DATA 624 Summer 2024

**Group 3: Engelman, Ferris, Fox, Jimenez**

HW Batch #1

6/15/2024

Table of Contents

[HA 2.1: 2](#_Toc169370876)

[HA 2.3 5](#_Toc169370877)

[HA 6.2 10](#_Toc169370878)

[KJ 3.1 15](#_Toc169370879)

[KJ 3.2 19](#_Toc169370880)

[HA 7.1-7.3 25](#_Toc169370881)

[HA 8.1 29](#_Toc169370882)

[HA 8.2 30](#_Toc169370883)

[HA 8.6 33](#_Toc169370884)

[HA 8.8 38](#_Toc169370885)

[Works Cited: 44](#_Toc169370886)

# HA 2.1:

## Use the help function to explore what the series gold, woolyrnq, and gas represent

# Load libraries for all problems

library(tidyverse)

library(dplyr)

library(caret)

library(ggplot2)

library(forcats)

library(fpp2)

library(mlbench)

library(RColorBrewer)

library(scales)

#-------------------------

# Section A – Explore datasets

#-------------------------

?gold

head(gold)

?woolyrnq

head(woolyrnq,8)

?gas

head(gas,24)

* **gold**: This series represents daily morning gold prices in US Dollars for the time period 1/1/1984 - 3/31/1989. It is a time series with 1,108 observations; no source is provided.

*Sample data:*

Time Series:  
Start = 1   
End = 6   
Frequency = 1   
[1] 306.25 299.50 303.45 296.75 304.40 298.35

* **woolyrnq:** This series represents quarterly production of woolen yarn in Australia in tonnes for the time period Mar 1965-Sept 1994. It is presented as a time series with 119 observations and was sourced from the Time Series Data Library (https://pkg.yangzhuoranyang.com/tsdl/).

*Sample data:*

Qtr1 Qtr2 Qtr3 Qtr4  
1965 6172 6709 6633 6660  
1966 6786 6800 6730 6765

* **gas:** This series represents monthly Australian gas production 1956-1995 (no units provided). It is presented as a time series sourced from the Australian Bureau of Statistics.

*Sample data:*

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec  
1956 1709 1646 1794 1878 2173 2321 2468 2416 2184 2121 1962 1825  
1957 1751 1688 1920 1941 2311 2279 2638 2448 2279 2163 1941 1878

## B. Use autoplot to plot each in separate plots:

#-------------------------

# Section B - Autoplot

#-------------------------

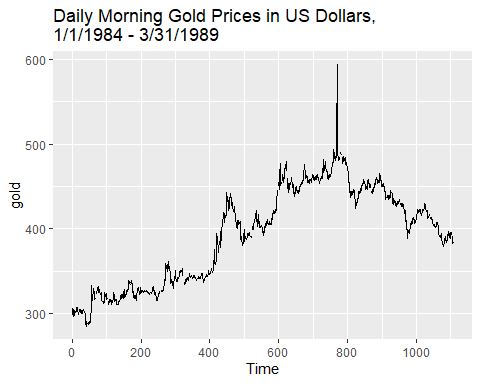
autoplot(gold) + ggtitle ("Daily Morning Gold Prices in US Dollars,

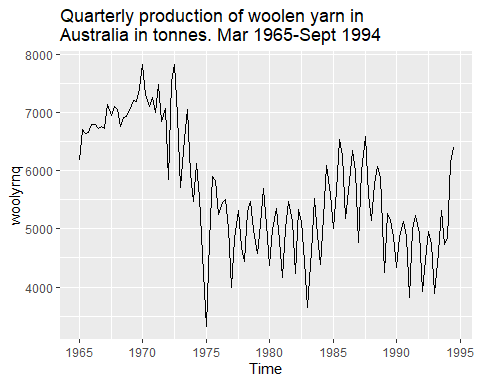
1/1/1984 - 3/31/1989")

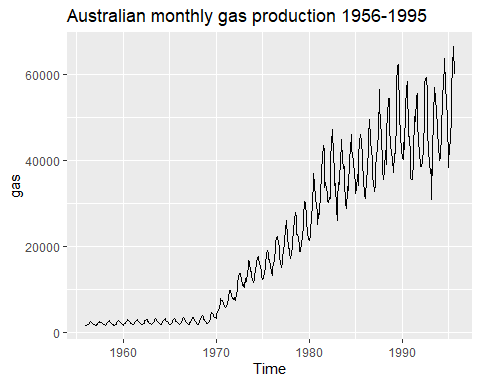
autoplot(woolyrnq) + ggtitle ("Quarterly production of woolen yarn in

Australia in tonnes. Mar 1965-Sept 1994")

autoplot(gas) + ggtitle ("Australian monthly gas production 1956-1995")







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

#-------------------------

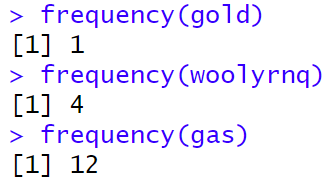
# Section C - Frequency

#-------------------------

frequency(gold)

frequency(woolyrnq)

frequency(gas)



By using the “frequency” function, we can see these patterns in our three sample time series:

* **gold**: This data is daily (business days) and has a frequency of one observation per unit of time (day).
* **woolyrnq**: This data is quarterly, and has a frequency of four observations per unit of time (year).
* **gas**: This data is monthly, and has a frequency of 12 observations per unit of time (year).

## D. Use which.max to spot the outlier in the gold series? Which was it?

#-------------------------

# Section D – which.max

#-------------------------

which.max(gold)



The outlier in the gold series was observation #770

# HA 2.3

**Prompt:**

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

## A. Read the data into R with the given script

After downloading the excel file from the book website, I imported the retail data using readxl package’s function read\_excel(), which functions similarly to read\_csv() but for excel files. As stated in the question problem, the skip=1 is necessary because the excel file has 2 headers.

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

After importing the retail data, I went ahead and observed the data set. It has 380 observations/ rows and 190 columns. It is time series data.

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

From the retail data set, I selected the A3349719C column to put through the timeseries ts() function. frequency argument of 12 was used because the data is monthly. The start argument of (1982,4) was set because April 1982 was the first data point. myts variable holds the time series object.

myts = ts(retaildata[,"A3349719C"],  
 frequency=12, start=c(1982,4))

## C. Explore your chosen retail time series using the following functions:

autoplot(), ggseasonplot(), ggsubseriesplot(), gglagplot(), ggAcf()

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

**Autoplot**

First, I plot the retail data with autoplot() . This function is nice because it automatically plots a line graph for time series object.

A graph showing a number of retail data

Description automatically generated

The retail data has seasonality. You can almost see the yearly pattern clearly, which I can best describe as one wavelength:

A graph showing a graph of a stock market

Description automatically generated with medium confidence

I can’t really tell anything about cyclicity using this plot, there might be some.

The trend of the graph is increasing. There are more sales over time.

**ggseasonplot**

Next, I use ggseasonplot().

A graph of retail data

Description automatically generated

ggseasonplot() shows the distribution of the data year by year. This better shows seasonality, with the increase being obvious around November of each year.

**ggsubseriesplot**

A graph showing the number of retail data

Description automatically generated

ggsubseriesplot() shows the distribution of the data for each of the months every year. The month of December’s mean is much higher than the mean of the other months, meaning December always had significantly increased sales than the earlier months.

**gglagplot**

A graph of retail data

Description automatically generated

gglagplot() produces a lag plot. For each lag plot, the pattern is repeating which supports seasonality.

**ggACF**

A graph of retail data

Description automatically generated

ggacf shows the autocorrelation plot. It supports seasonality with peaks every 12 months. It also shows significant autocorrelation since the bars are above the blue dotted lines.

