Data 624: Predictive Analytics HW 1

Group 2: Alice Friedman, Kayleah Griffen, Josh Iden, Michael Ippolito

6/18/2023

## Introduction

This homework assignment includes problems from:

1. Hyndman & Athanasopoulos. “Forecasting: Principles and Practice”
2. Kuhn & Johnson. “Applied Predictive Modeling”

This accompanies readings from KJ 1,2 and 3 and HA 1,2,6,7 and 8.

## Week 1 Homework Solutions

### HA 2.1

Using the help function (?) to explore what the series gold, woolyrnq and gas represent, we get the following results in the description:

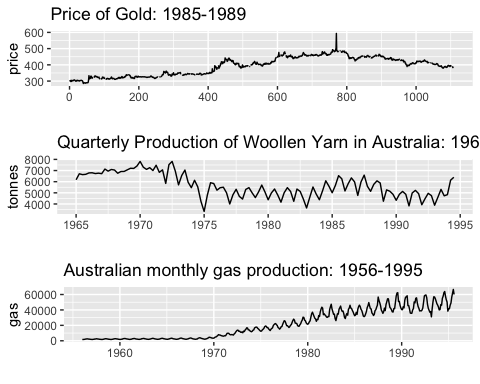
?gold “Daily morning gold prices in US dollars. 1 January 1985 – 31 March 1989”

?woolyrnq “Quarterly production of woollen yarn in Australia: tonnes. Mar 1965 – Sep 1994”

?gas Australian monthly gas production: 1956–1995.

#### a. Use autoplot() to plot each of these in separate plots.

library(fpp2)  
library(gridExtra)  
library(tidyverse)  
  
  
  
p1 = autoplot(gold) + ggtitle('Price of Gold: 1985-1989') + ylab('price') + xlab('')  
p2 = autoplot(woolyrnq) + ggtitle('Quarterly Production of Woollen Yarn in Australia: 1965 - 1994') + ylab('tonnes') + xlab('')  
p3 = autoplot(gas) + ggtitle('Australian monthly gas production: 1956-1995') + ylab('gas') + xlab('')  
  
grid.arrange(p1, p2, p3, nrow=3)



#### b. What is the frequency of each series?

Gold <- frequency(gold)  
  
Wool <- frequency(woolyrnq)  
  
Gas <- frequency(gas)  
  
Freqs <- data.frame(  
 Series = c("Gold", "Wool", "Gas"),  
 Frequency = c(Gold, Wool, Gas),  
 Seasonality = c("Daily", "Quaterly", "Monthly")  
 )   
Freqs %>% print()

## Series Frequency Seasonality  
## 1 Gold 1 Daily  
## 2 Wool 4 Quaterly  
## 3 Gas 12 Monthly

Based on frequency and the help function you can tell the gold observations are taken daily, the woolyrnq is quarterly, and the gas is monthly.

#### c. Use which.max() to spot the outlier in the gold series. Which observation was it?

which.max determines the index of the maximum of the numeric vector. This occurs at day 770 in the gold series. We can check what date this is using lubridate package.

library(lubridate)  
  
d <- ymd("1985/01/01") + which.max(gold)  
  
paste0('day:', which.max(gold) , ' date:', d, ' price: $', gold[which.max(gold)])

## [1] "day:770 date:1987-02-10 price: $593.7"

The max price of gold occurred on February 10, 1987.

### HA 2.3

#### a. Download some monthly Australian retail data from the book website (<https://otexts.com/fpp2/extrafiles/retail.xlsx>). These represent retail sales in various categories for different Australian states, and are stored in a MS-Excel file.

Head of the downdloaded dataset is below.

library(readxl)  
retaildata <- readxl::read\_excel("retail.xlsx", skip=1)  
head(retaildata)

## # A tibble: 6 × 190  
## `Series ID` A3349335T A3349627V A3349338X A3349398A A3349468W  
## <dttm> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1982-04-01 00:00:00 303. 41.7 63.9 409. 65.8  
## 2 1982-05-01 00:00:00 298. 43.1 64 405. 65.8  
## 3 1982-06-01 00:00:00 298 40.3 62.7 401 62.3  
## 4 1982-07-01 00:00:00 308. 40.9 65.6 414. 68.2  
## 5 1982-08-01 00:00:00 299. 42.1 62.6 404. 66   
## 6 1982-09-01 00:00:00 305. 42 64.4 412. 62.3  
## # ℹ 184 more variables: A3349336V <dbl>, A3349337W <dbl>, A3349397X <dbl>,  
## # A3349399C <dbl>, A3349874C <dbl>, A3349871W <dbl>, A3349790V <dbl>,  
## # A3349556W <dbl>, A3349791W <dbl>, A3349401C <dbl>, A3349873A <dbl>,  
## # A3349872X <dbl>, A3349709X <dbl>, A3349792X <dbl>, A3349789K <dbl>,  
## # A3349555V <dbl>, A3349565X <dbl>, A3349414R <dbl>, A3349799R <dbl>,  
## # A3349642T <dbl>, A3349413L <dbl>, A3349564W <dbl>, A3349416V <dbl>,  
## # A3349643V <dbl>, A3349483V <dbl>, A3349722T <dbl>, A3349727C <dbl>, …

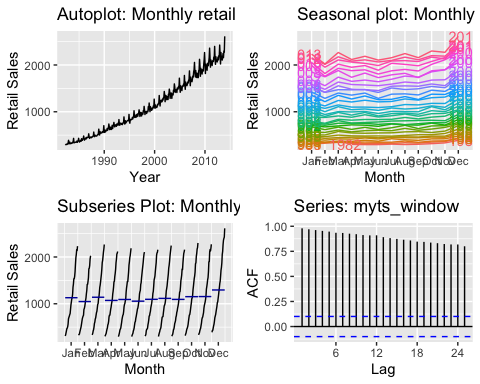
#### b. Select one of the time series as follows (but replace the column name with your own chosen column):

myts <- ts(retaildata[,"A3349335T"],  
 frequency=12, start=c(1982,4))

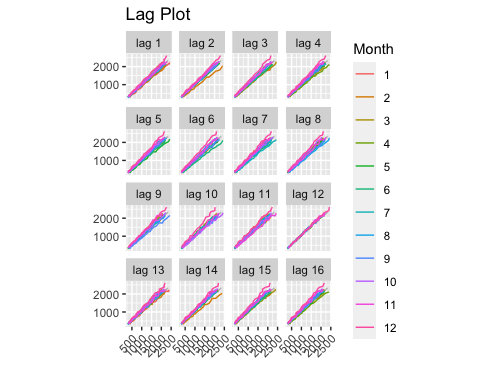
#### c. Explore your chosen retail time series using the following functions: autoplot(), ggseasonplot(), ggsubseriesplot(), gglagplot(), ggAcf()

Below we plotted autoplot(), ggseasonplot(), ggsubseriesplot(), & ggAcf() in a grid. The lag plot requires a lot of space so it is plotted seperately!

ap <- autoplot(myts) +  
 ggtitle("Autoplot: Monthly retail sales in various categories for different Australian states") +  
 xlab("Year") +  
 ylab("Retail Sales")  
  
sp <- ggseasonplot(myts, year.labels=TRUE, year.labels.left=TRUE) +  
 ylab("Retail Sales") +  
 ggtitle("Seasonal plot: Monthly retail sales")  
  
gsp <- ggsubseriesplot(myts) +  
 ylab("Retail Sales") +  
 ggtitle("Subseries Plot: Monthly retail sales")  
  
myts\_window <- window(myts, start = c(1982, 4))  
win <- ggAcf(myts\_window)  
  
  
grid.arrange(ap, sp, gsp, win, ncol=2)



lagged <- gglagplot(myts\_window) +   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1)) +   
 ggtitle("Lag Plot")  
print(lagged)



#### Can you spot any seasonality, cyclicity and trend? What do you learn about the series?

* From the autoplot, we can see that there is a clear, increasing trend, as well as seasonality.
* From the seasonal trend, it is easier to see that the seasonality has an annual frequency which is additionally following a steady upward trend over time.
* The subseries plot again shows the trend clearly – within each month there is a steady upward trend that appears to have roughly the same slope every month indicating that the trend is consistent across years.
* The window plot shows a clear trend (rather than white noise) which indicates that there is strong autocorrelation in the series.
* Finally, the lagged scatterplots show that the relationships are strongly positive across all lags, but dramatically positive at lag 12. This indicates a high annual correlation. We can see this in the ACF plot as well. We also notice that the correlation, while remaining positive, becomes less strongly positive as time goes by, and then peaks again every 12 months, again pointing to the 12 monthly frequency.

