834  
## 2 1.2252952 1.2313635 1.1887444 0.9919176 0.8330834  
## 3 1.2252952 1.2313635 1.1887444 0.9919176 0.8330834  
## 4 1.2252952 1.2313635 1.1887444 0.9919176 0.8330834  
## 5 1.2252952 1.2313635 1.1887444 0.9919176 0.8330834

# trend-cycle indices   
fit$trend

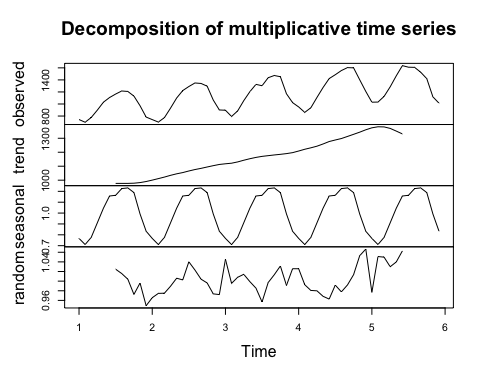
## Jan Feb Mar Apr May Jun Jul  
## 1 NA NA NA NA NA NA 976.9583  
## 2 1000.4583 1011.2083 1022.2917 1034.7083 1045.5417 1054.4167 1065.7917  
## 3 1117.3750 1121.5417 1130.6667 1142.7083 1153.5833 1163.0000 1170.3750  
## 4 1208.7083 1221.2917 1231.7083 1243.2917 1259.1250 1276.5833 1287.6250  
## 5 1374.7917 1382.2083 1381.2500 1370.5833 1351.2500 1331.2500 NA  
## Aug Sep Oct Nov Dec  
## 1 977.0417 977.0833 978.4167 982.7083 990.4167  
## 2 1076.1250 1084.6250 1094.3750 1103.8750 1112.5417  
## 3 1175.5000 1180.5417 1185.0000 1190.1667 1197.0833  
## 4 1298.0417 1313.0000 1328.1667 1343.5833 1360.6250  
## 5 NA NA NA NA NA

## 6.7.2.c Do the results support the graphical interpretation from part (a)?

Yes, the original data exhibits an upward trend with annual seasons, and the classical decomposition captures that.

## 6.7.2.d Compute and plot the seasonally adjusted data.

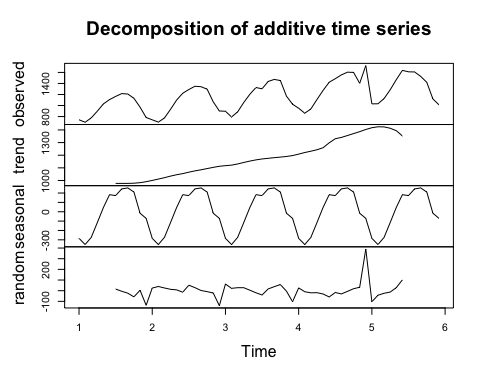
# plot of decomposition   
plot(fit)



## 6.7.2.e Change one observation to be an outlier (e.g., add 500 to one observation), and recompute the seasonally adjusted data. What is the effect of the outlier?

The outlier doesn't seem to affect the trend-cycle component, but it certainly changes the magnitudes of the seasonal and random components (although the shape of the seasonal component still looks reasonably the same).

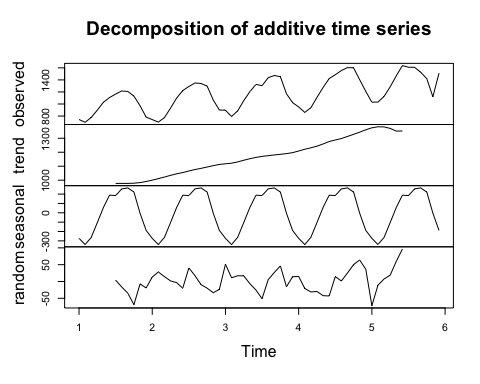
set.seed(123)  
plastics2 = plastics   
plastics2[sample.int(60, 1)] = plastics2[sample.int(60, 1)] + 500   
  
# plot decomposition   
fit2 = decompose(plastics2)  
plot(fit2)



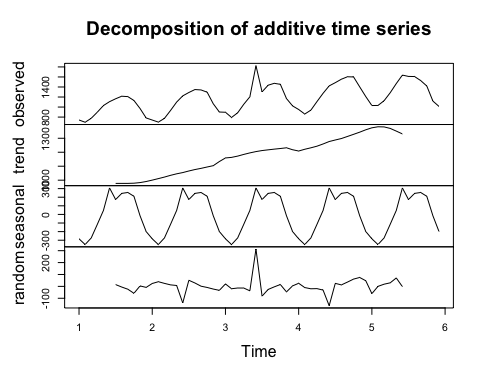
## 6.7.2.f Does it make any difference if the outlier is near the end rather than in the middle of the time series?

The location of the outlier does not seem to make much difference. In both cases, the outlier causes the seasonal and random components to be much larger than when no outlier exists.

plastics\_end = plastics  
plastics\_end[60] = plastics\_end[60] + 500  
  
plastics\_mid = plastics  
plastics\_mid[30] = plastics\_mid[30] + 500  
  
# plot decomposition   
fit\_end = decompose(plastics\_end)  
plot(fit\_end)

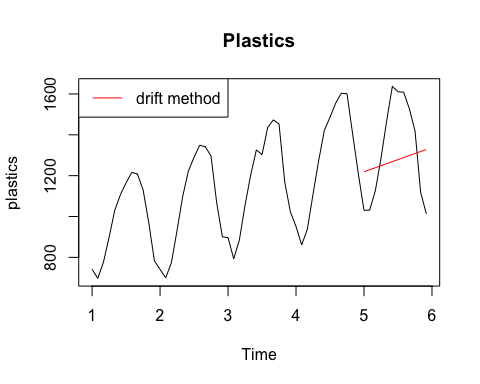


fit\_mid = decompose(plastics\_mid)  
plot(fit\_mid)



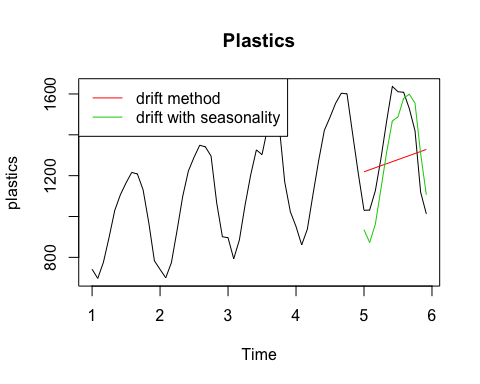
## 6.7.2.g Use a random walk with drift to produce forecasts of the seasonally adjusted data.