**Code for reference:**

# Load libraries

library(fpp2)

#-------------------------

# Section A

#-------------------------

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

#-------------------------

# Section B

#-------------------------

myts = ts(retaildata[,"A3349719C"],

frequency=12, start=c(1982,4))

#-------------------------

# Section C

#-------------------------

autoplot(myts) + ggtitle("Time Series Plot of Retail Data")

ggseasonplot(myts) + ggtitle("Seasonal Plot of Retail Data")

ggsubseriesplot(myts) + ggtitle("Subseries Plot of Retail Data")

gglagplot(myts) + ggtitle("Lag Plot of Retail Data")

ggAcf(myts) + ggtitle("ACF Plot of Retail Data")

# HA 6.2

**Prompt:**

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

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

In the graph below, you can see the overall sales. There appears to be a clear cyclic pattern. It is very consistent and repeats yearly. The pattern seems to shift upward after each year, which indicates a possible increasing trend.

```{r}

library(fma)

library(ggplot2)

data(plastics)

plot(plastics, main = "Monthly Sales of Product A for a Plastics Manufacturer",

xlab="Year", ylab="Sales (in thousands of dollars")

```

A graph of a sales report

Description automatically generated with medium confidence

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

The first section below shows the values of the indices for the trend-cycle and then the seasonal cycle. The values for the trend-cycle show an increase until right near the end, at which point they decrease slightly. The values for the seasonal cycle are exactly the same for each month. All the values for January are equal, all the values for February are equal, and so on. Different months have different values. In order, they increase until September. After September, they begin to decrease.

```{r}

decomposition <- decompose(plastics, type = "multiplicative")

trend\_cycle <- decomposition$trend

seasonal <- decomposition$seasonal

trend\_cycle

seasonal

```

A screenshot of a computer

Description automatically generated

The second section below shows a graph of these indices along with the original data and a remainder component.

```{r}

autoplot(decomposition) +

ggtitle("Multiplicative Decomposition of Monthly Plastic Sales") +

xlab("Year") +

ylab(“Sales (in thousands of dollars)”) +

theme\_minimal()

```

A graph of a graph showing the growth of plastic sales

Description automatically generated

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

Yes, the results support the graphical interpretation from part a. After multiplicative decomposition, it is clear to see the upward trend that was assumed in part a, and the consistent cyclical pattern is also shown in the results. The seasonal indices additionally show that the cycle repeats after every year.

## Compute and plot the seasonally adjusted data.

In this section, the seasonally adjusted data consists of all the data without the seasonal component. The seasonal component is divided out because it was originally calculated by using multiplicative decomposition. The first plot shows the original data, and the second plot shows the adjusted data. The two graphs together demonstrate the difference between original data and seasonally-adjusted data.

```{r}

decomposed\_data <- decompose(plastics, type = "multiplicative")

seasonal <- decomposed\_data$seasonal

seasonally\_adjusted <- plastics / seasonal

plot(plastics, main = "Original Data", ylab = "Sales (in thousands of dollars)", col =

"blue")

plot(seasonally\_adjusted, main = "Seasonally Adjusted Data",

ylab = "Sales (in thousands of dollars)", col = "purple")

```

A graph with blue lines

Description automatically generated A graph with purple line

Description automatically generated

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

The outlier of 50 was changed for the month of April in the second year of data collection. The results below were computed after this outlier was added to the data.

The outlier changed the seasonally-adjusted data. The rest of the data appeared to have the same upward trend as the original data when looking at the scale on the y-axis. The outlier in the data was seen as a major dip in Sales. This dip is much different from the original seasonally-adjusted data. This example demonstrates the power of outliers and the need to identify them when analyzing data.

```{r}

plastics[16] <- 50

decomposed\_data <- decompose(plastics, type = "multiplicative")

seasonal <- decomposed\_data$seasonal

seasonally\_adjusted <- plastics / seasonal

plot(plastics, main = "Original Data", ylab = "Sales (in thousands of dollars)", col =

"blue")

plot(seasonally\_adjusted, main = "Seasonally Adjusted Data", ylab = "Sales (in thousands

of dollars)", col = "purple")

```

A graph showing a line

Description automatically generated A graph showing a line

Description automatically generated with medium confidence

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

To determine if the location of the outlier results in any difference, an outlier was added near the end of the time series and compared to the results of data with the other outlier.

Based on the graph from below, the outlier still changed the graph, but since the outlier was at the end, it did not appear to have much of an effect on the cyclical pattern of the data as the other outlier did. The outlier from the previous question resulted in small peaks throughout the graph, while this outlier did not appear to result in any peaks. Instead, the graph shows an increasing trend and then a major dip where the outlier occurred.

```{r}

data(plastics)

plastics[58] <- 75

decomposed\_data <- decompose(plastics, type = "multiplicative")

seasonal <- decomposed\_data$seasonal

seasonally\_adjusted <- plastics / seasonal

plot(plastics, main = "Original Data", ylab = "Sales (in thousands of dollars)", col =

"blue")

plot(seasonally\_adjusted, main = "Seasonally Adjusted Data", ylab = "Sales (in thousands

of dollars)", col = "purple")

```

A graph with blue lines

Description automatically generated A graph with purple line

Description automatically generated

# KJ 3.1

**Prompt:**

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

The data can be accessed via:

data(Glass)  
str(Glass)

## 'data.frame': 214 obs. of 10 variables:  
## $ RI : num 1.52 1.52 1.52 1.52 1.52 ...  
## $ Na : num 13.6 13.9 13.5 13.2 13.3 ...  
## $ Mg : num 4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...  
## $ Al : num 1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...  
## $ Si : num 71.8 72.7 73 72.6 73.1 ...  
## $ K : num 0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...  
## $ Ca : num 8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...  
## $ Ba : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Fe : num 0 0 0 0 0 0.26 0 0 0 0.11 ...  
## $ Type: Factor w/ 6 levels "1","2","3","5",..: 1 1 1 1 1 1 1 1 1 1 ...

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

**Density Plots**

Using density plots reveals a lot about the distribution of the data. Some of the predictors appear to be pretty normally distributed, albeit with slight skew, while others don’t.

* Al appears to be very normally distributed with a very slight right skew.
* Ba has a very large percentage of zeros. This can cause major issues for many models and skews the distribution to the right.
* Ca has a slight right skew.
* Fe, like Ba, has a very large percentage of zeros.
* K has a significant amount of zeros while and most of the data is concentrated around 0.1 with what appear to be some major outliers around 0.6.
* Mg has what appears to be a normal distribution with a significant left skew. It also has a significant amount of zeros resulting in the appearance of a bimodal distribution. This is likely an illusion.
* Na, like Al, appears to be very normally distributed with a very slight right skew.
* Si appears to be very normally distributed with a slight left skew.
* RI (the refractive index) appears to be very normally distributed with a very slight right skew.

Glass |>  
 select(-Type) |>  
 gather(key = "variable", value = "value") |>   
 ggplot(aes(x = value)) +   
 geom\_histogram(aes(y = after\_stat(density)), bins = 20, fill = '#4E79A7', color = 'black') +   
 stat\_density(geom = "line", color = "red") +  
 facet\_wrap(~ variable, scales = 'free') +  
 theme(strip.text = element\_text(size = 5)) +  
 theme\_bw()

A graph of different values

Description automatically generated with medium confidence

**Correlation Matrix**

A correlation matrix can be used to understand the relationships between the predictors. The below plot reveals some significant relationships between the predictors. Most notably, there is a positive relationship of 0.81 between Ca and RI, this is an extremely strong relationship and should be considered when utilizing the data. There is also a significant positive relationship between Al and Ba (0.48), and negative relationships between RI and Si (-0.54), Mg and Ba (-0.49), and Mg and Al (-0.48) among some lesser but still potentially significant relationships. All these relationships should be considered when building a model and for any inference that may be drawn from the data.

As noted above, some of the predictors have interesting distributions that might be improved with transformations. After those transformations, the relationships between the predictors may, and in some cases is likely to, change.

q <- cor(Glass |>  
 select(-Type))  
  
ggcorrplot(q, type = "upper", outline.color = "white",  
 ggtheme = theme\_classic,  
 colors = c("#F28E2B", "white", "#4E79A7"),  
 lab = TRUE, show.legend = F, tl.cex = 10, lab\_size = 3)

A graph of periodic table

Description automatically generated

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

**Boxplots**

As noted above by the density plots, there are definitely outliers in the data and some predictors are skewed. We can get additional information about the distribution of the data by using boxplots.

The below boxplots confirm the presence of outliers in the data. The most notable outliers are in variables K and Ca, with Ba also having some significant outliers. The boxplots also confirm the presence of skew in the data. The most notable skew is in variables Ba and Fe, which have a significant amount of zeros.

Glass |>  
 select(-Type) |>  
 gather(key = "variable", value = "value") |>  
 ggplot(aes(x = variable, y = value)) +  
 geom\_boxplot(fill = '#4E79A7', color = 'black') +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1)) +  
 theme\_bw()

A graph with black and blue lines

Description automatically generated

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

There definitely should be some transformations of the data. As noted above, some of the predictors have significant skew and can be improved with transformations. A box-cox transformation to find the optimal lambda for each predictor would be a good place to start. However, due to the large number of zeros in some of the variables, a constant should be added to the data before applying the transformation to avoid undefined values from the transformation.

Additionally, the relationships between the predictors should be considered when transforming the data. For example, the strong relationship between Ca and RI should be considered when transforming the data and it might be decided to either use one or the other or to use the same transformation for both. It also might be decided to not transform any variables with relatively small skewness in order to not lose the relationships between predictors.