### HA 6.2

The plastics data set consists of the monthly sales (in thousands) of product A for a plastics manufacturer for five years.

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

# ?plastics  
# "Monthly sales of product A for a plastics manufacturer."  
  
autoplot(plastics) +  
 ggtitle("Monthly sales of product A for a plastics manufacturer") +  
 xlab("Year") +  
 ylab("Sales")

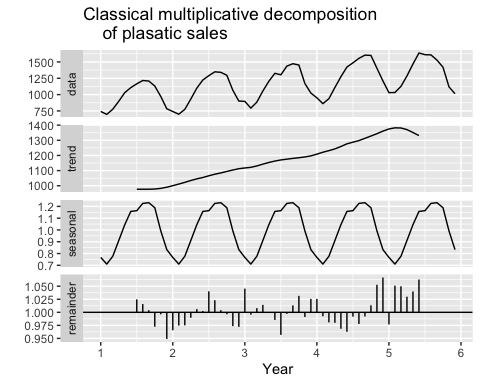


Based on the autoplot there does appear to be seasonality to the data, the plastic sales are highest May - October (peaking in August usually) and lower November - April (lowest in February). There is a consistent upward trend-cycle as well.

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

We use the decompose function with type="mutliplicative" and then autoplot to decompose the series and then evalaute the results.

plastics %>% decompose(type="multiplicative") %>%  
 autoplot() + xlab("Year") +  
 ggtitle("Classical multiplicative decomposition  
 of plasatic sales")



The trend cycle shows a strong seasonal component, with a yearly frequency, and has an increasing trend up until just past year 5 when it begins decreasing. There is some remainder as well. The remainder values below 1 indicate that there is some “leakage” of the trend-cycle component into the remainder component - the trend-cycle estimate has over-smoothed the drop in data.

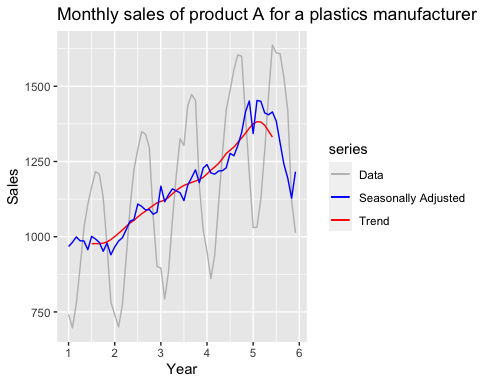
#### c. Do the results support the graphical interpretation from part a?

Yes, the graphical interpretation from part A aligns with the multiplicative decomposition. The yearly seasonal pattern was noted in both a and b. The only part that was not captured well is the drop off in the trend after year 5.

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

We can compute the seasonally adjusted fit by applying the seasadj function to the decomposed time series.

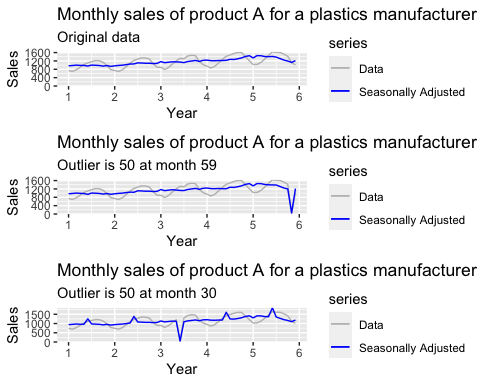
fit <- plastics %>%  
 decompose(type="multiplicative")   
  
autoplot(plastics, series="Data") +  
 autolayer(trendcycle(fit), series="Trend") +  
 autolayer(seasadj(fit), series="Seasonally Adjusted") +  
 xlab("Year") + ylab("Sales") +  
 ggtitle("Monthly sales of product A for a plastics manufacturer") +  
 scale\_colour\_manual(values=c("gray","blue","red"),  
 breaks=c("Data","Seasonally Adjusted","Trend"))



#### 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?

Our approach is to create a funciton that allows us to quickly check different outliers and positions, then compare several outputs to the original seasonally adjusted plot.

#Function that creates plots based on the position (n) value (v) and whether it's an outlier or not (outlier)  
outlier <- function(n, v, outlier){  
   
 # if it's an ourlier, add the outlier and change the plot subtitle  
 if(outlier==TRUE){  
 plastics[n] <- v  
 subtitle <- paste("Outlier is", v, "at month", n)  
 } else {  
 subtitle <- "Original data"  
 }  
   
 #fit to the decomp  
 fit <- plastics %>% decompose(type="multiplicative")   
   
 #plot  
 autoplot(plastics, series="Data") +  
 autolayer(seasadj(fit), series="Seasonally Adjusted") +  
 xlab("Year") + ylab("Sales") +  
 labs(  
 title = "Monthly sales of product A for a plastics manufacturer",  
 subtitle = subtitle  
 ) +  
 scale\_colour\_manual(values=c("gray","blue"),  
 breaks=c("Data","Seasonally Adjusted","Trend")) +   
 scale\_y\_continuous(expand = c(0, 0), limits = c(0, NA))  
  
   
}  
  
# plot the original and two variations in a grid  
grid.arrange(  
 outlier(1, 1, FALSE),  
 outlier(59, 50, TRUE),  
 outlier(30, 50, TRUE),   
 ncol=1)



The outlier in the middle causes the seasonally adjusted fit to hallucinate seasons that don’t exist! At or near the end, there is no discernible effect on the overall seasonally adjusted fit.

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

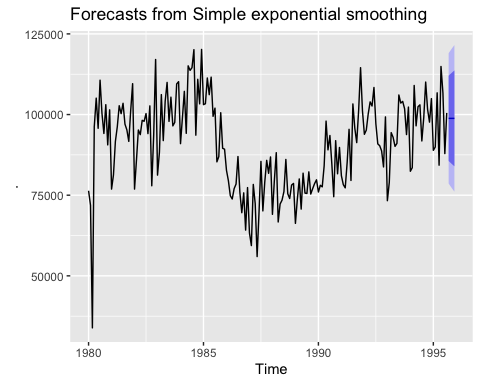
If the outlier is at the end vs in the middle, there is more of an effect on the seasonally adjusted data. The estimate of the trend cycle is unavailable for the first and last few observations - without this there is also no estimate of the remainder component. Due to this, all of the outlier is passed on to the seasonally adjusted rather than having some of it put in the remainder or in the trend. If the outlier is in the middle some of the variance is put into the cycle component and causes some “hallucinated” peaks

## Week 2 Homework Solutions

### HA 7.1 Consider the pigs series – the number of pigs slaughtered in Victoria each month.

### 7.1 a Use the ses function to find the optimal levels of alpha and l-zero, and generate forecasts for the next 4 months.

fit\_pigs <- pigs %>% ses(h=4)  
  
fit\_pigs %>% autoplot()



#summary(fit\_pigs)

By callng summary on the results of the ses object, we can see that the optimal values of alpha and are 0.2971 and 77260.0561, respectively.

The first four predicted values are given by calling the mean operator on the summary object, as shown below.

summary(fit\_pigs)$mean

## Sep Oct Nov Dec  
## 1995 98816.41 98816.41 98816.41 98816.41

### 7.1 b Compute a 95% prediction interval for the first forecast using y-hat is plus or minus 1.96s where s is the standard deviation of the residuals. Compare your interval with the interval produced by R.

# margin of error  
me <- 1.96\*sd(fit\_pigs$residuals)  
  
# first forecast  
f1 <- fit\_pigs$mean[1]

The first forecast from the ses prediction on pigs is 9.881641^{4} with a 95% margin of error plus or minus 2.013644^{4}. A comparison to results from R is shown below. The results are within 1/10 of 1% from the manually calculated values.

library(dplyr)  
  
#Make a table to review results  
labels\_7.1b <- c("My Results", "R")  
  
df\_pigs <- data.frame(  
 c(f1-me, f1+me),   
 c(fit\_pigs$lower[5], fit\_pigs$upper[5]))  
  
colnames(df\_pigs) <- labels\_7.1b  
row.names(df\_pigs) <- c("Lower 95% Bound", "Upper 95% Bound")  
df\_pigs <- df\_pigs %>% mutate(  
 `% Diff` = round((`My Results` - R)/R\*100,2)  
)  
  
df\_pigs %>% t()

## Lower 95% Bound Upper 95% Bound  
## My Results 78679.97 118952.84  
## R 78611.97 119020.84  
## % Diff 0.09 -0.06

### HA 7.3

#### Modify your function from the previous exercise to return the sum of the squared errors rather than the forecast of the next observations. The use the optim function to find the optimal methods of alpha and . Do you get the same values as the ses() function?