# create training and test sets   
plastics\_train = window(plastics, start=1, end=4.99)  
plastics\_test = window(plastics, start=5)  
  
# fit random walk with drift on training set   
plastics\_driftfit = rwf(plastics\_train, h = 12, drift = TRUE)  
  
# plot   
plot(plastics, main = 'Plastics')  
lines(plastics\_driftfit$mean, col = 2)  
legend('topleft', lty = 1, col = c(2),  
 legend = c('drift method'))



## 6.7.2.h Reseasonalize the results to give forecasts on the original scale.

# multiply by seasonality index   
plastics\_driftfitSeas = plastics\_driftfit$mean \* fit$seasonal[49:60]   
  
# plot   
plot(plastics, main = 'Plastics')  
lines(plastics\_driftfit$mean, col = 2)  
lines(plastics\_driftfitSeas , col = 3)  
legend('topleft', lty = 1, col = c(2, 3),  
 legend = c('drift method', 'drift with seasonality'))



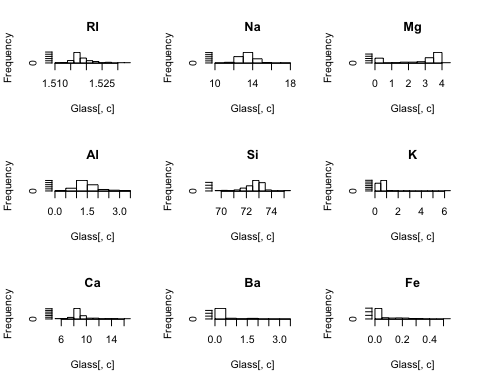
BHao\_HW3

# 3.1

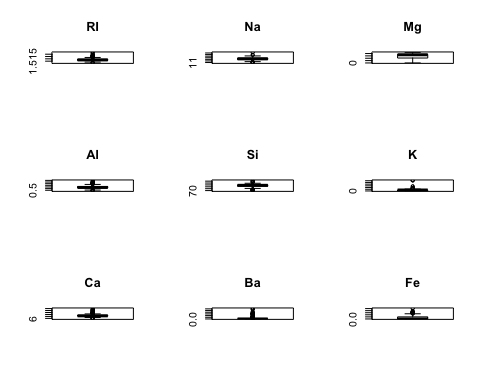
library(mlbench)  
data(Glass)

## 3.1.a. Using visualizations, explore the predictor variables to understand their distributions as well as the relationships between predictors. (given the rich set of visuals in R, this is a bit lacking)

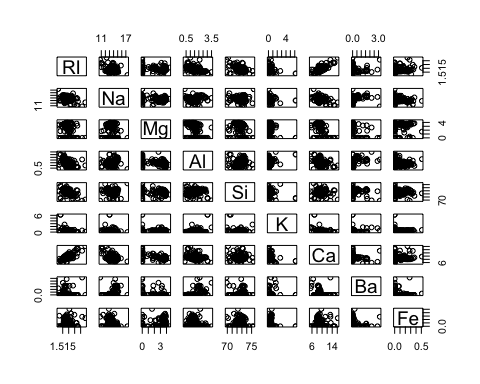
preds = colnames(Glass)  
  
# check histograms for each predictor   
par(mfrow = c(3, 3))  
for (c in 1:(length(preds)-1)){  
 hist(Glass[, c], main = preds[c])  
}



# check boxplots for each predictor   
par(mfrow = c(3, 3))  
for (c in 1:(length(preds)-1)){  
 boxplot(Glass[, c], main = preds[c])  
}



# check scatter plot matrix   
pairs(Glass[, 1:9])



## 3.1.b. Do their appear to be any outliers in the data? Are any predictors skewed?

Based on the histograms and boxplots, the following predictors appear to contain outliers: RI, Na, Al, Si, K, Ca, Ba, Fe. However, K, Ba, Fe are highly skewed to the right, so it's harder to see if the observations in the tails are in fact outliers. YOU ARE TO SHOW YOUR WORK! YOUR CODE!

## 3.1.c. Are there any relevant transformations of one or more predictors that might improve the classification model?

Center and scale - since the predictors have significantly different scales, depending on the classification model, centering and scaling may improve classification results.

Log transformation - since predictors K, Ba and Fe are highly skewed, we can consider using a log transformation. WHERE IS IT??????

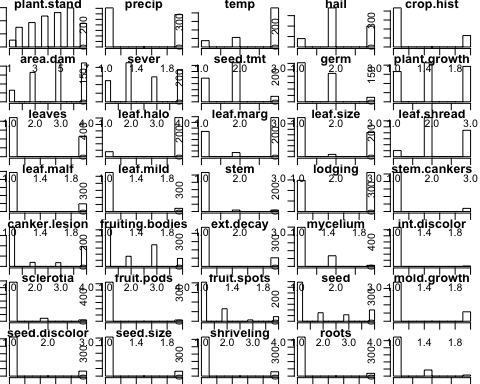
# 3.2

data("Soybean")

## 3.2.a. Investigate the frequency distributions for the categorical predictors. Are any of the distributions degenerate in the ways discussed earlier in this chapter?

Although based on the distributions several predictors only have a couple values, only three predictors meet the rules outlined in the chapter for degenerate variables: stem, int.discolor, fruit.pods

preds = colnames(Soybean[, 2:length(colnames(Soybean))])  
  
# check historgrams for each predictor   
par(mfrow = c(7, 5), mar = c(.5, .5, .5, .5))  
for (c in 2:length(colnames(Soybean))){  
 hist(as.numeric(Soybean[, c]),   
 main = preds[c]) # need to change margins to prevent error  
}



# calculate fraction of unique values and ratio of frequency between 1st and 2nd most prevalent value   
df = data.frame(pred = as.character(),  
 deg = as.character())  
for (c in 2:length(colnames((Soybean)))){  
 uniq\_frac = (length(unique(Soybean[, c])) / length(Soybean[, c])) < 0.1   
  
 # determine frequency of top 1 and 2 values   
 var1\_freq = sort(table(Soybean[, c]), decreasing = TRUE, useNA = 'ifany')[1]  
 var2\_freq = sort(table(Soybean[, c]), decreasing = TRUE, useNA = 'ifany')[2]  
 freq\_ratio= (var1\_freq / var2\_freq) > 20   
 uniq\_frac & freq\_ratio  
 df\_temp = data.frame(preds[c], uniq\_frac & freq\_ratio)  
 df = rbind(df, df\_temp)  
}  
colnames(df)= c('pred', 'deg')  
  
# return only those predictors that meet degenerate rules   
df[df['deg'] == TRUE, ]

## pred deg  
## 07 stem TRUE  
## 013 int.discolor TRUE  
## 015 fruit.pods TRUE

## 3.2.b. Roughly 18% of the data are missing. Are there particular predictors that are more likely to be missing? Is the pattern of missing data related to the classes?

Yes, the first table below shows the predictors by number of NAs, most to least.

The second table below shows the number of complete rows (i.e. no NAs) by Class. As shown, only phytophthora-rot has complete and incomplete records. All other classes have records that are all complete or are all missing values.

