Homework One

Group Two

Vinicio Haro
Sang Yoon (Andy) Hwang
Julian McEachern
Jeremy O'Brien
Bethany Poulin

15 October 2019

Contents

Overview

We created this together, each member doing all of the problems and then voting on what to include and what to leave out of our final submission. In most cases, we used the guts of one members work and supplemented it with plots, metrics and interpretations from some or all other team member answers.

Dependencies

The following R packages were used in the completion of this work. Loading them all at the beggining should ensure that any code found in this document runs correctly.

```
#Textbook Packages
library(fpp2)
library(AppliedPredictiveModeling)
library(mlbench)

#Processing
library(tidyverse)
library(RCurl)
library(readxl)

#Graphing
library(ggplot2)
library(grid)
library(gridExtra)
library(lemon)

#Math
library(tsfeatures)
library(caret)
library(forecast)
library(randomForest)
library(seasonal)
library(psych)
library(corrplot)
require(e1071)

#Formatting
library(knitr)
require(default)
library(kableExtra)
library(magrittr)
```

Data Files

Where data is not included via an R package, and needs to be supplied , we referrenced a folder parallel to our code as such '/data/retail.xlsx'. In order for our code to read properly, you should create a parallel folder `data` to host such files.

1 Assignment One

- Hyndman 2.1
- Hyndman 2.3

Hyndman Exercise 2.1

> Use the help function to explore what the series `gold`, `woolyrnq` and `gas` represent.

```
data("gold"); #help("gold")
data("woolyrnq"); #help("woolyrnq")
data('gold'); #help("gas")
```

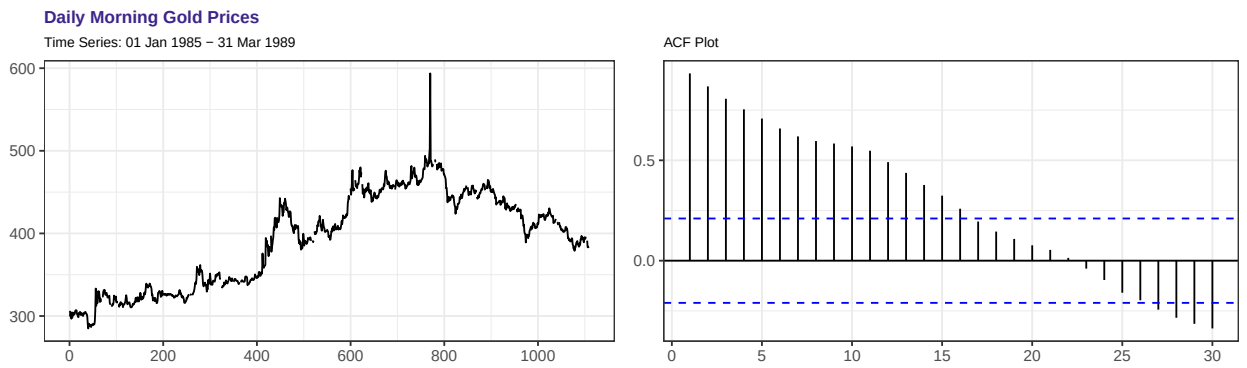
Per the `help` function, the `gold` series represents "daily morning gold prices in US dollars. 1 January 1985-31 March 1989"; the `woolyrnq` series represents "quarterly production of woollen yarn in Australia: tonnes. Mar 1965-Sep 1994"; and, the `gas` series represents "Australian monthly gas production: 1956-1995."

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

The time plots below were generated from the `autoplot` function.

`gold`

```
p1 <- autoplot(gold) + labs(title = "Daily Morning Gold Prices",
  subtitle = "Time Series: 01 Jan 1985 - 31 Mar 1989",
  x = "Day", y = "Price (USD)") + theme_bw() + theme()

p2 <- ggAcf(gold) + labs(title = "", subtitle = "ACF Plot") +
  theme_bw() + theme()

grid.arrange(p1, p2, nrow = 1)
```

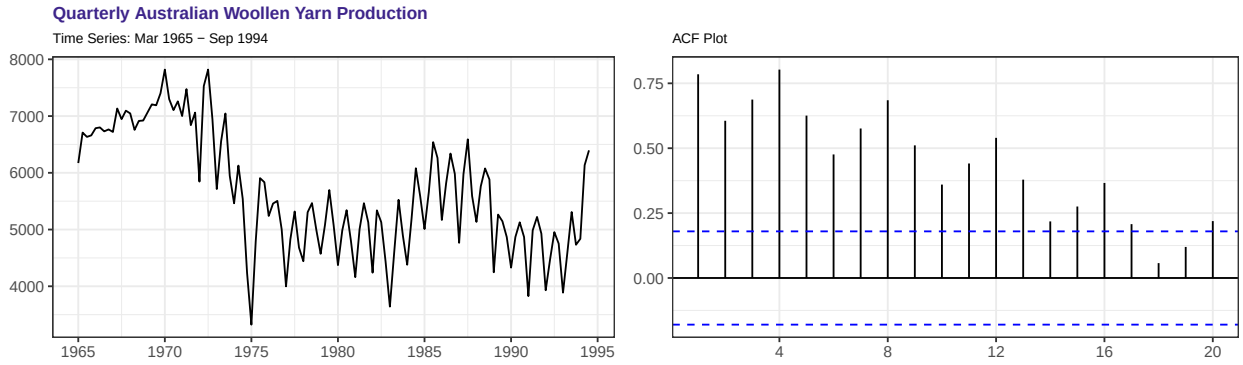


The `gold` series demonstrates a general, incremental trend until a large spike around day 775 (February 15, 1997). Thereafter the value of gold decreases over the remaining until the end of the timespan (March 31, 1989).

Based on the ACF plot, the trend increases until lag 22, after which it proceeds to decrease. The slightly scalloped shape between lags 5 and 10 suggest the possibility of some seasonality.

woolyrnq

```
p1 <- autoplot(woolyrnq) + labs(title = "Quarterly Australian Woollen Yarn Production",
  subtitle = "Time Series: Mar 1965 - Sep 1994", x = "Year",
  y = "Quantity (Tons)") + theme_bw() + theme()

p2 <- ggAcf(woolyrnq) + labs(title = "", subtitle = "ACF Plot") +
  theme_bw() + theme()

grid.arrange(p1, p2, nrow = 1)
```

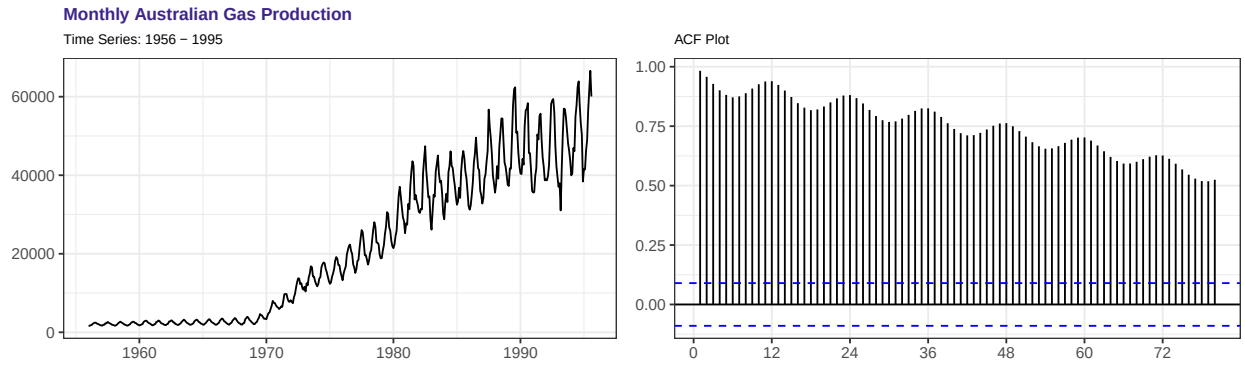


The `woolyrng` series exhibits a decremental trend, with a dramatic dip in early 1975.

The slight “U” shape repeating every 4 lags in the ACF plot suggests some seasonality, which can be described as mild cyclic behavior.

gas

```
p1 <- autoplot(gas) + labs(title = "Monthly Australian Gas Production",
  subtitle = "Time Series: 1956 - 1995", x = "Year", y = "Quantity") +
  theme_bw() + theme()

p2 <- ggAcf(gas, lag = 80) + labs(title = "", subtitle = "ACF Plot") +
  theme_bw() + theme()

grid.arrange(p1, p2, nrow = 1)
```



Lastly, the `gas` plot shows monthly changes to Australian gas production. As production takes off around 1970 in an overall increasing trend, the seasonal peaks and valleys also grow wider apart, suggesting a multiplicative relationship.

A moderate "U" shape repeats every 12 lags in the ACF plot, indicating a seasonal pattern.

b. What is the frequency of each series? Hint: apply the `frequency()` function.

```
frequency(gold)
frequency(woolyrnq)
frequency(gas)
```

The frequency of timeseries shows us the number of observations within a singular seasonal pattern.

- **Gas Frequency:** 12

- **Yarn Frequency:** 4
- **Gold Frequency:** 1

The output of the `frequency` function indicates that `gold` is an annual time series, `woolyrnq` is a quarterly series, and `gas` is a monthly series.

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

```
which.max(gold)
```

The `which.max()` function returns the index of the maximum value in a series:

Gold Maximum Value: 770

This number aligns with the observed spike in gold price from the time series plot in part (a).

Hyndman Exercise 2.3

> Download some monthly Australian retail data from the book website. These represent retail sales in various categories for different Australian states, and are stored in a MS-Excel file.

a. You can read the data into R with the following script:

```
retaildata <- readxl::read_excel("data/retail.xlsx", skip = 1)
```

The second argument (skip=1) is required because the Excel sheet has two header rows.

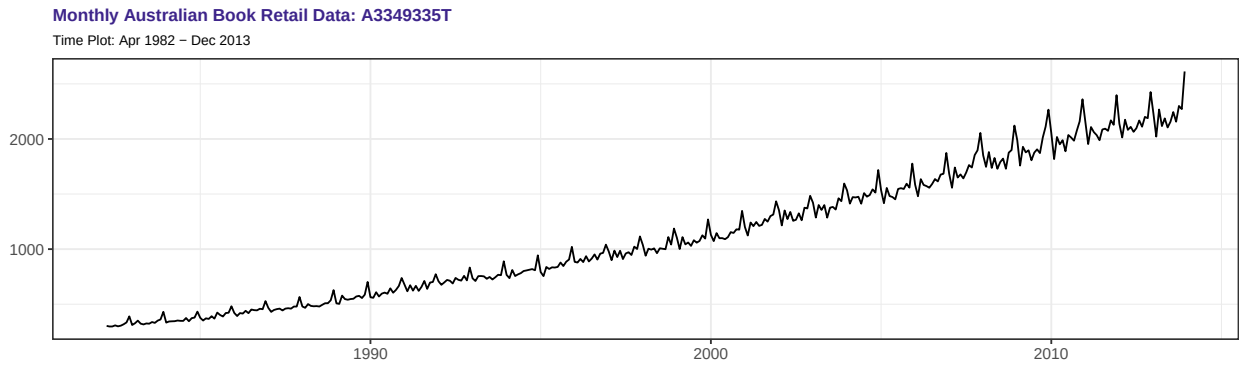
b. Select one of the time series as follows (but replace the column name with your own chosen column):

```
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()`.

Autoplot

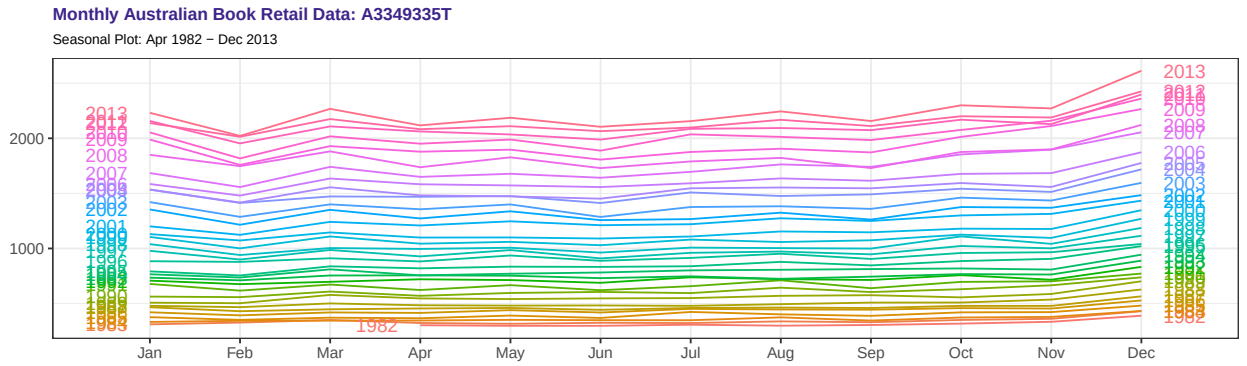
```
autoplot(myts) + labs(title = "Monthly Australian Book Retail Data: A3349335T",
  subtitle = "Time Plot: Apr 1982 - Dec 2013", x = "Year",
  y = "Sales") + theme_bw() + theme()
```



The output of the `autoplot` function demonstrate a general, incremental trend over the observed period, with a more pronounced peaks and valleys in the seasonal pattern.

Seasonal Plot

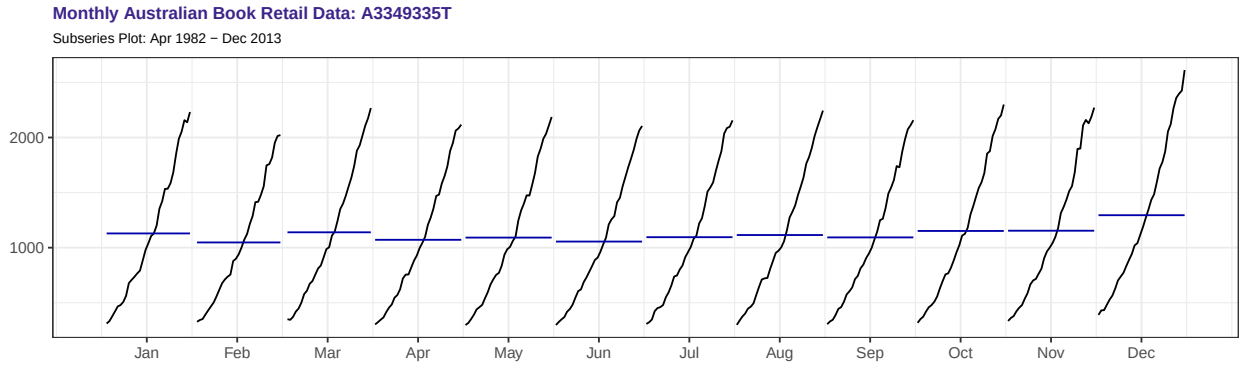
```
ggseasonplot(myts, year.labels = TRUE, year.labels.left = TRUE) +
  labs(title = "Monthly Australian Book Retail Data: A3349335T",
    subtitle = "Seasonal Plot: Apr 1982 - Dec 2013",
    x = "Month", y = "Sales") + theme_bw() + theme()
```



Seasonal plots shows the observed data plotted for each year within the series. The seasons overlap to highlight underlying seasonal patterns and the years in which these trends occur. The seasonal plot below reveals that book sales tend to decline over January and February, rise in March and bump around over spring and summer before increasing over fall months into December. The overall increasing trend is evidenced by the roughly chronological progression in years on the right and left margins.

Subseries Plot

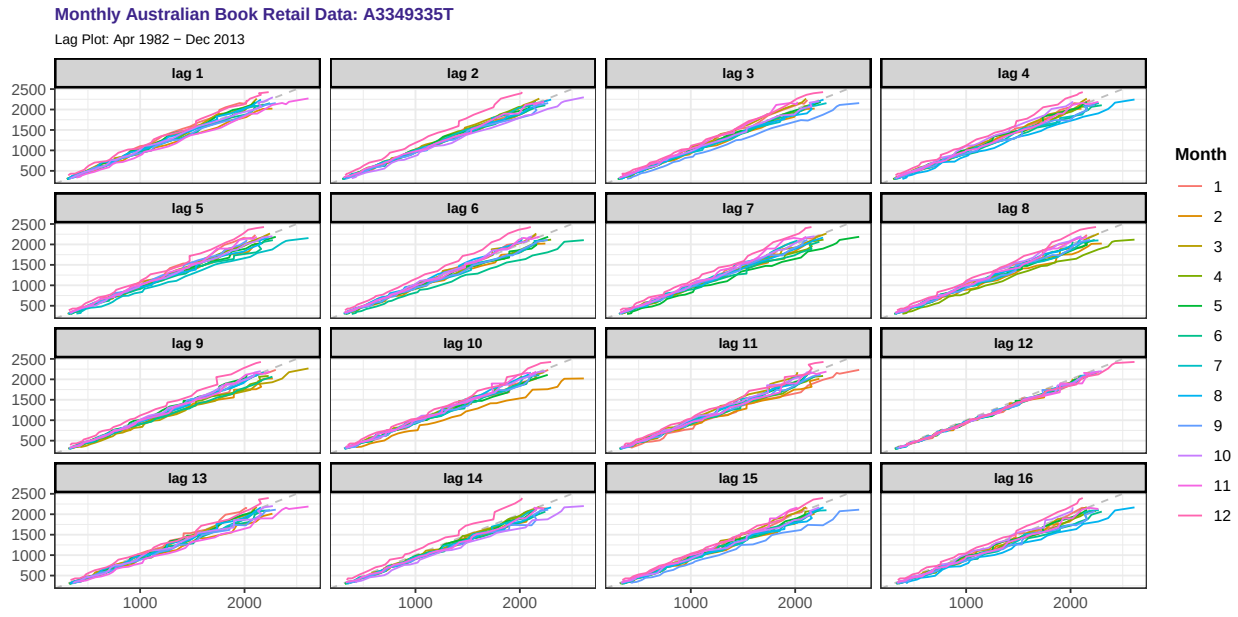
```
ggsubseriesplot(myts) + labs(title = "Monthly Australian Book Retail Data: A3349335T",
  subtitle = "Subseries Plot: Apr 1982 - Dec 2013", x = "Month",
  y = "Sales") + theme_bw() + theme()
```



The subseries plot provides a different way to view underlying seasonal patterns by separating each season in a year into an individual plot. The horizontal line shows the mean value of each month. The plot below reveals that, on average, the month of February sees the lowest sales and the month of December the highest over the observed time period.

Lag Plots

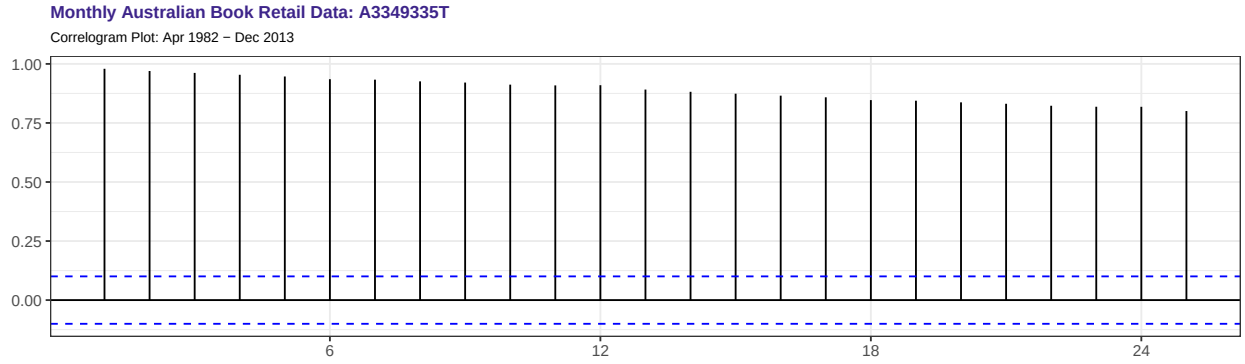
```
gglagplot(myts) + labs(title = "Monthly Australian Book Retail Data: A3349335T",
  subtitle = "Lag Plot: Apr 1982 - Dec 2013", x = "Month",
  y = "Sales") + theme_bw() + theme()
```



Lag plots are used to examine the correlation between the X and Y axis over a fixed period of time with a scatterplot. Per the text, "each graph shows y_t plotted against $y_t - k$ for different values of k." The lag plot below shows an overall positive relationship at each lag, indicating a strong seasonal relationship in the series.

Autocorrelation Plot

```
ggAcf(myts) + labs(title = "Monthly Australian Book Retail Data: A3349335T",
  subtitle = "Correlogram Plot: Apr 1982 - Dec 2013", x = "Month",
  y = "Sales") + theme_bw() + theme()
```



Lastly, our autocorrelation plot (also referred to as correlogram plot) measures the linear relationship between the time series' lagged values. Below we used the autocorrelation function `ggAcf` to examine this relationship for the selected variable from the book retail data. As $r1$ shows the largest lag and $r25$ the smallest, this decrease in positive values as lags accrue clearly indicates a trend in the series.

2 Assignment Two

- Hyndman 6.2

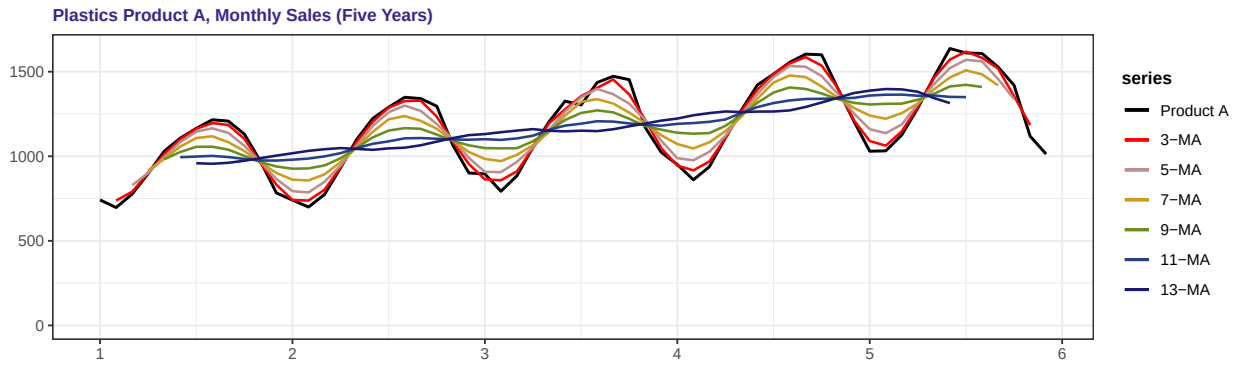
Hyndman 6.2

> The `plastics` dataset consists of the monthly sales (in thousands) of product A for a plastics manufacturer for five years.