Note: In the previous exercise, we were asked to write our own function to implement simple exponential smoothing. This is shown below.

T <- length(pigs) %>% as.numeric()  
  
my\_smooth <- function(y, alpha, level) {  
   
 T <- length(y) %>% as.numeric()  
   
 my\_fitted <- c()  
   
 #set first value   
 my\_fitted[1] <- level  
  
 for (i in 2:(T+1)) {  
 my\_fitted[i] <- alpha \* y[i-1] + (1 - alpha) \* my\_fitted[i-1]  
 }  
   
 #return last value  
 return(my\_fitted)  
 }  
  
alpha <- 0.291  
level <- pigs[1]  
  
my\_smooth(pigs, alpha=alpha, level=level)[T+1]

## [1] 98822.32

my\_smooth(pigs, alpha=alpha, level=level/2)[T+1]

## [1] 98822.32

my\_smooth(pigs, alpha=1/T, level=level)[T+1]

## [1] 85451.49

my\_smooth(pigs, alpha=1/T, level=level/2)[T+1]

## [1] 71439.99

ses(pigs, alpha=alpha)$mean[1]

## [1] 98822.32

The optim() function using the default method (Nelder and Mead) produced different values for α and (0.5179980 and 0.1307012, respectively). Changing the method to BFGS (Broyden, Fletcher, Goldfarb and Shanno) made the results very close to the one produced by the ses() function (0.297142 and 77273.160331, respectively).

# Function to implement SES, modified to return the sum of squared errors instead of the forecast of the next observation  
my\_ses\_mod <- function(y, params) {  
  
 # Split out parameters for optim() function   
 alpha <- params[1]  
 level <- params[2]  
   
 # Initialize vector to store forecast values  
 fitted <- c()  
 fitted[1] <- level  
   
 # Set T to be length of time series  
 T <- length(y)  
   
 # Iterate over terms  
 for (j in seq(2, T)) {  
 fitted[j] <- alpha \* (y[j - 1]) + (1 - alpha) \* fitted[j - 1]  
 }  
   
 # Convert fitted values to time series  
 fitted <- ts(fitted, frequency=12, start=c(1980, 1))  
   
 # Calculate residuals  
 residuals <- y - fitted  
   
 # Calculate sum of squared errors  
 sse=sum(residuals \*\* 2)  
  
 # Return fitted values, residuals, and sum of squared errors  
 # Return sse  
 return(sse)  
  
}  
  
# Calculate SSE using modified custom function  
fit2 <- my\_ses\_mod(pigs, c(0.2971, 77260.0561))  
  
# Compare this SSE to SSE from original fit using sse()  
  
ans7.1b <- data\_frame(  
 MyFit = c(fit2),  
 sesFit = c(sum(fit\_pigs$residuals \*\* 2))  
)

## Warning: `data\_frame()` was deprecated in tibble 1.1.0.  
## ℹ Please use `tibble()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.

ans7.1b <- ans7.1b %>%   
 mutate(  
 `% Diff` = round((MyFit - sesFit)/sesFit\*100 , 2)  
)  
  
print(ans7.1b)

## # A tibble: 1 × 3  
## MyFit sesFit `% Diff`  
## <dbl> <dbl> <dbl>  
## 1 19765613579. 19765613407. 0

# Optimize α and ℓ\_0 using optim() function  
fit3 <- optim(par=c(0, 0), fn=my\_ses\_mod, y=pigs)  
fit3$par # produces alpha=0.5179980, l\_0=0.1307012

## [1] 0.5179980 0.1307012

# Try optimize again using method='BFGS'  
fit4 <- optim(par=c(0, 0), fn=my\_ses\_mod, y=pigs, method='BFGS')  
fit4$par

## [1] 0.297142 77273.160331

## KJ 3.1

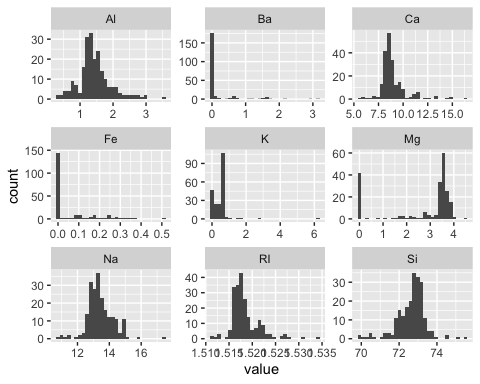
The UC Irvine Machine Learning Repository6 contains a data set related to glass identification. The data consist of 214 glass samples labeled as one of seven class categories. There are nine predictors, including the refractive index and percentages of eight elements: Na, Mg, Al, Si, K, Ca, Ba, and Fe. The data can be accessed via: > library(mlbench) > data(Glass) > str(Glass)

library(mlbench)  
data(Glass)

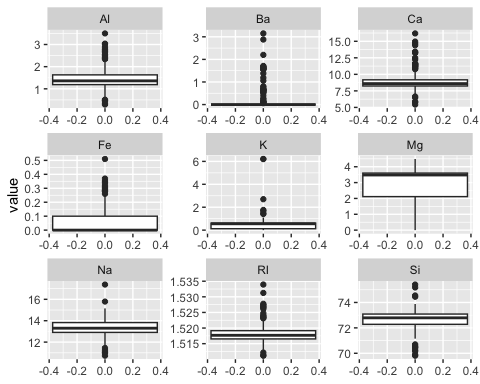
### 3.1a

Using visualizations, explore the predictor variables to understand their distributions as well as the relationships between predictors.

# Gather  
Glass\_long <- Glass %>% select(-Type) %>% gather()   
  
# Hist plots  
Glass\_long %>% ggplot(aes(value)) + geom\_histogram() + facet\_wrap(~key, scales = "free")



# Box plots  
Glass\_long %>% ggplot(aes(y=value))+geom\_boxplot()+facet\_wrap(~key, scales = "free")



A colorized correlation plot is used to understand relationships between predictors.

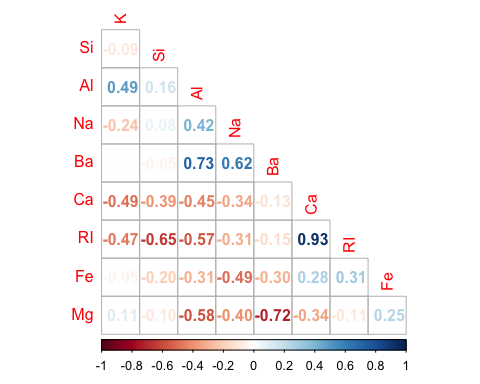
library(corrplot)  
library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

# Filter for just quantitative vars  
glassQuant <- Glass %>% select (-Type)  
  
glasscorr <- cor(glassQuant)  
# Corr chart  
corrplot(cor(glasscorr), method="number", order = "AOE", type = "lower", diag = FALSE)



# Find correlations past cutoff  
print(paste0("Candidate for removal due to high correlation: ", findCorrelation(glasscorr, cutoff=0.8, exact=T, verbose=T, names=T)))

## Compare row 7 and column 1 with corr 0.81   
## Means: 0.319 vs 0.217 so flagging column 7   
## All correlations <= 0.8   
## [1] "Candidate for removal due to high correlation: Ca"

There is some collinearity between RI and Ca. There is some correlation between a few of the other variables: RI and Si, Mg and AI, and Ba and AI.

The RI-Ca correlation is the only one to take particular note of, as it could impact the types of modeling sensitive to those effect (e.g. linear models). Using the heuristic algorithm proposed by K&J (and using the findCorrelation() function), calcium is a candidate for removal due to high correlation.

All variables except Type are quantitative; Type is a categorical variable with 6 levels (1, 2, 3, 5, 6, and 7).

### 3.1b Do there appear to be any outliers? Are any predictors skewed?

As we observed from the histograms, all of the predictors are skewed to some degree, but the worst offenders are Mg, which appears to be bimodal, and K, Ba, and Fe, which appear to be right-skewed. We can confirm this by using the skewness function:

library(psych)  
describe(Glass) %>% select(skew)

## skew  
## RI 1.60  
## Na 0.45  
## Mg -1.14  
## Al 0.89  
## Si -0.72  
## K 6.46  
## Ca 2.02  
## Ba 3.37  
## Fe 1.73  
## Type\* 1.04

As far as outliers, let’s take a look the boxplots above. There are a lot of outliers present (greater than 1.5X outside the interquartile range) in each predictor with the exception of Mg.