Another transformation that might be made is to scale and center the data (but not around zero if using a box-cox transformation as mentioned above) to ensure that all predictors are on the same scale.

# KJ 3.2

**Prompt:** The soybean data can 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.

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

# Load Soybean dataset

data(Soybean)

# Create df excluding outcome variable "Class"

soy\_data <- select(Soybean,-Class)

#----------------------------------------

# Zero-Var:

#----------------------------------------

# Identify Zero-Var: Count unique values for each column and extract those where count = 1

unique\_values <- soy\_data %>%

summarise\_all(n\_distinct)

zero\_var <- names(unique\_values)[unique\_values == 1]

zero\_var

#----------------------------------------

# Near-Zero-Var:

#----------------------------------------

# Identify Near-Zero-Var: Use function in caret pkg

near\_zero <- soy\_data %>%

nearZeroVar()

near\_zero\_vars <- (names(soy\_data)[near\_zero])

near\_zero\_vars

# Calculate ratios for the above: Create df to hold the ratios

df\_soy\_freq <- tibble(

var = character(),

most\_freq = character(),

most\_freq\_count = numeric(),

sec\_most\_freq = character(),

sec\_most\_freq\_count = numeric(),

ratio = numeric()

)

# Iterate through the near-zero-var variables above:

for (col\_name in (names(soy\_data)[near\_zero])) {

soy\_var <- soy\_data[[col\_name]]

soy\_var\_freq <- table(soy\_var) %>%

sort(decreasing = TRUE)

freq\_1 <- soy\_var\_freq[1]

freq\_2 <- soy\_var\_freq[2]

freq\_ratio <- freq\_1/freq\_2

df\_soy\_freq <- bind\_rows(df\_soy\_freq,tibble(

var = col\_name,

most\_freq = names(soy\_var\_freq[1]),

most\_freq\_count = freq\_1,

sec\_most\_freq = names(soy\_var\_freq[2]),

sec\_most\_freq\_count = freq\_2,

ratio = freq\_ratio

)

)

}

#df\_soy\_freq$ratio <- as.numeric(df\_soy\_freq$ratio)

df\_soy\_freq

# Create bar chart for each degenerate variable

for (i in near\_zero\_vars) {

soy\_plot <- ggplot(soy\_data, aes\_string(x=i)) +

geom\_bar(fill = "darkslategray4") +

ggtitle(i) +

xlab(i) +

ylab('Frequency')

print(soy\_plot)

}

There were no “zero variance” variables in this dataset, as all predictors had more than one unique value:

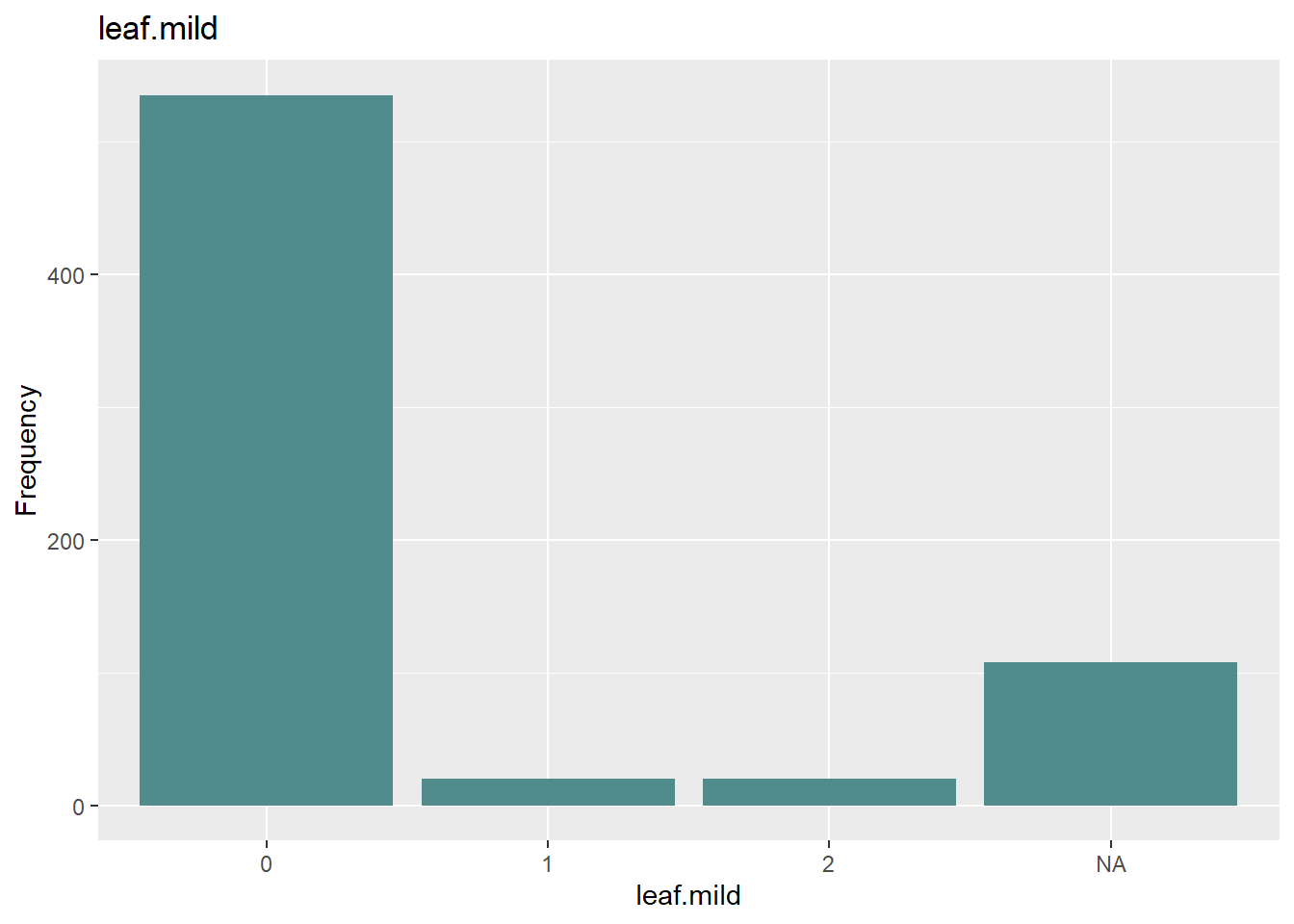
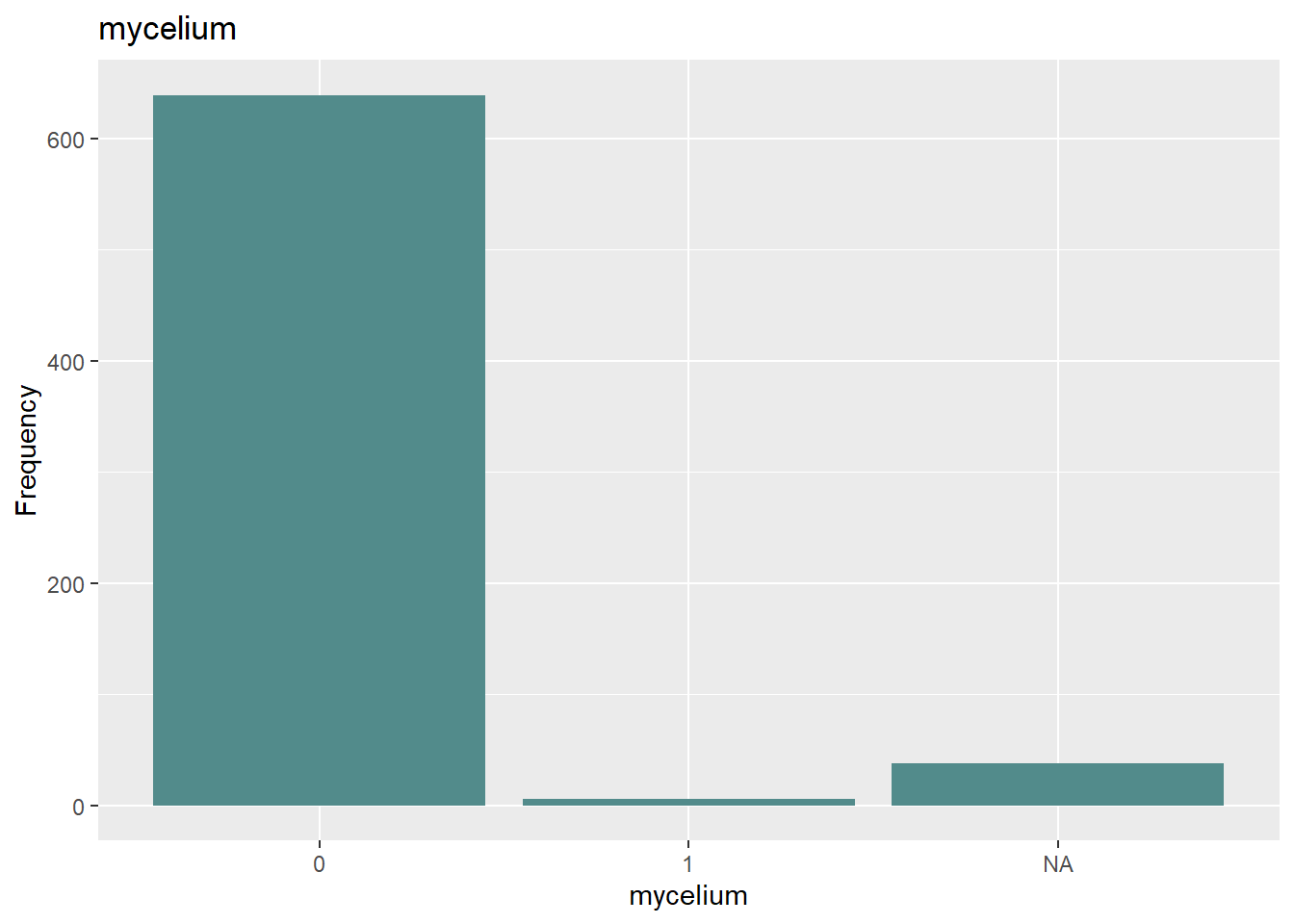