```
help(plastics)
str(plastics)
```

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

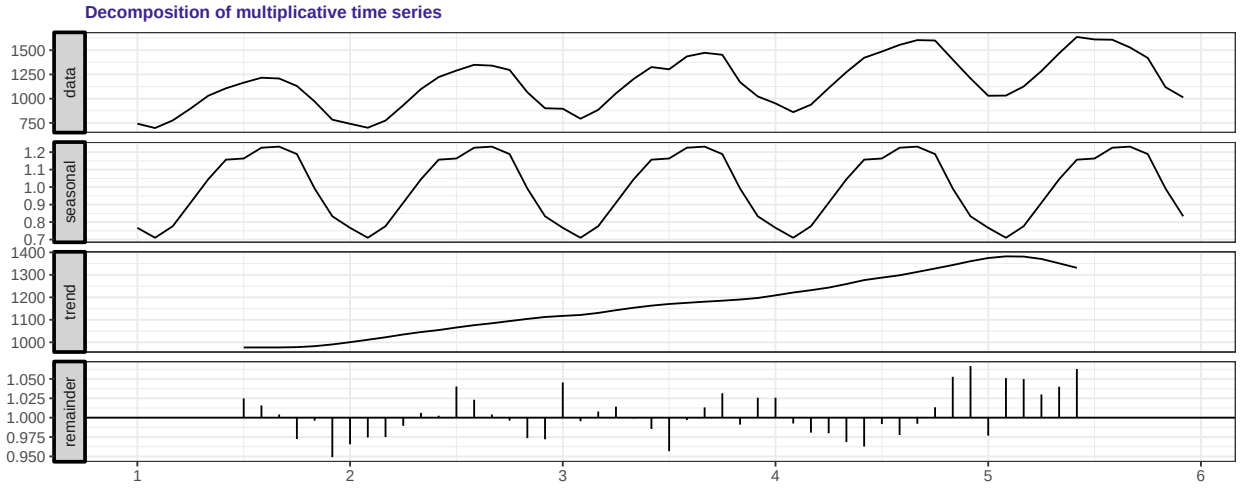
```
autoplot(plastics, main = "Plastics Product A, Monthly Sales (Five Years)",
  series = "Product A", size = 0.8) + autolayer(ma(plastics,
  order = 3), series = "3-MA", size = 0.7) + autolayer(ma(plastics,
  order = 5), series = "5-MA", size = 0.7) + autolayer(ma(plastics,
  order = 7), series = "7-MA", size = 0.7) + autolayer(ma(plastics,
  order = 9), series = "9-MA", size = 0.7) + autolayer(ma(plastics,
  order = 11), series = "11-MA", size = 0.7) + autolayer(ma(plastics,
  order = 13), series = "13-MA", size = 0.7) + ylab(label = "") +
  ylim(0, max(plastics)) + scale_color_manual(values = c(`Product A` = "black",
  `3-MA` = "red", `5-MA` = "rosybrown", `7-MA` = "goldenrod3",
  `9-MA` = "olivedrab", `11-MA` = "royalblue4", `13-MA` = "midnightblue"),
  breaks = c("Product A", "3-MA", "5-MA", "7-MA", "9-MA",
    "11-MA", "13-MA")) + theme_bw() + theme()
```



For this question, we plotted the monthly plastics time series together with moving averages of order 3, 5, 7, 9, 11, and 13. There's a clear seasonal fluctuation, exhibiting troughs at the beginning of each year and peaks over August through October. Additionally, there's a consistent upward trend.

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

```
# Produce multiplicative decomposition of time series
plastics_decomp <- plastics %>% decompose(type = "multiplicative")

# Plot
plastics %>% decompose(type = "multiplicative") %>% autoplot() +
  xlab("Month") + ylab("Sales") + theme_bw() + theme()
```




```
# Output the trend strength and seasonal strength metrics
plastics %>% tsfeatures() %>% select(trend, seasonal_strength) %>%
  rename(trend_str = trend, season_str = seasonal_strength)
```

```
FALSE # A tibble: 1 x 2
FALSE   trend_str season_str
FALSE       <dbl>      <dbl>
FALSE 1     0.919      0.964
```

```
# The tsfeatures package provides statistics on strength
# of trend and seasonality, which doesn't seem to be a
# built-in feature of the forecast or other time series
# packages:
# https://rdrr.io/github/robjhyndman/tsfeatures/man/stl_features.html

# Output the seasonal index
(function(x) print(paste(min(x), max(x))))(plastics_decomp$figure)
```

```
FALSE [1] "0.710335708434527 1.23136353496152"
```

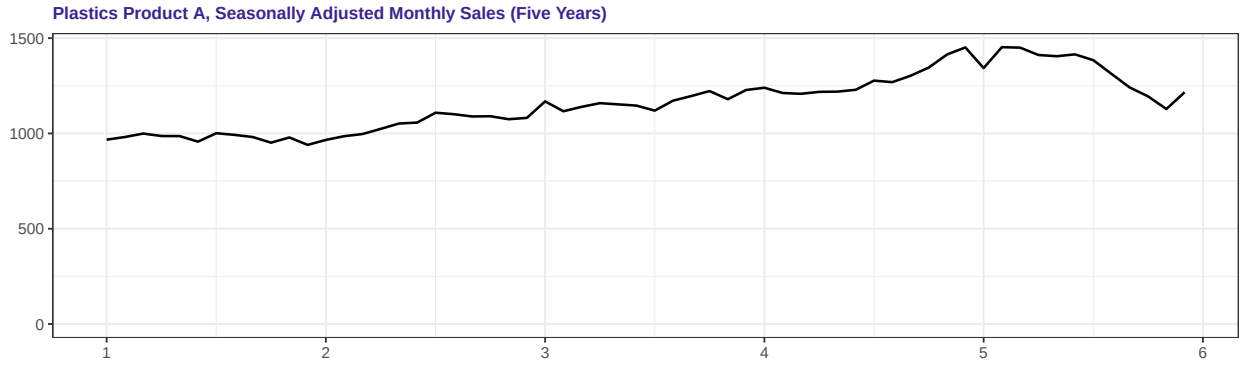
The multiplicative decomposition, with F_S of .963 (close to the maximum of 1), substantiates the strong seasonal trend observed in the data. The seaonsal index ranges between .71 and 1.23. The trend is also strong, with F_T of .92 (also close to the maximum of 1), which supports our answer for c..

c. Do the results support the graphical interpretations from part a?

Yes, when decomposed, the trend becomes very clearly upward, while a rather unvarying seasonal component emerges. The one very interesting component is the random, component, it does seem to imply an unexpained mechanism (with some variability over time).

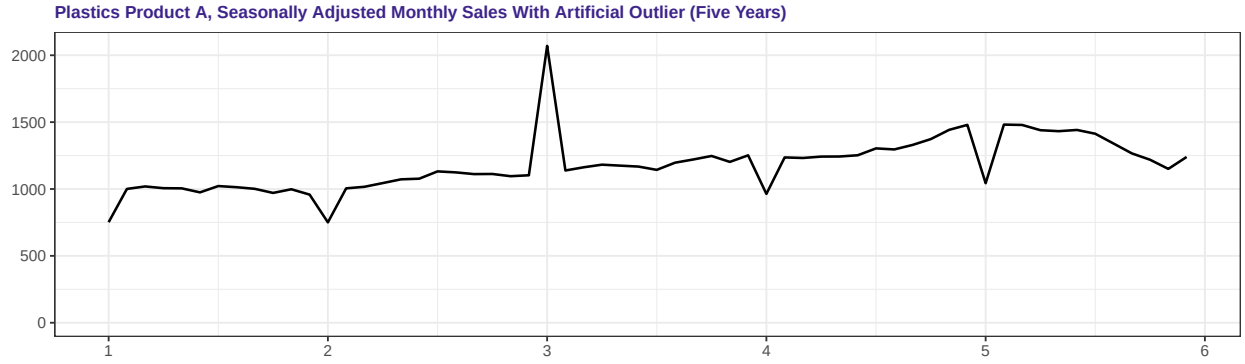
d. Compute and plot the seasonally adjusted data.

```
# Compute the seasonally adjusted monthly plastics time
# series
plastics_seasadj <- plastics/plastics_decomp$seasonal

# Plot the seasonally adjusted monthly plastics time
# series
autoplot(plastics_seasadj, main = "Plastics Product A, Seasonally Adjusted Monthly Sales (Five Years)",
  series = "Product A (Seasonally Adjusted)", size = 0.7,
  color = "black") + theme_bw() + theme() + ylab(label = "") +
  ylim(0, max(plastics_seasadj))
```



e. Change one observation to be an outlier and recompute the seasonally adjusted data. What is the effect of the outlier?

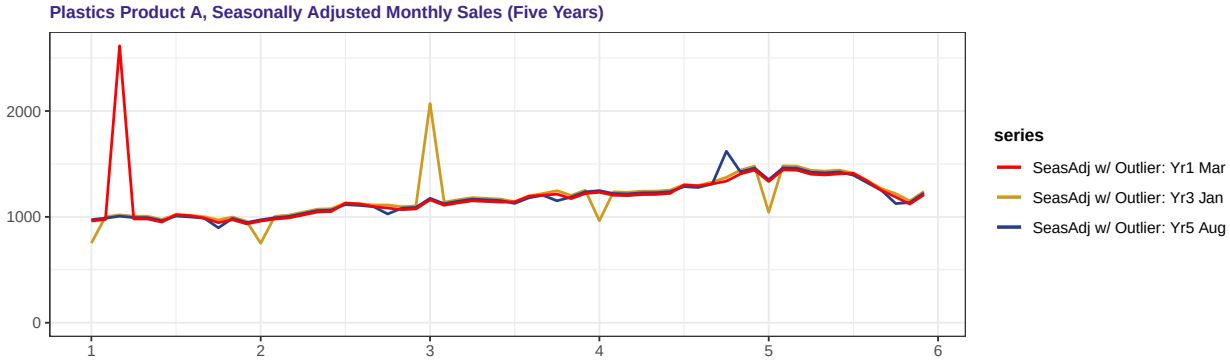
```
# Artifically change single observation to create outlier
# between 1.5x IQR in middle of time series
plastics_2 <- plastics
plastics_2[25] <- quantile(plastics_2)[4] * 1.5
plastics_2_decomp <- decompose(plastics_2, type = "multiplicative")
plastics_2_seasadj <- plastics_2/plastics_2_decomp$seasonal

# Plot the seasonally adjusted monthly plastics time
# series
autoplot(plastics_2_seasadj, main = "Plastics Product A, Seasonally Adjusted Monthly Sales With Artifici
  series = "Product A (Seasonally Adjusted)", size = 0.7,
  color = "black") + theme_bw() + theme() + ylab(label = "") +
  ylim(0, max(plastics_2_seasadj))
```



The effect is to deflate the corresponding month in other years in the seasonally adjusted data. In this case, the first month (January) of Year 3 was artifically inflated to approximately 2,111, which is 1.5 times larger than the upper bound of the inter-quartile range (the minimum value for an outlier). The second months of years 1-2 and 4-5 have all fallen to compensate together for this outlier and now represent troughs.

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

```
# Create outliers at beginning and end of respective time
# series
plastics_3 <- plastics
plastics_3[46] <- quantile(plastics_3)[4] * 1.5
plastics_3_decomp <- decompose(plastics_3, type = "multiplicative")
plastics_3_seasadj <- plastics_3/plastics_3_decomp$seasonal

plastics_4 <- plastics
plastics_4[3] <- quantile(plastics_4)[4] * 1.5
plastics_4_decomp <- decompose(plastics_4, type = "multiplicative")
plastics_4_seasadj <- plastics_4/plastics_4_decomp$seasonal

# autoplot(plastics_4_seasadj)

# Plot different outlier time series for comparison
autoplot(plastics_2_seasadj, main = "Plastics Product A, Seasonally Adjusted Monthly Sales (Five Years)
  series = "SeasAdj w/ Outlier: Yr3 Jan", size = 0.75) +
  autolayer(plastics_3_seasadj, series = "SeasAdj w/ Outlier: Yr5 Aug",
    size = 0.75) + autolayer(plastics_4_seasadj, series = "SeasAdj w/ Outlier: Yr1 Mar",
  size = 0.75) + theme_bw() + theme() + ylab(label = "") +
  ylim(0, max(c(max(plastics_seasadj), max(plastics_2_seasadj),
    max(plastics_3_seasadj), max(plastics_4_seasadj)))) +
  scale_color_manual(values = c(SeasAdj = "black", `SeasAdj w/ Outlier: Yr1 Mar` = "red",
    `SeasAdj w/ Outlier: Yr3 Jan` = "goldenrod3", `SeasAdj w/ Outlier: Yr5 Aug` = "royalblue4"),
    breaks = c("SeasAdj", "SeasAdj w/ Outlier: Yr1 Mar",
      "SeasAdj w/ Outlier: Yr3 Jan", "SeasAdj w/ Outlier: Yr5 Aug"))
```


Plastics Product A, Seasonally Adjusted Monthly Sales (Five Years)
2000
1000
0
1
2
3
4
5
6
series
SeasAdj w/ Outlier: Yr1 Mar
SeasAdj w/ Outlier: Yr3 Jan
SeasAdj w/ Outlier: Yr5 Aug

3 Assignment Three

- Kuhn & Johnson 3.1
- Kuhn & Johnson 3.2

Kuhn & Johnson 3.1

> The UC Irvine Machine Learning Repository 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.

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

Descriptive Statistics

The output of the `describe( )` function suggests some variables to take closer looks at. RI, Mg, K, Ca, Ba and Fe appear skewed; and RI, Na, Si, K , Ca, Ba, and (to a lesser extent) Fe exhibit some challenging tails (kurtosis != 3.0).

Table 3.1: Summary Statistics of Glass data

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
RI	1	214	1.52	0.00	1.52	1.52	0.00	1.51	1.53	0.02	1.60	4.72	0.00
Na	2	214	13.41	0.82	13.30	13.38	0.64	10.73	17.38	6.65	0.45	2.90	0.06
Mg	3	214	2.68	1.44	3.48	2.87	0.30	0.00	4.49	4.49	-1.14	-0.45	0.10
Al	4	214	1.44	0.50	1.36	1.41	0.31	0.29	3.50	3.21	0.89	1.94	0.03
Si	5	214	72.65	0.77	72.79	72.71	0.57	69.81	75.41	5.60	-0.72	2.82	0.05
K	6	214	0.50	0.65	0.56	0.43	0.17	0.00	6.21	6.21	6.46	52.87	0.04
Ca	7	214	8.96	1.42	8.60	8.74	0.66	5.43	16.19	10.76	2.02	6.41	0.10
Ba	8	214	0.18	0.50	0.00	0.03	0.00	0.00	3.15	3.15	3.37	12.08	0.03
Fe	9	214	0.06	0.10	0.00	0.04	0.00	0.00	0.51	0.51	1.73	2.52	0.01
Type*	10	214	2.54	1.71	2.00	2.31	1.48	1.00	6.00	5.00	1.04	-0.29	0.12

Skewness

We can also look at the skew of our numeric variables using the `e1071` package:

```
skewer<-function(df){
  new_skew = setNames(data.frame(matrix(ncol = 2, nrow = 0)), c("Element", "Skew"))
    for(name in colnames(df)){
    skew <-skewness(df[,name])
    temp = data.frame(cbind('Element' =name, 'Skew' =skew))
    new_skew <-rbind(new_skew, temp)}
```

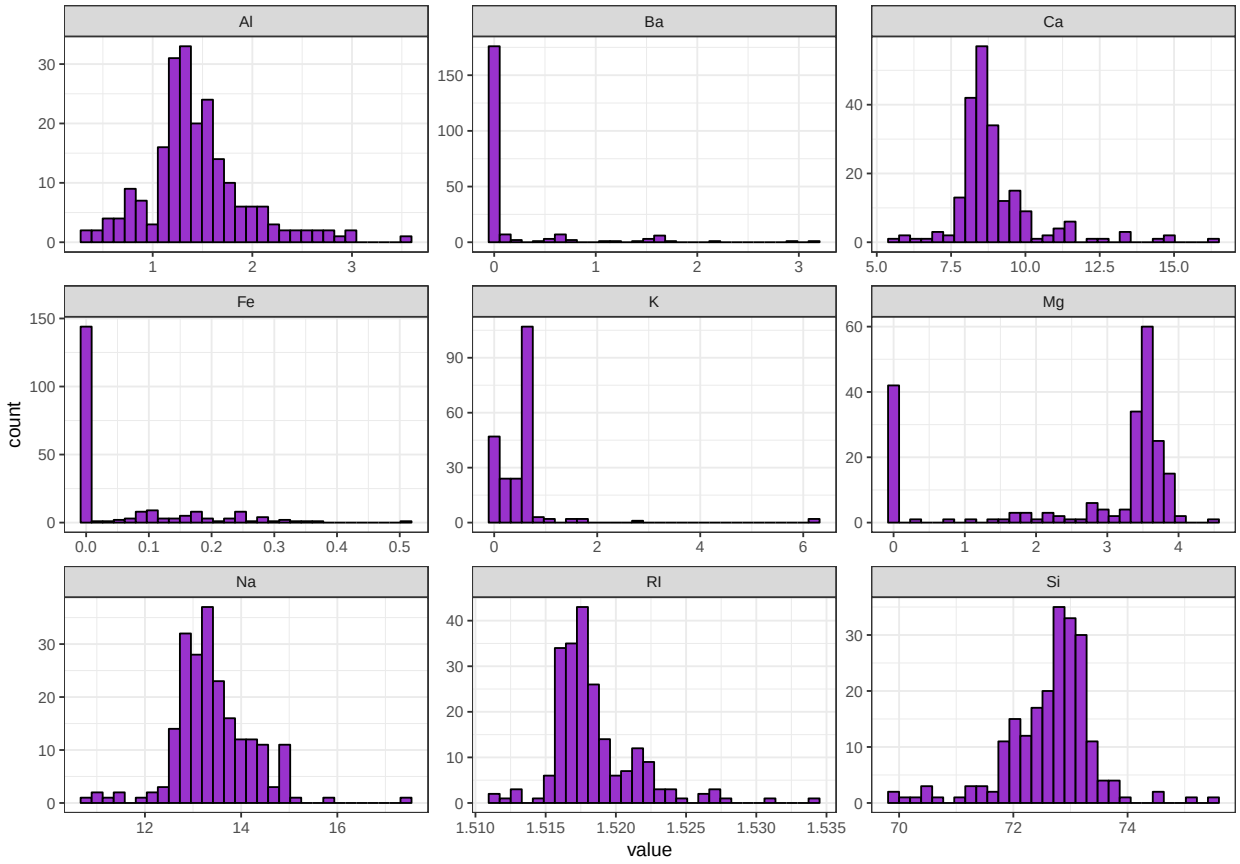
```
  return(new_skew)
}

skewer(Glass[, 1:9])%>%
  kable("latex") %>%
  kable_styling(latex_options = c("hold_position", "striped"))
```

Element	Skew
RI	1.60271508274373
Na	0.447834258917133
Mg	-1.13645227846653
Al	0.89461041611312
Si	-0.720239210805621
K	6.46008889572281
Ca	2.01844629445302
Ba	3.36867996880571
Fe	1.7298107095598

Distribution

Histograms visually illustrate the challenges with these distributions.

```
Glass %>% keep(is.numeric) %>% gather() %>% ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") + geom_histogram(fill = "darkorchid",
  color = "black") + theme_bw()
```



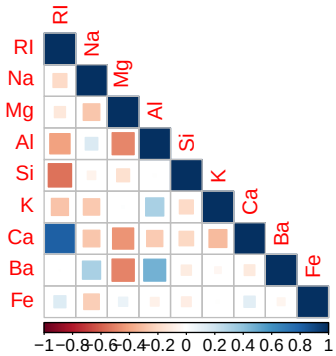
Correlation

We used a corrplot to look at the correleations between numeric glass variables.

There is some significant positive correlation between Ca and RI; slightly less between Ba and Al, Ba and Na, and K and Al.

There are some obvious negative correlations between Si and RI, as well as Al and RI, Al and Mg, Ca and Mg, and Ba and Mg.

```
glassCor <- cor(Glass[-10])
corrplot(glassCor, method = "square", type = "lower")
```



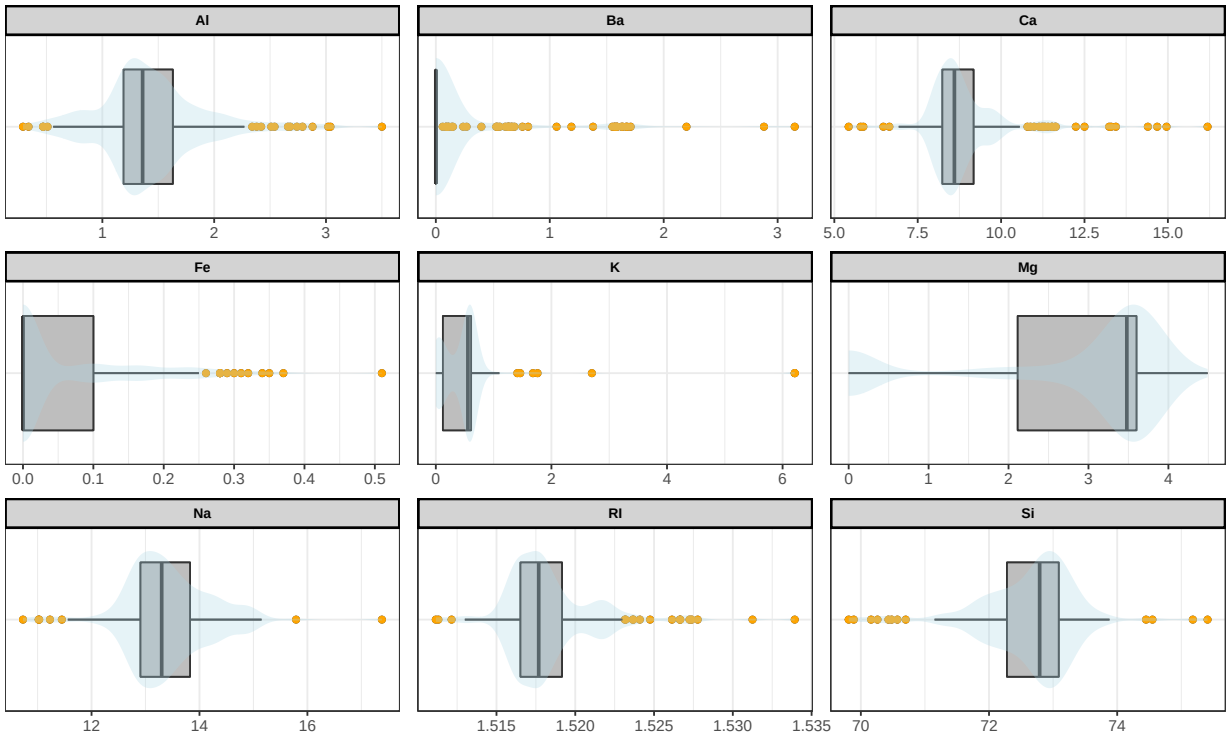
b. Do there appear to be any outliers in the data? Are any predictors skewed?

Boxplot

The long tails of some of the predictors visible in histograms are indicative of outliers. The plot below shows a boxplot and violin plot for each variable. Many of the variables appear to have outliers; `Ba`, `Fe` and `K` appear to have the most potential outliers. Handling these outliers will depend on a mix of reference data and statistical techniques.

Skewness is also visible in the histograms, `Mg` is bimodal. `Ba`, `K` and `Fe` are right skewed. The remaining variables don't appear to be normally distributed - though they exhibit central tendency, there are signs that they are leptokurtic or platykurtic. This suggests exploring skew and possibly transformations might be of use down the line.