The third table below shows the number of NAs for each predictor for each Class. Here, we see that typically when a Class is missing data for a predictor, all observations in that Class are missing data for that particular predictor. You should note - There are a total of 2337 missing values spread across the predictors.

library(dplyr)  
library(tidyr)  
  
Soybean %>% select(-one\_of(c('Class'))) %>%   
 summarise\_each(funs(sum(is.na(.)))) %>%  
 gather(pred, n\_na) %>%   
 arrange(desc(n\_na))

## pred n\_na  
## 1 hail 121  
## 2 sever 121  
## 3 seed.tmt 121  
## 4 lodging 121  
## 5 germ 112  
## 6 leaf.mild 108  
## 7 fruiting.bodies 106  
## 8 fruit.spots 106  
## 9 seed.discolor 106  
## 10 shriveling 106  
## 11 leaf.shread 100  
## 12 seed 92  
## 13 mold.growth 92  
## 14 seed.size 92  
## 15 leaf.halo 84  
## 16 leaf.marg 84  
## 17 leaf.size 84  
## 18 leaf.malf 84  
## 19 fruit.pods 84  
## 20 precip 38  
## 21 stem.cankers 38  
## 22 canker.lesion 38  
## 23 ext.decay 38  
## 24 mycelium 38  
## 25 int.discolor 38  
## 26 sclerotia 38  
## 27 plant.stand 36  
## 28 roots 31  
## 29 temp 30  
## 30 crop.hist 16  
## 31 plant.growth 16  
## 32 stem 16  
## 33 date 1  
## 34 area.dam 1  
## 35 leaves 0

any\_na = Soybean %>% complete.cases()  
Soybean %>% mutate(any\_na = any\_na) %>%   
 group\_by(Class, any\_na) %>%   
 summarise(n = n()) %>%   
 spread(any\_na, n) %>%   
 arrange(`TRUE`, `FALSE`, Class) %>%  
 rename('has\_NAs' = `FALSE`, 'no\_NAs' = `TRUE`)

## Source: local data frame [19 x 3]  
## Groups: Class [19]  
##   
## Class has\_NAs no\_NAs  
## <fctr> <int> <int>  
## 1 phytophthora-rot 68 20  
## 2 bacterial-blight NA 20  
## 3 bacterial-pustule NA 20  
## 4 charcoal-rot NA 20  
## 5 diaporthe-stem-canker NA 20  
## 6 downy-mildew NA 20  
## 7 phyllosticta-leaf-spot NA 20  
## 8 powdery-mildew NA 20  
## 9 purple-seed-stain NA 20  
## 10 rhizoctonia-root-rot NA 20  
## 11 anthracnose NA 44  
## 12 brown-stem-rot NA 44  
## 13 alternarialeaf-spot NA 91  
## 14 frog-eye-leaf-spot NA 91  
## 15 brown-spot NA 92  
## 16 herbicide-injury 8 NA  
## 17 cyst-nematode 14 NA  
## 18 diaporthe-pod-&-stem-blight 15 NA  
## 19 2-4-d-injury 16 NA

Soybean %>% mutate(n = NaN) %>%  
 group\_by(Class) %>%  
 summarise\_each(funs(sum(is.na(.))))

## # A tibble: 19 × 37  
## Class date plant.stand precip temp hail  
## <fctr> <int> <int> <int> <int> <int>  
## 1 2-4-d-injury 1 16 16 16 16  
## 2 alternarialeaf-spot 0 0 0 0 0  
## 3 anthracnose 0 0 0 0 0  
## 4 bacterial-blight 0 0 0 0 0  
## 5 bacterial-pustule 0 0 0 0 0  
## 6 brown-spot 0 0 0 0 0  
## 7 brown-stem-rot 0 0 0 0 0  
## 8 charcoal-rot 0 0 0 0 0  
## 9 cyst-nematode 0 14 14 14 14  
## 10 diaporthe-pod-&-stem-blight 0 6 0 0 15  
## 11 diaporthe-stem-canker 0 0 0 0 0  
## 12 downy-mildew 0 0 0 0 0  
## 13 frog-eye-leaf-spot 0 0 0 0 0  
## 14 herbicide-injury 0 0 8 0 8  
## 15 phyllosticta-leaf-spot 0 0 0 0 0  
## 16 phytophthora-rot 0 0 0 0 68  
## 17 powdery-mildew 0 0 0 0 0  
## 18 purple-seed-stain 0 0 0 0 0  
## 19 rhizoctonia-root-rot 0 0 0 0 0  
## # ... with 31 more variables: crop.hist <int>, area.dam <int>,  
## # sever <int>, seed.tmt <int>, germ <int>, plant.growth <int>,  
## # leaves <int>, leaf.halo <int>, leaf.marg <int>, leaf.size <int>,  
## # leaf.shread <int>, leaf.malf <int>, leaf.mild <int>, stem <int>,  
## # lodging <int>, stem.cankers <int>, canker.lesion <int>,  
## # fruiting.bodies <int>, ext.decay <int>, mycelium <int>,  
## # int.discolor <int>, sclerotia <int>, fruit.pods <int>,  
## # fruit.spots <int>, seed <int>, mold.growth <int>, seed.discolor <int>,  
## # seed.size <int>, shriveling <int>, roots <int>, n <int>

## 3.2.c. Develop a strategy for handling missing data, either by eliminating predictors or imputation.

Since so many Classes consistently have missing data, one possibility is to create a dummy variable that indicates such a class. Another approach might be to use two different models, one for those with complete data and another for ones with incomplete data (and drop the empty predictors from the latter model).

I don't think imputation makes sense here because the missing values seem to be missing completely most of the time for a given class, so imputing based on other classes would just create a static (zero variance) value for that particular class and predictor combination. GOOD

BHao\_HW4

## 7.8.1.a. Plot the series and discuss the main features of the data.

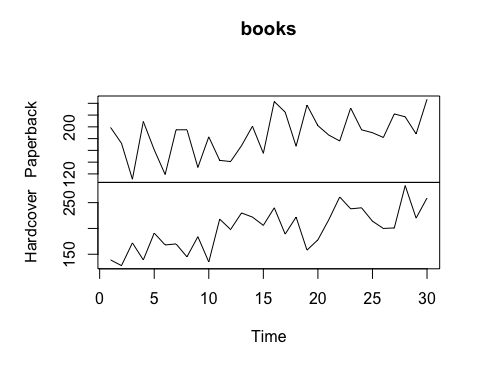
The data represent daily unit sales for paperback and hardcover books at the same store over 30 days. Optically, it appears that sales trend upward, but there are no obvious patterns in day-to-day sales.

library(fma)

## Loading required package: forecast

## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2018c.  
## 1.0/zoneinfo/America/Los\_Angeles'

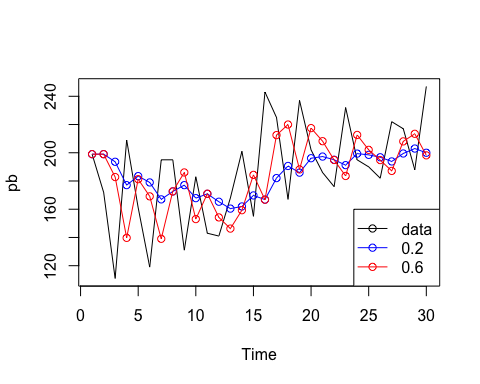
plot(books)



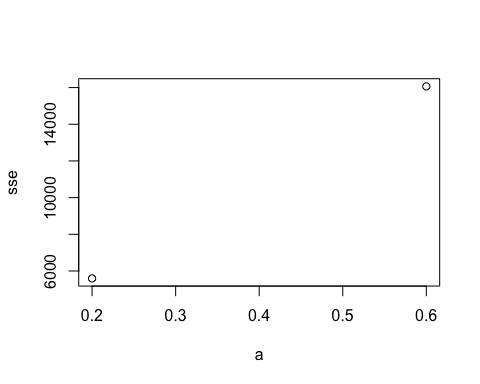
## 7.8.1.b. Use simple exponential smoothing with the ses function (setting initial="simple") and explore different values of αα for the paperback series. Record the within-sample SSE for the one-step forecasts. Plot SSE against αα and find which value of αα works best. What is the effect of αα on the forecasts?