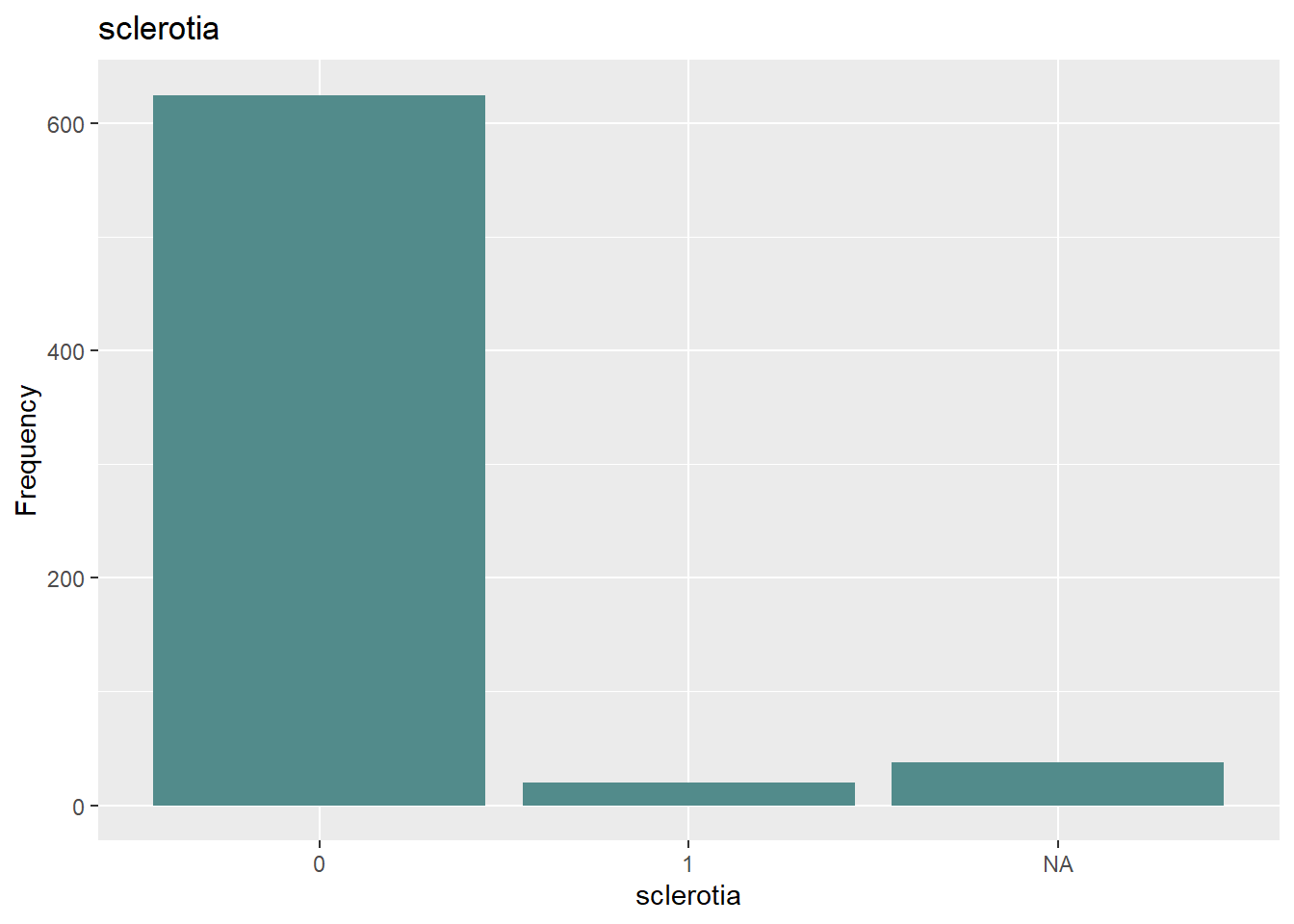
However, three variables exhibited “near-zero variance”: their ratio of most-common to second-most-common variables exceeded 20:1 (Kuhn, 44). Those three variables were:

[1] "leaf.mild" "mycelium" "sclerotia"

The table and bar charts below display the frequency of the most-common vs. the second-most-common values for each of these three degenerate variables.

var most\_freq most\_freq\_count sec\_most\_freq sec\_most\_freq\_count ratio  
 <chr> <chr> <dbl> <chr> <dbl> <dbl>  
1 leaf.mild 0 535 1 20 26.8  
2 mycelium 0 639 1 6 106.0   
3 sclerotia 0 625 1 20 31.2





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

#--------------------------

# Missing Values by Predictor:

#--------------------------

missing\_pred <- soy\_data %>%

summarise\_all(~sum(is.na(.))/n()) %>%

pivot\_longer(everything(),names\_to = "predictor", values\_to = "pct\_na") %>%

arrange(desc(pct\_na))

tail(missing\_pred)

missing\_plot <- missing\_pred %>%

ggplot(aes(x = reorder(predictor, pct\_na), y = pct\_na)) +

geom\_col(fill = "seagreen") +

coord\_flip() +

scale\_y\_continuous(

labels = scales::percent,

breaks = breaks\_width(.02,0)) +

labs(

title = "% Missing Values by Predictor",

x = "Predictor",

y = "% Missing Values")

missing\_plot

#--------------------------

# Missing Values by Class:

#--------------------------

# add column to count of missing values across all predictor variables

missing\_outcomes <- Soybean %>%

mutate(

count\_na = rowSums(is.na(Soybean[,-1])),

count\_total = 35) %>%

group\_by(Class) %>%

summarise(

total\_rows = n(),

total\_values = sum(count\_total),

total\_na= sum(count\_na),

pct\_na = total\_na/total\_values) %>%

arrange(desc(pct\_na))

head(missing\_outcomes, 5)