```
# boxplot with violin plot overlaid for all variables
Glass %>% keep(is.numeric) %>% gather() %>% group_by(key) %>%
  ggplot(data = ., aes(x = "", y = value)) + geom_boxplot() +
  geom_boxplot(outlier.colour = "orange", fill = "grey") +
  geom_violin(alpha = 0.3, color = NA, fill = "lightblue") +
  labs(x = NULL, y = NULL) + theme_bw() + theme(axis.ticks.y = element_blank()) +
  facet_wrap(~key, scales = "free", ncol = 3) + coord_flip()
```



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

Below we conduct a simple exploration of histograms for skewed distributions using `BoxCoxTrans( )`:

```
skews <- c("Mg", "Fe", "Ba", "Ca", "K")
```

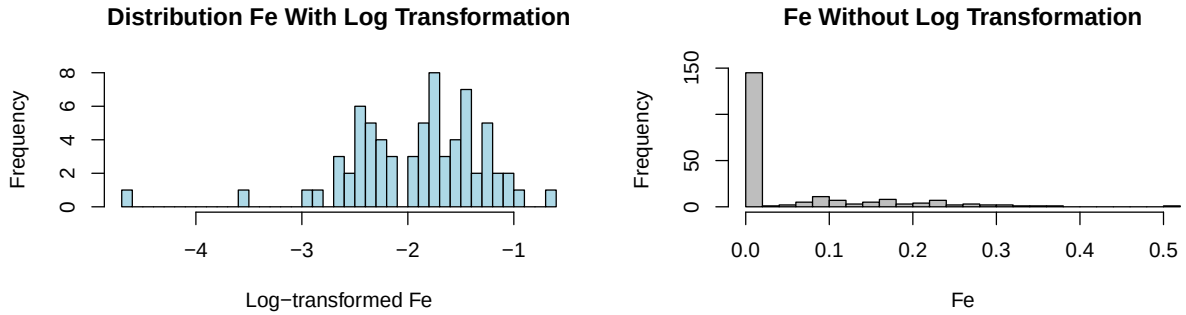
```r
# MASS::boxcox(Glass)
for (skw in skews) {
  print(skw)
  print(BoxCoxTrans(Glass[, skw]))
}
```

```
FALSE [1] "Mg"
FALSE Box-Cox Transformation
FALSE
FALSE 214 data points used to estimate Lambda
FALSE
FALSE Input data summary:
FALSE    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
FALSE   0.000   2.115   3.480   2.685   3.600   4.490
FALSE
FALSE Lambda could not be estimated; no transformation is applied
FALSE
FALSE [1] "Fe"
FALSE Box-Cox Transformation
FALSE
FALSE 214 data points used to estimate Lambda
FALSE
FALSE Input data summary:
FALSE    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
FALSE 0.00000 0.00000 0.00000 0.05701 0.10000 0.51000
FALSE
FALSE Lambda could not be estimated; no transformation is applied
FALSE
FALSE [1] "Ba"
FALSE Box-Cox Transformation
FALSE
FALSE 214 data points used to estimate Lambda
FALSE
FALSE Input data summary:
FALSE    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
FALSE   0.000   0.000   0.000   0.175   0.000   3.150
FALSE
FALSE Lambda could not be estimated; no transformation is applied
FALSE
FALSE [1] "Ca"
FALSE Box-Cox Transformation
FALSE
FALSE 214 data points used to estimate Lambda
FALSE
FALSE Input data summary:
FALSE    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
FALSE   5.430   8.240   8.600   8.957   9.172  16.190
FALSE
FALSE Largest/Smallest: 2.98
FALSE Sample Skewness: 2.02
FALSE
FALSE Estimated Lambda: -1.1
FALSE
```

```
FALSE [1] "K"
FALSE Box-Cox Transformation
FALSE
FALSE 214 data points used to estimate Lambda
FALSE
FALSE Input data summary:
FALSE    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
FALSE  0.0000  0.1225  0.5550  0.4971  0.6100  6.2100
FALSE
FALSE Lambda could not be estimated; no transformation is applied
```

A simple log transformation shows improvements in the distribution of the iron variable Fe. However it does introduce outliers on the left; additionally, all the zero values of Fe go to -Inf and are not reflected at all in this distribution.

```
par(mfrow = c(1, 2))
hist(log(Glass$Fe), breaks = 30, col = "lightblue", main = "Distribution Fe With Log Transformation",
  xlab = "Log-transformed Fe")
hist(Glass$Fe, breaks = 30, col = "gray", main = "Fe Without Log Transformation",
  xlab = "Fe")
```



We then examined the first few rows of the data before and after transformations to see what changes the Box-Cox method suggests.

```
head(Glass) %>% kable(caption = "Glass (without transformations)",
  digits = 2, "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

Table 3.2: Glass (without transformations)

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1.52	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00	1
1.52	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00	1
1.52	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00	1
1.52	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00	1
1.52	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00	1
1.52	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1

```
trans <- preProcess(Glass, method = c("center", "scale",
  "BoxCox"))
trans_glass <- predict(trans, Glass)
```

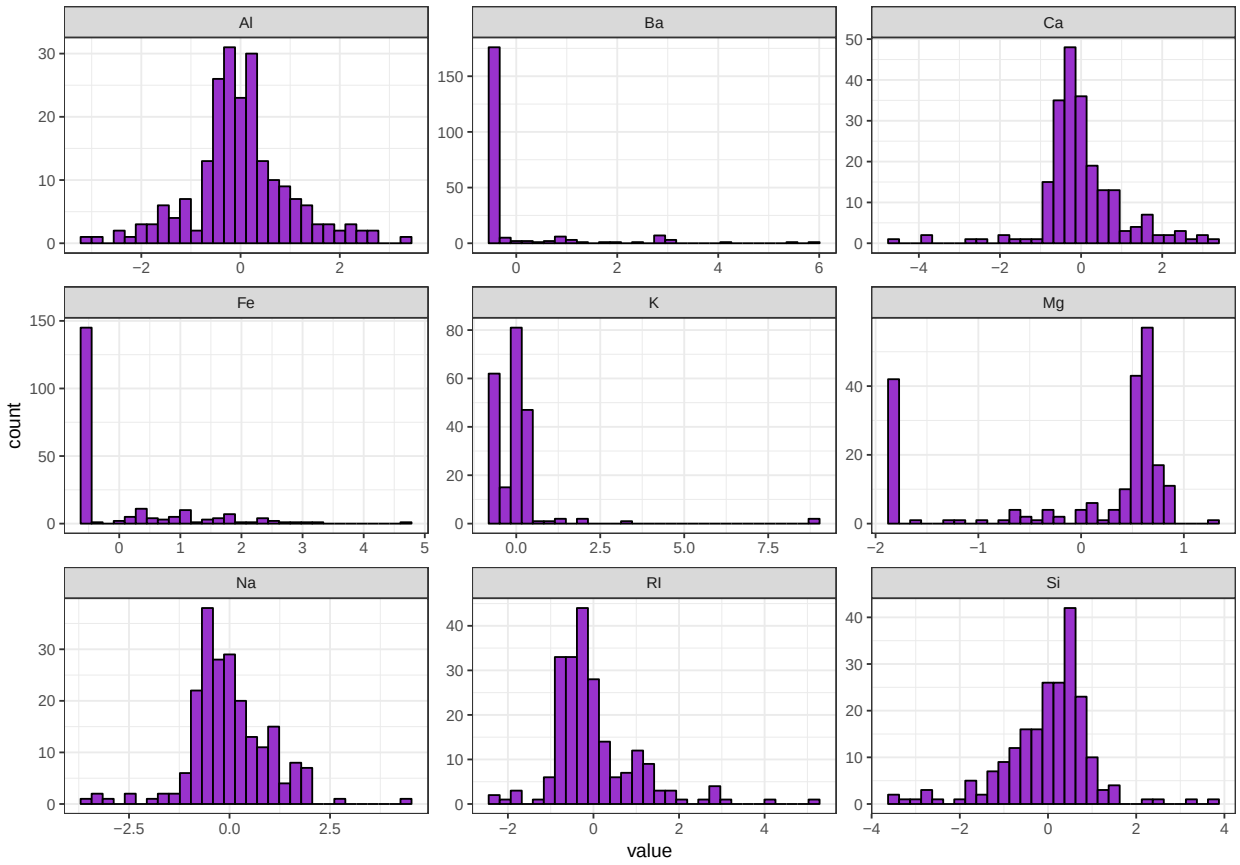
```
head(trans_glass) %>% kable(caption = "Glass (with transformations)",
  digits = 2, "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

Table 3.3: Glass (with transformations)

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0.88	0.31	1.25	-0.66	-1.13	-0.67	-0.01	-0.35	-0.59	1
-0.25	0.61	0.63	-0.09	0.10	-0.03	-0.88	-0.35	-0.59	1
-0.72	0.18	0.60	0.27	0.44	-0.16	-0.93	-0.35	-0.59	1
-0.23	-0.22	0.70	-0.23	-0.06	0.11	-0.49	-0.35	-0.59	1
-0.31	-0.14	0.65	-0.34	0.55	0.08	-0.63	-0.35	-0.59	1
-0.79	-0.75	0.64	0.43	0.41	0.22	-0.63	-0.35	2.08	1

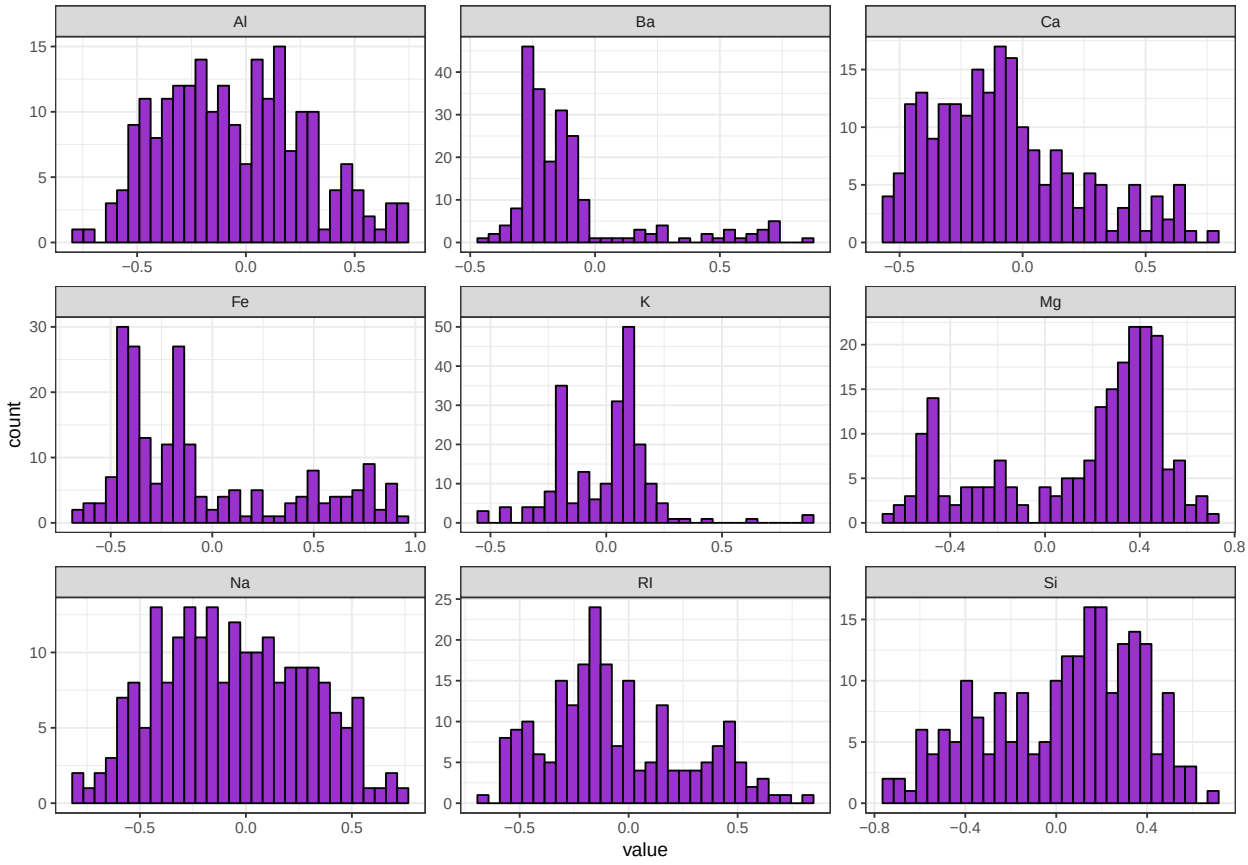
Finally, we looked at the histograms of our variables after automated application of the Box-Cox transformations.

```
trans_glass %>% keep(is.numeric) %>% gather() %>% ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") + geom_histogram(fill = "darkorchid",
  color = "black") + theme_bw()
```



While the transformation produces marginal improvement in most of the variables the only one with noteworthy changes is Ca. Given the number of outliers and skew in this data, it might be useful to attempt a `spatialSign()` transformation.

```
trans <- preProcess(Glass, method = c("center", "scale",
  "spatialSign"))

trans_glass <- predict(trans, Glass)

trans_glass %>% keep(is.numeric) %>% gather() %>% ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") + geom_histogram(fill = "darkorchid",
  color = "black") + theme_bw()
```



Without Spatial Sign:

```
head(Glass) %>% kable("latex", digits = 3) %>% kable_styling(latex_options = "hold_position")
```

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1.521	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00	1
1.518	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00	1
1.516	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00	1
1.518	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00	1
1.517	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00	1
1.516	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1

With Spatial Sign:

```
head(trans_glass) %>% kable("latex", digits = 3) %>% kable_styling(latex_options = "hold_position")
```

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0.386	0.126	0.555	-0.306	-0.499	-0.297	-0.064	-0.156	-0.259	1
-0.178	0.423	0.455	-0.122	0.073	-0.019	-0.568	-0.252	-0.419	1
-0.474	0.099	0.395	0.125	0.288	-0.108	-0.545	-0.232	-0.385	1
-0.193	-0.202	0.580	-0.258	-0.044	0.093	-0.431	-0.293	-0.487	1
-0.227	-0.123	0.473	-0.299	0.404	0.059	-0.454	-0.257	-0.426	1
-0.304	-0.290	0.246	0.134	0.158	0.084	-0.239	-0.135	0.799	1

The spatial sign-transformed distributions are the most normal and best for modeling. While custom transformations of each variable might prove even better robust, the simple spatial sign grooming and pre-processing method seems practical and useful for this data set.

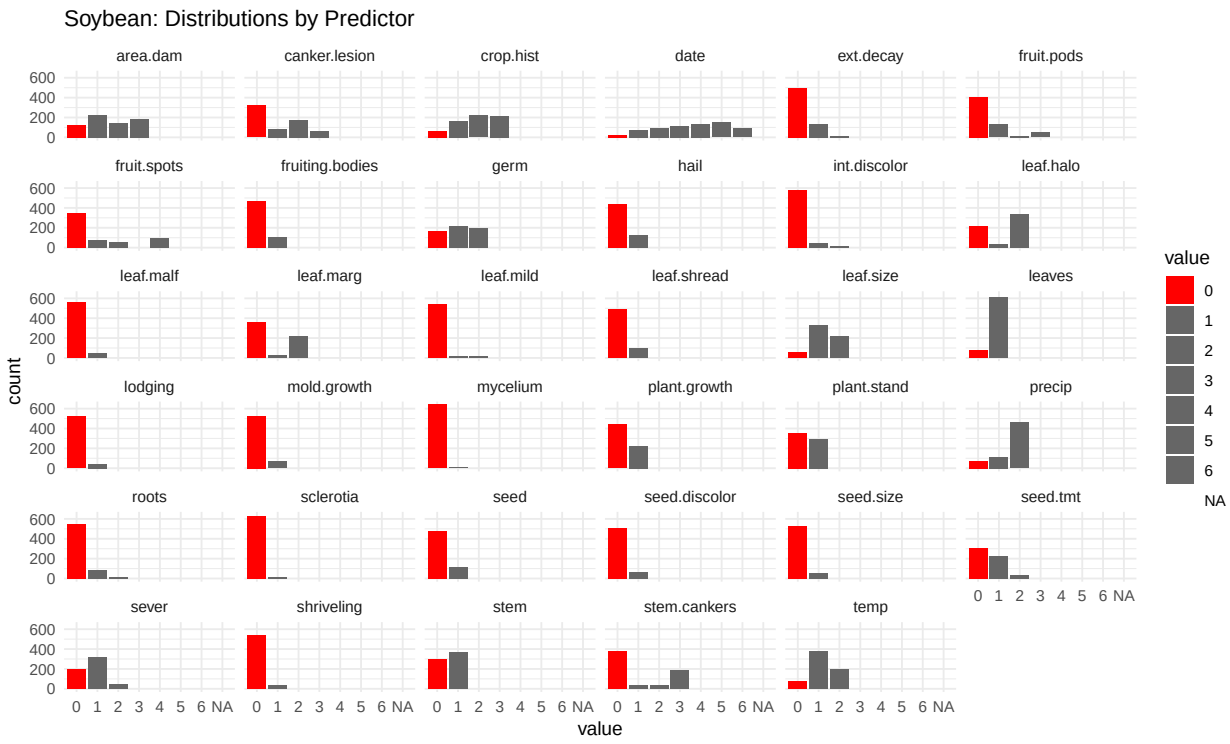
Kuhn & Johnson 3.2

> 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.

```
data(Soybean)
# str(Soybean)
```

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

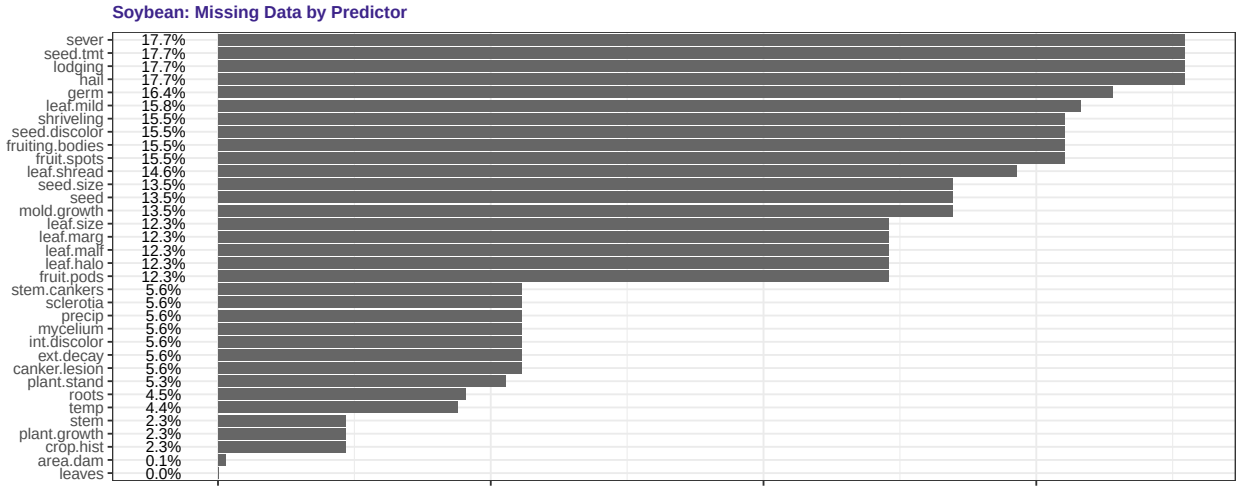
```
# Tidy dataset, removing non-numeric variables
Soybean %>% select(-Class) %>% gather() %>%
# Depict distribution of each class within categories
ggplot(aes(value, fill = value)) + geom_bar() +
# Code all 0 factor values as red for easier visual
# detection of degenerate distribution
scale_fill_manual(values = c("red", rep("grey40", 7))) +

# Facet plots by predictor and attenuate chart look
facet_wrap(~key) + theme_minimal() + labs(title = "Soybean: Distributions by Predictor")
```



Distributions are regarded as degenerate when they have a unique values with extremely low frequencies, i.e. 'predictors with a single value for the vast majority of samples'. `mycelium` and `sclerotica` both fall into this category, and arguably `leaf.mild`, `lodging`, `seed.discolor`, `seed.size`, and `shriveling` could also be considered.

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?**

```
# Tidy dataset, removing non-numeric variables
Soybean %>% select(-Class, -date) %>% summarise_all(funs(perc_missing = sum(is.na((.))/nrow(Soybean)))) %>%
  rename_all(funs(str_replace(., "_perc_missing", ""))) %>%
  gather() %>% ggplot(aes(x = reorder(key, value), y = value)) +
  geom_bar(stat = "identity", fill = "grey40") + # Annotate bar chart with percentage missing
geom_text(aes(label = scales::percent(value), y = -0.01),
  size = 3, position = position_dodge(width = 0.9)) + coord_flip() +
  labs(title = "Soybean: Missing Data by Predictor", x = "",
    y = "") + theme_bw() + theme(axis.text.x = element_blank())
```



Of the 18% of incomplete cases, the predictors `sever`, `seed.tmt`, and `lodging`, and `hail` are missing in almost all of them.

Apart from `Class` and `date`, `leaves` is the only other predictor that's present for all cases.