In this case, the smaller alpha value 0.2 performed much better in terms of SSE than did the larger alpha value 0.6.

pb = books[, 'Paperback']  
  
fit1 = ses(pb, alpha = 0.2, initial = 'simple')  
fit2 = ses(pb, alpha = 0.6, initial = 'simple')  
plot(pb)  
lines(fitted(fit1), col='blue', type='o')  
lines(fitted(fit2), col='red' , type='o')  
legend('bottomright', lty=1, col=c(1,'blue','red'),  
 c('data', expression(a = 0.2), expression(a = 0.6)),  
 pch=1)



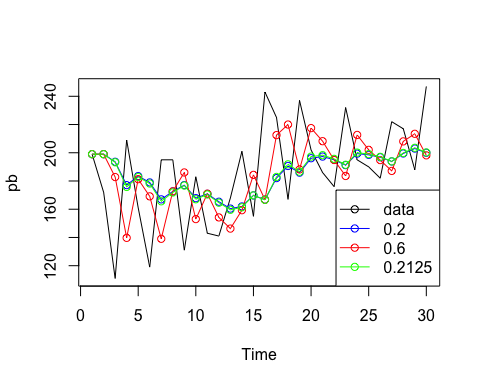
sse1 = sum((fitted(fit1) - mean(pb))^2)  
sse2 = sum((fitted(fit2) - mean(pb))^2)  
  
plot(data.frame(a = c(0.2, 0.6), sse = c(sse1, sse2)))



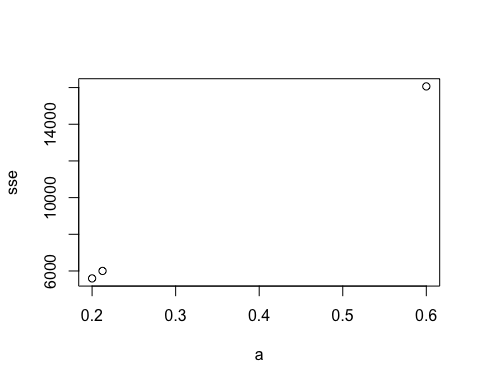
## 7.8.1.c. Now let ses select the optimal value of αα. Use this value to generate forecasts for the next four days. Compare your results with 2.

In this case, ses selected 0.1685 as the optimal alpha value; however, the SSE was still higher than the 0.2 alpha value above.

fit3 = ses(pb, initial = 'simple')  
  
plot(pb)  
lines(fitted(fit1), col='blue' , type='o')  
lines(fitted(fit2), col='red' , type='o')  
lines(fitted(fit3), col='green', type='o')  
legend('bottomright', lty=1, col=c(1,'blue','red','green'),  
 c('data', expression(a = 0.2), expression(a = 0.6), expression(a = 0.2125)),  
 pch=1)



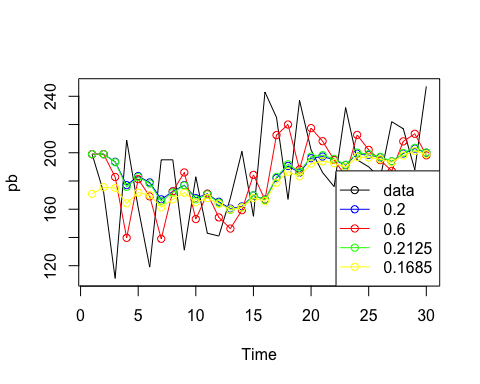
sse3 = sum((fitted(fit3) - mean(pb))^2)  
  
plot(data.frame(a = c(0.2, 0.6, 0.2125), sse = c(sse1, sse2, sse3)))



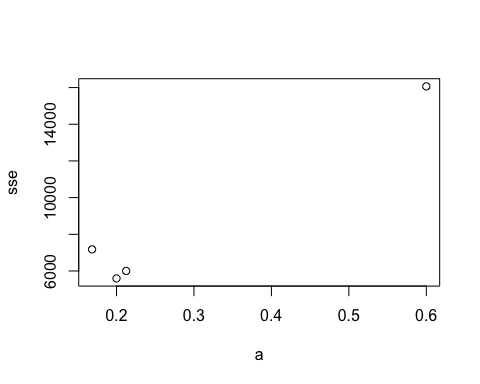
## 7.8.1.d. Repeat but with initial="optimal". How much difference does an optimal initial level make?

Interestingly, allowing ses to select the optimal initial value, the resulting SSE was higher than than above.

fit4 = ses(pb, initial = 'optimal')  
  
plot(pb)  
lines(fitted(fit1), col='blue' , type='o')  
lines(fitted(fit2), col='red' , type='o')  
lines(fitted(fit3), col='green' , type='o')  
lines(fitted(fit4), col='yellow', type='o')  
legend('bottomright', lty=1, col=c(1,'blue','red','green','yellow'),  
 c('data', expression(a = 0.2), expression(a = 0.6), expression(a = 0.2125), expression(a = 0.1685)),  
 pch=1)



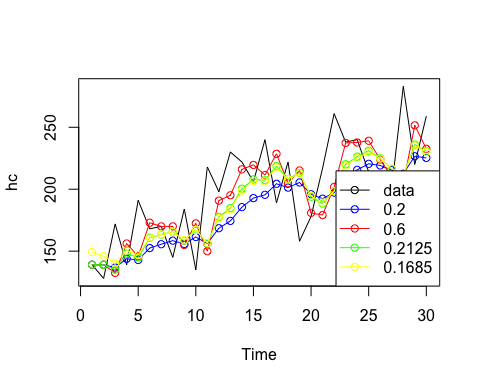
sse4 = sum((fitted(fit4) - mean(pb))^2)  
  
plot(data.frame(a = c(0.2, 0.6, 0.2125, 0.1685), sse = c(sse1, sse2, sse3, sse4)))



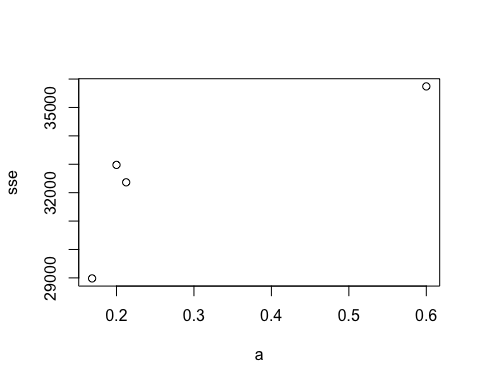
## 7.8.1.e Repeat steps (b)–(d) with the hardcover series.