## KJ 3.2

### KJ 3.2 (a)

The soybean data can also be found at the UC Irvine Machine Learning Repository. Data were collected to predict disease in 683 soybeans. The 35 predictors are mostly categorical and include information on the environmental conditions (e.g., temperature, precipitation) and plant conditions (e.g., left spots, mold growth). The outcome labels consist of 19 distinct classes.

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

Using the nearZeroVar() function, these variables appear to be degenerate as described in K&J as having near-zero variance:

* leaf.mild
* mycelium
* sclerotia

As shown in the summary output, each of these columns has a dominant class that far exceeds its next most populous class.

# Load data  
data(Soybean)  
#str(Soybean)  
  
# Generate frequency tables to look for degenerate distributions  
multifunc2 <- function(x) {  
 return(prop.table(table(x)))  
}  
  
# Degenerate vars: if both of these are true:  
# 1. fraction of unique vals over the sample size < 10%  
# 2. ratio of freq of the most prevalent to the freq of the second most prevalent is > 20  
nzv <- nearZeroVar(Soybean)  
#print(colnames(Soybean[,nzv]))  
  
# Examine NZV columns  
summary(Soybean[,nzv])

## leaf.mild mycelium sclerotia   
## 0 :535 0 :639 0 :625   
## 1 : 20 1 : 6 1 : 20   
## 2 : 20 NA's: 38 NA's: 38   
## NA's:108

### KJ 3.2 (b)

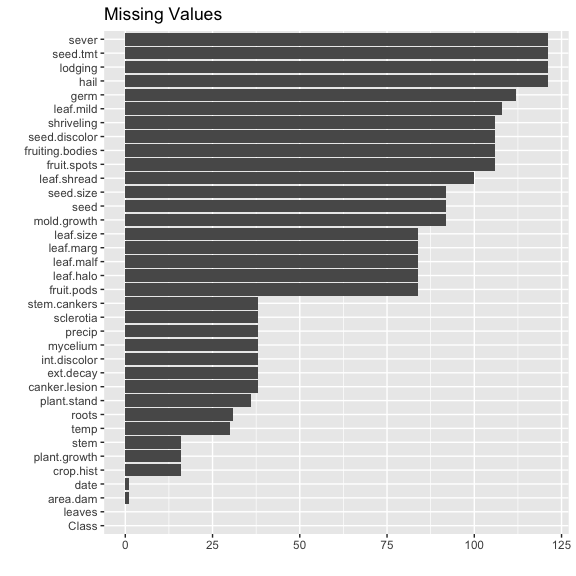
**Roughly 18% of the data are missing. Are there particular predictors that are more likely to be missing? Is the pattern of missing data related to the classes?**

As shown in the md.pattern plot below, there are 562 (of 683) complete cases, or roughly 82% of the data. Of the 18% of the data that have over 15% missing values, most of the variables are related to plant pathology:

* hail
* sever
* seed.tmt
* germ
* leaf.mild
* lodging
* fruiting.bodies
* fruit.spots
* seed.discolor

We verified that the missing values weren’t due to the absence of the specific pathology, as negative cases were also reported. So the values do appear to be missing as opposed to being intentionally or structurally absent.

col\_sums = colSums(is.na(Soybean))  
  
data.frame(variable = names(col\_sums), missing = col\_sums, row.names = NULL) |>   
 ggplot(aes(y=reorder(variable,missing), x=missing)) +  
 geom\_col() +  
 ggtitle('Missing Values') + xlab('') + ylab('')



We see a bunch of predictors with more than 80 NAs, related to seed (seed.tmt,seed.size,seed.discolor,seed), leaf information (leaf.size,leaf.shread,leaf.mild,leaf.marg,leaf.malf,leaf.halo), fruit (fruiting.bodies,fruit.spots,fruit.pods), and a few others that are not of a clear category (shriveling,sever,mold.growth,lodging,hail,germ).

Checking to see how these NAs relate by class:

Soybean |>  
 group\_by(class = Class) |>  
 summarize(total\_na = sum(is.na(across(-Class))),  
 pct\_na=sum(is.na(across(-Class)))  
 / n()) |>  
 arrange(desc(pct\_na))

## # A tibble: 19 × 3  
## class total\_na pct\_na  
## <fct> <int> <dbl>  
## 1 2-4-d-injury 450 28.1  
## 2 cyst-nematode 336 24   
## 3 herbicide-injury 160 20   
## 4 phytophthora-rot 1214 13.8  
## 5 diaporthe-pod-&-stem-blight 177 11.8  
## 6 alternarialeaf-spot 0 0   
## 7 anthracnose 0 0   
## 8 bacterial-blight 0 0   
## 9 bacterial-pustule 0 0   
## 10 brown-spot 0 0   
## 11 brown-stem-rot 0 0   
## 12 charcoal-rot 0 0   
## 13 diaporthe-stem-canker 0 0   
## 14 downy-mildew 0 0   
## 15 frog-eye-leaf-spot 0 0   
## 16 phyllosticta-leaf-spot 0 0   
## 17 powdery-mildew 0 0   
## 18 purple-seed-stain 0 0   
## 19 rhizoctonia-root-rot 0 0

We see that five of the 19 classes are responsible for all of the NAs. This indicates to me a likelihood that certain measurements are either applicable or not available to these classes. This should be taken into consideration when considering how to deal with these missing values.

### KJ 3.2 (c)

**Develop a strategy for handling missing data, either by eliminating predictors or imputation.**

The degenerate values should be removed as they offer no predictive value.

Because NA values only appear in 5 of the 14 classes, missingness is potentially informative in predicting a class. Two ways to deal with this while retaining the predictive value of the missingness is to **1)** create an **indicator variable** that represents the presence or absence of missing values in a specific variable, or **2)** create a **missingness pattern variable** that indicates whether a specific combination of variables is missing or not.

It’s also possible to apply pattern-based imputation using the mice or amelia packages in R.

# Create copy of the original data set  
sb2 <- Soybean  
  
# Remove the observations in which 19 or more variables have missing values  
most\_complete <- rowSums(is.na(sb2)) < 19  
sb2 <- sb2[most\_complete,]  
  
# Recalculate number of rows contain missing values  
some\_missing <- rowSums(is.na(sb2)) > 0  
print(paste0("Proportion of observations with missing values after removing those with high proportion of missing variables: ",   
 round(nrow(sb2[some\_missing,]) / nrow(sb2), 3)))

## [1] "Proportion of observations with missing values after removing those with high proportion of missing variables: 0.047"

print(paste0("Proportion of observations retained from original data set: ",  
 round(nrow(sb2) / nrow(Soybean), 3)))

## [1] "Proportion of observations retained from original data set: 0.864"

An alternative approach would be to remove the three variables with degenerate classes:

* leaf.mild
* mycelium
* sclerotia

and to additionally remove those with a high proportion of data:

* hail
* sever
* seed.tmt
* germ
* leaf.mild (also degenerate)
* lodging
* fruiting.bodies
* fruit.spots
* seed.discolor

However, that still leaves 17.7% of observations with at least one missing value. So a combination of removing some observations along with the above variables would be more effective. The disadvantage of this is that 11 variables would be removed–a full 30% of all variables.

# Create copy of original data set  
sb3 <- Soybean  
  
# Remove the observations in which 24 or more variables have missing values  
most\_complete <- rowSums(is.na(sb3)) < 24  
sb3 <- sb3[most\_complete,]  
  
# Remove variables with degenerate classes or having high proportion of missing values (11 all together)  
sb3 <- sb3 %>%  
 select(-leaf.mild, -mycelium, -sclerotia, -hail, -sever, -seed.tmt, -germ, -lodging, -fruiting.bodies, -fruit.spots, -seed.discolor)  
  
# Recalculate number of rows contain missing values  
#some\_missing <- rowSums(is.na(sb3)) > 0  
#print(paste0("Proportion of observations with missing values after removing degenerate variables: ",   
# round(nrow(sb3[some\_missing,]) / nrow(sb3), 3)))  
#print(paste0("Proportion of observations retained from original data set: ",  
# round(nrow(sb3) / nrow(Soybean), 3)))

Another approach would be to impute the missing data. The MICE package (multivariate imputation by chained equations) is an appropriate way to do this. Even using MICE, the observations with over 24 variables missing would seem to remain problematic, so we’ll remove those first before imputing. As shown below, we now have a dataset with no missing cases, and over 95% of the original observations were retained.