```
kable(table(Soybean$Class, complete.cases(Soybean)), caption = "NA Values",
  "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

Table 3.4: NA Values

	FALSE	TRUE
2-4-d-injury	16	0
alternarialeaf-spot	0	91
anthracnose	0	44
bacterial-blight	0	20
bacterial-pustule	0	20
brown-spot	0	92
brown-stem-rot	0	44
charcoal-rot	0	20
cyst-nematode	14	0
diaporthe-pod-&-stem-blight	15	0
diaporthe-stem-canker	0	20
downy-mildew	0	20
frog-eye-leaf-spot	0	91
herbicide-injury	8	0
phyllosticta-leaf-spot	0	20
phytophthora-rot	68	20
powdery-mildew	0	20
purple-seed-stain	0	20
rhizoctonia-root-rot	0	20

We should check to see if this is the result of chance, or if there are systematic issues (i.e. the data generating process, measurement challenges, recording errors, data loss, etc.) that could explain this.

```
# Calculate total cases in Soybean set
total_cases <- nrow(Soybean)
```

```
# Tidy dataset, calculating complete cases by predictor
# and predictor cases overall
Soybean %>% mutate(complete_cases = complete.cases(Soybean)) %>%
  group_by(Class) %>% summarize(cases = n(), complete_cases = sum(complete_cases),
  completeness = complete_cases/cases, proportion_allcases = cases/total_cases) %>%

# Display only predictors with missing data
filter(completeness != 1) %>% arrange(desc(proportion_allcases)) %>%

# Attenuate table look
mutate(completeness = scales::percent(completeness), proportion_allcases = scales::percent(proportion_a
  select(Class, cases, complete_cases, completeness, proportion_allcases) %>%
  rename(class = Class) %>% kable(caption = "Complete Cases",
  label = NULL, "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

Table 3.5: Complete Cases

class	cases	complete_cases	completeness	proportion_allcases
phytophthora-rot	88	20	22.7%	12.9%
2-4-d-injury	16	0	0.0%	2.3%
diaporthe-pod-&-stem-blight	15	0	0.0%	2.2%
cyst-nematode	14	0	0.0%	2.0%
herbicide-injury	8	0	0.0%	1.2%

When completeness is examined from the viewpoint of `Class`, it becomes apparent that five classes are responsible for all missing data: `phytophthora-rot`, `2-4-d-injury`, `diaporthe-pod-&-stem-blight`, `cyst-nematode`, and `herbicide-injury`. Diagnosis of the cause of missing data should focus on these classes.

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

There is no simple method for handling missing data AND preserving the global utility of it. Because the classes with missing data have many incomplete cases in the observations, imputation that could be useful for the less empty classes would introduce a lot of error for the more empty ones.

If performing imputation, it would be important to create a new, separate flag variable so that models with high sensitivity can use native values differently than imputed ones (attributing bias to the 'flag') - an approach we have used in logistic regression with some success. It may also be useful to reduce dimensionality by extracting the most variance through PCA or LDA.

Additionally, purging the most empty four or five classes and impute the remainder could be a beneficial approach. Alternatively, we might remove the four emptiest classes, establish which variables are most important using PCA, and then see how those variables are represented by the four fragile classes with high NA counts; if those classes are less impacted, it might be possible to impute across the remaining variables for all classes using flags for the imputed values and preserving a good amount of predictive power.

4 Assignment Four

- Hyndman 7.1
- Hyndman 7.3

Hyndman 7.1

> Consider the `pigs` series – the number of pigs slaughtered in Victoria each month.

a. Use the `ses()` function in R to find the optimal values of α and ℓ_0, and generate forecasts for the next four months.

Forecast Summary

The forecast summary below shows us the optimal values of α and ℓ_0, as estimated by the `ses` function in R.

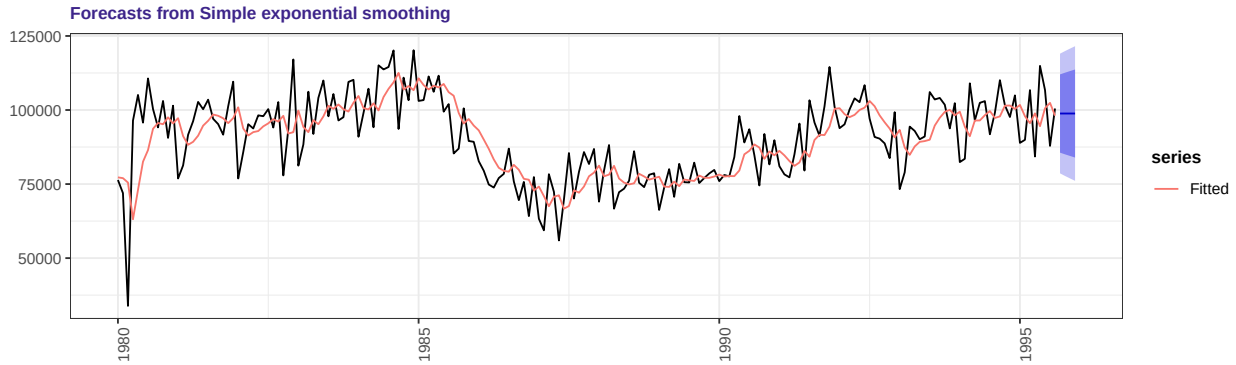
```
pigsdata <- window(pigs, start=1980)
fc <- ses(pigsdata, h=4)
summary(fc)
```

```
FALSE
FALSE Forecast method: Simple exponential smoothing
FALSE
FALSE Model Information:
FALSE Simple exponential smoothing
FALSE
FALSE Call:
FALSE  ses(y = pigsdata, h = 4)
FALSE
FALSE   Smoothing parameters:
FALSE     alpha = 0.2971
FALSE
FALSE   Initial states:
FALSE     l = 77260.0561
FALSE
FALSE   sigma:  10308.58
FALSE
FALSE      AIC     AICc      BIC
FALSE 4462.955 4463.086 4472.665
FALSE
FALSE Error measures:
FALSE                     ME    RMSE      MAE       MPE
FALSE Training set 385.8721 10253.6 7961.383 -0.922652
FALSE                  MAPE      MASE       ACF1
```

```
FALSE Training set 9.274016 0.7966249 0.01282239
FALSE
FALSE Forecasts:
FALSE          Point Forecast    Lo 80    Hi 80    Lo 95
FALSE Sep 1995       98816.41 85605.43 112027.4 78611.97
FALSE Oct 1995       98816.41 85034.52 112598.3 77738.83
FALSE Nov 1995       98816.41 84486.34 113146.5 76900.46
FALSE Dec 1995       98816.41 83958.37 113674.4 76092.99
FALSE             Hi 95
FALSE Sep 1995 119020.8
FALSE Oct 1995 119894.0
FALSE Nov 1995 120732.4
FALSE Dec 1995 121539.8
```

Visualization

```
autoplot(fc) + autolayer(fitted(fc), series = "Fitted") +
  ylab("Pigs") + xlab("Year") + theme_bw() + theme()
```



b. Compute a 95% prediction interval for the first forecast using $\hat{y} \pm 1.96s$ where s is the standard deviation of the residuals. Compare your interval with the interval produced by R.

Formula calculations of the confidence interval were obtained using $\bar{x} \pm 1.96s$ and compared to the calculations from the `ses` function in R.

```
# Formula Calculatioin
lower <- fc$mean[1] - 1.96 * sd(fc$residuals)
upper <- fc$mean[1] + 1.96 * sd(fc$residuals)

# R Calculation
lowerR <- as.numeric(fc$lower[1, "95%"])
upperR <- as.numeric(fc$upper[1, "95%"])

# Table Output
lower <- cbind(`95_perct` = "Lower", Computed = lower, R = lowerR)
upper <- cbind(`95_perct` = "Upper", Computed = upper, R = upperR)
as.data.frame(rbind(lower, upper)) %>% kable("latex") %>%
  kable_styling(latex_options = "hold_position")
```

95_perct	Computed	R
Lower	78679.9672534162	78611.9684577467
Upper	118952.844969765	119020.843765435

Hyndman 7.3

> Modify your function from the previous exercise to return the sum of squared errors rather than the forecast of the next observation. Then use the `optim()` function to find the optimal values of α and ℓ_0. Do you get the same values as the `ses()` function?

Function

The prior question asks us to write a function to implement simple exponential smoothing. As this was a challenge, we made reference to stackexchange while crafting what follows.

```
# Custom SES Function
custom_ses = function(ts, alpha, l0) {
  N = length(ts)
  y_hat = c(l0, 0 * (2:N))
  for (j in 1:(N - 1)) {
    new = alpha * ts[j] + (1 - alpha) * y_hat[j]
    y_hat[j + 1] = new
  }
  return(y_hat)
}

# SSE function
custom_sse = function(fn, ts) {
  alpha <- fn[1]
  l0 <- fn[2]
  N = length(ts)
  fit = custom_ses(ts, alpha, l0)
  res = ts - fit
  out = sum(res^2)/(N - 1)
  return(out)
}
```

Optimization

The `optim` function optimizes the parameters α and ℓ_0 by using the initial values in the `par` argument to maximize output from the vector defined in our custom functions.

```
# `optim` Function
optimize <- optim(par = c(0.1, 10000), fn = custom_sse, ts = pigsdata)

# Optimized Parameters
alpha = optimize$par[1]
l0 = optimize$par[2]
```

```
# SES Model Parameters
ses_alpha <- fc$model$fit$par[1]
ses_l0 <- fc$model$fit$par[2]
```

The calculated differences between the R estimated paremeters and optimized parameters from the custom function were minimal, as is visible in the comparison below below:

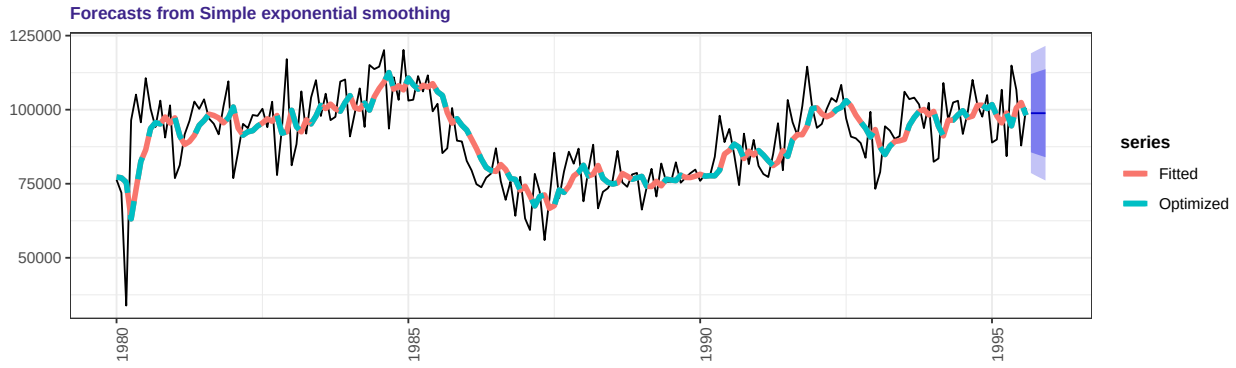
```
# Table Output
lower <- cbind(`SES Function` = "Custom Optimized", alpha = alpha,
  l0 = l0)
upper <- cbind(SES = "R Estimated", alpha = ses_alpha, l0 = ses_l0)
as.data.frame(rbind(lower, upper)) %>% kable("latex") %>%
  kable_styling(latex_options = c("hold_position", "striped"))
```

SES Function	alpha	l0
Custom Optimized	0.297214493242571	77260.9584487885
R Estimated	0.297148833372095	77260.0561458528

Visualization

A timeseries plot of our fitted (estimated) and custom (optimized) values are very similar, such that the difference between the two is difficult to discern in the plot.

```
fit = custom_ses(pigsdata, alpha = alpha, l0 = l0)
ts_fit <- ts(fit, start = 1980, frequency = 12)

autoplot(fc) + autolayer(fitted(fc), series = "Fitted", size = 1.5) +
  autolayer(ts_fit, series = "Optimized", linetype = "dashed",
    size = 1.5) + ylab("Pigs") + xlab("Year") + theme_bw() +
  theme()
```



5 Assignment Five

- Hyndman 7.5
- Hyndman 7.6
- Hyndman 7.10

```
# Textbook Packages
library(fpp2)
library(AppliedPredictiveModeling)
library(mlbench)

# Processing
library(tidyverse)

# Graphing
library(ggplot2)
library(grid)
library(gridExtra)
library(lemon)

# Math
library(caret)
library(forecast)
library(randomForest)
library(seasonal)
library(psych)
library(corrplot)

# Formatting
library(knitr)
library(kableExtra)
library(default)
```

Hyndman 7.5

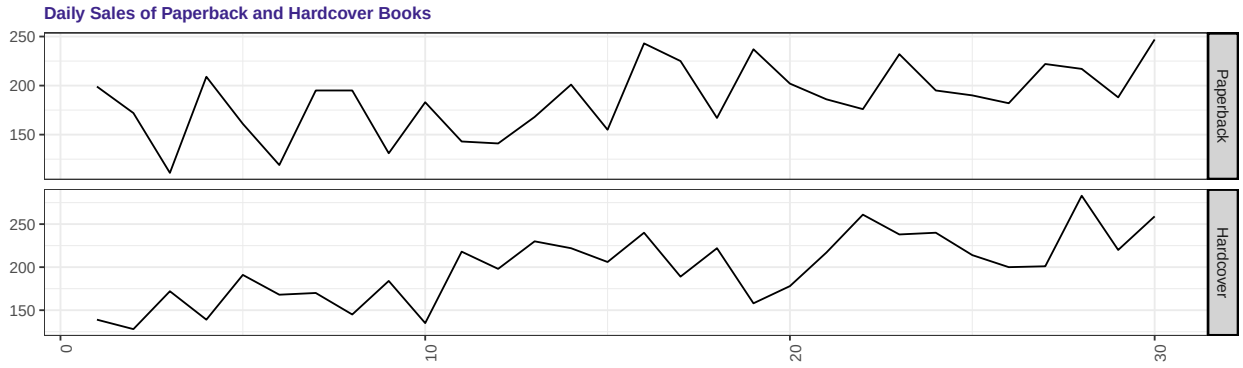
> Data set books contains the daily sales of paperback and hardcover books at the same store. The task is to forecast the next four days' sales for paperback and hardcover books.

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

Timeseries Plot

The `books` data shows an overall positive trend over the 30-day period for paperback and hardcover books. There is heavy fluctuation between days, suggesting seasonality within the time series. While paperbacks and hardcovers both have a similar range of unit sales, peaks and valleys are not well synchronized.

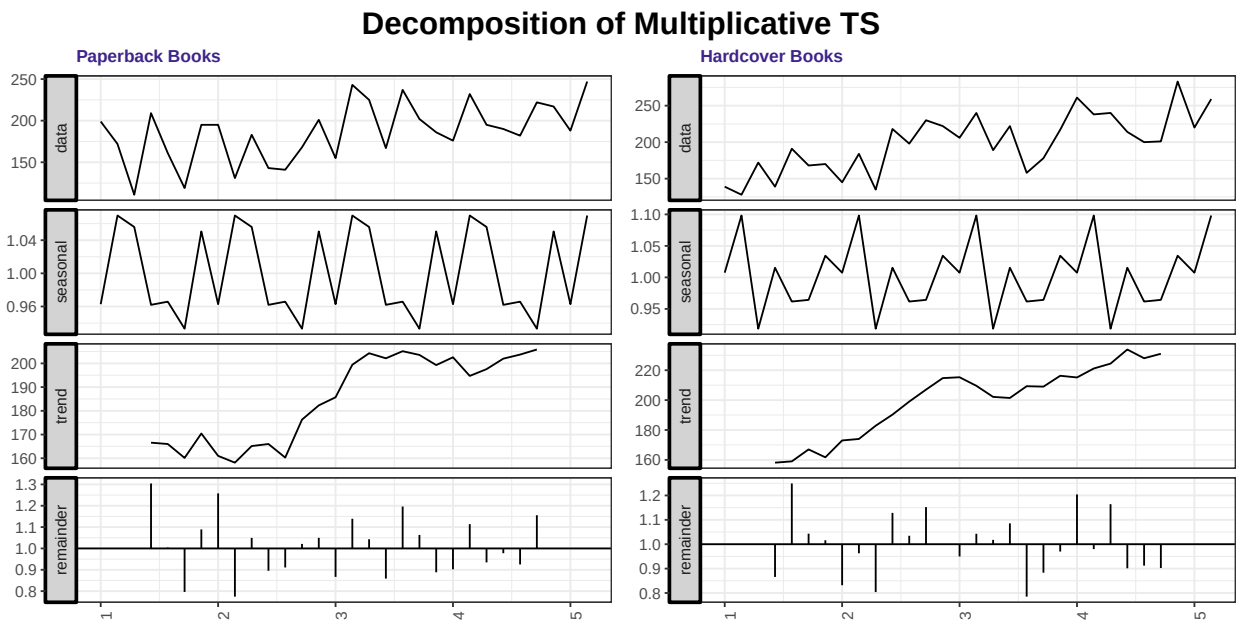
```
autoplot(books, facet = TRUE) + labs(title = "Daily Sales of Paperback and Hardcover Books",
  x = "Days", y = "Sales") + theme_bw() + theme()
```



Decomposed Plot

We can look at the decomposed timeseries to better understand the trend and seasonality on a weekly basis. Sales tend to peak three times per week and increase throughout the entire month. The paperback books trend increases more drastically between weeks two and four, while hardcover books appear to have smoother incline. Hardcover books also show sharper fluctuations in the seasonal component.

```
paperback <- books[, 1]
hardcover <- books[, 2]

pb_plot <- ts(paperback, frequency = 7) %>% decompose(type = "multiplicative") %>%
  autoplot() + labs(title = "Paperback Books", x = "Weeks") +
  theme_bw() + theme()

hc_plot <- ts(hardcover, frequency = 7) %>% decompose(type = "multiplicative") %>%
  autoplot() + labs(title = "Hardcover Books", x = "Weeks") +
  theme_bw() + theme()

grid.arrange(pb_plot, hc_plot, ncol = 2, top = textGrob("Decomposition of Multiplicative TS",
  gp = gpar(fontface = "bold", cex = 1.5)))
```



b. Use the `ses()` function to forecast each series, and plot the forecasts.

Paperback SES

```
pb_ses <- ses(paperback, h = 4)
summary(pb_ses)
```

```
FALSE
FALSE Forecast method: Simple exponential smoothing
FALSE
FALSE Model Information:
FALSE Simple exponential smoothing
FALSE
FALSE Call:
FALSE  ses(y = paperback, h = 4)
FALSE
FALSE   Smoothing parameters:
FALSE     alpha = 0.1685
FALSE
FALSE   Initial states:
FALSE     l = 170.8271
FALSE
FALSE   sigma:  34.8183
FALSE
FALSE      AIC     AICc      BIC
FALSE 318.9747 319.8978 323.1783
FALSE
FALSE Error measures:
FALSE                     ME     RMSE     MAE       MPE
FALSE Training set 7.175981 33.63769 27.8431 0.4736071
FALSE                   MAPE      MASE       ACF1
```

```
FALSE Training set 15.57784 0.7021303 -0.2117522
FALSE
FALSE Forecasts:
FALSE    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
FALSE 31       207.1097 162.4882 251.7311 138.8670 275.3523
FALSE 32       207.1097 161.8589 252.3604 137.9046 276.3147
FALSE 33       207.1097 161.2382 252.9811 136.9554 277.2639
FALSE 34       207.1097 160.6259 253.5935 136.0188 278.2005
```

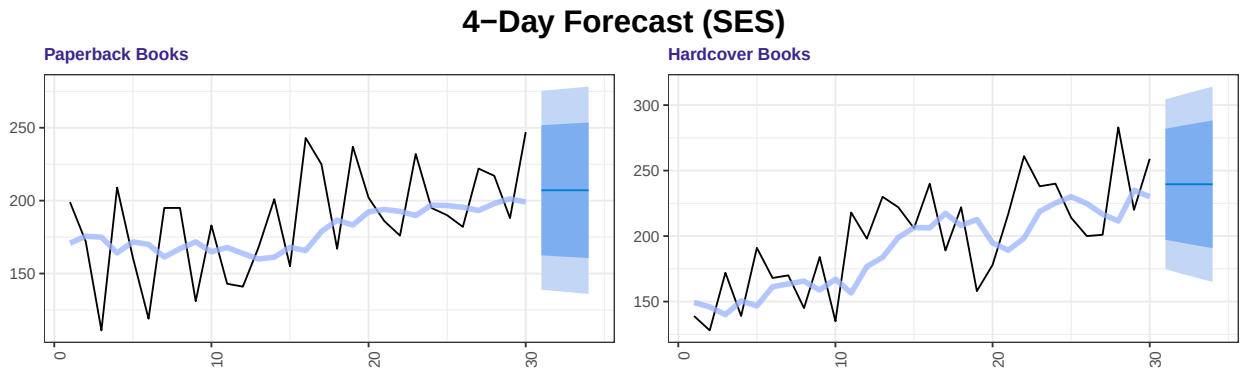
Hardcover SES

```
hc_ses <- ses(hardcover, h = 4)
summary(hc_ses)
```

```
FALSE
FALSE Forecast method: Simple exponential smoothing
FALSE
FALSE Model Information:
FALSE Simple exponential smoothing
FALSE
FALSE Call:
FALSE  ses(y = hardcover, h = 4)
FALSE
FALSE   Smoothing parameters:
FALSE     alpha = 0.3283
FALSE
FALSE   Initial states:
FALSE     l = 149.2861
FALSE
FALSE   sigma:  33.0517
FALSE
FALSE      AIC     AICc      BIC
FALSE 315.8506 316.7737 320.0542
FALSE
FALSE Error measures:
FALSE                     ME     RMSE      MAE      MPE
FALSE Training set 9.166735 31.93101 26.77319 2.636189
FALSE                  MAPE      MASE       ACF1
FALSE Training set 13.39487 0.7987887 -0.1417763
FALSE
FALSE Forecasts:
FALSE    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
FALSE 31       239.5601 197.2026 281.9176 174.7799 304.3403
FALSE 32       239.5601 194.9788 284.1414 171.3788 307.7414
FALSE 33       239.5601 192.8607 286.2595 168.1396 310.9806
FALSE 34       239.5601 190.8347 288.2855 165.0410 314.0792
```

Plot

```r
pb_ses_plot <- autoplot(pb_ses, size = 1, fcol = "#0044cc") +
  autolayer(fitted(pb_ses), series = "Fitted", alpha = 0.75,
    size = 1.5, color = "#99b3ff") + labs(title = "Paperback Books",
  y = "Sales", x = "Days") + theme_bw() + theme()

hc_ses_plot <- autoplot(hc_ses, size = 1, fcol = "#0044cc") +
  autolayer(fitted(hc_ses), series = "Fitted", alpha = 0.75,
    size = 1.5, color = "#99b3ff") + labs(title = "Hardcover Books",
  y = "Sales", x = "Days") + theme_bw() + theme()

grid.arrange(pb_ses_plot, hc_ses_plot, ncol = 2, top = textGrob("4-Day Forecast (SES)",
  gp = gpar(fontface = "bold", cex = 1.5)))
```



As SES may work well when their is no clear trend or seasonal pattern. As it generates the next probable value and uses that for all future forecasting values, however `ses()` may not be the best long-term forecasting model for this particular data.

c. Compute the RMSE values for the training data in each case.

The root mean squared error (RMSE) is calculated below and tells us the spread between the residual errors from our predicted and observed using the SES method. The RMSE for paperback books is slightly higher than hardcover, suggesting better accuracy in the later's predictions.

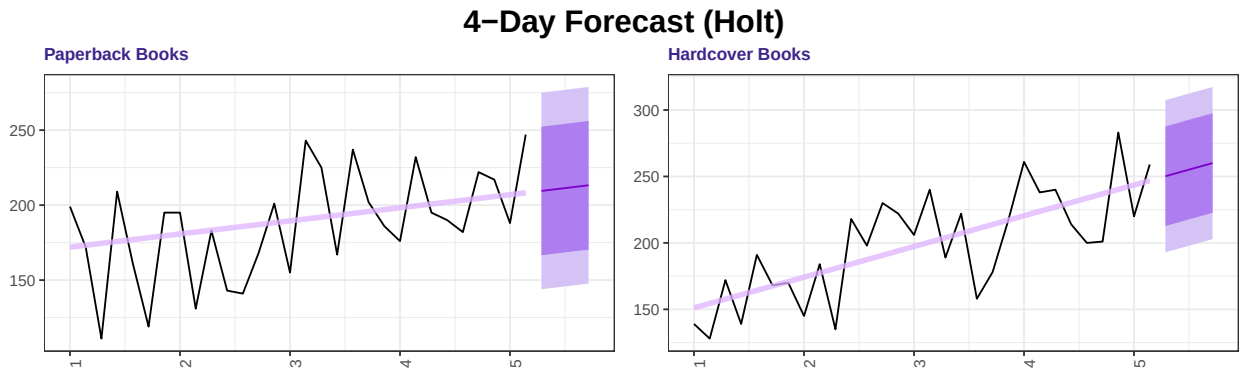
```r
# manual calculation
sqrt(mean(pb_ses$residuals^2))
sqrt(mean(hc_ses$residuals^2))

# caret package
RMSE(fitted(pb_ses), paperback)
RMSE(fitted(hc_ses), hardcover)
```

Paperback RMSE: 33.6376868
Hardcover RMSE: 31.931015

Hyndman 7.6

a. Now apply Holt's linear method to the paperback and hardback series and compute four-day forecasts in each case.