In the case of hardcovers, allowing ses to select the optimal initial and alpha values resulted in the lowest SSE by far.

hc = books[, 'Hardcover']  
  
fit1 = ses(hc, alpha = 0.2, initial = 'simple')  
fit2 = ses(hc, alpha = 0.6, initial = 'simple')  
fit3 = ses(hc, initial = 'simple')  
fit4 = ses(hc, initial = 'optimal')  
  
plot(hc)  
lines(fitted(fit1), col='blue' , type='o')  
lines(fitted(fit2), col='red' , type='o')  
lines(fitted(fit3), col='green' , type='o')  
lines(fitted(fit4), col='yellow', type='o')  
legend('bottomright', lty=1, col=c(1,'blue','red','green','yellow'),  
 c('data', expression(a = 0.2), expression(a = 0.6), expression(a = 0.2125), expression(a = 0.1685)),  
 pch=1)



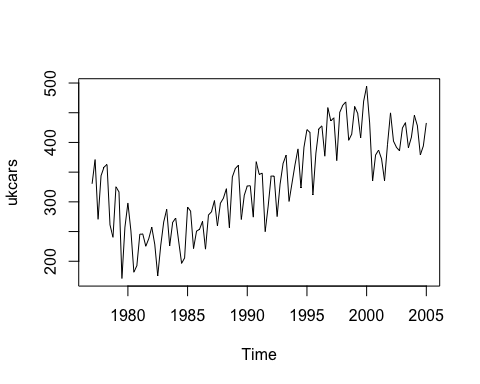
sse1 = sum((fitted(fit1) - mean(hc))^2)  
sse2 = sum((fitted(fit2) - mean(hc))^2)  
sse3 = sum((fitted(fit3) - mean(hc))^2)  
sse4 = sum((fitted(fit4) - mean(hc))^2)  
  
plot(data.frame(a = c(0.2, 0.6, 0.2125, 0.1685), sse = c(sse1, sse2, sse3, sse4)))



## 7.8.3.a. Plot the data and describe the main features of the series.

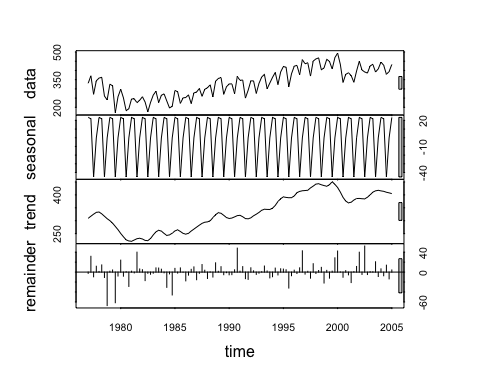
There is a clear upward trend in the data over time as well as clear seasonality.

library(expsmooth)  
plot(ukcars)



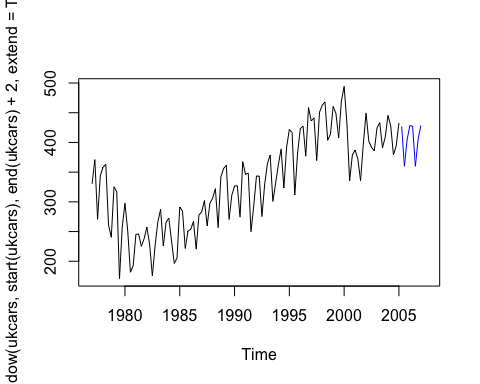
## 7.8.3.b. Decompose the series using STL and obtain the seasonally adjusted data.

fit\_stl = stl(ukcars, s.window = 'periodic', robust = TRUE)  
plot(fit\_stl)



## 7.8.3.c. Forecast the next two years of the series using an additive damped trend method applied to the seasonally adjusted data. Then reseasonalize the forecasts. Record the parameters of the method and report the RMSE of the one-step forecasts from your method.

ukcars\_seasAdj = fit\_stl$time.series[, 'trend']  
reSeas = as.numeric(tail(fit\_stl$time.series[, 'seasonal'], 8)) # use prev 8 periods to reseasonalize forecast   
  
fit\_damped = ses(ukcars\_seasAdj, h = 8, damped = TRUE)  
fit\_damped\_reSeas = fit\_damped$mean + reSeas # combine with forecasted mean   
  
plot(window(ukcars, start(ukcars), end(ukcars) + 2, extend = TRUE)) # extend window to include full forecast   
lines(fit\_damped\_reSeas, col='blue')

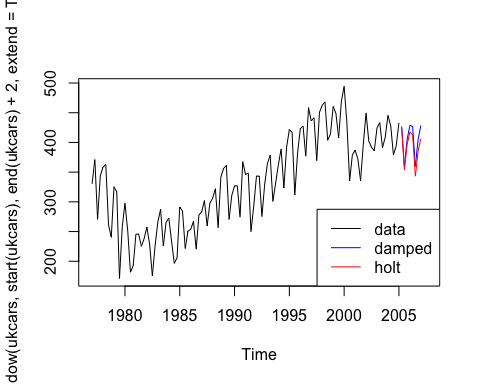


# output model parameters   
summary(fit\_damped)

##   
## Forecast method: Simple exponential smoothing  
##   
## Model Information:  
## Simple exponential smoothing   
##   
## Call:  
## ses(y = ukcars\_seasAdj, h = 8, damped = TRUE)   
##   
## Smoothing parameters:  
## alpha = 0.9999   
##   
## Initial states:  
## l = 309.5501   
##   
## sigma: 7.5104  
##   
## AIC AICc BIC   
## 995.8765 996.0967 1004.0587   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.8455707 7.510414 6.240617 0.2092296 1.972227 0.3181851  
## ACF1  
## Training set 0.766302  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## 2005 Q2 405.09 395.4650 414.7150 390.3698 419.8101  
## 2005 Q3 405.09 391.4789 418.7011 384.2736 425.9064  
## 2005 Q4 405.09 388.4201 421.7598 379.5957 430.5843  
## 2006 Q1 405.09 385.8415 424.3385 375.6519 434.5281  
## 2006 Q2 405.09 383.5696 426.6104 372.1774 438.0026  
## 2006 Q3 405.09 381.5157 428.6643 369.0362 441.1438  
## 2006 Q4 405.09 379.6269 430.5531 366.1475 444.0325  
## 2007 Q1 405.09 377.8688 432.3112 363.4588 446.7212

## 7.8.3.d. Forecast the next two years of the series using Holt's linear method applied to the seasonally adjusted data. Then reseasonalize the forecasts. Record the parameters of the method and report the RMSE of of the one-step forecasts from your method.

fit\_holt = holt(ukcars\_seasAdj, h = 8)  
fit\_holt\_reSeas = fit\_holt$mean + reSeas   
  
plot(window(ukcars, start(ukcars), end(ukcars) + 2, extend = TRUE))  
lines(fit\_damped\_reSeas, col='blue')  
lines(fit\_holt\_reSeas , col='red')  
legend('bottomright', lty=1, col=c(1,'blue','red'),  
 c('data', 'damped', 'holt'))