# Create copy of original data set  
sb4 <- Soybean  
  
# Remove the observations in which 24 or more variables have missing values  
most\_complete <- rowSums(is.na(sb4)) < 24  
sb4 <- sb4[most\_complete,]  
  
# Recalculate number of rows contain missing values  
some\_missing <- rowSums(is.na(sb4)) > 0  
print(paste0("Proportion of observations with missing values after removing degenerate variables: ",   
 round(nrow(sb4[some\_missing,]) / nrow(sb4), 3)))  
print(paste0("Proportion of observations retained from original data set: ",  
 round(nrow(sb4) / nrow(Soybean), 3)))  
  
# Impute missing data  
imp <- mice(sb4, maxit=5, m=5, seed=777)  
  
# Complete the data set using imputed values  
sb4 <- complete(imp)  
  
# Recalculate number of rows contain missing values  
some\_missing <- rowSums(is.na(sb4)) > 0  
print(paste0("Proportion of observations with missing values after removing degenerate variables: ",   
 round(nrow(sb4[some\_missing,]) / nrow(sb4), 3)))  
print(paste0("Proportion of observations retained from original data set: ",  
 round(nrow(sb4) / nrow(Soybean), 3)))

## Week 3 Homeowork Solutions

### HA 8.1

#### HA 8.1 (a)

Figure 8.31 shows the ACFs for 36 random numbers, 360 random numbers and 1,000 random numbers.

**Explain the differences among these figures. Do they all indicate that the data are white noise?**

A picture containing text, screenshot, diagram, line

Description automatically generated

*Figure 8.31: Left: ACF for a white noise series of 36 numbers. Middle: ACF for a white noise series of 360 numbers. Right: ACF for a white noise series of 1,000 numbers.*

According to the book, “autocorrelation measures the linear relationship between lagged values of a time series,” and the autocorrelation coefficients are plotted to show the autocorrelation function (ACF) in a plot called a correlogram. If there is no autocorrelation, this is called white noise and the autocorrelation should be close to 0. To qualify as white noise, 95% of the spikes in the ACF need to be within where T is the length of the time series. The bounds are plotted as the blue dashed lines on the correlogram.

The difference among the figures is the length of the time series - that is why the blue dashed lines are at different values. For the longer time series, the bounds are the smallest, and for the shortest time series the bounds are the largest. Because more than 95% of the spikes lie within the bounds for each of the figures, they all indicate white noise. Series X2 does appear to have potentially two spikes just outside of the bounds, but we still qualified it as white noise because some of the other spikes are almost too small to see, and therefore we estimated that 95% of the spikes still lie within the bounds.

#### HA 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?**

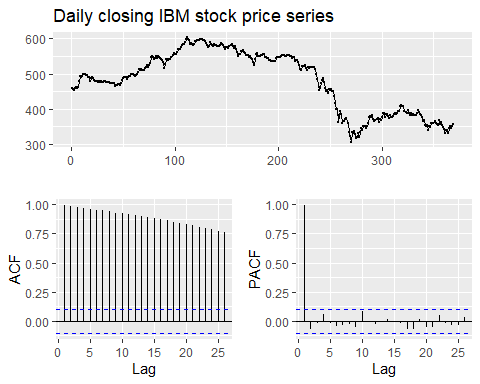
The critical values are those that occur above or below , where T is the number of samples in the series. Since the denominator (square root of sample size) increases for each series while the numerator is held constant, the critical value will grow narrower as T gets larger. Moreover, the autocorrelations are different in each figure because each figure represents random numbers, therefore the autocorrelations are different from one another.

### HA 8.2

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

As shown below, the ACF plot exhibits a downward decay consistent with autocorrelated values for each successive lag. If the series were stationary, the ACF plot would exhibit a drop to zero very soon after lag 1. The PACF only exhibits a single spike above significance at lag 1, indicating that an ARIMA (1,1,0) model would be appropriate, but the PACF by itself isn’t useful at determining whether differencing is needed.

# Load data  
data(ibmclose)  
  
# Plot  
ggtsdisplay(ibmclose, main='Daily closing IBM stock price series')



### HA 8.6

#### HA 8.6 (a)

Use R to simulate and plot some data from simple ARIMA models.

**Use the following R code to generate data from an AR(1) model with = 0.6 and = 1. The process starts with = 0.**

y <- ts(numeric(100)) e <- rnorm(100) for(i in 2:100) y[i] <- 0.6\*y[i-1] + e[i]

# Make the R code provided in the problem into a function  
genTS\_AR1 <- function(phi1) {  
  
 y <- ts(numeric(100))  
 e <- rnorm(100)  
 for(i in 2:100)  
 y[i] <- (phi1 \* y[i - 1]) + e[i]  
 return(y)  
}  
  
# Generate initial time series  
set.seed(777)  
y <- genTS\_AR1(0.6)

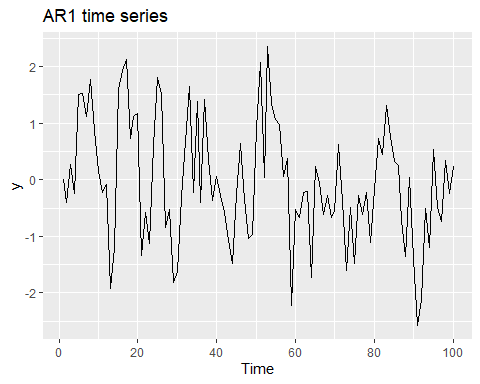
An AR(p) model is an autoregressive model with an order of p; in our case the order is 1. The order of 1 indicates how many lagged values of y are included, so in this case it is 1 lagged y value. The code above successfully generated the data from an AR(1) model with the given parameters.

#### HA 8.6 (b)

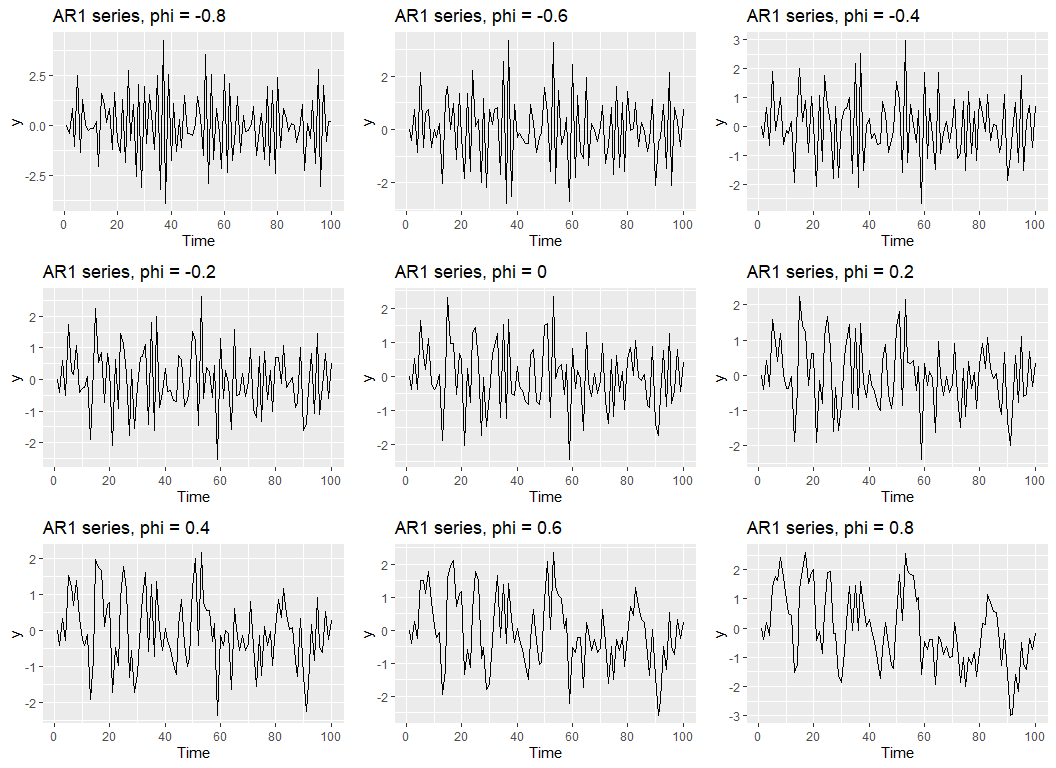
**Produce a time plot for the series. How does the plot change as you change ?**

More negative values of make each successive value in the series more likely to have the opposite sign than the previous value, resulting in a plot that appears to fluctuate rapidly, with denser peaks and valleys. As approaches zero, any deviation between successive values is entirely due to random error, and therefore appears as white noise. More positive values of make each successive value more likely to have the same sign as the previous value, resulting in a series that appears to have wider-spaced peaks and valleys.