```
hardcover <- ts(books[, 2], start = 1, frequency = 7)
softbound <- ts(books[, 1], start = 1, frequency = 7)

hb <- ses(hardcover, h = 5)
sb <- ses(softbound, h = 5)

acc_soft <- accuracy(sb)
acc_hard <- accuracy(hb)

holt_soft <- holt(softbound, seasonal = "additive", h = 4)
holt_hard <- holt(hardcover, seasonal = "additive", h = 4)

acc_holt_hard <- accuracy(holt_hard)
acc_holt_soft <- accuracy(holt_soft)

pb_holt_plot <- autoplot(holt_soft, size = 1, fcol = "#7733ff") +
  autolayer(fitted(holt_soft), series = "Fitted", alpha = 0.75,
    size = 1.5, color = "#e0b3ff") + labs(title = "Paperback Books",
  y = "Sales", x = "Days") + theme_bw() + theme()

hc_holt_plot <- autoplot(holt_hard, size = 1, fcol = "#7733ff") +
  autolayer(fitted(holt_hard), series = "Fitted", alpha = 0.75,
    size = 1.5, color = "#e0b3ff") + labs(title = "Hardcover Books",
  y = "Sales", x = "Days") + theme_bw() + theme()

grid.arrange(pb_holt_plot, hc_holt_plot, ncol = 2, top = textGrob("4-Day Forecast (Holt)",
  gp = gpar(fontface = "bold", cex = 1.5)))
```



As would be expected, Holt's linear method generates a linear trend forecast.

b. Compare the RMSE measures of Holt's method for the two series to those of simple exponential smoothing in the previous question. Discuss the merits of the two forecasting methods for these data sets.

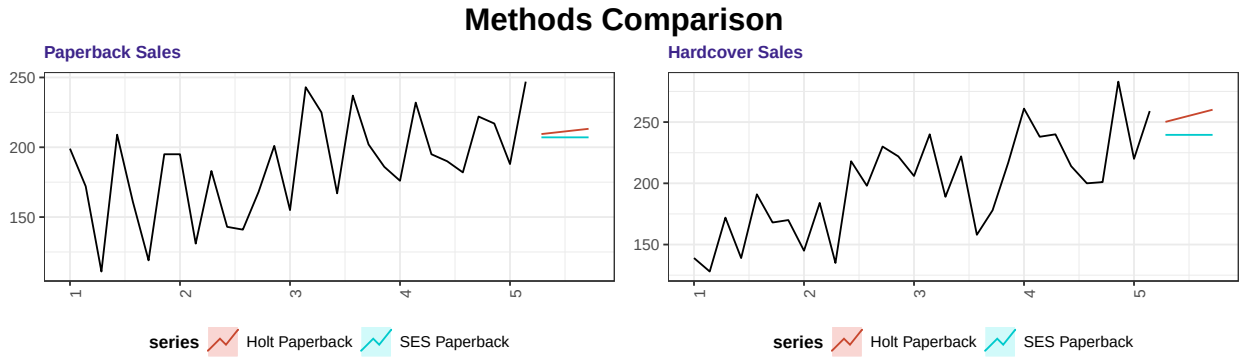
> Remember that Holt's method is using one more parameter than SES.

RMSE Exponential Smoothed Hardcover Sales: 31.931
RMSE Exponential Smoothed Paperback Sales: 33.638
RMSE Holt-Winters Hardcover Sales: 27.194
RMSE Holt-Winters Paperback Sales: 31.137

The Holt-Winter method does indeed reduce the root mean squared errors for both the paperback and hardcover book sales. This would suggest that there is in fact some level of trend acting on the book sales series.

c. Compare the forecasts for the two series using both methods. Which do you think is best?

```
fc_hh <- forecast(holt_hard)
fc_sh <- forecast(holt_soft)

fc_hes <- forecast(hb, h = 4)
fc_ses <- forecast(sb, h = 4)

p1 <- autoplot(softbound) + autolayer(fc_ses, series = "SES Paperback",
  PI = FALSE) + autolayer(fc_sh, series = "Holt Paperback",
  PI = FALSE) + labs(title = "Paperback Sales", x = "Year",
  y = "Daily Sales of Paperback Books") + theme_bw() +
  theme(legend.position = "bottom")


p2 <- autoplot(hardcover) + autolayer(fc_hes, series = "SES Paperback",
  PI = FALSE) + autolayer(fc_hh, series = "Holt Paperback",
  PI = FALSE) + labs(title = "Hardcover Sales", x = "Year",
  y = "Daily Sales of Hardcover Books") + theme_bw() +
  theme(legend.position = "bottom")

grid.arrange(p1, p2, nrow = 1, top = textGrob("Methods Comparison",
  gp = gpar(fontface = "bold", cex = 1.5)))
```



While Holt's method provided better accuracy measures (RMSE), it is a tough choice to pick one model over the other because neither is particularly robust over the long term. the SES is providing a continuous flat prediction which we can see is clearly not how the series has evolved in the past. On the other hand Holt's method does account for the trend but not the seasonality, so at some point it too would fail. Choosing a model from the two should be done based on your end need (ses being more conservative than holt's method) and with the understanding that more than a few observations forward and either of them will likely underperform.

d. Calculate a 95% prediction interval for the first forecast for each series, using the RMSE values and assuming normal errors. Compare your intervals with those produced using ses and holt.

```
pi_ses_s <- 1.96 * acc_soft[2]
pi_ses_h <- 1.96 * acc_hard[2]

pi_hl_s <- 1.96 * acc_holt_soft[2]
pi_hl_s <- 1.96 * acc_holt_hard[2]

l_interval_ses_s <- round(fc_ses$mean[1] - pi_ses_s, 3)
l_interval_ses_h <- round(fc_hes$mean[1] - pi_ses_h, 3)

l_interval_hl_s <- round(fc_sh$mean[1] - pi_hl_s, 3)
l_interval_hl_h <- round(fc_hh$mean[1] - pi_hl_s, 3)

u_interval_ses_s <- round(fc_ses$mean[1] + pi_ses_s, 3)
u_interval_ses_h <- round(fc_hes$mean[1] + pi_ses_h, 3)

u_interval_hl_s <- round(fc_sh$mean[1] + pi_hl_s, 3)
u_interval_hl_h <- round(fc_hh$mean[1] + pi_hl_s, 3)
```

	Model Lower	Model Upper	Calculated Lower	Calculated Upper
SES Softbound	138.867	275.352	141.18	273.04
SES Hardbound	174.78	304.34	176.975	302.145
Holt Softbound	143.913	275.021	156.167	262.766
Holt Hardbound	192.922	307.426	196.874	303.473

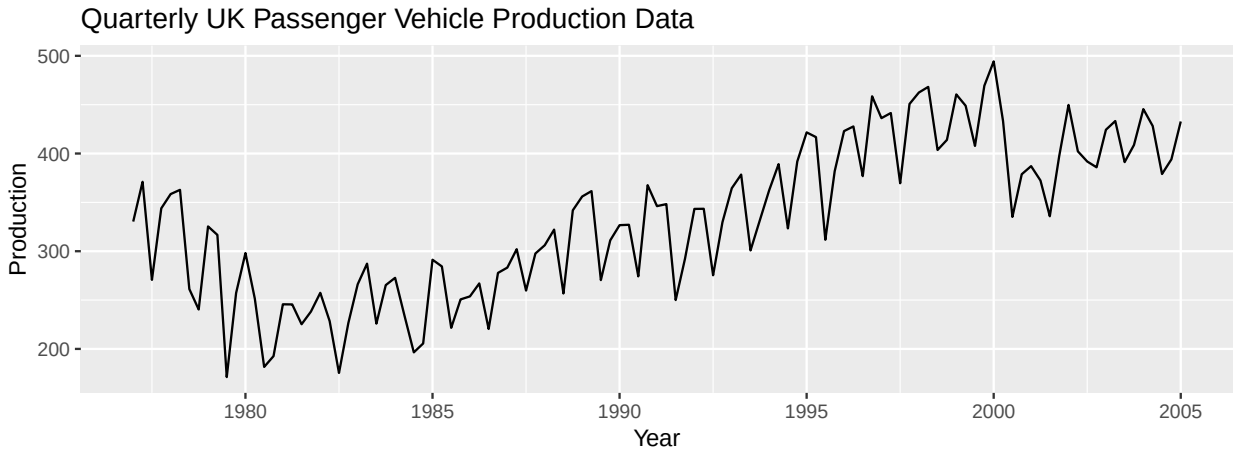
The manually calculated and library-generated prediction intervals are not exactly the same, but they are close. Manual intervals are a bit less consistent with the library computed interval values using the Holt's linear method than they are with exponentially smoothed (SES) version. Additionally, as SES entails a flat forecast with higher RMSE, its interval is slightly wider than that generated by Holt's linear method.

Hyndman 7.10

For this exercise use data set ukcars, the quarterly UK passenger vehicle production data from 1977Q1–2005Q1.

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

```
ukcdata <- window(ukcars, start = 1977, end = 2005)
autoplot(ukcdata) + labs(title = "Quarterly UK Passenger Vehicle Production Data",
  y = "Production", x = "Year")
```



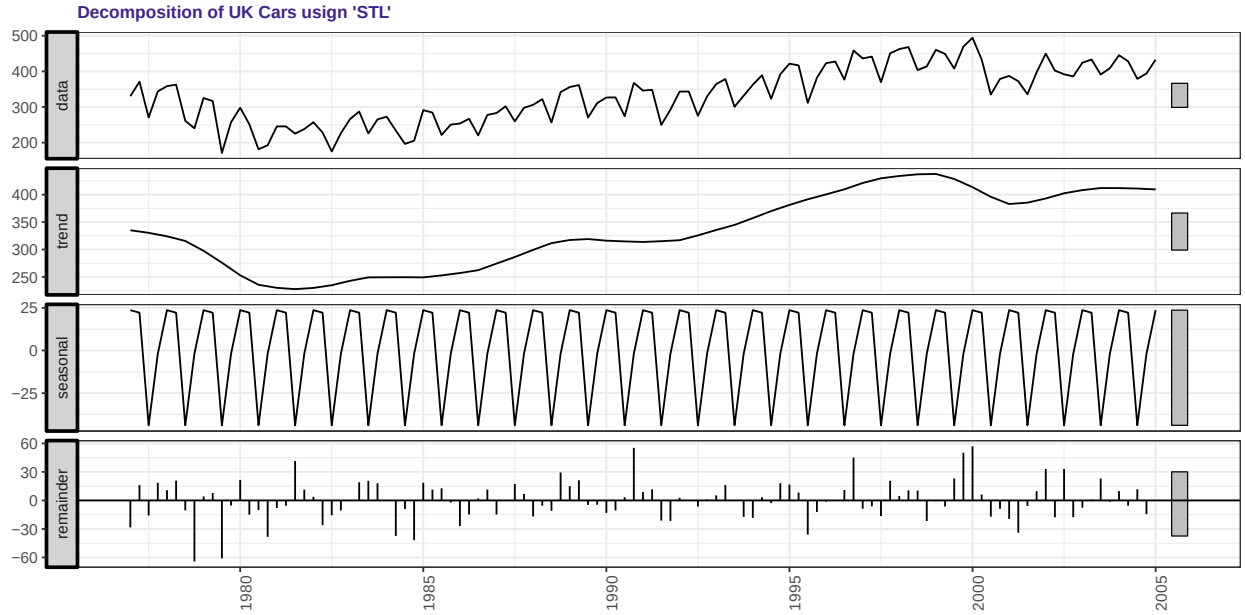
There appears to be a general upward trend in production from 1982-Q3 to 2000-Q1. The overall time series shows fluxuations between years suggesting an underlying seasonality component. There also is some less structured periodicity and random-like variation.

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

STL Decomposition

Through decomposition, we can confirm the overall increasing trend described in part (a). There are consistant seasonal fluxations which can we can futher evaluate using a seasonal plot.

```
ukc_stl <- stl(ukcars, t.window = 13, s.window = "periodic",
  robust = TRUE)

ukc_stl %>% autoplot() + ggtitle("Decomposition of UK Cars usign 'STL' ") +
  theme_bw() + theme()
```

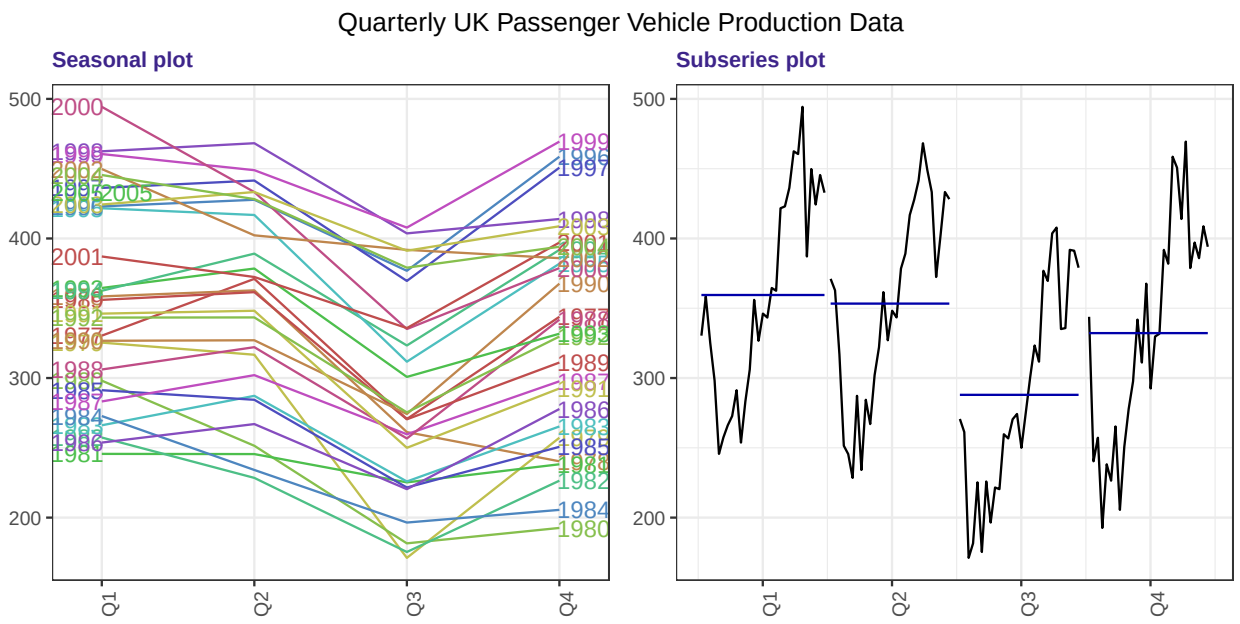


Seasonal Plots

With the seasonal plot, we can see that our overal seasonality component increases between Q1-Q2, decreases Q2-Q3, and increases Q3-Q4. The intensity of the fluxuations vary by year. There are a few exceptions to this observation, for example, 2000 shows a two quarter decrease between Q1-Q3.

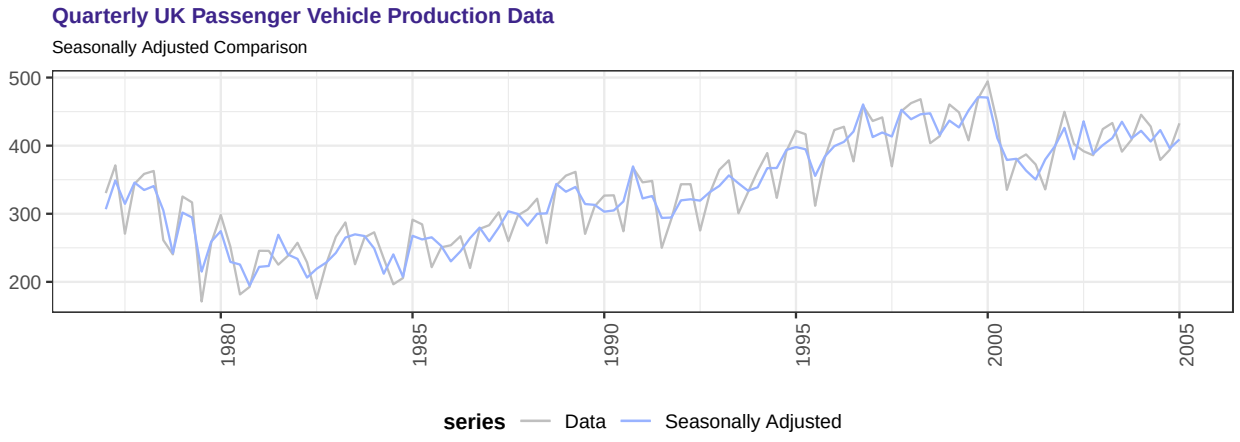
The subseries plot further confirms our seasonal pattern and shows that the smallest pattern changes occur between Q1-Q2. The decreases and increases between Q2-Q4 are much more distingushable.

```
ukc_s_plot <- ggseasonplot(ukcdata, year.labels = TRUE, year.labels.left = T,
  col = rainbow(12, s = 0.6, v = 0.75)) + labs(title = "Seasonal plot",
  y = "Production") + theme_bw() + theme()

ukc_ss_plot <- ggsubseriesplot(ukcdata) + labs(title = "Subseries plot",
  y = "Production") + theme_bw() + theme()

grid.arrange(ukc_s_plot, ukc_ss_plot, ncol = 2, top = "Quarterly UK Passenger Vehicle Production Data")
```



We can capture the seasonal component observed in the data and control for the changes overtime using a seasonal adjustment. We compared the seasonally adjusted data to the observed data for `ukcars` below:

```
ukc_seas <- seasadj(ukc_stl)

autoplot(ukcdata, series = "Data") + autolayer(ukc_seas,
  series = "Seasonally Adjusted") + labs(title = "Quarterly UK Passenger Vehicle Production Data",
  subtitle = "Seasonally Adjusted Comparison", x = "Year",
  y = "Production") + scale_colour_manual(values = c("#bfbfbf",
  "#99b3ff"), breaks = c("Data", "Seasonally Adjusted",
  "Trend")) + theme_bw() + theme(legend.position = "bottom")
```



c. Forecast the next two years of the series using an additive damped trend method applied to the seasonally adjusted data. (This can be done in one step using stlf() with arguments etsmodel="AAN", damped=TRUE.)

As the data is seasonally adjusted, we use an AAN model (error = A, trend = A, and seasonal = N). Damped models are often better at forecasting long-run values.

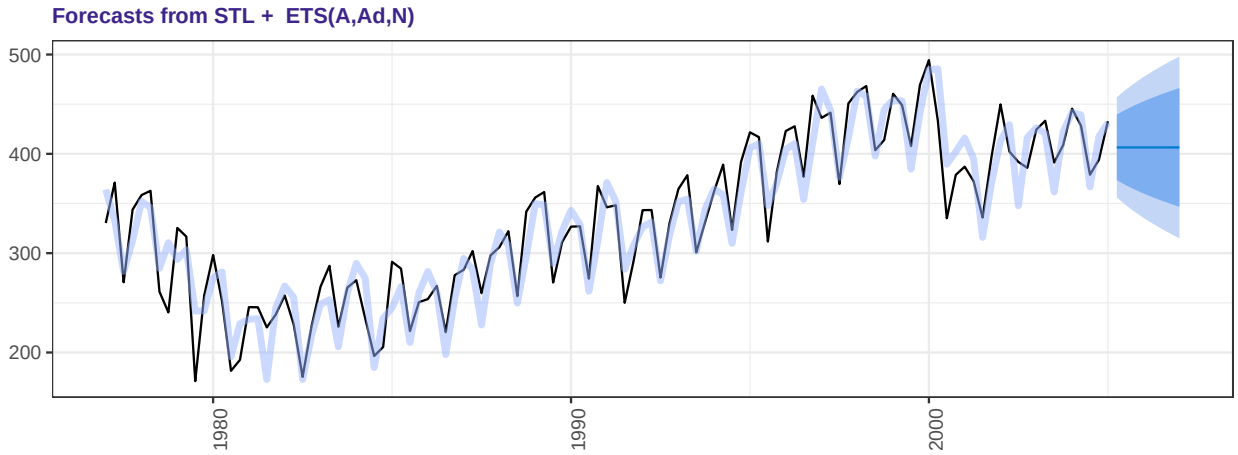
Forecast

```
ukc_stlf1 <- stlf(ukcdata, s.window = "periodic", etsmodel = "AAN",
  damped = TRUE, h = 8)
rmse_holt_d <- accuracy(ukc_stlf1)[2]

kable(forecast(ukc_stlf1), "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
2005 Q2	406.4679	373.4863	439.4495	356.0269	456.9089
2005 Q3	406.4673	368.4766	444.4580	348.3655	464.5691
2005 Q4	406.4668	364.0533	448.8804	341.6009	471.3327
2006 Q1	406.4664	360.0485	452.8842	335.4764	477.4563
2006 Q2	406.4660	356.3620	456.5700	329.8385	483.0934
2006 Q3	406.4656	352.9279	460.0033	324.5867	488.3444
2006 Q4	406.4652	349.7005	463.2299	319.6511	493.2794
2007 Q1	406.4649	346.6465	466.2834	314.9805	497.9493

Plot

```
autoplot(ukc_stlf1, size = 1.25, fcol = "#0044cc") + autolayer(fitted(ukc_stlf1),
  series = "Fitted", alpha = 0.5, size = 1.5, color = "#99b3ff") +
  labs(y = "Production", x = "Years") + theme_bw() + theme()
```



d. Forecast the next two years of the series using Holt's linear method applied to the seasonally adjusted data (as before but with damped=FALSE).

We again use an AAN model (error = A, trend = A, seasonality = N) - this time without damping, so point forecasts are a little bit higher.