# output model parameters   
summary(fit\_holt)

##   
## Forecast method: Holt's method  
##   
## Model Information:  
## Holt's method   
##   
## Call:  
## holt(y = ukcars\_seasAdj, h = 8)   
##   
## Smoothing parameters:  
## alpha = 0.9999   
## beta = 0.9999   
##   
## Initial states:  
## l = 299.8252   
## b = 14.2267   
##   
## sigma: 5.0564  
##   
## AIC AICc BIC   
## 910.4623 911.0231 924.0993   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set -0.1504732 5.056389 3.901235 -0.0008004432 1.218107 0.1989089  
## ACF1  
## Training set 0.3659724  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## 2005 Q2 402.3147 395.8347 408.7947 392.4044 412.2251  
## 2005 Q3 399.5397 385.0511 414.0283 377.3813 421.6981  
## 2005 Q4 396.7647 372.5210 421.0084 359.6872 433.8422  
## 2006 Q1 393.9897 358.5007 429.4787 339.7140 448.2654  
## 2006 Q2 391.2147 343.1625 439.2669 317.7251 464.7042  
## 2006 Q3 388.4397 326.6306 450.2487 293.9108 482.9685  
## 2006 Q4 385.6646 308.9999 462.3293 268.4161 502.9132  
## 2007 Q1 382.8896 290.3460 475.4332 241.3564 524.4228

## 7.8.3.e. Now use ets() to choose a seasonal model for the data.

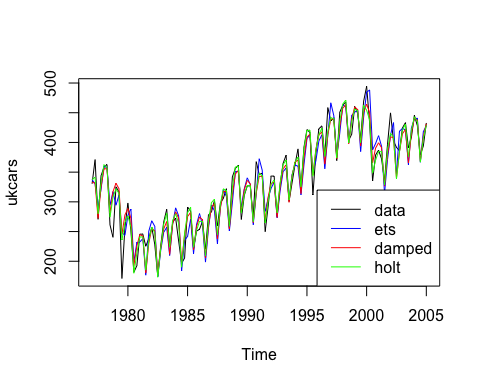
fit\_ets = ets(ukcars)  
  
# output model parameters  
summary(fit\_ets)

## ETS(A,N,A)   
##   
## Call:  
## ets(y = ukcars)   
##   
## Smoothing parameters:  
## alpha = 0.6267   
## gamma = 1e-04   
##   
## Initial states:  
## l = 313.0916   
## s=-1.1271 -44.606 21.5553 24.1778  
##   
## sigma: 25.2579  
##   
## AIC AICc BIC   
## 1277.980 1279.047 1297.072   
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 1.324962 25.25792 20.12508 -0.1634983 6.609629 0.6558666  
## ACF1  
## Training set 0.01909295

## 7.8.3.f. Compare the RMSE of the fitted model with the RMSE of the model you obtained using an STL decomposition with Holt's method. Which gives the better in-sample fits?

The linear Holt method produced the lowest in-sample RMSE.

plot(ukcars)  
lines(fitted(fit\_ets), col='blue')  
# although the below two comparisons are not exactly fair since they are using the actual seasonality estimates   
lines(fitted(fit\_damped) + fit\_stl$time.series[, 'seasonal'], col='red')  
lines(fitted(fit\_holt) + fit\_stl$time.series[, 'seasonal'], col='green')  
legend('bottomright', lty=1, col=c(1,'blue','red','green'),  
 c('data', 'ets', 'damped', 'holt'))



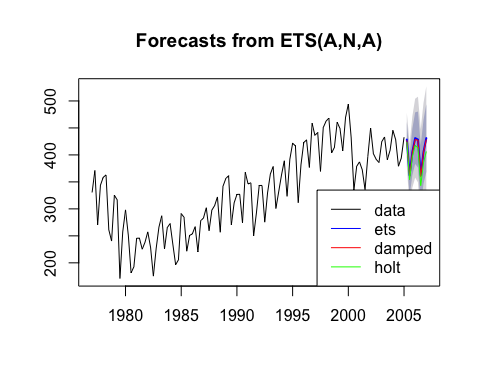
data.frame(c('ets', 'damped', 'holt'),   
 c(sqrt(fit\_ets$mse), sqrt(fit\_damped$model$mse), sqrt(fit\_holt$model$mse)))

## c..ets....damped....holt..  
## 1 ets  
## 2 damped  
## 3 holt  
## c.sqrt.fit\_ets.mse...sqrt.fit\_damped.model.mse...sqrt.fit\_holt.model.mse..  
## 1 25.257919  
## 2 7.510414  
## 3 5.056389

## 7.8.3.g. Compare the forecasts from the two approaches? Which seems most reasonable?

The ets and damped forecasts are right on top of one another. The Holt model forecasts slightly lower values. It's not clear which is the most reasonable, as they all appear quite reasonable.

plot(forecast(fit\_ets, h = 8))  
lines(fit\_damped\_reSeas, col='red')  
lines(fit\_holt\_reSeas , col='green')  
legend('bottomright', lty=1, col=c(1,'blue','red','green'),  
 c('data', 'ets', 'damped', 'holt'))



BHao\_HW5

## 8.1 Figure 8.24 shows the ACFs for 36 random numbers, 360 random numbers and for 1,000 random numbers.

### 8.1.a. Explain the differences among these figures. Do they all indicate the data are white noise?

The differences between the three figures is the number of random values used, 36, 360 and 1000, respectively. Yes, all figures indicate the data are white noise because there are no significant spikes?

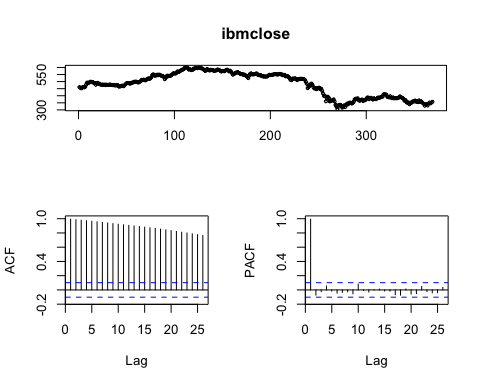
### 8.1.b. Why are the critical values at different distances from the mean of zero? Why are the autocorrelations different in each figure when they each refer to white noise?

When there are more random values, the closer the critical values are to the mean of zero. I assume the reason is similar to why standard errors decrease with the number of observations. Each figure refers to white noise, the when the range of random values is wider, the lower the autocorrelations between the values.

## 8.2 A classic example of a non-stationary series is the daily closing IBM stock prices (data set ibmclose). Use R to plot the daily closing prices for IBM stock and the ACF and PACF. Explain how each plot shows the series is non-stationary and should be differenced.

The plot of IBM's closing price shows a downward trend. The ACF plot shows significant autocorrelation between values and their lagged counterparts. The PACF plot shows the correlation between values and their lagged counterparts but with the effects of the in-between values removed. The downward trend and significant autocorrelation in the ACF chart indicate that the series is non-stationary and should be differenced.

tsdisplay(ibmclose)

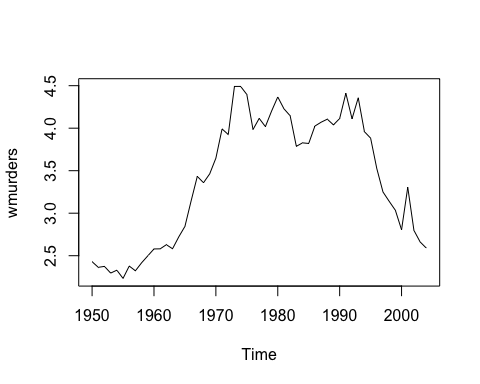


## 8.6 Consider the number of women murdered each year (per 100,000 standard population) in the United States (data set wmurders).