# plot distribution of n/a values by class

missing\_plot2 <- missing\_outcomes %>%

ggplot(aes(x = reorder(Class, pct\_na), y = pct\_na)) +

geom\_col(fill = "lightblue") +

coord\_flip() +

scale\_y\_continuous(

labels = scales::percent,

breaks = breaks\_width(.05,0)) +

labs(

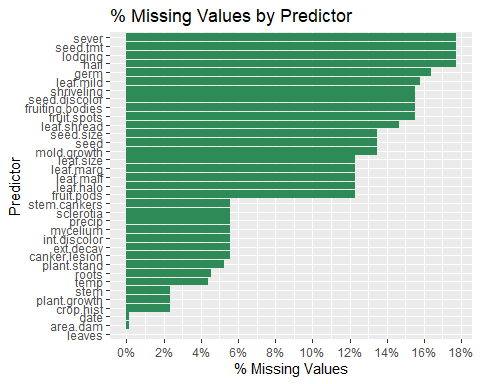
title = "% Missing Values by Class",

x = "Class",

y = "% Missing Values")

missing\_plot2

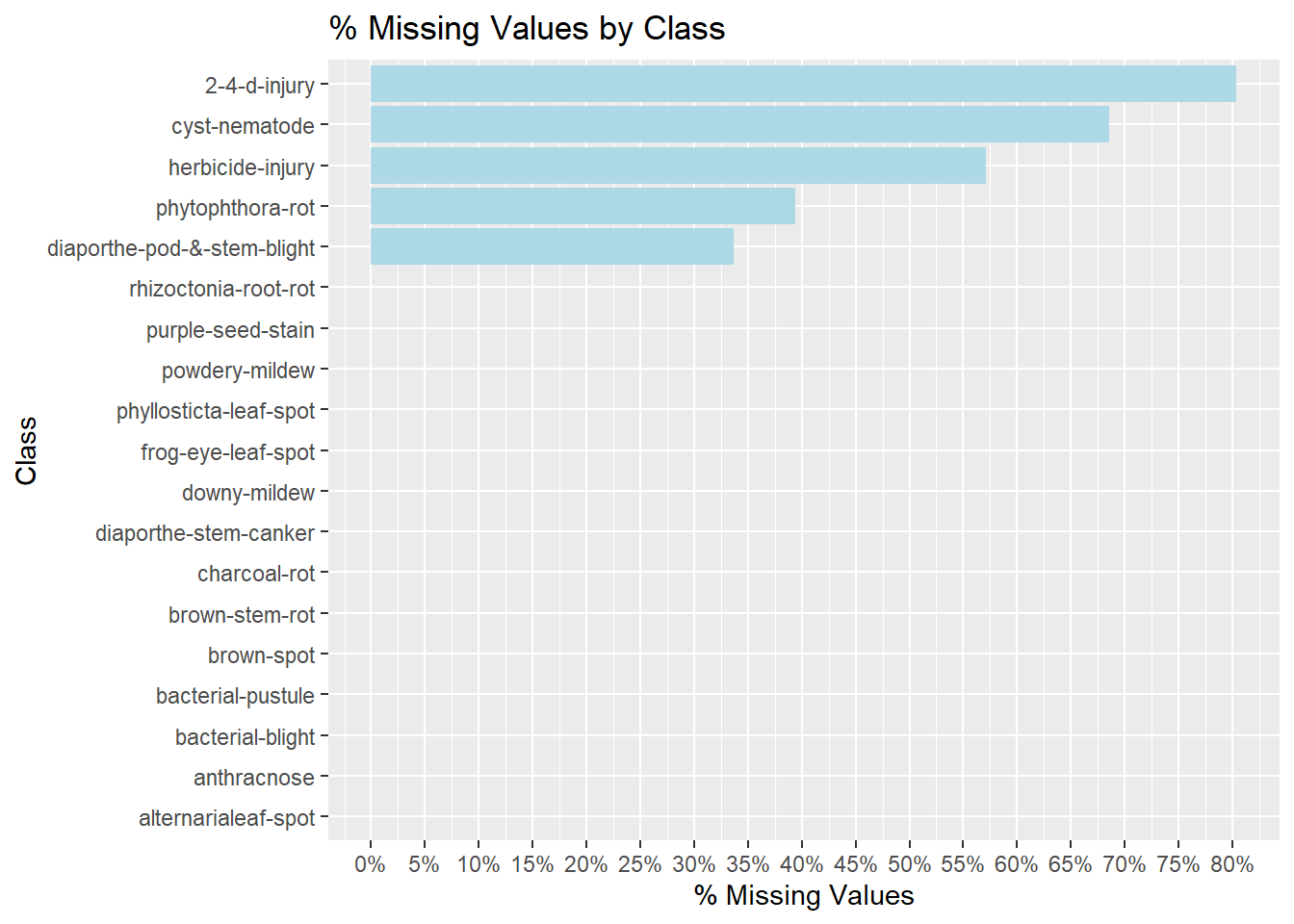
**Predictors:** Some predictors are much more likely than others to have missing values. The bar chart below shows the rate of missing data for all 35 predictor variables. Note that only one variable (leaves) has no missing values at all, while the rest range widely from <1% (area.dam, date) to 17.73% (sever, seed.tmt, lodging, hail).



**Classes:** There is also a strong relationship between the outcome variable (class) and missing data. The summary grid and bar chart below illustrate the proportion of NA predictor values to total possible predictor values for each Class (number of rows \* 35 total predictor variables).

Only five Classes had any missing values, but these were all quite significant, ranging from 33.7% to 80.4% of the total possible values.

Class total\_rows total\_values total\_na pct\_na  
 <fct> <int> <dbl> <dbl> <dbl>  
1 2-4-d-injury 16 560 450 0.804  
2 cyst-nematode 14 490 336 0.686  
3 herbicide-injury 8 280 160 0.571  
4 phytophthora-rot 88 3080 1214 0.394  
5 diaporthe-pod-&-stem-blight 15 525 177 0.337



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

To develop an effective strategy to handle missing data, one should carefully evaluate the data and engage with the subject matter experts to understand the context around missing values. Given the strong relationship between the NA values and the Classes, at least some of the missing data is likely to be meaningful and should not be eliminated without due consideration.

For example, it may be that a value is missing for a given variable because the plant never matured (perhaps a predictor like “fruiting bodies”). In cases like this, the missing data may considered “informative missing” and should be included.

Alternately, a missing value may indicate a value beyond a measurable threshold (“censored missing”). In this case, the data is still meaningful and should receive a value equal to the threshold or a random number between zero and the threshold value.

Or a value might simply be missing due to data quality, collection, or other reasons, and such a variable might be imputed using K-nearest neighbor or correlating it to another variable, if a strong correlation exists.

Of course, if the predictor with significant missing data is highly correlated to another variable, it might also be eliminated as redundant to reduce complexity, if no additional value would be gained by keeping it.

# HA 7.1-7.3

**Prompt:**

Consider the pigs series — the number of pigs slaughtered in Victoria each month.

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

**Simple Exponential Smoothing & Forecasts**

# Use the ses() function to find optimal values and generate forecasts  
fit <- ses(pigs, h = 4)  
  
# Extracting the optimal parameters  
alpha\_opt <- fit$model$par["alpha"]  
level\_opt <- fit$model$par["l"]  
  
# Print the results  
#print(fit)  
cat("Optimal alpha:", alpha\_opt, "\n")

## Optimal alpha: 0.2971488

cat("Optimal level (l0):", level\_opt, "\n")

## Optimal level (l0): 77260.06

# Forecasts for the next 4 months  
forecasts <- fit$mean  
print(forecasts)

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

The optimum alpha for the simple exponential smoothing is 0.297 and the optimum level is 77260.06. Using these values, the ses() function generated forecasts for the next four months. The forecasts were all 98816.41.

A feature of simple exponential smoothing is that the forecast for the next period is equal to the level of the series at the current period. Therefore, all future months will have the same forecast, albeit with wider successive prediction intervals.

## 7.1 (b) Compute a 95% prediction interval for the first forecast using where is the standard deviation of the residuals.

Compare your interval with the interval produced by R.

**Prediction Interval**

# Standard deviation of the residuals  
s <- sd(residuals(fit))  
  
# First forecast  
first\_forecast <- forecasts[1]  
  
# 95% prediction interval  
lower\_bound <- first\_forecast - 1.96 \* s  
upper\_bound <- first\_forecast + 1.96 \* s  
man\_ci\_95 <- c(lower\_bound, upper\_bound)  
  
# Compare with interval produced by R  
lower\_95\_first <- fit$lower[1, "95%"]  
upper\_95\_first <- fit$upper[1, "95%"]  
ses\_ci\_95 <- c(lower\_95\_first, upper\_95\_first)  
  
cat("95% prediction interval (manual):", man\_ci\_95, "\n")

## 95% prediction interval (manual): 78679.97 118952.8

cat("95% prediction interval (ses()):", ses\_ci\_95, "\n")

## 95% prediction interval (ses()): 78611.97 119020.8

The 95% prediction interval for the first forecast using the manual calculation is (78679.97-118952.8), while the interval produced by the ses() function is (78611.97-119020.8). The discrepancy is likely due to the rounding of the standard deviation, the inexactness of the 1.96 multiplier, the specific implementation of the ses() function, floating-point errors, or a combination of these factors. The intervals are close enough to be considered equivalent for practical purposes.

## 7.2 Write your own function to implement simple exponential smoothing.

The function should take arguments y (the time series), alpha (the smoothing parameter ) and level (the initial level ). It should return the forecast of the next observation in the series. Does it give the same forecast as ses()?