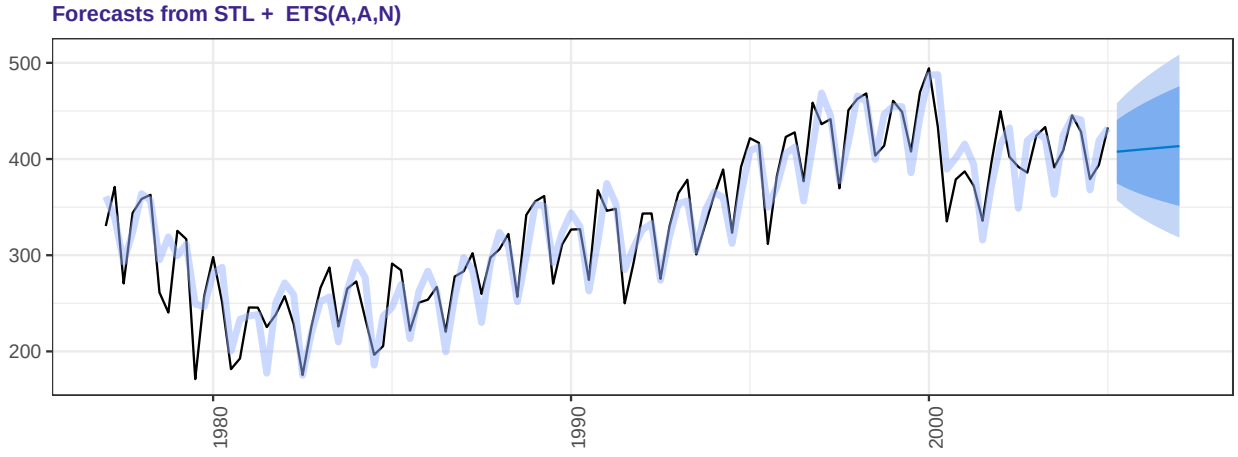
Forecast

```
ukc_stlf2 <- stlf(ukcdata, s.window = "periodic", etsmodel = "AAN",
  damped = FALSE, h = 8)
rmse_holt_u <- accuracy(ukc_stlf2)[2]

kable(forecast(ukc_stlf2), "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
2005 Q2	407.6326	374.6715	440.5938	357.2229	458.0424
2005 Q3	408.4644	369.9399	446.9889	349.5463	467.3825
2005 Q4	409.2962	365.9146	452.6778	342.9497	475.6426
2006 Q1	410.1280	362.3794	457.8765	337.1029	483.1531
2006 Q2	410.9598	359.2102	462.7093	331.8157	490.1039
2006 Q3	411.7915	356.3277	467.2554	326.9669	496.6162
2006 Q4	412.6233	353.6776	471.5690	322.4737	502.7730
2007 Q1	413.4551	351.2210	475.6892	318.2763	508.6339

Plot

```
autoplot(ukc_stlf2, size = 1.25, fcol = "#0044cc") + autolayer(fitted(ukc_stlf2),
  series = "Fitted", alpha = 0.5, size = 1.5, color = "#99b3ff") +
  labs(y = "Production", x = "Years") + theme_bw() + theme()
```



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

ETS automatically chooses the best model by minimizing AIC, which results in a error = A, trend = N, seasonal = A model.

Seasonal Model

```
ukc_ets <- ets(ukcdata)
summary(ukc_ets)
```

```
FALSE ETS(A,N,A)
FALSE
FALSE Call:
FALSE  ets(y = ukcdata)
FALSE
FALSE   Smoothing parameters:
FALSE     alpha = 0.6199
FALSE     gamma = 1e-04
FALSE
FALSE   Initial states:
FALSE     l = 314.2568
FALSE     s = -1.7579 -44.9601 21.1956 25.5223
FALSE
FALSE   sigma:  25.9302
FALSE
FALSE      AIC     AICc      BIC
FALSE 1277.752 1278.819 1296.844
FALSE
FALSE Training set error measures:
FALSE                     ME     RMSE      MAE        MPE
FALSE Training set 1.313884 25.23244 20.17907 -0.1570979
FALSE                  MAPE      MASE       ACF1
FALSE Training set 6.629003 0.6576259 0.02573334
```

```
ukc_ets_fc <- ukc_ets %>% forecast(h = 8)
rmse_ets <- accuracy(ukc_ets)[2]
```

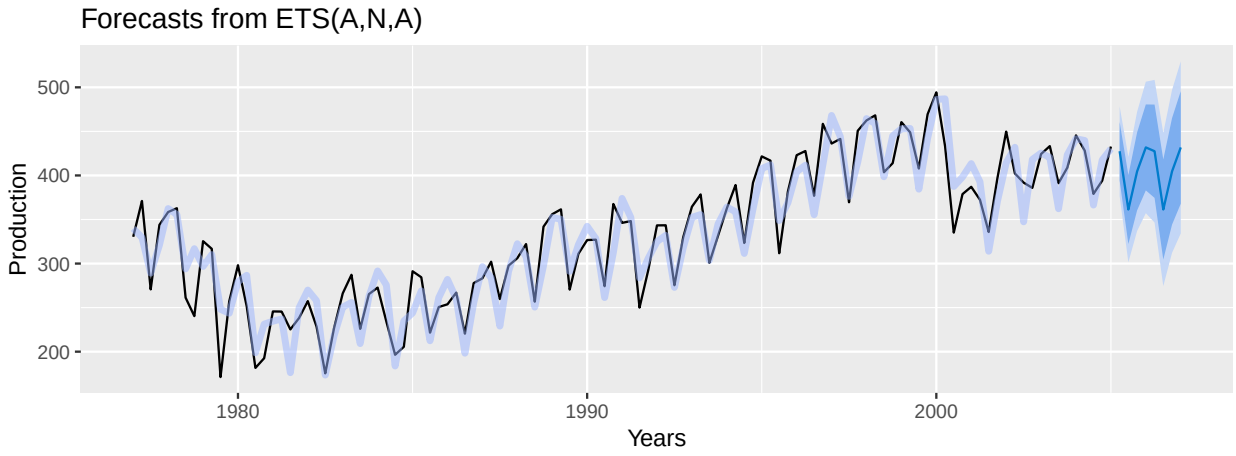
Forecast

```
kable(ukc_ets_fc, "latex") %>% kable_styling(latex_options = c("hold_position",
  "striped"))
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
2005 Q2	427.4885	394.2576	460.7195	376.6662	478.3109
2005 Q3	361.3329	322.2353	400.4305	301.5383	421.1275
2005 Q4	404.5358	360.3437	448.7280	336.9497	472.1219
2006 Q1	431.8154	383.0568	480.5741	357.2455	506.3854
2006 Q2	427.4885	374.5571	480.4200	346.5369	508.4401
2006 Q3	361.3329	304.5345	418.1313	274.4672	448.1986
2006 Q4	404.5358	344.1174	464.9542	312.1338	496.9378
2007 Q1	431.8154	367.9809	495.6500	334.1890	529.4419

Plot

```
autoplot(ukc_ets_fc, size = 1.25, fcol = "#0044cc") + autolayer(fitted(ukc_ets),
  series = "Fitted", alpha = 0.5, size = 1.5, color = "#99b3ff") +
  labs(y = "Production", x = "Years")
```



f. Compare the RMSE of the ETS model with the RMSE of the models you obtained using STL decompositions. Which gives the better in-sample fits?

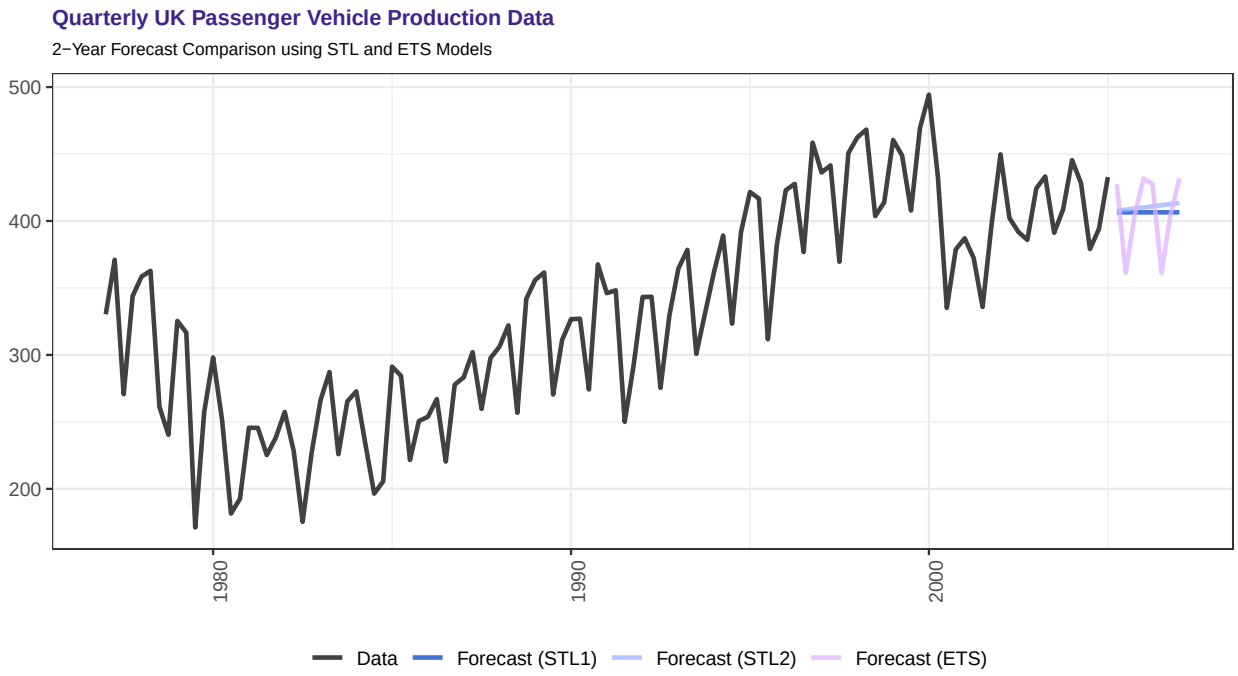
Our STL decomponsition, using additive damped trend method with no seasonal components (A, A_d, N), produced the best in-sample fit. This model provided us with the lowest RMSE, suggesting the highest degree of accuracy between our predicted and observed data. We should note, however, there is a very small variation in RMSE between all STL and ETS models.

- **STL:** 25.2604123
- **HOLT:** 25.1598573
- **ETS:** 25.2324409

g. Compare the forecasts from the three approaches? Which seems most reasonable?

The forecasts from the ETS (A, N, A) model appears most reasonable and fits our data best. This is not surprising as the `ets` function automatically selects the methodology based on the provided data. The forecasts using STL were both linear, whereas the ETS model was able to predict seasonal pattern changes across our 2-year (8-quarter) prediction.

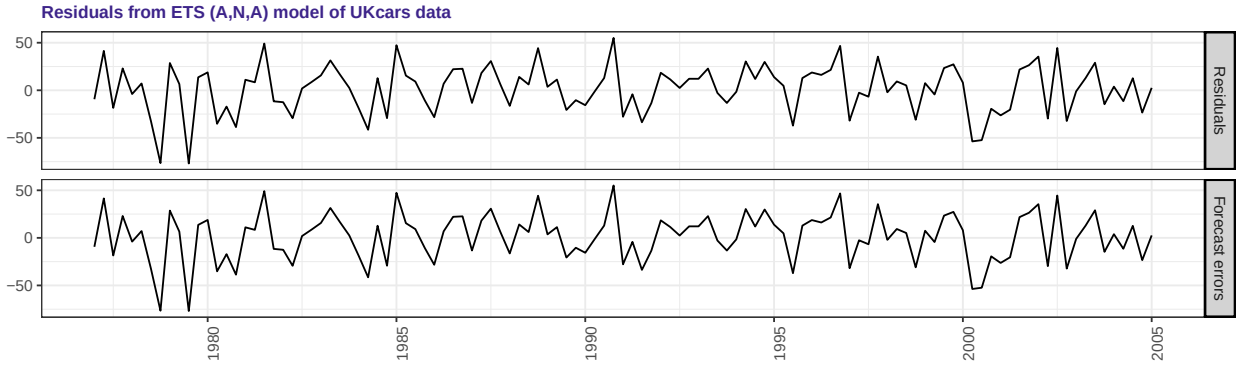
```
ukc_series <- cbind(Data = ukcdata, `Forecast (STL1)` = ukc_stlf1[["mean"]],
  `Forecast (STL2)` = ukc_stlf2[["mean"]], `Forecast (ETS)` = ukc_ets_fc[["mean"]])

autoplot(ukc_series, size = 1, alpha = 0.75) + scale_color_manual(name = "",
  values = c("#000000", "#0044cc", "#99b3ff", "#e0b3ff")) +
  labs(title = "Quarterly UK Passenger Vehicle Production Data",
    subtitle = "2-Year Forecast Comparison using STL and ETS Models",
    x = "Years", y = "Production") + theme_bw() + theme(legend.position = "bottom")
```



h. Check the residuals of your preferred model.

The plots below contains our residuals data from the preferred ETS model. We observed constant variance and normally distribution of our residuals, suggesting our model has adequately captured the information from the `ukcars` data

```
cbind(Residuals = residuals(ukc_ets), `Forecast errors` = residuals(ukc_ets,
  type = "innovation")) %>% autoplot(facet = TRUE) + labs(title = "Residuals from ETS (A,N,A) model of U
  x = "Year") + theme_bw() + theme()
```



6 Assignment Six

- Hyndman 8.1
- Hyndman 8.2
- Hyndman 8.6
- Hyndman 8.8

Hyndman 8.1

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

ACF plots (also known as correlograms) are used to graph the autocorrelation coeffients of lag values in a timeseries. In white noise series, we expect autocorrelation to be close to zero. This component is observed in the ACF plots by examining the dashed blue lines.

Using this knowledge, we can determine that all three series are white noise because they all show just a small spread between these blue lines and hover around 0. Only two lag lines come close to exceeding this boundary, but none are significant or escape the denoted critical value used to determine autocorrelation.

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 blue line boundary on an ACF plot is calculated using the formula $\pm 2/\sqrt{T-d}$, where T represents the length of time in a series and d the number of differences. Thus as T grows larger, the denominator also gets larger, making the resulting quotient smaller.

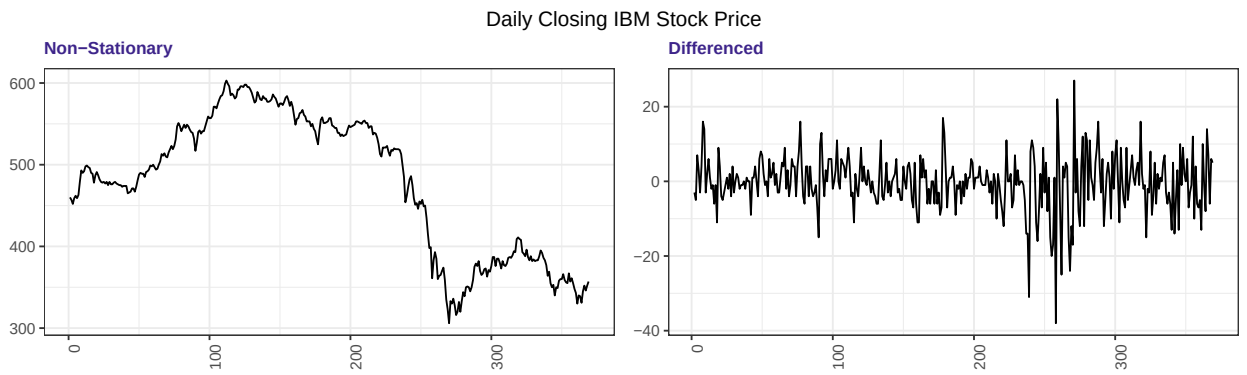
This means that the areas defined by the boundaries $\pm 2/\sqrt{T-d}$ decrease as the number values in series T increases.

Hyndman 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.

a. Time Series Plot

```
p1 <- autoplot(ibmclose) + labs(title = "Non-Stationary",
  y = "Price", x = "Days") + theme_bw() + theme()
p2 <- autoplot(diff(ibmclose)) + labs(title = "Differenced",
  y = "Price", x = "Days") + theme_bw() + theme()

grid.arrange(p1, p2, ncol = 2, top = "Daily Closing IBM Stock Price")
```



Daily closing prices from the `ibmclose` dataset are non-stationary and show an overall trend component. Seasonality is harder to distinguish at this frequency. There may be a cyclical component where pattern changes occur over a non-specific period of time, however, there are not enough observations to determine this.

We can compare this plot to a differenced timeseries, in which the stock prices are stationary and do not depend on the time at which the series is observed. Stationary models are required for autoregression modeling and analysis.

b. ACF Plot

```
p1 <- ggAcf(ibmclose) + theme_bw() + theme()
p2 <- ggAcf(diff(ibmclose)) + theme_bw() + theme()

grid.arrange(p1, p2, ncol = 2, top = "Daily Closing IBM Stock Price")
```

In our `ibmclose` series, we can identify large, positive, and slowly decreasing lags. This further indicates that our data is non-stationary, trended, and autocorrelated. Seasonal fluxuations do not appear present within our lags.

Autoregressive models, however, require stationary data. The `ibmclose` data could be differenced to meet this requirement. The `diff(ibmclose)` series has no signs of autocorrelation outside the 95% limit (± 0.1) as shown by the dashed blue lines. This plot appears similiar to a white noise series.

c. PACF Plot

```
p1 <- ggPacf(ibmclose) + ggtitle("`ibmclose Series ACF") +
  theme_bw() + theme()
p2 <- ggPacf(diff(ibmclose)) + ggtitle("`ibmclose` ACF Differenced Series") +
```

```
  theme_bw() + theme()

grid.arrange(p1, p2, ncol = 2, top = "Daily Closing IBM Stock Price")
```

Lastly, the PACF plot is used to look at partial autocorrelation. This plot analyzes the correlation of residuals after removing the effect explained by previous lags. Our Pacf plot of the `ibmclose` series a sharp cut off after the first lag.

Pacf is very beneficial in determining order of moving averages component of a non-seasonal ARIMA model. In the differentiated data, there are two spikes outside the 95% limit. These can be ignored though as they do not occur in our initial lags and fall just outside the ± 0.1 limit. The differentiated Pacf plot further suggests a white-noise ARIMA model of the stationary data.

Hyndman 8.6

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

a. Use the following R code to generate data from an AR(1) model with $\phi_1 = 0.6$ and $\sigma^2 = 1$. The process starts with $y_1 = 0$.

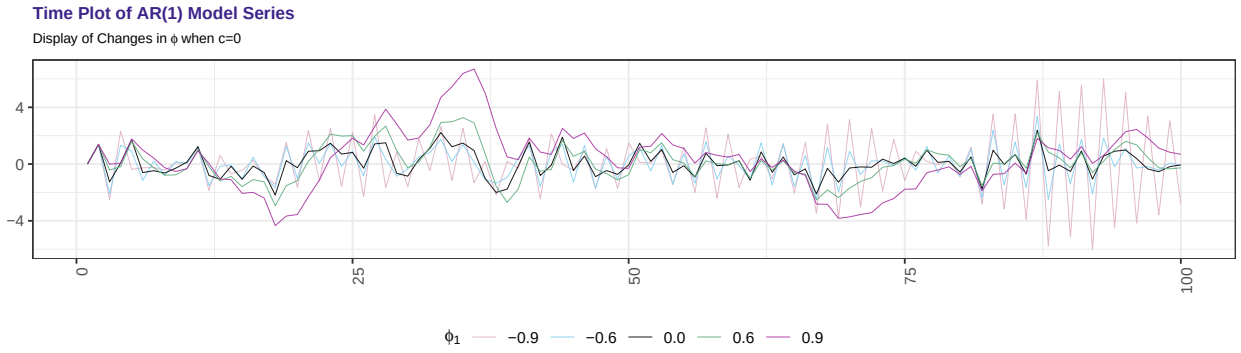
```
# Sample Code
y <- ts(numeric(100))
e <- rnorm(100)
for (i in 2:100) y[i] <- 0.6 * y[i - 1] + e[i]

# Function

AR1 <- function(phi1, sigma2, c) {
  # add warnings for expected constraints for AR1
  # parameters values
  if (phi1 < -1)
    warning("'phi1' should be > -1 for AR(1) models")
  if (phi1 > 1)
    warning("'phi1' should be < 1 for AR(1) models")
  if (phi1 < 1 & phi1 > -1)
    message("Model: AR(1) ")

  # set random number generation (RNG) state
  set.seed(5)
```

```r
  # generate time series 1:100
  y <- ts(numeric(100))

  # RNG of epsilon with constant variance (sigma^2)
  e <- rnorm(100, sd = sigma2)

  # apply formula to all but the first ts observation so
  # that y1=0
  for (i in 2:100) {
    y[i] <- c + phi1 * y[i - 1] + e[i]
  }
  return(y)
}

# Set parameters per instruction
phi1 = 0.6
sigma2 = 1
c = 0

# Generate Data
AR1(phi1, sigma2, c)
```

```
FALSE Time Series:
FALSE Start = 1
FALSE End = 100
FALSE Frequency = 1
FALSE   [1]  0.00000000  1.38435934 -0.42487626 -0.18478299
FALSE   [5]  1.60057108  0.35743467 -0.25770559 -0.78999466
FALSE   [9] -0.75977043 -0.31775404  1.03697792 -0.17959270
FALSE  [13] -1.18814822 -0.87042329 -1.59401401 -1.09539455
FALSE  [17] -1.25454982 -2.93669665 -1.52120074 -1.17207585
FALSE  [21]  0.19726644  1.06022926  2.10409946  1.96922076
FALSE  [25]  2.00054139  0.90684298  1.96269486  2.67639075
FALSE  [29]  0.94875235 -0.28354403  0.14578862  1.19716734
FALSE  [33]  2.93376098  2.97736022  3.26563792  2.91095659
FALSE  [37]  0.73704131 -1.55824796 -2.69713465 -1.76088891
FALSE  [41]  0.49352702 -0.50630697 -0.37836310  1.66865009
FALSE  [45]  0.54462112  0.88899603 -0.35361089 -0.67241111
FALSE  [49] -1.12777515 -0.74587625  1.01572281  0.79715979
FALSE  [53]  1.50031873  0.30835641  0.07281319 -0.88126517
FALSE  [57]  0.22454569  0.02211835 -0.05081992  0.20278334
FALSE  [61] -1.01491280  0.24588274 -0.43084077  0.23785708
FALSE  [65] -0.61734369 -0.71179248 -2.52940461 -1.81934505
FALSE  [69] -2.36399047 -1.69806039 -1.22293356 -0.95937432
FALSE  [73] -0.22859614 -0.10478984  0.35065739  0.05504595
FALSE  [77]  1.00651297  0.72499792  0.62417244 -0.18838160
FALSE  [81]  0.38538720 -1.51107017  0.06888699  0.01724932
FALSE  [85]  0.68603407 -0.29868916  2.20801915  0.85137948
FALSE  [89]  0.43505513 -0.26080698  0.76956295 -0.60067340
FALSE  [93]  0.19662982  1.01870848  1.60117077  1.34431055
FALSE  [97]  0.46000252 -0.26418774 -0.34106824 -0.26394059
```

b. Produce a time plot for the series. How does the plot change as you change ϕ_1?

```
nam <- expression(phi[1])

AR1_phi1 <- suppressMessages(cbind(`-0.9` = AR1(-0.9, 1,
  0), `-0.6` = AR1(-0.6, 1, 0), `0.0` = AR1(0, 1, 0), `0.6` = AR1(0.6,
  1, 0), `0.9` = AR1(0.9, 1, 0)))

pallete = c("#dfb1c5", "#86d1f1", "#000000", "#5ead7f", "#a931a1")
# palette <- brewer.pal(n = 5 , name = 'RdYlGn')
palette <- c("#D7191C", "#C51B8A", "#555555", "#756BB1",
  "#2C7BB6")
autoplot(AR1_phi1, size = 0.25, alpha = 0.9) + scale_color_manual(name = nam,
  values = pallete) + labs(title = "Time Plot of AR(1) Model Series",
  subtitle = expression("Display of Changes in" ~ phi ~
    "when c=0"), y = "Value") + theme_bw() + theme(legend.position = "bottom")
```



- The AR(1) Model at $\phi = 0$ should be the equivalent of white noise.
- The y_t is expected to oscillate around the mean when $\phi < 0$.
- We would expect y_t to resemble a random walk when $\phi = 1$ and $c = 0$.