# Generate initial time plot for series y  
y %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle('AR1 time series')



# Generate additional plots, varying phi1  
# (for an AR(1) model, -1 < phi < 1)  
plt <- list()  
phis <- seq(-0.8, 0.8, 0.2)  
for (i in seq(1, length(phis))) {  
 set.seed(777)  
 ynew <- genTS\_AR1(phis[i])  
 plt[[i]] <- ynew %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle(paste0('AR1 series, phi = ', phis[i]))  
}  
do.call('grid.arrange', c(plt, nrow=3))



#### HA 8.6 (c)

**Write your own code to generate data from an MA(1) model with = 0.6 and = 1.**

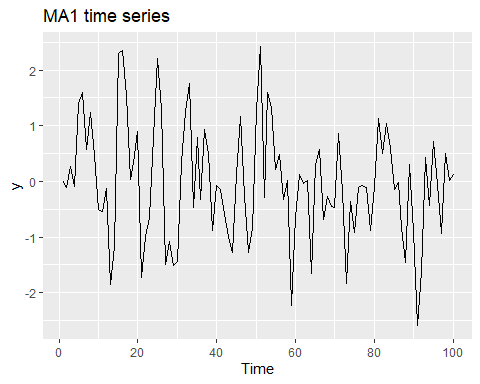
# Function to generate MA(1) model given theta1  
genTS\_MA1 <- function(theta1) {  
  
 y <- ts(numeric(100))  
 e <- rnorm(100)  
 for(i in 2:100)  
 y[i] <- (theta1 \* e[i - 1]) + e[i]  
 return(y)  
}  
  
# Generate initial time series  
set.seed(777)  
yma1 <- genTS\_MA1(0.6)

#### HA 8.6 (d)

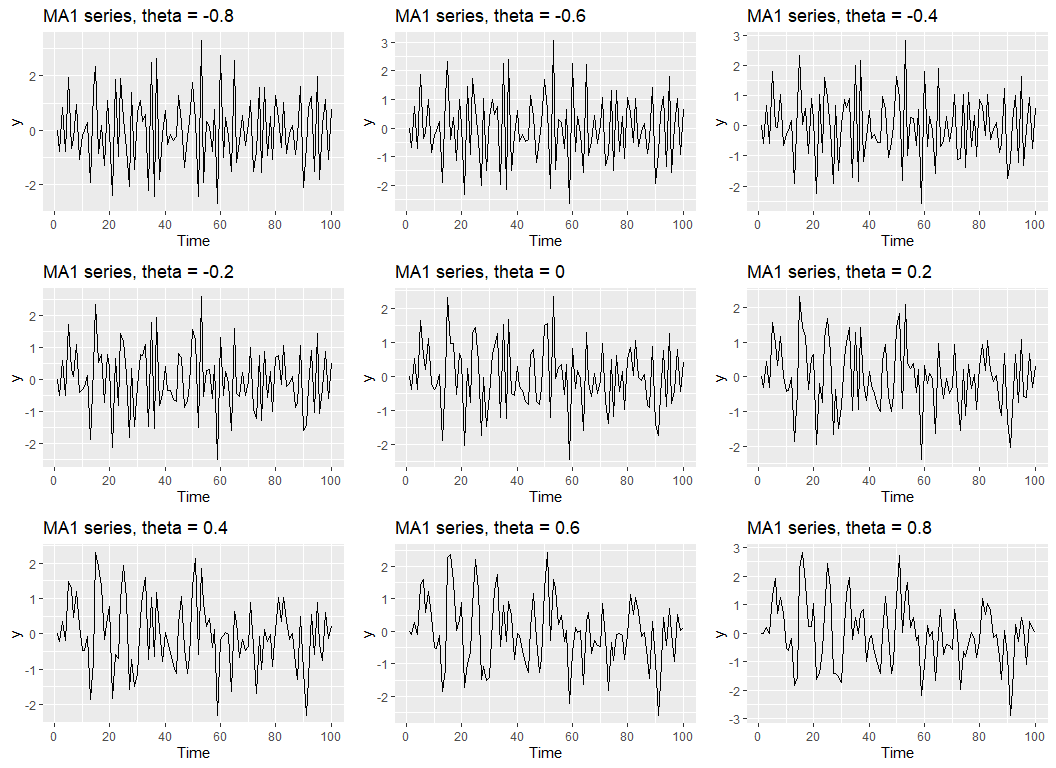
**Produce a time plot for the series. How does the plot change as you change ?**

The peaks and valleys seem to widen as values of grow from negative to positive. This is because of the positive sign in front of the regression term . Since since is a random number between -1 and 1 with a mean of 0 and standard deviation of 1, then multiplying one such number by a negative value and adding it to another random number with mean 0 and sd 1 will be more likely to result in a number with the opposite sign as the previous, resulting in the tighter peaks and valleys seen when is negative. The opposite is true when is positive: i.e., it is more likely that a number with the same sign is generated, meaning that that the current trend will continue. This results in a plot that appears to have peaks and valleys that are more spread out.

# Generate time plot for MA1 series  
yma1 %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle('MA1 time series')



# Generate additional plots, varying theta1  
# (for an MA(1) model, -1 < theta < 1)  
pltma1 <- list()  
thetas <- seq(-0.8, 0.8, 0.2)  
for (i in seq(1, length(thetas))) {  
 set.seed(777)  
 ynew <- genTS\_MA1(thetas[i])  
 pltma1[[i]] <- ynew %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle(paste0('MA1 series, theta = ', thetas[i]))  
}  
do.call('grid.arrange', c(pltma1, nrow=3))



#### HA 8.6 (e)

**Generate data from an ARMA(1,1) model with = 0.6, = 0.6 and = 1.**

# Function to generate ARMA(1,1) model given phi1 and theta1  
genTS\_ARMA1\_1 <- function(phi1, theta1) {  
  
 y <- ts(numeric(100))  
 e <- rnorm(100)  
 for(i in 2:100)  
 y[i] <- (phi1 \* y[i - 1]) + (theta1 \* e[i - 1]) + e[i]  
 return(y)  
}  
  
# Generate initial time series  
set.seed(777)  
yarma1\_1 <- genTS\_ARMA1\_1(0.6, 0.6)

#### HA 8.6 (f)

**Generate data from an AR(2) model with = −0.8, = 0.3 and = 1. (Note that these parameters will give a non-stationary series.)**

# Function to generate AR(2) model given phi1 and phi2  
genTS\_AR2 <- function(phi1, phi2) {  
  
 y <- ts(numeric(100))  
 e <- rnorm(100)  
 for(i in 3:100)  
 y[i] <- (phi2 \* y[i - 2]) + (phi1 \* y[i - 1]) + e[i]  
 return(y)  
}  
  
# Generate initial time series  
set.seed(777)  
yar2 <- genTS\_AR2(-0.8, 0.3)

#### HA 8.6 (g)

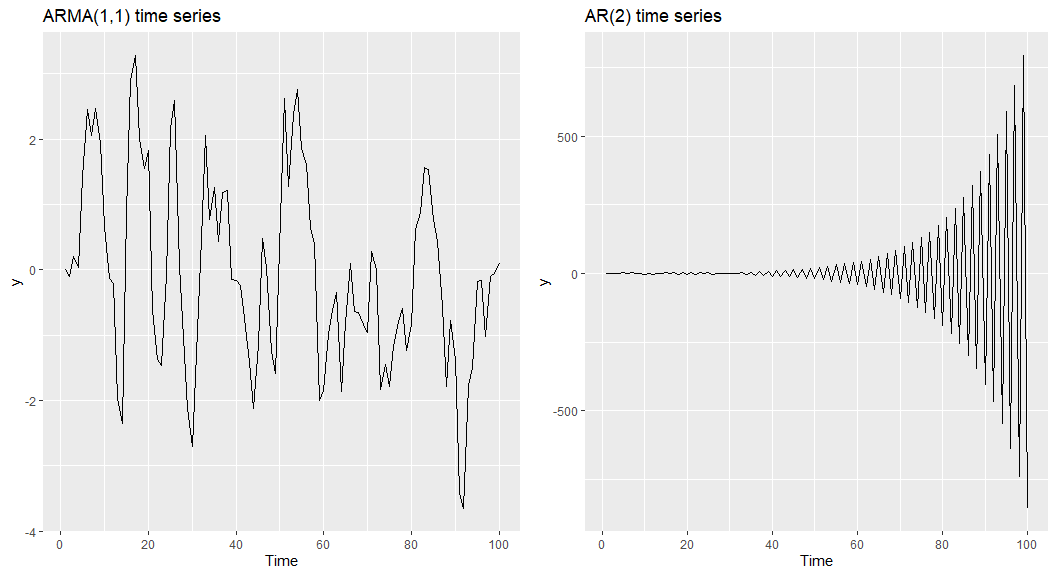
**Graph the latter two series and compare them.**