**Simple Exponential Smoothing Custom Function**

# Function for simple exponential smoothing  
simple\_exponential\_smoothing <- function(y, alpha, level) {  
 n <- length(y)  
 forecast <- numeric(n + 1)  
 forecast[1] <- level  
   
 for (t in 1:n) {  
 forecast[t + 1] <- alpha \* y[t] + (1 - alpha) \* forecast[t]  
 }  
   
 return(forecast[n + 1])  
}  
  
# Compare with ses() forecast  
my\_forecast <- simple\_exponential\_smoothing(pigs, alpha\_opt, level\_opt)  
cat("My forecast:", my\_forecast, "\n")

## My forecast: 98816.41

cat("ses() forecast:", forecasts[1], "\n")

## ses() forecast: 98816.41

The custom function for simple exponential smoothing produced a forecast of 98816.41, which is the same as the forecast generated by the ses() function. This result confirms that the custom function is correctly implementing simple exponential smoothing. It should be noted that the alpha and level parameters used in the custom function are the optimal values obtained from the ses() function. This means that while the custom function can produce the same forecast as ses(), while using the same parameters, it is still dependent on the ses() function for the optimal values.

## 7.3 Modify your function from the previous exercise to return the sum of squared errors

Then use the optim() function to find the optimal values of and . Do you get the same values as the ses() function?

**Sum of Squared Errors and Optimization**

# Function for simple exponential smoothing returning sum of squared errors  
simple\_exponential\_smoothing\_sse <- function(y, alpha, level) {  
 n <- length(y)  
 forecasts <- numeric(n)  
 forecasts[1] <- level  
 sse <- 0  
   
 for (t in 2:n) {  
 forecasts[t] <- alpha \* y[t-1] + (1 - alpha) \* forecasts[t-1]  
 sse <- sse + (y[t] - forecasts[t])^2  
 }  
   
 return(sse)  
}  
  
# Use the optim() function to find the optimal alpha and level  
initial\_params <- c(alpha = 0.2, level = pigs[1])  
optim\_result <- optim(initial\_params, function(params) {  
 simple\_exponential\_smoothing\_sse(pigs, params[1], params[2])  
})  
  
optimal\_alpha <- optim\_result$par[1]  
optimal\_level <- optim\_result$par[2]  
  
cat("Optimized alpha:", optimal\_alpha, "\n")

## Optimized alpha: 0.2950617

cat("Optimized level:", optimal\_level, "\n")

## Optimized level: 78332.9

# Compare with ses() optimal values  
cat("ses() alpha:", alpha\_opt, "\n")

## ses() alpha: 0.2971488

cat("ses() level:", level\_opt, "\n")

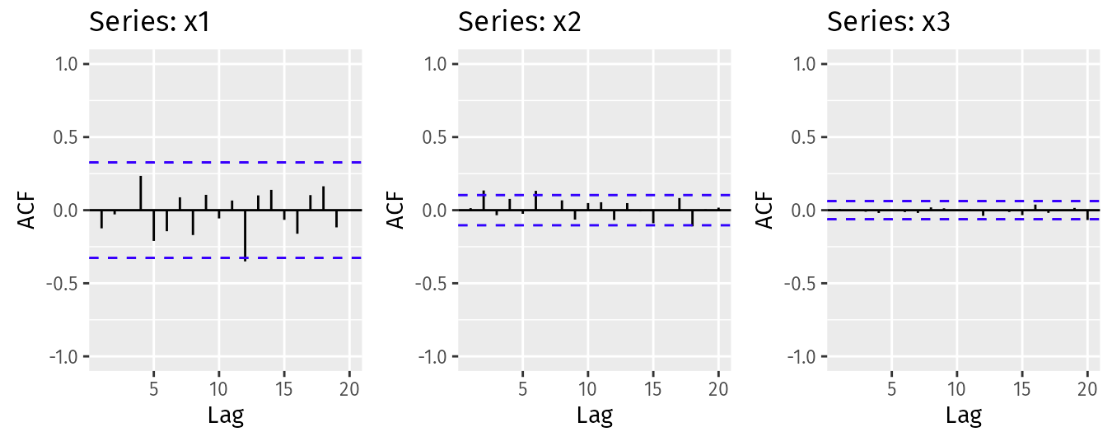
## ses() level: 77260.06

The optimization process using the optim() function found an optimal alpha of 0.2971488 and an optimal level of 77260.06. These values are extremely similar to the optimal values obtained from the ses() function. This result indicates that the custom function, which minimizes the sum of squared errors, can find the same optimal values for alpha and level as the ses() function. The similarity in the optimal values further confirms the accuracy of the custom function in implementing simple exponential smoothing.

# HA 8.1

**Prompt:**

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

****

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

Yes, all of the above figures indicate that the data are white noise. There is no autocorrelation: the ACFs all fall within +/- , indicated by the dashed blue lines on each chart, so there is no significant relationship between lagged values in the time series.

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

The critical values, indicated by the dashed blue lines, represent +/- where T is the length of the time series. As T increases from 36 to 360 to 1,000, the is 6, ~18.97, and ~31.62 respectively.

This has the effect of increasing the denominator and therefore decreasing the value of , narrowing the critical values as seen above. As long as the ACF values fall between the critical values, the data are white noise in all of these examples regardless of the differences in the absolute value of the critical values.

# HA 8.2

**Prompt:**

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.

#### Plotting Daily Closing Prices

The data set ibmclose is already nicely cleaned and organized after looking at it. It is time series data of the IBM stock at closing after each day. To plot the daily closing prices, I used autoplot() and added labels using ggtitle() and labs().

A graph showing the price of ibm stock

Description automatically generated

For the span of the data set, IBM stock experienced an initial sharp period of growth where the stock priced increased ~$50 in a less than 25 day time period. Afterwards, there was a slow decline; followed by an upward trend that lasted about 75 days (at around day 110). At this point, the stock price reached its peak price of ~$600 and then the price continued to decline until it reached its minimum of ~$300 160 days later (at around day 275). After this period there was a short time frame of growth, followed by decreasing.

I can’t really tell seasonality from this just because it is data from one year. Maybe there is some cyclical effects that come from the day of the week (maybe different prices at closing Monday vs. closing Friday) but its difficult to tell with this plot. The decline in stock prices possibly correlate to some negative news related to the company around that time period.

## Plotting ACF

The Autocorrelation Function (ACF) plot is a visualization that shows how the values of a time series are related to each other at different time lags. To plot, I use the ACF() function.

A graph of a number of data

Description automatically generated with medium confidence

The plot shows that the autocorrelation starts very high (close to 1) at lag 1 and gradually decreases as the lag increases. This means the daily closing prices are highly correlated with their recent past values and the influence of past prices diminishes over time. The lags are not statistically significant because it exceeds the bounds of the confidence interval (dotted blue line).

## Plotting PACF

Partial Autocorrelation Function (PACF) measures the correlation between a time series and its lagged values, but this time accounting for the relationships of all shorter lags. Unlike the ACF, the PACF isolates the direct effect of a specific lag. That way, we can tell which specific lag is significant or not. To plot this, I use the pacf() function.

A graph of a company's closing price

Description automatically generated

The first lag has a significant partial autocorrelation (close to 1), which supports a strong direct relationship with the previous day’s price. Lags more than 1 day have partial autocorrelations that are much lower and within the confidence intervals (blue dashed lines), meaning these lags do not have significant direct effects. IBMs stock price is really really influenced by the previous day’s stock price.

**Code for reference:**

# Load libraries

library(fpp2)

data("ibmclose")

autoplot(ibmclose) + ggtitle("Daily Closing Prices of IBM Stock") +

labs(y = "Price", x = "Date")

acf(ibmclose, main = "ACF of IBM Closing Prices")

pacf(ibmclose, main = "PACF of IBM Closing Prices")

# HA 8.6

**Prompt:**

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

## Use the following R code to generate data from an AR(1) model with Φ1 = 0.6 and variance = 1. The process starts with y1 = 0.

```{r}

set.seed(85)

y <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

y[i] <- 0.6\*y[i-1] + e[i]

```

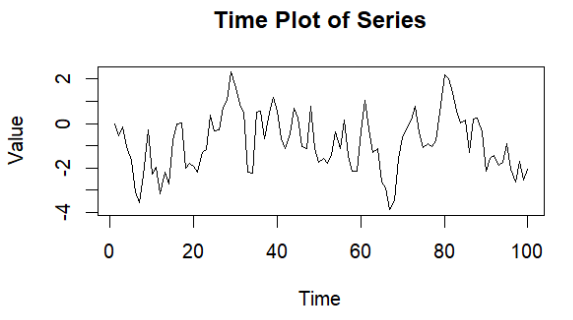
## Produce a time plot for the series. How does the plot change as you change autoregressive operator 1?