As expected we observe these patterns in the time series plots above. The variation and pattern observed in y changes as the ϕ parameter changes with the negative values of ϕ occilating around the mean, and the positive numbers becoming more like a random walk as ϕ approaches 1.

c. Write your own code to generate data from an MA(1) model with $\theta_1 = 0.6$ and $sigma^2 = 1$.

```
# Function
MA1 <- function(theta1, sigma2, c) {
  # add warnings for expected constraints for MA1
  # parameters values
  if (theta1 < 0)
    warning("'theta' should be > 0 for MA(1) models")
  if (theta1 > 1)
    warning("'theta' should be < 1 for MA(1) models")
```

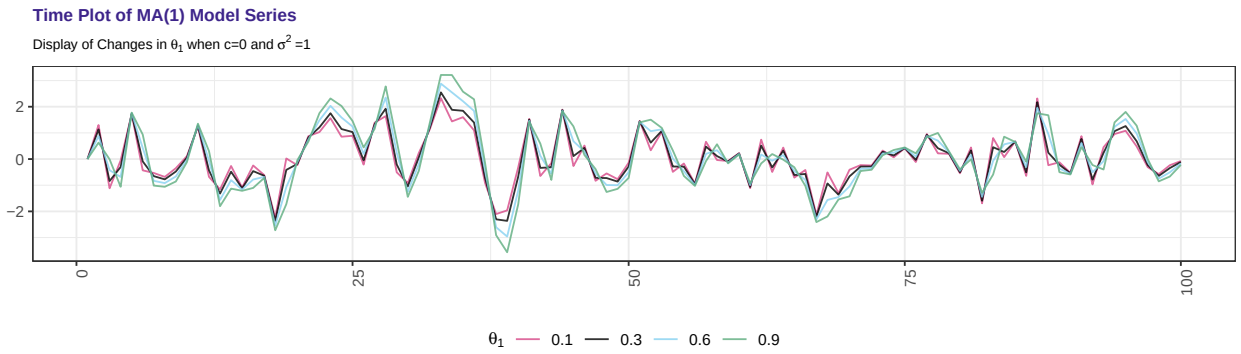
```
  if (theta1 < 1 & theta1 > 0)
    message("Model: MA(1)")

  # set random number generation (RNG) state
  set.seed(5)

  # generate time series 1:100
  y <- ts(numeric(100))

  # RNG of epsilon with constant variance (sigma^2)
  e <- rnorm(100, sd = sigma2)

  # apply formula to all but the first ts observation so
  # that y1=0
  for (i in 2:100) {
    y[i] <- c + theta1 * e[i - 1] + e[i]
  }
  return(y)
}

# Set parameters per instruction
theta1 = 0.6
sigma2 = 1
c = 0

# Generate Data
MA1(theta1, sigma2, c)
```

```
FALSE Time Series:
FALSE Start = 1
FALSE End = 100
FALSE Frequency = 1
FALSE   [1]  0.00000000  0.87984606 -0.42487626 -0.68315235
FALSE   [5]  1.75352653  0.42395654 -0.83391117 -0.91867114
FALSE   [9] -0.66699642 -0.03335596  1.31049528 -0.06520125
FALSE  [13] -1.56146027 -0.80576992 -1.16628065 -0.78204216
FALSE  [17] -0.68070478 -2.54235462 -1.06956280 -0.11486505
FALSE  [21]  0.74489870  1.48217656  2.03308354  1.58753823
FALSE  [25]  1.24306558  0.19792351  1.24249996  2.34992727
FALSE  [29]  0.24218220 -1.24704470 -0.19576223  1.29924319
FALSE  [33]  2.88127707  2.54637998  2.20948397  1.83910690
FALSE  [37] -0.43858835 -2.60619233 -2.96246952 -1.19991965
FALSE  [41]  1.46449549  0.12761304 -0.55603283  1.85092060
FALSE  [45]  0.68083183  0.28828200 -0.54967449 -0.99244968
FALSE  [49] -1.00047523 -0.50380825  1.42172187  1.06567524
FALSE  [53]  1.13465852  0.02137888 -0.46730155 -0.99227348
FALSE  [57]  0.19833295  0.33937381 -0.13165637  0.19482074
FALSE  [61] -0.99661763  0.17288074 -0.06547217  0.14933929
FALSE  [65] -0.46224101 -0.79742103 -2.30716088 -1.56309975
FALSE  [69] -1.45340481 -1.04309618 -0.37189699 -0.34807258
FALSE  [73]  0.21165994  0.24058491  0.43295200  0.09277030
FALSE  [77]  0.88027631  0.70518138  0.26182778 -0.44938085
FALSE  [81]  0.16068512 -1.44325279 -0.06985240  0.56123459
```

```
FALSE  [85]  0.66123475 -0.30489892  1.96104688  0.95890758
FALSE  [89] -0.35983176 -0.56730359  0.61294310 -0.50678289
FALSE  [93] -0.08041284  1.23495090  1.53038403  0.97757550
FALSE  [97] -0.11641896 -0.74813954 -0.50666914 -0.16883301
```

d. Produce a time plot for the series. How does the plot change as you change θ_1?

```
nam <- expression(theta[1])

MA1_theta1 <- cbind(`0.1` = suppressMessages(MA1(0.1, 1,
  0)), `0.3` = suppressMessages(MA1(0.3, 1, 0)), `0.6` = suppressMessages(MA1(0.6,
  1, 0)), `0.9` = suppressMessages(MA1(0.9, 1, 0)))

pallete = c("#d74887", "#000000", "#86d1f1", "#5ead7f")

autoplot(suppressMessages(MA1_theta1), size = 0.5, alpha = 0.8) +
  scale_color_manual(name = nam, values = pallete) + labs(title = "Time Plot of MA(1) Model Series",
  subtitle = expression("Display of Changes in" ~ theta[1] ~
    "when c=0 and" ~ sigma^2 ~ "=1"), y = "Value") +
  theme_bw() + theme(legend.position = "bottom")
```



- When θ_1 increases, we observe higher peaks and troughs in the y_t value.
- In a MA(1) model, we expect the weight of the moving average to be the most constant when θ_1 nears the value of 1.
- Conversely, the most recent observations have more influence on y_t than past observations when $\theta_1 < 1$.

e. Generate data from an ARMA(1,1) model with $\phi_1 = 0.6$, $\theta_1 = 0.6$, and $\sigma^2 = 1$.

```
# Reference ARMA(1,1) Equation:
# http://feldman.faculty.pstat.ucsb.edu/174-03/lectures/l7.pdf

# Function
ARMA11 <- function(phi1, theta1, sigma2, c) {
  # add warnings for expected constraints for ARMA(1,1)
  # parameters values
  if (phi1 + theta1 == 0)
```

```
    warning("'phi1 + theta1' should be != 0 for ARMA(1,1) models ")
  if (phi1 + theta1 != 0)
    message("Model: ARMA(1,1) ")
  # set random number generation (RNG) state
  set.seed(5)
  # generate time series 1:100
  y <- ts(numeric(100))
  # RNG of epsilon with constant variance (sigma^2)
  e <- rnorm(100, sd = sigma2)
  # apply formula to all but the first ts observation so
  # that y1=0
  for (t in 2:100) {
    y[t] <- phi1 * y[t - 1] + e[t] + theta1 * e[t - 1]
  }
  return(y)
}

# Set parameters per instruction
phi1 = 0.6
theta1 = 0.6
sigma2 = 1
c = 0

# Generate Data
e <- ARMA11(phi1, theta1, sigma2, c)
e
```

```
FALSE Time Series:
FALSE Start = 1
FALSE End = 100
FALSE Frequency = 1
FALSE   [1]  0.0000000000  0.8798460550  0.1030313765
FALSE   [4] -0.6213335253  1.3807264174  1.2523923926
FALSE   [7] -0.0824757385 -0.9681565867 -1.2478903744
FALSE  [10] -0.7820901808  0.8412411703  0.4395434538
FALSE  [13] -1.2977342005 -1.5844104364 -2.1169269154
FALSE  [16] -2.0521983137 -1.9120237673 -3.6895688773
FALSE  [19] -3.2833041265 -2.0848475291 -0.5060098161
FALSE  [22]  1.1785706714  2.7402259426  3.2316737972
FALSE  [25]  3.1820698623  2.1071654268  2.5067992194
FALSE  [28]  3.8540068025  2.5545862835  0.2857070734
FALSE  [31] -0.0243379816  1.2846404017  3.6520613133
FALSE  [34]  4.7376167699  5.0520540320  4.8703393236
FALSE  [37]  2.4836152476 -1.1160231776 -3.6320834222
FALSE  [40] -3.3791697028 -0.5630063277 -0.2101907567
FALSE  [43] -0.6821472829  1.4416322330  1.5458111717
FALSE  [46]  1.2157687011  0.1797867268 -0.8845776470
FALSE  [49] -1.5312218200 -1.4225413395  0.5681970658
FALSE  [52]  1.4065934747  1.9786146046  1.2085476466
FALSE  [55]  0.2578270332 -0.8375772590 -0.3042134086
FALSE  [58]  0.1568457636 -0.0375489122  0.1722913893
FALSE  [61] -0.8932427934 -0.3630649348 -0.2833111260
FALSE  [64] -0.0206473879 -0.4746294400 -1.0821986927
```

```
FALSE  [67] -2.9564800984 -3.3369878127 -3.4555974986
FALSE  [70] -3.1164546743 -2.2417697913 -1.6931344528
FALSE  [73] -0.8042207310 -0.2419475248  0.2877834804
FALSE  [76]  0.2654403854  1.0395405377  1.3289057009
FALSE  [79]  1.0591711977  0.1861218637  0.2723582413
FALSE  [82] -1.2798378496 -0.8377551086  0.0585815204
FALSE  [85]  0.6963836639  0.1129312785  2.0288056504
FALSE  [88]  2.1761909659  0.9458828156  0.0002260989
FALSE  [91]  0.6130787600 -0.1389356349 -0.1637742176
FALSE  [94]  1.1366863741  2.2123958591  2.3050130133
FALSE  [97]  1.2665888468  0.0118137699 -0.4995808814
FALSE [100] -0.4685815348
```

f. Generate data from an AR(2) model with $\phi_1 = -0.8$, $\phi_2 = 0.3$, and $\sigma^2 = 1$. (Note that these parameters will give a non-stationary series.)

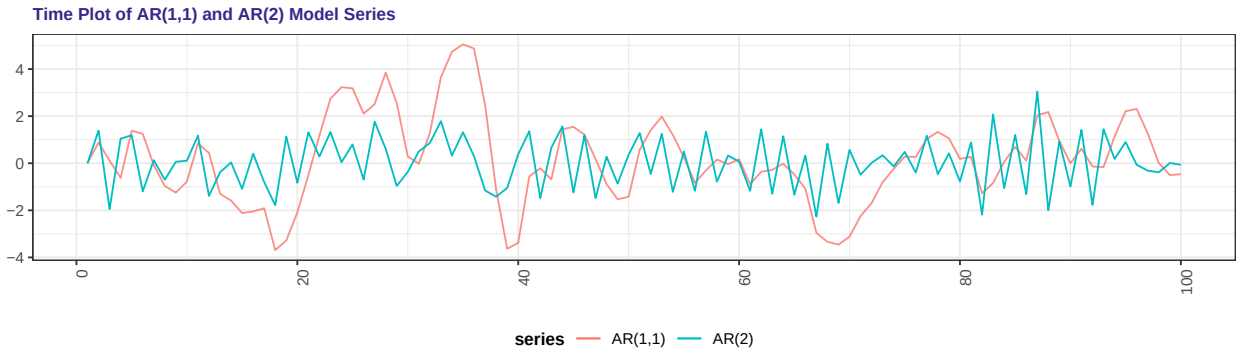
```r
# Function
AR2 <- function(phi1, phi2, sigma2, c) {
  # add warnings for expected constraints for AR2
  # parameters values
  if (phi2 < -1)
    warning("Warning: 'phi2' should be > -1 for AR(2) model. ")
  if (phi2 > 1)
    warning("Warning: 'phi2' should be < 1 for AR(2) model. ")
  if (phi1 + phi2 > 1)
    warning("Warning: 'phi1+phi2' should be < 1 for AR(2) model. ")
  if (phi2 - phi1 > 1)
    warning("Warning: 'phi2-phi1' should be < 1 for AR(2) model. ")
  if (phi2 > -1 & phi1 < 1 & phi1 + phi2 < 1 & phi2 - phi1 <
    1)
    message("Model: AR(2). ")
  # set random number generation (RNG) state
  set.seed(5)
  # generate time series 1:100
  y <- ts(numeric(100))
  # RNG of epsilon with constant variance (sigma^2)
  e <- rnorm(100, sd = sigma2)
  # apply formula to all but the first ts observation so
  # that y1=0
  for (i in 2:100) {
    y[i] <- c + phi1 * y[i - 1] + phi2 * y[i - 1] + e[i]
  }
  return(y)
}

# Set parameters per instruction
phi1 = -0.8
phi2 = 0.3
sigma2 = 1
c = 0

# Generate Data
```

```
f <- AR2(phi1, phi2, sigma2, c)
f
```

```
FALSE Time Series:
FALSE Start = 1
FALSE End = 100
FALSE Frequency = 1
FALSE   [1]  0.000000000  1.384359343 -1.947671534
FALSE   [4]  1.043978534  1.189451606 -1.197633784
FALSE   [7]  0.126650507 -0.698696566  0.063574648
FALSE  [10]  0.106320901  1.174469893 -1.389014401
FALSE  [13] -0.385885399  0.035408344 -1.089464212
FALSE  [16]  0.405745965 -0.800186077 -1.783873721
FALSE  [19]  1.132754117 -0.825732465  1.313378178
FALSE  [22]  0.285180305  1.325371751  0.044075214
FALSE  [25]  0.796971323 -0.691967510  1.764572828
FALSE  [28]  0.616487414 -0.965325801 -0.370132539
FALSE  [31]  0.500981308  0.859203514  1.785858815
FALSE  [34]  0.324174232  1.317134671  0.293006497
FALSE  [37] -1.156035894 -1.422454791 -1.050958477
FALSE  [40]  0.382871112  1.358624813 -1.481735588
FALSE  [43]  0.666288874  1.562523518 -1.237830700
FALSE  [46]  1.181138712 -1.477577868  0.278544358
FALSE  [49] -0.863600665  0.362589177  1.281953975
FALSE  [52] -0.453250890  1.248648306 -1.216158986
FALSE  [55]  0.495878838 -1.172892505  1.339751051
FALSE  [58] -0.782484596  0.327151370  0.069699609
FALSE  [61] -1.171432608  1.440546727 -1.298643782
FALSE  [64]  1.145683430 -1.332899646  0.325063552
FALSE  [67] -2.264860897  0.830728167 -1.687747526
FALSE  [70]  0.564207653 -0.486201147  0.017486388
FALSE  [73]  0.338285258 -0.136774786  0.481918683
FALSE  [76] -0.396307818  1.171639302 -0.464729508
FALSE  [79]  0.421538445 -0.773654293  0.885243311
FALSE  [82] -2.184924149  2.067991172 -1.058078459
FALSE  [85]  1.204723705 -1.312671457  3.043568375
FALSE  [88] -1.995216200  0.921835543 -0.982757828
FALSE  [91]  1.417426049 -1.771124196  1.442595964
FALSE  [94]  0.179432603  0.900229382 -0.066506604
FALSE  [97] -0.313330512 -0.383523994  0.009206404
FALSE [100] -0.063902852
```

g. Graph the latter two series and compare them.

```
autoplot(suppressMessages(e), series = "AR(1,1)", size = 0.5,
  alpha = 0.8) + autolayer(suppressMessages(f), series = "AR(2)") +
  labs(title = "Time Plot of AR(1,1) and AR(2) Model Series",
    y = "Value") + theme_bw() + theme(legend.position = "bottom")
```



Assumptions:
* When $|\theta_1|$ nears 0 in an ARMA(1,1) model, the data follows a pattern similiar to an AR(1) model.
* When $|\phi_1|$ nears zero, it behaves similarly to a MA(1) model.

Our ARMA(1,1) model from 8.6(e). has parameters of $\phi_1 = 0.6$ and $\theta_1 = 0.6$, thus we do not expect explict AR(1) or MA(1) patterns.

Our AR(2) series shows some evidence of trend. This is likely due to the parameters of the AR(2) model from 8.6(f) being set to generate non-staionary data.

Conversely, the parameters in the ARMA(1, 1) model from 8.6(e) generate stationary.

Hyndman 8.8

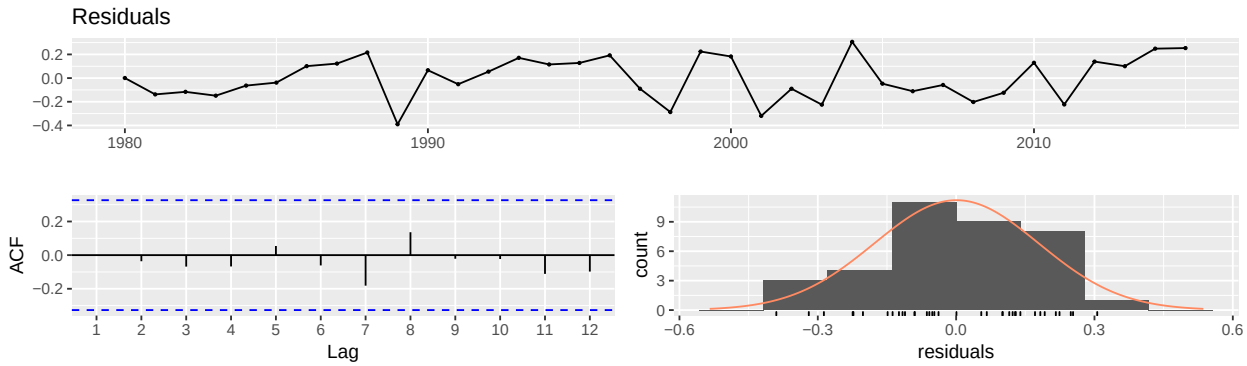
Consider `austa`, the total international visitors to Australia (in millions) for the period 1980-2015.

a. 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.

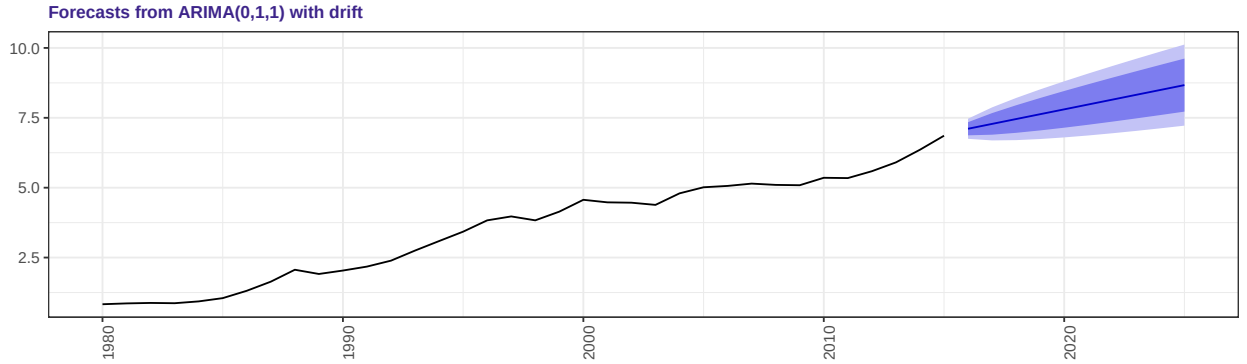
```
data(austa)
(fit <- auto.arima(austa))
```

```
FALSE Series: austa
FALSE ARIMA(0,1,1) with drift
FALSE
FALSE Coefficients:
FALSE          ma1   drift
FALSE       0.3006  0.1735
FALSE s.e.  0.1647  0.0390
FALSE
FALSE sigma^2 estimated as 0.03376:  log likelihood=10.62
FALSE AIC=-15.24   AICc=-14.46   BIC=-10.57
```

```
residuals(fit) %>% checkresiduals()
```




```
fit %>% forecast(h = 10) %>% autoplot() + theme_bw() + theme()
```



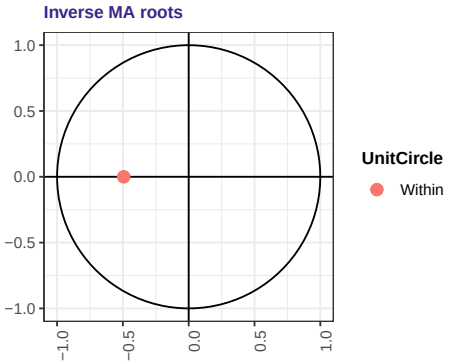
The results of `auto.arima()` suggest single-differencing and applying an MA(1) model with the equation $y_t = c + y_{t-1} - .3006\epsilon_{t-1}\epsilon_t$

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.

```
(fit_ma <- Arima(austa, order = c(0, 1, 1), include.mean = FALSE))
```

```
FALSE Series: austa
FALSE ARIMA(0,1,1)
FALSE
FALSE Coefficients:
FALSE          ma1
FALSE       0.4936
FALSE s.e.  0.1265
FALSE
FALSE sigma^2 estimated as 0.04833:  log likelihood=3.73
FALSE AIC=-3.45   AICc=-3.08   BIC=-0.34
```

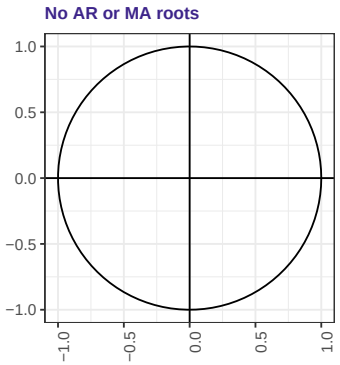
```
fit_ma %>% autoplot() + theme_bw() + theme()
```




```
(fit_noma <- Arima(austa, order = c(0, 1, 0), include.mean = FALSE))
```

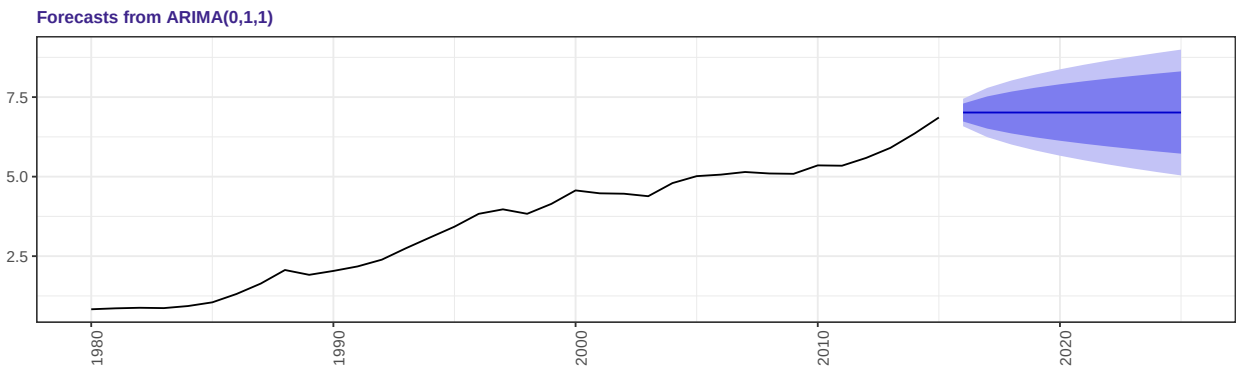
```
FALSE Series: austa
FALSE ARIMA(0,1,0)
FALSE
FALSE sigma^2 estimated as 0.06423:  log likelihood=-1.62
FALSE AIC=5.24   AICc=5.36   BIC=6.8
```

```
fit_noma %>% autoplot() + theme_bw() + theme()
```



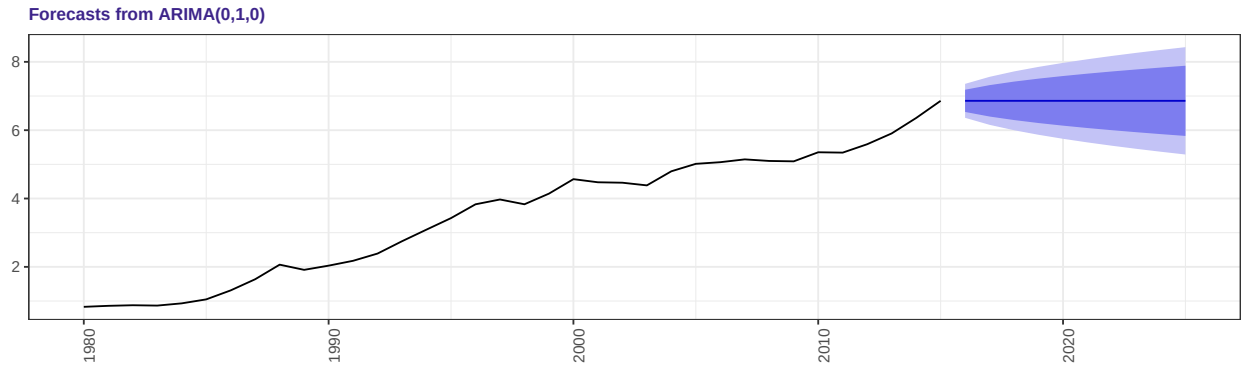
With MA

```
forecast(fit_ma, h = 10) %>% autoplot() + theme_bw() + theme()
```



Without MA

```
forecast(fit_noma, h = 10) %>% autoplot() + theme_bw() +
  theme()
```



Comparing

```
nm <- forecast(fit_noma, h = 10)
accuracy(nm)
```

```
FALSE                      ME      RMSE       MAE      MPE
FALSE Training set 0.1674969 0.2498996 0.1981155 5.471861
FALSE                    MAPE      MASE      ACF1
FALSE Training set 6.479997 0.9723354 0.2583422
```

```
m <- forecast(fit_ma, h = 10)
accuracy(m)
```

```
FALSE                      ME      RMSE       MAE      MPE
FALSE Training set 0.1150344 0.2136411 0.1714887 3.817629
FALSE                    MAPE      MASE       ACF1
FALSE Training set 5.649995 0.8416532 -0.1892256
```

```
difference <- m$mean[1] - nm$mean[1]
```

Mean of MA: 7.015837 (the forecast value for the MA)
Mean of Non-MA: 6.858953 (the forecast value for the Non-MA)
Mean Differences: 0.156884

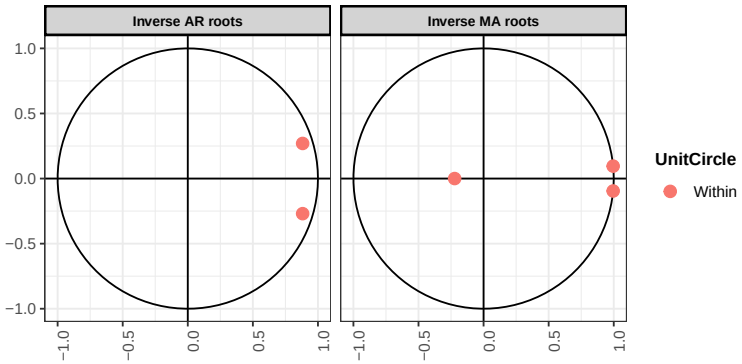
The two models are quite similar: their forecasts differ by .16, with approximately the same confidence windows (shifted given the different means). With a better RMSE, the (0,1,1) model may be prefereable.