The ARMA(1,1) model is stationary. Introducing an MA(1) term into an AR(1) model shouldn’t (and doesn’t) have any effect on producing any cyclicity, seasonality, or trend. On the other hand, the AR(2) model with the given and values does introduce stark seasonality into the plot. It is noted that when > 1, similar seasonality can be produced. For example, similar plots can be produced with the following values:

|  |  |
| --- | --- |
| -0.9 | 0.2 |
| -0.8 | 0.3 |
| -0.6 | 0.5 |
| -0.4 | 0.7 |
| -0.2 | 0.9 |

This is because any random variation introduced by the term will be counteracted by adding the two terms together, which are guaranteed to have an absolute value greater than 1.

# Generate time plot for the ARMA(1,1)series  
p1 <- yarma1\_1 %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle('ARMA(1,1) time series')  
  
# Generate time plot for the AR(2) series  
p2 <- yar2 %>%  
 autoplot() +  
 ylab('y') +  
 ggtitle('AR(2) time series')  
  
# Arrange plots on grid  
grid.arrange(p1, p2, nrow=1)



### HA 8.8

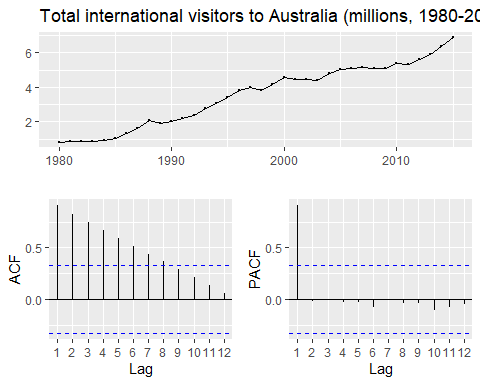
#### HA 8.8 (a)

Consider austa, the total international visitors to Australia (in millions) for the period 1980-2015.

**Use auto.arima() to find an appropriate ARIMA model. What model was selected? Check that the residuals look like white noise. Plot forecasts for the next 10 periods.**

First, we will plot the international visitors to Australia to get an idea for the data.

# Load data  
data(austa)  
  
# Plot  
austa %>%  
 ggtsdisplay(main='Total international visitors to Australia (millions, 1980-2015)')



Now, we can use auto.arima to select a model automatically.

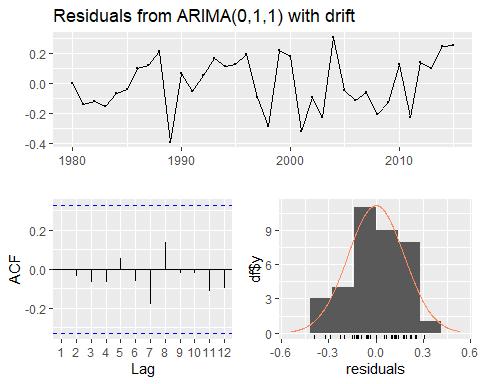
# Fit auto.arima model and show results  
fit <- auto.arima(austa)  
summary(fit)

## Series: austa   
## ARIMA(0,1,1) with drift   
##   
## Coefficients:  
## ma1 drift  
## 0.3006 0.1735  
## s.e. 0.1647 0.0390  
##   
## sigma^2 = 0.03376: log likelihood = 10.62  
## AIC=-15.24 AICc=-14.46 BIC=-10.57  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.0008313383 0.1759116 0.1520309 -1.069983 5.513269 0.7461559  
## ACF1  
## Training set -0.000571993

The model selected was ARIMA(0,1,1) with drift.

We need to check the residuals from the chosen model by plotting the ACF of the residuals and making sure they look like white noise.

# Plot residuals  
checkresiduals(fit)

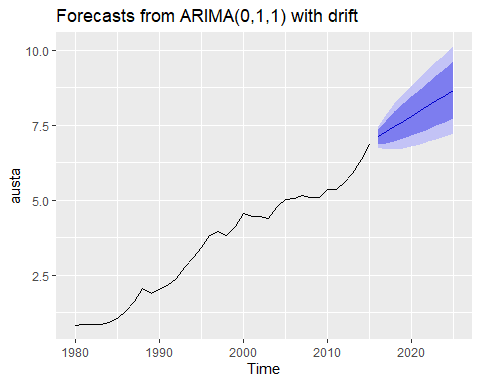


##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1) with drift  
## Q\* = 2.297, df = 6, p-value = 0.8905  
##   
## Model df: 1. Total lags used: 7

The residuals are all within the bounds and therefor classify as white noise.

The next 10 time periods can be forecast.

# Plot forecasts for the next ten periods  
fit %>%  
 forecast(h=10) %>%  
 autoplot()



In summary, the auto.arima() function fit an ARIMA(0,1,1) model with drift. Using the checkresiduals() function, residuals appear as white noise, all lagged values are within the threshold limits, and the Ljung-Box p-value of 0.8905 exceeds 0.05. We can therefore conclude that this is an appropriate model to use.

#### HA 8.8 (b)

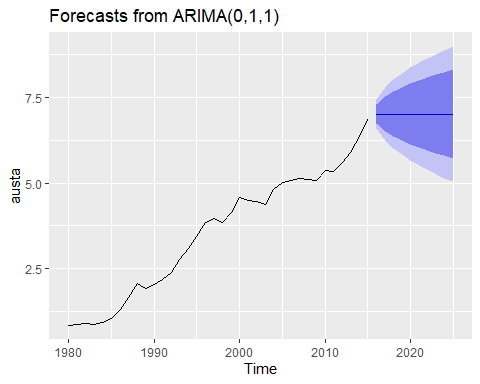
**Plot forecasts from an ARIMA(0,1,1) model with no drift and compare these to part a. Remove the MA term and plot again.**

First we can create forecasts from the ARIMA(0,1,1) model with no drift.

# Model using ARIMA(0,1,1) with no drift  
fit2 <- Arima(austa, order=c(0,1,1), include.drift=F)  
summary(fit2)

## Series: austa   
## ARIMA(0,1,1)   
##   
## Coefficients:  
## ma1  
## 0.4936  
## s.e. 0.1265  
##   
## sigma^2 = 0.04833: log likelihood = 3.73  
## AIC=-3.45 AICc=-3.08 BIC=-0.34  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.1150344 0.2136411 0.1714887 3.817629 5.649995 0.8416532  
## ACF1  
## Training set -0.1892256

# Plot  
fit2 %>%  
 forecast(h=10) %>%  
 autoplot()



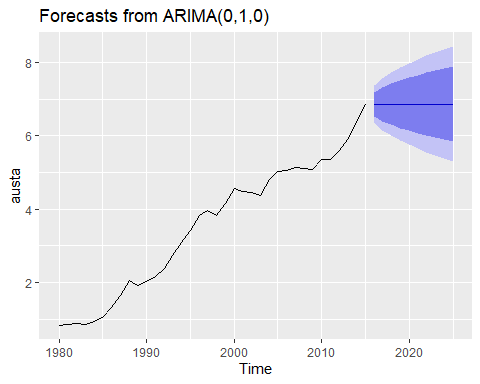
Compared to part A rather than having an increasing trend there is now a flat line projection.

Next we can remove the MA term and replot.

# Remove the MA term, i.e., model using ARIMA(0,1,0) with no drift  
fit3 <- Arima(austa, order=c(0,1,0), include.drift=F)  
summary(fit3)

## Series: austa   
## ARIMA(0,1,0)   
##   
## sigma^2 = 0.06423: log likelihood = -1.62  
## AIC=5.24 AICc=5.36 BIC=6.8  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.1674969 0.2498996 0.1981155 5.471861 6.479997 0.9723354  
## ACF1  
## Training set 0.2583422

# Plot  
fit3 %>%  
 forecast(h=10) %>%  
 autoplot()



As shown above, allowing drift in the model causes trend to be included in the forecast values, while point forecasts for the no-drift model are held constant. Removing the MA term generated similar point forecasts, but had a lower AICc than the ARIMA(0,1,1) model, indicating that the MA term may not be needed and that a simpler, random-walk model may be better.