This is the time plot.

```{r}

plot(y, type = "l", main = "Time Plot of Series", xlab = "Time", ylab = "Value")

```



Next, the autoregressive operator was changed to 0.25, 1, and 2. The plots of these three variations are below. As you can see from the results, lowering the autoregressive operator slightly caused the values to appear a bit more random. A pattern was not obvious. When increasing the value to 1, the graph appeared to have a slight cyclical pattern. When increasing the value to 1.5, an obvious curve appeared. The values began around 0 and then experienced an exponential change in the positive direction.

```{r}

set.seed(40)

y <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

y[i] <- 0.25\*y[i-1] + e[i]

plot(y, type = "l", main = "Time Plot of Series - Variation 1", xlab = "Time", ylab = "Value")

y <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

y[i] <- 1\*y[i-1] + e[i]

plot(y, type = "l", main = "Time Plot of Series - Variation 2", xlab = "Time", ylab = "Value")

y <- ts(numeric(100))

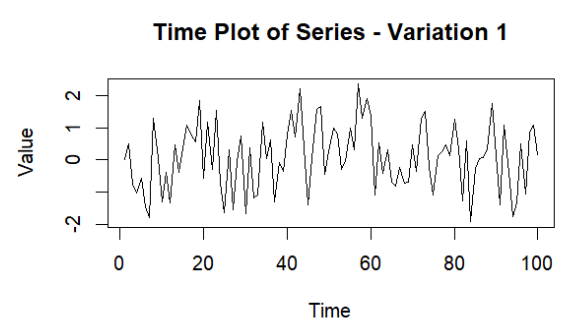
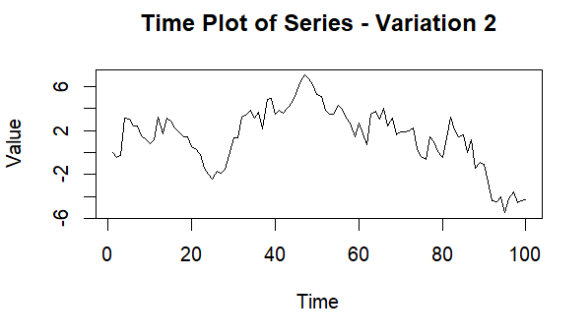
e <- rnorm(100)

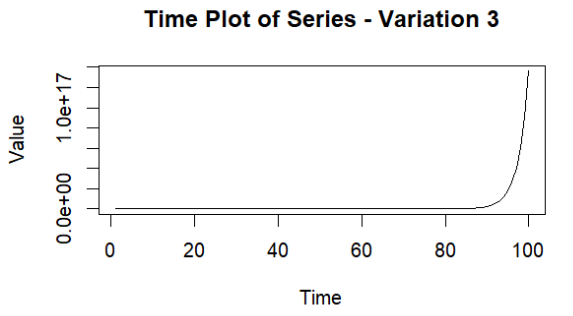
for(i in 2:100)

y[i] <- 1.5\*y[i-1] + e[i]

plot(y, type = "l", main = "Time Plot of Series - Variation 3", xlab = "Time", ylab = "Value")

```



## Write your own code to generate data from an MA(1) model with θ1 = 0.6 variance = 1.

```{r}

set.seed(25)

e <- rnorm(100, mean = 0, sd = sqrt(1))

y <- ts(numeric(100))

for(i in 2:100) {

y[i] <- e[i] + 0.6 \* e[i-1]

}

```

## Produce a time plot for the series. How does the plot change as you change theta 1?

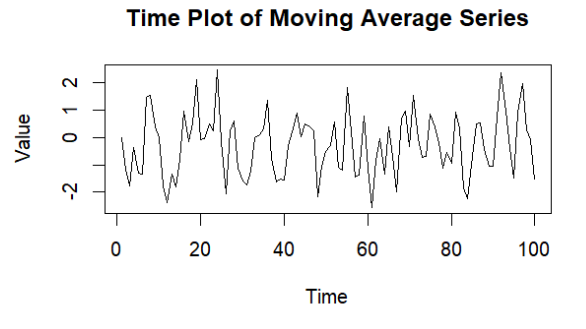
This is the plot.

```{r}

# Plot the time series

plot(y, type = "l", main = "Time Plot of Moving Average Series", xlab = "Time", ylab = "Value")

```



To answer the second part of this question, values 0.25, 1, and 1.5 were used again. The plots with these three variations of theta are shown below. The graphs did not appear to change in any significant way.

```{r}

set.seed(25)

e <- rnorm(100, mean = 0, sd = sqrt(1))

y <- ts(numeric(100))

for(i in 2:100) {

y[i] <- e[i] + 0.25 \* e[i-1]

}

plot(y, type = "l", main = "Time Plot of Moving Average Series - Vatiation 1", xlab = "Time", ylab

= "Value")

set.seed(25)

e <- rnorm(100, mean = 0, sd = sqrt(1))

y <- ts(numeric(100))

for(i in 2:100) {

y[i] <- e[i] + 1 \* e[i-1]

}

plot(y, type = "l", main = "Time Plot of Moving Average Series - Variation 2", xlab = "Time", ylab

= "Value")

set.seed(25)

e <- rnorm(100, mean = 0, sd = sqrt(1))

y <- ts(numeric(100))

for(i in 2:100) {

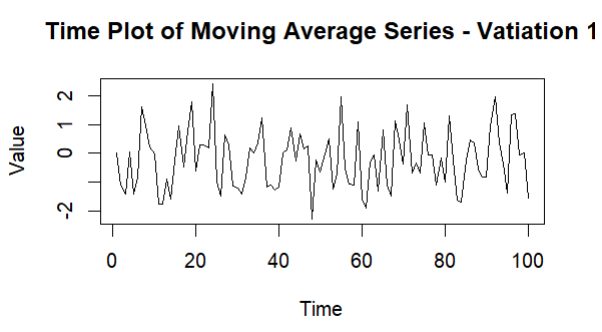
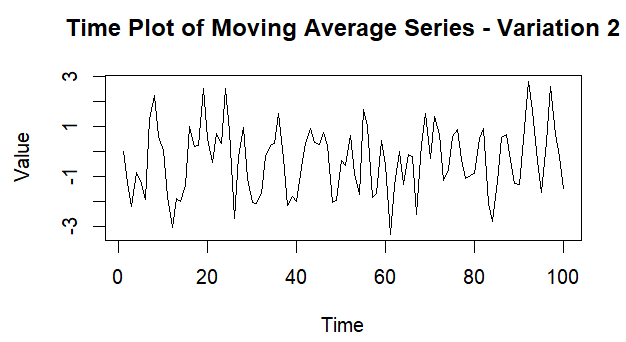
y[i] <- e[i] + 1.5 \* e[i-1]

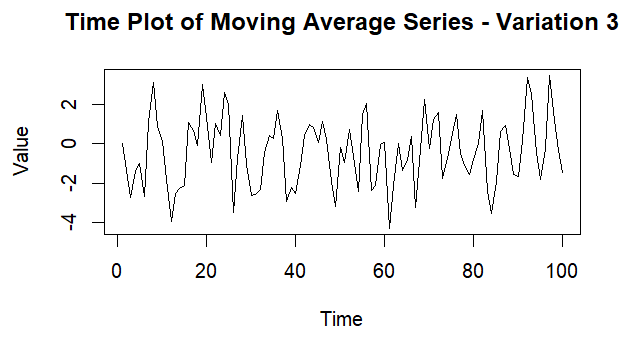
}

plot(y, type = "l", main = "Time Plot of Moving Average Series - Variation 3", xlab = "Time", ylab

= "Value")

```



## Generate data from an ARMA(1,1) model with Φ1 = 0.6, θ1 = -0.6, and variance = 1.

```{r}

set.seed(899)

e <- rnorm(100, mean = 0, sd = sqrt(1))

y <- ts(numeric(100))

for(i in 2:100) {

y[i] <- 0.6 \* y[i-1] + e[i] + -0.6 \* e[i-1]

}

```

## Generate data from an AR(2) model with Φ1 = -0.8, Φ2 = 0.3, and variance = 1. (These will give a non-stationary series.)

```{r}

set.seed(101)

# Generate the white noise errors with the specified variance

e <- rnorm(100, mean = 0, sd = sqrt(1))

# Initialize the time series

y2 <- ts(numeric(100))

# Generate the AR(2) series

for(i in 3:100) {

y2[i] <- -0.8 \* y2[i-1] + 0.3 \* y2[i-2] + e[i]

}