(c.) Plot forecasts from an ARIMA(2,1,3) model with drift.

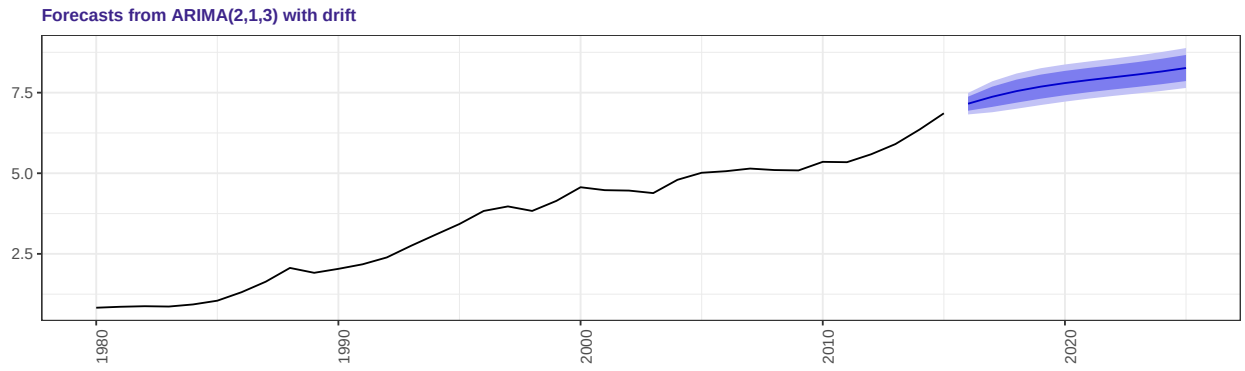
```
(fit_arima <- Arima(austa, order = c(2, 1, 3), include.drift = TRUE))
```

```
FALSE Series: austa
FALSE ARIMA(2,1,3) with drift
FALSE
FALSE Coefficients:
FALSE          ar1      ar2      ma1     ma2     ma3   drift
FALSE       1.7648  -0.8513  -1.7684  0.5571  0.2224  0.1725
FALSE s.e.  0.1136   0.1182   0.2433  0.4351  0.2150  0.0051
FALSE
FALSE sigma^2 estimated as 0.0276:  log likelihood=13.72
FALSE AIC=-13.44   AICc=-9.29   BIC=-2.55
```

```
fit_arima %>% autoplot() + theme_bw() + theme()
```




```
fit_arima %>% forecast(h = 10) %>% autoplot() + theme_bw() +
  theme()
```




```
accuracy(forecast(fit_arima))
```

```
FALSE                      ME      RMSE       MAE       MPE
FALSE Training set -0.004129818 0.1491068 0.114857 -1.203567
FALSE                    MAPE      MASE        ACF1
FALSE Training set 4.326425 0.5637093 -0.01736817
```

Visually this model seems to improve the forecsts. It takes into account the upwasrd trend missing in the (0,1,0) and (0,1,1) forecasts; it offers a much tighter confidence window throughout the prediction range; and, lastly, its RMSE of .14 suggests a better forecast.

> Remove the constant and see what happens

```
(fit_arima <- Arima(austa, order = c(2, 1, 3), include.constant = FALSE))
fit_arima %>% autoplot()
```

In the current state, the coefficients for the AR component breed non-stationarity, such that the model throws an error.

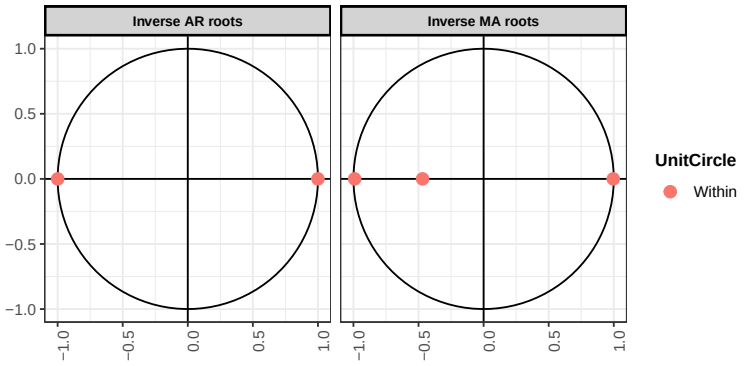
```
Error in stats::arima(x = x, order = order, seasonal = seasonal, include.mean = include.mean,
: non-stationary AR part from CSS
```

This can be corrected by switching to the traditional “Maximum Likelihood” method or “CSS” method.

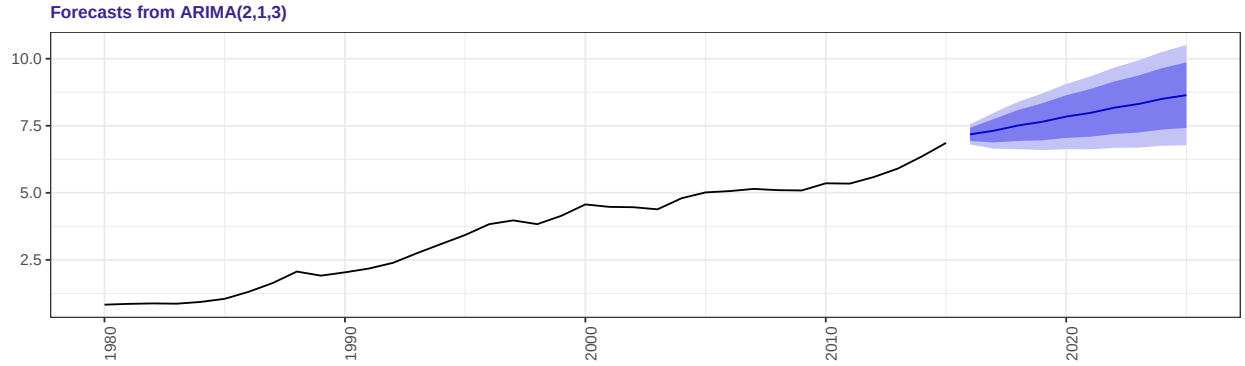
```
(fit_arima <- Arima(austa, order = c(2, 1, 3), include.constant = FALSE,
  method = "ML"))
```

```
FALSE Series: austa
FALSE ARIMA(2,1,3)
FALSE
FALSE Coefficients:
FALSE          ar1     ar2     ma1      ma2      ma3
FALSE       0.0004  0.9996  0.4633  -0.9893  -0.4625
FALSE s.e.  0.0031  0.0031  0.2283   0.0329   0.2261
FALSE
FALSE sigma^2 estimated as 0.03568:  log likelihood=9.24
FALSE AIC=-6.48   AICc=-3.48   BIC=2.85
```

```
fit_arima %>% autoplot() + theme_bw() + theme()
```




```
fit_arima %>% forecast(h = 10) %>% autoplot() + theme_bw() +
  theme()
```




```
accuracy(forecast(fit_arima))
```

```
FALSE                      ME      RMSE       MAE      MPE
FALSE Training set 0.03972287 0.1724412 0.1402048 1.685829
FALSE                    MAPE      MASE        ACF1
FALSE Training set 4.592996 0.6881143 -0.07513543
```

This approach leads to a much wider confidence interval and unit roots moved to the extremes of the acceptable stationary range (-1,0) and (1,0) for both the MA and AR components (suggesting only marginal stationarity).

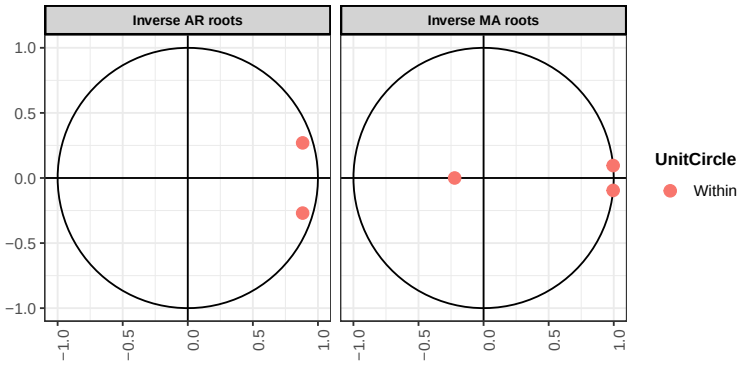
Including the constant term does seem to improve model performance and yield a more compact prediction interval.

In order to compare apples to apples, we also used the 'ML' method on the model when we add the drift.

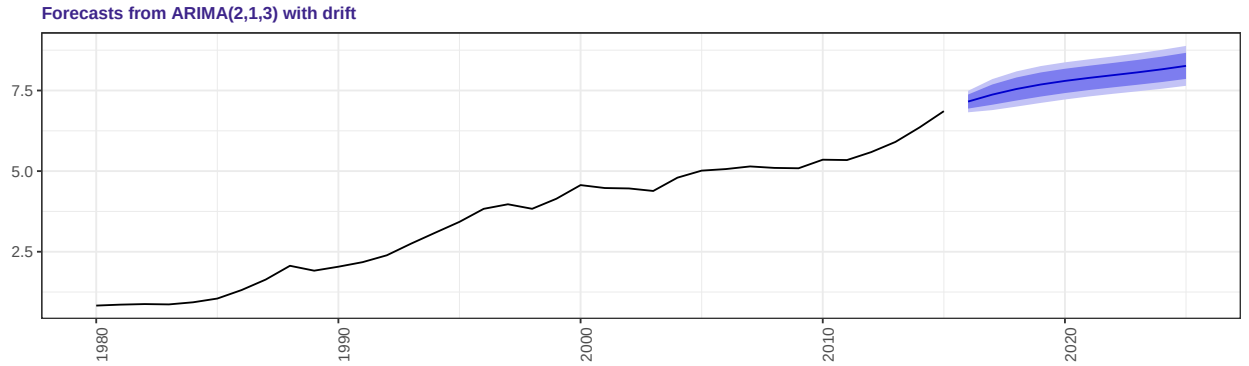
```
(fit_arima <- Arima(austa, order = c(2, 1, 3), include.drift = TRUE,
  method = "ML"))
```

```
FALSE Series: austa
FALSE ARIMA(2,1,3) with drift
FALSE
FALSE Coefficients:
FALSE          ar1      ar2      ma1     ma2     ma3   drift
FALSE       1.7648  -0.8513  -1.7684  0.5571  0.2224  0.1725
FALSE s.e.  0.1136   0.1182   0.2433  0.4351  0.2150  0.0051
FALSE
FALSE sigma^2 estimated as 0.0276:  log likelihood=13.72
FALSE AIC=-13.44   AICc=-9.29   BIC=-2.55
```

```
fit_arima %>% autoplot() + theme_bw() + theme()
```



```
fit_arima %>% forecast(h = 10) %>% autoplot() + theme_bw() +
  theme()
```




```
accuracy(forecast(fit_arima))
```

```
FALSE                       ME      RMSE      MAE       MPE
FALSE Training set -0.004129818 0.1491068 0.114857 -1.203567
FALSE                    MAPE      MASE        ACF1
FALSE Training set 4.326425 0.5637093 -0.01736817
```

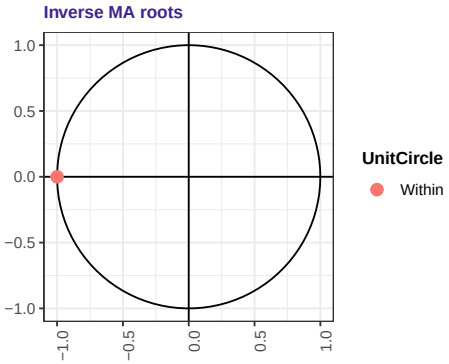
This forecast is very similar to the original (2,1,3) model, but it compares more favorably in terms of training error metrics. The AICc is considerably lower with constant (and drift) than without, as are the forecasting RMSE and resulting prediction interval. In short, it makes sense to allow for drift.

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

```
(fit_ma <- Arima(austa, order = c(0, 0, 1), include.mean = TRUE))
```

```
FALSE Series: austa
FALSE ARIMA(0,0,1) with non-zero mean
FALSE
FALSE Coefficients:
FALSE          ma1    mean
FALSE       1.0000  3.5515
FALSE s.e.  0.0907  0.3090
FALSE
FALSE sigma^2 estimated as 0.9347:  log likelihood=-50.64
FALSE AIC=107.28   AICc=108.03   BIC=112.04
```

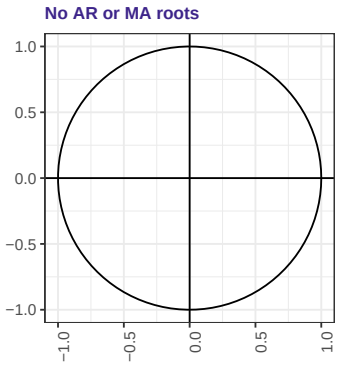
```
fit_ma %>% autoplot() + theme_bw() + theme()
```




```
(fit_noma <- Arima(austa, order = c(0, 0, 0), include.mean = FALSE))
```

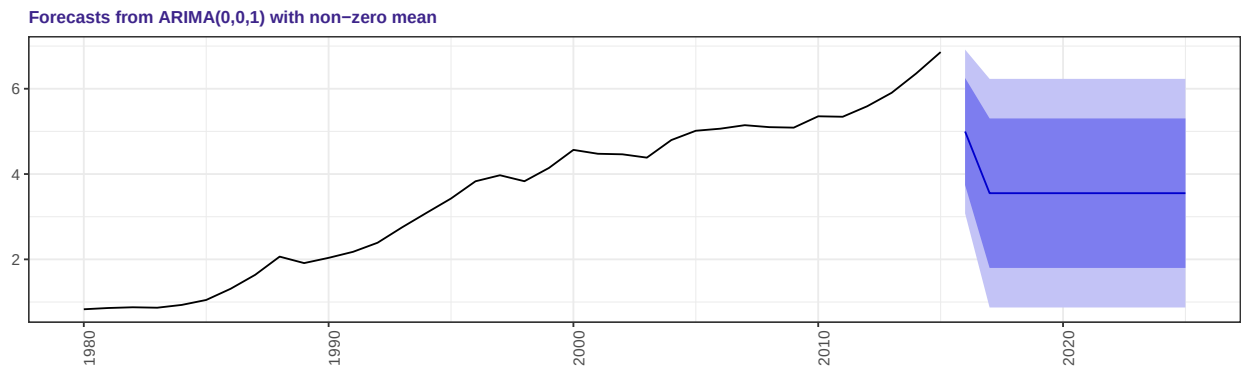
```
FALSE Series: austa
FALSE ARIMA(0,0,0) with zero mean
FALSE
FALSE sigma^2 estimated as 15.72:  log likelihood=-100.67
FALSE AIC=203.34   AICc=203.46   BIC=204.93
```

```
fit_noma %>% autoplot() + theme_bw() + theme()
```



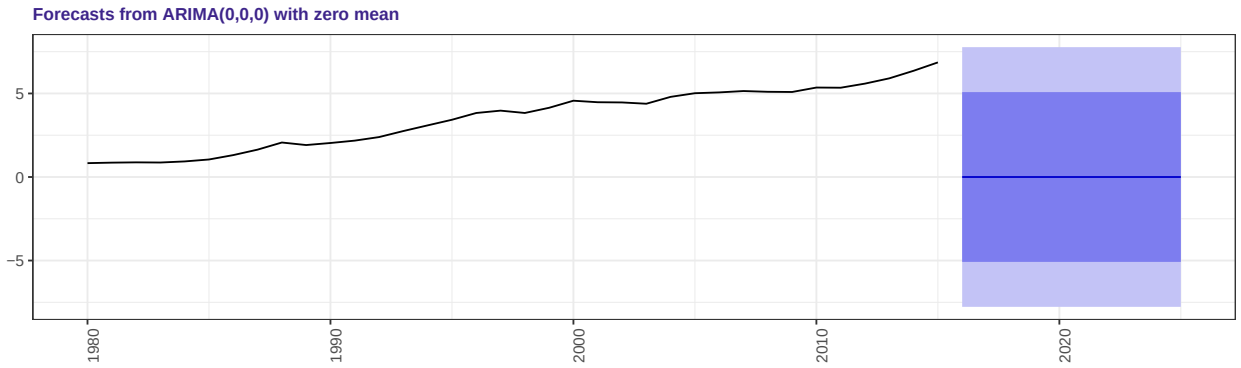
With MA

```
forecast(fit_ma, h = 10) %>% autoplot() + theme_bw() + theme()
```



Without MA

```
forecast(fit_noma, h = 10) %>% autoplot() + theme_bw() +
  theme()
```



Even without examination of error measures its evident that these forecasts are not particularly useful. The moving average forecast begins accounting for the trend, but then deteriorates to the mean; the (0,0,0) model simply predicts a static mean over an obviously more complex time series than the forecasts reflect.

Accuracy

```
accuracy(forecast(fit_ma, h = 10))
```

```
FALSE                      ME      RMSE       MAE       MPE
FALSE Training set 0.01997167 0.9395462 0.8063899 -26.87733
FALSE                    MAPE     MASE      ACF1
FALSE Training set 42.56419 3.957698 0.8109528
```

```
accuracy(forecast(fit_noma, h = 10))
```

```
FALSE                     ME     RMSE      MAE MPE MAPE
FALSE Training set 3.541428 3.964979 3.541428 100  100
FALSE                   MASE      ACF1
FALSE Training set 17.38105 0.9099807
```

This assessment gains further support given the high RMSE for the MA(1) and even higher RMSE for the (0,0,0) model. Neither are particularly useful forecasts.

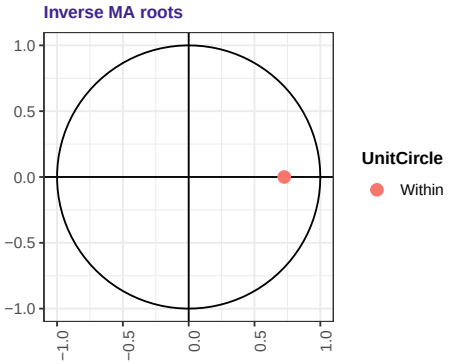
(e.) Plot forecasts from an ARIMA(0,2,1) model with no constant.

```
(fit_ma <- Arima(austa, order = c(0, 2, 1), include.mean = TRUE))
```

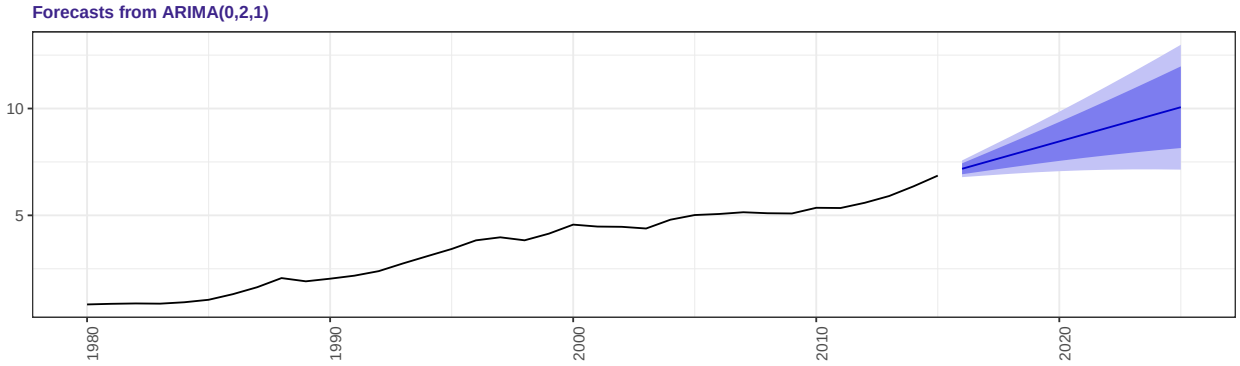
```
FALSE Series: austa
FALSE ARIMA(0,2,1)
```

```
FALSE
FALSE Coefficients:
FALSE           ma1
FALSE       -0.7263
FALSE s.e.   0.2277
FALSE
FALSE sigma^2 estimated as 0.03969:  log likelihood=6.74
FALSE AIC=-9.48   AICc=-9.09   BIC=-6.43
```

```
fit_ma %>% autoplot() + theme_bw() + theme()
```




```
forecast(fit_ma, h = 10) %>% autoplot() + theme_bw() + theme()
```




```
accuracy(forecast(fit_ma, h = 10))
```

```
FALSE                      ME      RMSE       MAE      MPE
FALSE Training set 0.02907781 0.1907537 0.1567877 1.216128
FALSE                    MAPE      MASE      ACF1
FALSE Training set 4.984081 0.7695018 0.1284994
```

Based on RMSE alone, this would seem a go-to model when compared with the (0,0,1), (0,0,0) and (0,1,1) models. It is close to as good as the (2,1,3) without drift, which had an RMSE of .17; however, the (2,1,3) with drift had the overall best RMSE of .14.

Given that the (2,1,3) model also had a tighter confidence window it makes sense to consider that there is some autoregression in this series and use the full ARIMA(2,1,3) with drift in predicting over this data set.