#### HA 8.8 (c)

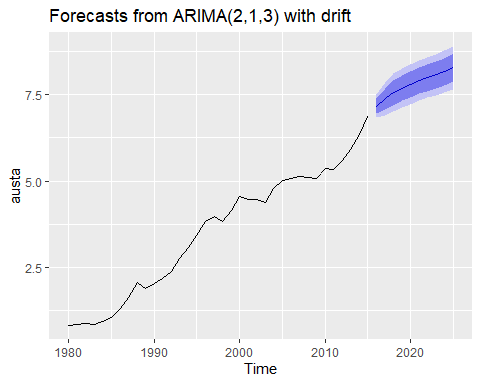
**Plot forecasts from an ARIMA(2,1,3) model with drift. Remove the constant and see what happens.**

First, plot forecasts from ARIMA(2,1,3) with drift.

# Fit ARIMA(2,1,3) model with drift  
fit4 <- Arima(austa, order=c(2,1,3), include.drift=T)  
summary(fit4)

## Series: austa   
## ARIMA(2,1,3) with drift   
##   
## Coefficients:  
## ar1 ar2 ma1 ma2 ma3 drift  
## 1.7648 -0.8513 -1.7684 0.5571 0.2224 0.1725  
## s.e. 0.1136 0.1182 0.2433 0.4351 0.2150 0.0051  
##   
## sigma^2 = 0.0276: log likelihood = 13.72  
## AIC=-13.44 AICc=-9.29 BIC=-2.55  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set -0.004129818 0.1491068 0.114857 -1.203567 4.326425 0.5637093  
## ACF1  
## Training set -0.01736817

# Plot  
fit4 %>%  
 forecast(h=10) %>%  
 autoplot()



Now remove the constant.

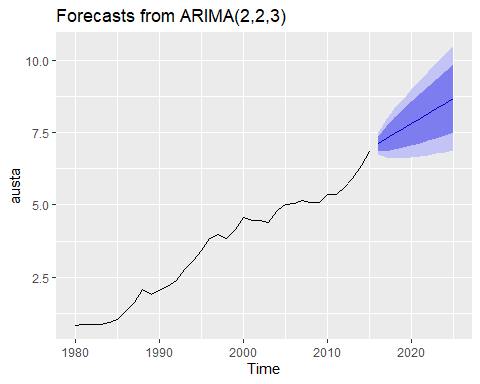
# Fit ARIMA(2,1,3) model with drift, removing constant;  
# generates an error - non-stationary series  
# fit5 <- Arima(austa, order=c(2,1,3), include.drift=T, include.constant=F)  
  
# Model using second-order differencing instead  
fit5 <- Arima(austa, order=c(2,2,3), include.drift=T, include.constant=F)  
summary(fit5)

## Series: austa   
## ARIMA(2,2,3)   
##   
## Coefficients:

## Warning in sqrt(diag(x$var.coef)): NaNs produced

## ar1 ar2 ma1 ma2 ma3  
## -0.1831 0.0032 -0.4918 -0.4398 -0.0685  
## s.e. NaN NaN NaN NaN 0.3921  
##   
## sigma^2 = 0.03844: log likelihood = 8.31  
## AIC=-4.62 AICc=-1.51 BIC=4.54  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.03113204 0.1759602 0.1476209 1.328102 4.710164 0.7245119  
## ACF1  
## Training set -0.03673752

# Plot  
fit5 %>%  
 forecast(h=10) %>%  
 autoplot()



Removing the constant from the model elicits an error from Arima() indicating that the AR part of the model is non-stationary. Performing another round of differencing fixes the problem, i.e. using an ARIMA(2,2,3) model, but it yields a model with a higher AICc than the ARIMA(2,1,3) model with drift.

#### HA 8.8 (d)

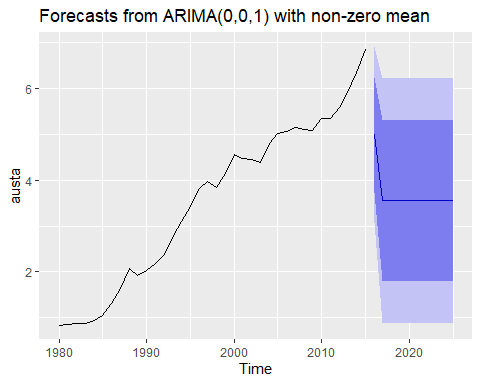
**Plot forecasts from an ARIMA(0,0,1) model with a constant. Remove the MA term and plot again.**

First plot forecast from ARIMA(0,0,1) with a constant.

# Model ARIMA(0,0,1) with constant  
fit6 <- Arima(austa, order=c(0,0,1), include.constant=T)  
summary(fit6)

## Series: austa   
## ARIMA(0,0,1) with non-zero mean   
##   
## Coefficients:  
## ma1 mean  
## 1.0000 3.5515  
## s.e. 0.0907 0.3090  
##   
## sigma^2 = 0.9347: log likelihood = -50.64  
## AIC=107.28 AICc=108.03 BIC=112.04  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.01997167 0.9395462 0.8063899 -26.87733 42.56419 3.957698  
## ACF1  
## Training set 0.8109528

# Plot forecasts  
fit6 %>%  
 forecast(h=10) %>%  
 autoplot()

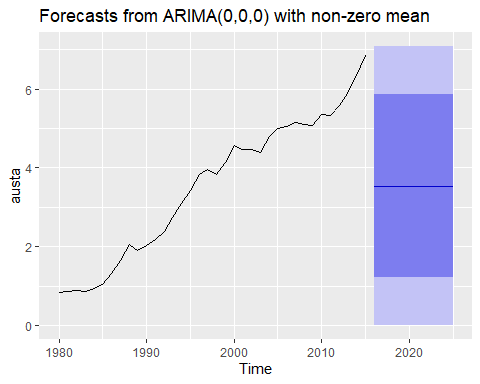


Now we can remove the MA term and plot again.

# Remove the MA term, i.e., model ARIMA(0,0,0) with constant  
fit7 <- Arima(austa, order=c(0,0,0), include.constant=T)  
summary(fit7)

## Series: austa   
## ARIMA(0,0,0) with non-zero mean   
##   
## Coefficients:  
## mean  
## 3.5414  
## s.e. 0.2972  
##   
## sigma^2 = 3.27: log likelihood = -71.9  
## AIC=147.8 AICc=148.17 BIC=150.97  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 5.00254e-13 1.783072 1.580774 -52.92811 82.83469 7.758317  
## ACF1  
## Training set 0.9099807

# Plot forecasts  
fit7 %>%  
 forecast(h=10) %>%  
 autoplot()



One moving average with a constant yields what appears to be the mean of the series as the forecast mean. Removing the constant expands the 95% prediction interval from zero to just greater than the last recorded observation.

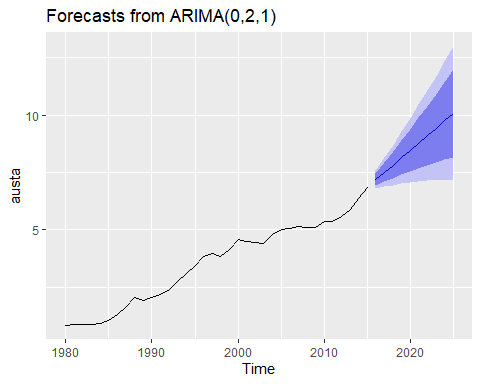
#### HA 8.8 (e)

**Plot forecasts from an ARIMA(0,2,1) model with no constant.**

# Remove the MA term, i.e., model ARIMA(0,0,0) with constant  
fit8 <- Arima(austa, order=c(0,2,1), include.constant=F)  
summary(fit8)

## Series: austa   
## ARIMA(0,2,1)   
##   
## Coefficients:  
## ma1  
## -0.7263  
## s.e. 0.2277  
##   
## sigma^2 = 0.03969: log likelihood = 6.74  
## AIC=-9.48 AICc=-9.09 BIC=-6.43  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.02907781 0.1907537 0.1567877 1.216128 4.984081 0.7695018  
## ACF1  
## Training set 0.1284994

# Plot forecasts  
fit8 %>%  
 forecast(h=10) %>%  
 autoplot()



This looks remarkably similar to the model with the auto.arima() function selected, albeit with a wider prediction interval.