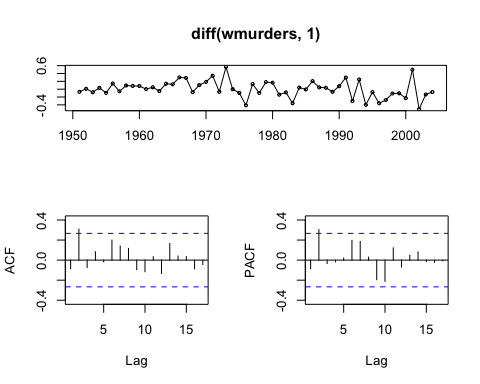
### 8.6.a. By studying appropriate graphs of the series in R, find an appropriate ARIMA(p,d,q) model for these data.

The data are clearly non-stationary as it trends up and then down over a long period of time; thus, we take the first difference which looks stationary.

plot(wmurders)



tsdisplay(diff(wmurders, 1))



The spike in lag2 of both ACF and PACF suggest evaluating ARIMA(2,1,0) and ARIMA(0,1,2) models along with a few similar variants. Based on the table below, the ARIMA(0,1,2) model produced the lowest AICc.

fit310 = Arima(wmurders, order=c(3,1,0))  
fit210 = Arima(wmurders, order=c(2,1,0))  
fit211 = Arima(wmurders, order=c(2,1,1))  
fit112 = Arima(wmurders, order=c(1,1,2))  
fit012 = Arima(wmurders, order=c(0,1,2))  
fit013 = Arima(wmurders, order=c(0,1,3))  
  
model = c('fit310', 'fit210', 'fit211', 'fit112', 'fit012', 'fit013')   
aicc = c(fit310$aicc, fit210$aicc, fit211$aicc, fit112$aicc, fit012$aicc, fit013$aicc)   
data.frame(model, aicc) %>% arrange(aicc)

## model aicc  
## 1 fit012 -12.94527  
## 2 fit210 -12.47872  
## 3 fit013 -10.63595  
## 4 fit112 -10.62525  
## 5 fit310 -10.17350  
## 6 fit211 -10.16966

### 8.6.b. Should you include a constant in the model? Explain.

No, when d >= 1 then c should be set to zero.

### 8.6.c. Write this model in terms of the backshift operator.

(1 - *B*)y = (1 + *B* + *B*^2)

where = -0.0660 and = 0.3712

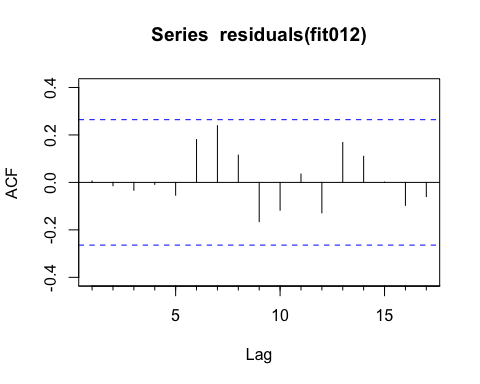
summary(fit012)

## Series: wmurders   
## ARIMA(0,1,2)   
##   
## Coefficients:  
## ma1 ma2  
## -0.0660 0.3712  
## s.e. 0.1263 0.1640  
##   
## sigma^2 estimated as 0.0422: log likelihood=9.71  
## AIC=-13.43 AICc=-12.95 BIC=-7.46  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE  
## Training set 0.0007242355 0.1997392 0.1543531 -0.08224024 4.434684  
## MASE ACF1  
## Training set 0.9491994 0.005880608

### 8.6.d. Fit the model using R and examine the residuals. Is the model satisfactory?

Yes, the residuals do look satisfactory.

Acf(residuals(fit012))



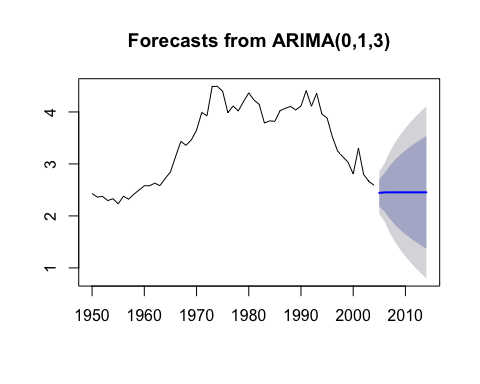
### 8.6.e. Forecast three times ahead. Check your forecasts by hand to make sure you know how they have been calculated.

forecast(fit012, h = 3)

## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## 2005 2.458450 2.195194 2.721707 2.055834 2.861066  
## 2006 2.477101 2.116875 2.837327 1.926183 3.028018  
## 2007 2.477101 1.979272 2.974929 1.715738 3.238464

### 8.6.f. Create a plot of the series with forecasts and prediction intervals for the next three periods shown.

plot(forecast(fit013))



### 8.6.g. Does auto.arima give the same model you have chosen? If not, which model do you think is better?

If I set d=1, then auto.arima finds the same model; however, without that , auto.arima selects ARIMA(0,2,3) which has a higher AICc.

auto.arima(wmurders, d = 1, stepwise = F, approximation = F)

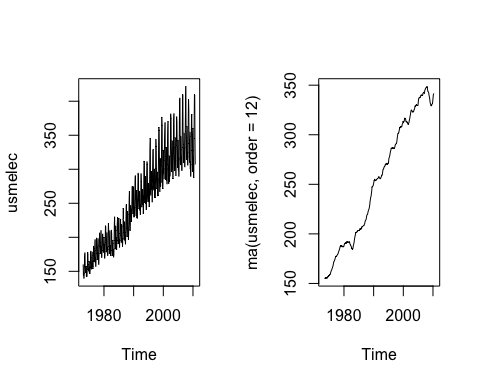
## Series: wmurders   
## ARIMA(0,1,2)   
##   
## Coefficients:  
## ma1 ma2  
## -0.0660 0.3712  
## s.e. 0.1263 0.1640  
##   
## sigma^2 estimated as 0.0422: log likelihood=9.71  
## AIC=-13.43 AICc=-12.95 BIC=-7.46

## 8.8 Consider the total net generation of electricity (in billion kilowatt hours) by the U.S. electric industry (monthly for the period 1985–1996). (Data set usmelec.) In general there are two peaks per year: in mid-summer and mid-winter.