```

## Graph the latter two series and compare them.

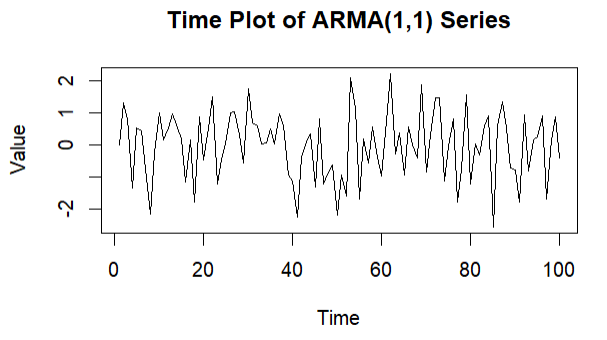
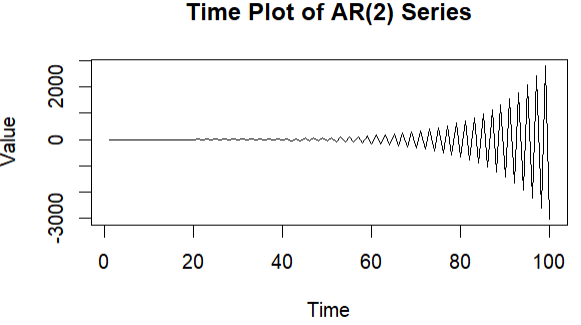
The ARMA(1,1) graph appears to be random data without any obvious pattern. The AR(2) graph appears to gradually increase in height of the values in both the negative and y directions, and it appears to look like a crescendo in music. The first graph is stationary while the second graph is not.

```{r}

plot(y, type = "l", main = "Time Plot of ARMA(1,1) Series", xlab = "Time", ylab = "Value")

plot(y2, type = "l", main = "Time Plot of AR(2) Series", xlab = "Time", ylab = "Value")

```

# HA 8.8

**Prompt:**

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

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

The auto.arima() function returns the best ARIMA model for a univariate time series. Here, I pass the austa time series object through the function.

Series: austa ARIMA(0,1,1) with drift

Coefficients: ma1 drift 0.3006 0.1735 s.e. 0.1647 0.0390

sigma^2 = 0.03376: log likelihood = 10.62 AIC=-15.24 AICc=-14.46 BIC=-10.57

The function suggests an ARIMA(0,1,1) model with drift. It has the following statistics:

* **Coefficients:**
  + MA1=0.3006 = 0.3006
  + Drift=0.1735 = 0.1735
* **Standard Errors:**
  + MA1=0.1647 = 0.1647
  + Drift=0.0390 = 0.0390

MA1 is the moving average of the model.

Now, I need to check if the residuals look like white noise. I do this by passing our model through the checkresiduals() function. This function performs a Ljung-Box test, as well as plots for the residuals.

A graph of residuals and drift

Description automatically generated

Ljung-Box test

data: Residuals from ARIMA(0,1,1) with drift Q\* = 2.297, df = 6, p-value = 0.8905

Model df: 1. Total lags used: 7

The Ljung-box test returned a p-value of 0.8905. This means that our model has no autocorrelation. This test follows the following assumptions, and we failed to reject the null:

* **Null Hypothesis (H0):** The residuals are independently distributed (i.e., there is no autocorrelation).
* **Alternative Hypothesis (H1):** The residuals are not independently distributed (i.e., there is autocorrelation).

The residuals appear to be normally distributed. The ACF plot shows no apparent pattern, and is not significant. This means that it is consistent with white noise and there is no autocorrelation. The time series of the residuals fluctuate around zero with no patterns.

Now, I will plot the forecast for the next 10 periods. For that, I use the forecast() function to create the forecast and autoplot() to plot:

A graph showing the growth of a car

Description automatically generated

The plot shows the forecasted values for austa, with the 95% prediction interval in dark blue and the 80% prediction interval in light blue.

## 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.

Now I have to forecast using ARIMA(0,1,1) without any drift and compare it to the previous forecast. To remove the drift, I set the argument include.drift=FALSE.

In the Arima() function, order follows the following structure: First term is the number of autoregressive terms, second term being the number of differencing terms, and the third being the moving average term.

A graph showing the growth of a number of vehicles

Description automatically generated with medium confidence

ARIMA(0,1,1) with no drift flat-lines (unlike with drift). This makes sense because without drift removes the difference of the previous data, which will remove the increasing trend.

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

The following is the plot of forecasted ARIMA (2,1,3) with drift.

A graph showing a graph of a forecast

Description automatically generated with medium confidence

This ARIMA(2,1,3) model gives us a forecast of upward trend. It is more reasonable and conservative than the plot in A, but has an increasing trend unlike the plot in B. This is probably because it uses more autoregressive and moving average terms compared to the other two.

Now I will remove the constants from this ARIMA model:

tryCatch({  
 c\_mod2 = Arima(austa, order=c(2,1,3), include.constant=FALSE)  
 c\_forecast2 = forecast(c\_mod2, h=10)  
 autoplot(c\_forecast2) + ggtitle("Forecasts from ARIMA(2,1,3) model without constant")  
}, error = function(e) {  
 print("ARIMA(2,1,3) without constant failed, trying a simpler model.")  
 c\_mod3 = Arima(austa, order=c(1,1,2), include.constant=FALSE)  
 c\_forecast3 = forecast(c\_mod3, h=10)  
 autoplot(c\_forecast3) + ggtitle("Forecasts from ARIMA(1,1,2) model without constant")  
})

Whenever I try running the ARIMA(2,1,3), the model gives an error and fails. After further research, I have learned that this ARIMA model is not appropriate for this data. This is because there are too many differencing terms. We need to try with a simpler model

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

The following is the ARIMA(0,0,1) with the constant:

A graph showing the growth of a number of forecasts

Description automatically generated

This pattern makes sense. Without a constant or drift, the model does not account for any trends or levels in the data. There is also now higher variability.

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

Now, I will plot the ARIMA(0,2,1) with no constant. Here we increase the middle term the differencing term by two:

A graph showing the growth of a forecast

Description automatically generated

This ARIMA model with two differencing terms accounts for and extrapolates the trend, resulting in an upward-sloping forecast with narrower prediction intervals. By differencing twice, the model effectively removes long-term trends and stabilizes the mean of the series. There is an upward trend in this model.

**Code for reference:**

# Load libraries

library(fpp2)

library(forecast)

data("austa")

a\_mod= auto.arima(austa)

print(a\_mod)

checkresiduals(a\_mod)

a\_forecast = forecast(a\_mod, h=10)

autoplot(a\_forecast) + ggtitle("Forecasts from auto.arima() model")+

annotate("text", x = 2017, y = max(a\_forecast$mean) \* 0.95,

label = "80% prediction interval", color = "darkblue", size = 4) +

annotate("text", x = 2017, y = max(a\_forecast$mean) \* 0.85,

label = "95% prediction interval", color = "darkblue", size = 4) +

theme(legend.position = "none")

b\_mod = Arima(austa, order=c(0,1,1), include.drift=FALSE)

b\_forecast = forecast(b\_mod, h=10)

autoplot(b\_forecast) + ggtitle("Forecasts from ARIMA(0,1,1) model with no drift")

c\_mod = Arima(austa, order=c(2,1,3), include.constant=TRUE, include.drift = TRUE)

c\_forecast = forecast(c\_mod, h=10)

autoplot(c\_forecast) + ggtitle("Forecasts from ARIMA(2,1,3) model with drift")

tryCatch({

c\_mod2 = Arima(austa, order=c(2,1,3), include.constant=FALSE)

c\_forecast2 = forecast(c\_mod2, h=10)

autoplot(c\_forecast2) + ggtitle("Forecasts from ARIMA(2,1,3) model without constant")

}, error = function(e) {

print("ARIMA(2,1,3) without constant failed, trying a simpler model.")

c\_mod3 = Arima(austa, order=c(1,1,2), include.constant=FALSE)

c\_forecast3 = forecast(c\_mod3, h=10)

autoplot(c\_forecast3) + ggtitle("Forecasts from ARIMA(1,1,2) model without constant")

})

d\_mod = Arima(austa, order=c(0,0,1), include.constant=TRUE)

d\_forecast = forecast(d\_mod, h=10)

autoplot(d\_forecast) + ggtitle("Forecasts from ARIMA(0,0,1) model with no constant")

e\_mod = Arima(austa, order=c(0,2,1), include.constant=FALSE)

e\_forecast = forecast(e\_mod, h=10)

autoplot(e\_forecast) + ggtitle("Forecasts from ARIMA(0,2,1) model with no constant")

# Works Cited:

Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. OTexts.com/fpp3. Accessed June 2024.

Kuhn, Max, et al. Applied Predictive Modeling. New York, Springer, 2016.