### 8.8.a. Examine the 12-month moving average of this series to see what kind of trend is involved.

Clearly, there is an upward trend in the data, and the variance seems to increase with time. Can you expound?

par(mfrow = c(1,2))  
plot(usmelec)  
plot(ma(usmelec, order = 12))

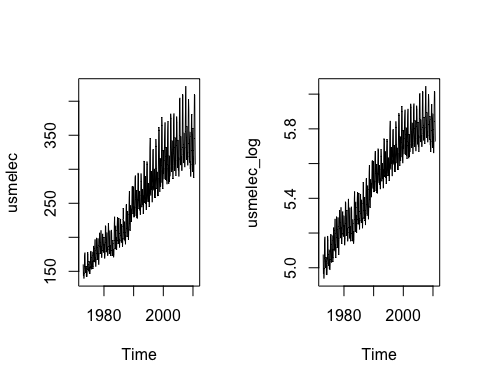


### 8.8.b. Do the data need transforming? If so, find a suitable transformation.

Given the increasing variance over time, taking the log will help stabilize the variance. A Box Cox is the right xfm……

usmelec\_log = log(usmelec)  
par(mfrow=c(1,2))

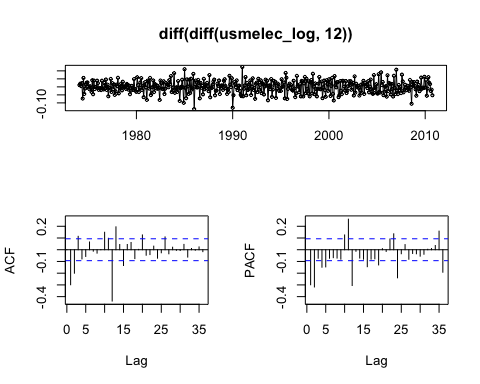
plot(usmelec)  
plot(usmelec\_log)



### 8.8.c. Are the data stationary? If not, find an appropriate differencing which yields stationary data.

No, the data are not stationary. Given the seasonality, we first take the seasonal difference and then take an additional first difference. Explain what you are doing, please

tsdisplay(diff(diff(usmelec\_log, 12)))



### 8.8.d. Identify a couple of ARIMA models that might be useful in describing the time series. Which of your models is the best according to their AIC values?

In the ACF chart, the significant spike at lag 1 suggessts a non-seasonal MA(1) component and the significant spike at lag 12 suggests a seasonal MA(1) component, so we start with an ARIMA(0,1,1)(0,1,1). We see similar spikes on the PACF chart, and thus can also try an ARIMA(1,1,0)(1,1,0) model. Of the models I tested, the ARIMA(1,1,1)(1,1,1) model produced the lowest AICc value. Hmmm….

However, that model did show a few spikes in its residual chart, so I took a look at a few more options, but it still produced the lowest AICc value.

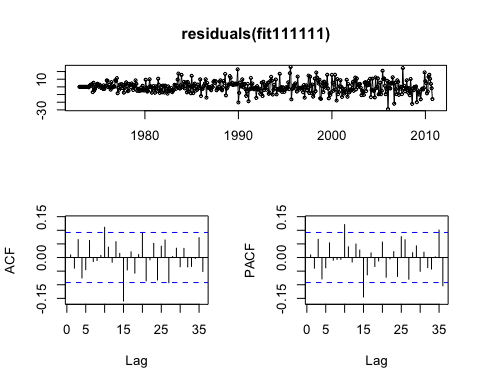
fit013011 = Arima(usmelec, order = c(0,1,3), seasonal = c(0,1,1))  
fit012011 = Arima(usmelec, order = c(0,1,2), seasonal = c(0,1,1))  
fit011011 = Arima(usmelec, order = c(0,1,1), seasonal = c(0,1,1))  
fit111111 = Arima(usmelec, order = c(1,1,1), seasonal = c(1,1,1))  
  
fit110310 = Arima(usmelec, order = c(1,1,0), seasonal = c(3,1,0))  
fit210210 = Arima(usmelec, order = c(2,1,0), seasonal = c(2,1,0))  
fit012012 = Arima(usmelec, order = c(0,1,2), seasonal = c(0,1,2))  
  
model2 = c('fit013011', 'fit012011', 'fit011011', 'fit111111', 'fit110310', 'fit210210', 'fit012012')   
aicc2 = c(fit013011$aicc, fit012011$aicc, fit011011$aicc, fit111111$aicc, fit110310$aicc, fit210210$aicc, fit012012$aicc)   
data.frame(model2, aicc2) %>% arrange(aicc2)

## model2 aicc2  
## 1 fit111111 3048.295  
## 2 fit012011 3048.427  
## 3 fit013011 3049.515  
## 4 fit012012 3050.158  
## 5 fit011011 3082.082  
## 6 fit110310 3113.753  
## 7 fit210210 3125.140

### 8.8.e. Estimate the parameters of your best model and do diagnostic testing on the residuals. Do the residuals resemble white noise? If not, try to find another ARIMA model which fits better.

Despite the few significant spikes in the residual charts, the ARIMA(1,1,1)(1,1,1) model produced the best AICc values of the models I tested. This model also produced a better AICc value than anything the auto.arima() function produced even with stepwise and approximation set to False.

tsdisplay(residuals(fit111111))



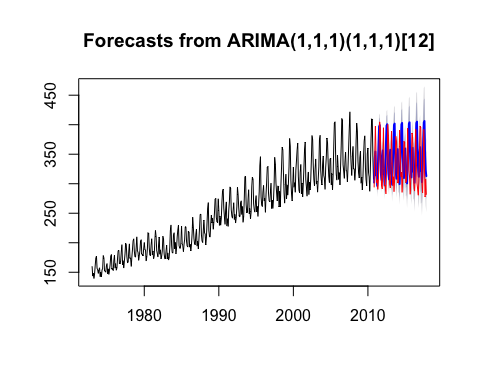
auto.arima(usmelec)

## Series: usmelec   
## ARIMA(1,0,2)(0,1,1)[12] with drift   
##   
## Coefficients:  
## ar1 ma1 ma2 sma1 drift  
## 0.9586 -0.4153 -0.2735 -0.7041 0.4268  
## s.e. 0.0210 0.0522 0.0525 0.0312 0.0676  
##   
## sigma^2 estimated as 56.23: log likelihood=-1519.25  
## AIC=3050.49 AICc=3050.68 BIC=3075.04

#auto.arima(usmelec, stepwise = F, approximation = F)

### 8.8.f. Forecast the next 15 years of generation of electricity by the U.S. electric industry. Get the latest figures from <http://data.is/zgRWCO> to check on the accuracy of your forecasts.

forecast111111 = forecast(fit111111, h = 85) # to match actual data   
  
# format actual data to compare with forecast   
actual = read.csv('/Users/brucehao/Google Drive/CUNY/git/DATA624/MER\_T07\_01.csv', stringsAsFactors = FALSE)  
actual\_recent = actual[actual$YYYYMM >= 201011, ]  
actual\_recent$Value = as.numeric(actual\_recent$Value)  
# there appears to be problem in the data where every 13 months the value is 10x what it should be   
actual\_recent$Value = ifelse(actual\_recent$Value > 1000, actual\_recent$Value/10, actual\_recent$Value)  
actual\_ts = ts(actual\_recent$Value, start=c(2010, 11), end=c(2017, 11), frequency=12)  
  
plot(forecast111111)  
lines(actual\_ts, col=2)



# calculate RMSE of prediction   
rmse = function(error){ sqrt(mean(error^2)) }  
mae = function(error){ mean(abs(error)) }  
error = forecast111111$mean - actual\_ts  
rmse(error)

## [1] 49.6056

### 8.8.f. How many years of forecasts do you think are sufficiently accurate to be usable?

Usability depends on the use case, but at least over the ~7 year period for which actual data is available, the model's error does not seem to increase over time.