Homework Set 1

DATA 624-01 Group 3

Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam

**Good Job. See Comments and Notes in-line in the document.**

**I would have liked to see more commentary. While you have a couple of nice features like the ToC and notations, in general, your HW was the hardest to grade and there were a few issues, again see in-line. I recommend you getting a copy of group 2 or maybe Group 1 and taking a look for ideas.**

**I am grading leniently this time, setting expectations - Report Grade is 92.**

**Table of Contents**

[**HA 2.1**](#_rbuniq85zz2q) **2**

[**HA 2.2**](#_klogzcw8n5bk) **5**

[**HA 6.2**](#_g0cn60fzxbd2) **8**

[**HA 7.1**](#_hu74ev55i1dy) **15**

[**HA 7.2**](#_wg0tabikqnmj) **18**

[**HA 7.3**](#_9yzokkteddtw) **18**

[**HA 8.1**](#_t1b03unc375l) **19**

[**HA 8.2**](#_gmrsgp6h544m) **21**

[**HA 8.6**](#_ac9c591ldyp3) **21**

[**KJ 3.1**](#_3znysh7) **25**

[**KJ 3.2**](#_2et92p0) **29**

[**HA 8.8**](#_tbhwyl6t9ha) **30**

**Key:**

Homework problems are stated in blue text

Answers are in black text

# HA 2.1

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

The data is stored in an R package called forecast. First, we import the package as shown in the first line of code with the function ‘library(forecast)’. We then review the vignette provided for each data set within the package using the help function. This will describe each data set in R. After reviewing each vignette, the results are interpreted.

```{r}

library(forecast)

help(gold, package = forecast)

tsdisplay(gold)

help(woolyrnq, package = forecast)

tsdisplay(woolyrnq)

help(gas, package = forecast)

tsdisplay(gas)

```

The gold data set contains the daily morning gold prices in US dollars from 1 January 1985 - 31 March 1989. The woolyrng data set contains quarterly production of woollen yarn in Australia measured in tonnes from March 1965 - September 1994. Lastly, the gas data set contains Australian monthly gas production from 1956-1995.

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

Our approach to this is to simply plot each data set within the parentheses of the autoplot() function. Separated plots are shown in the order gold, woolyrnq, and gas.

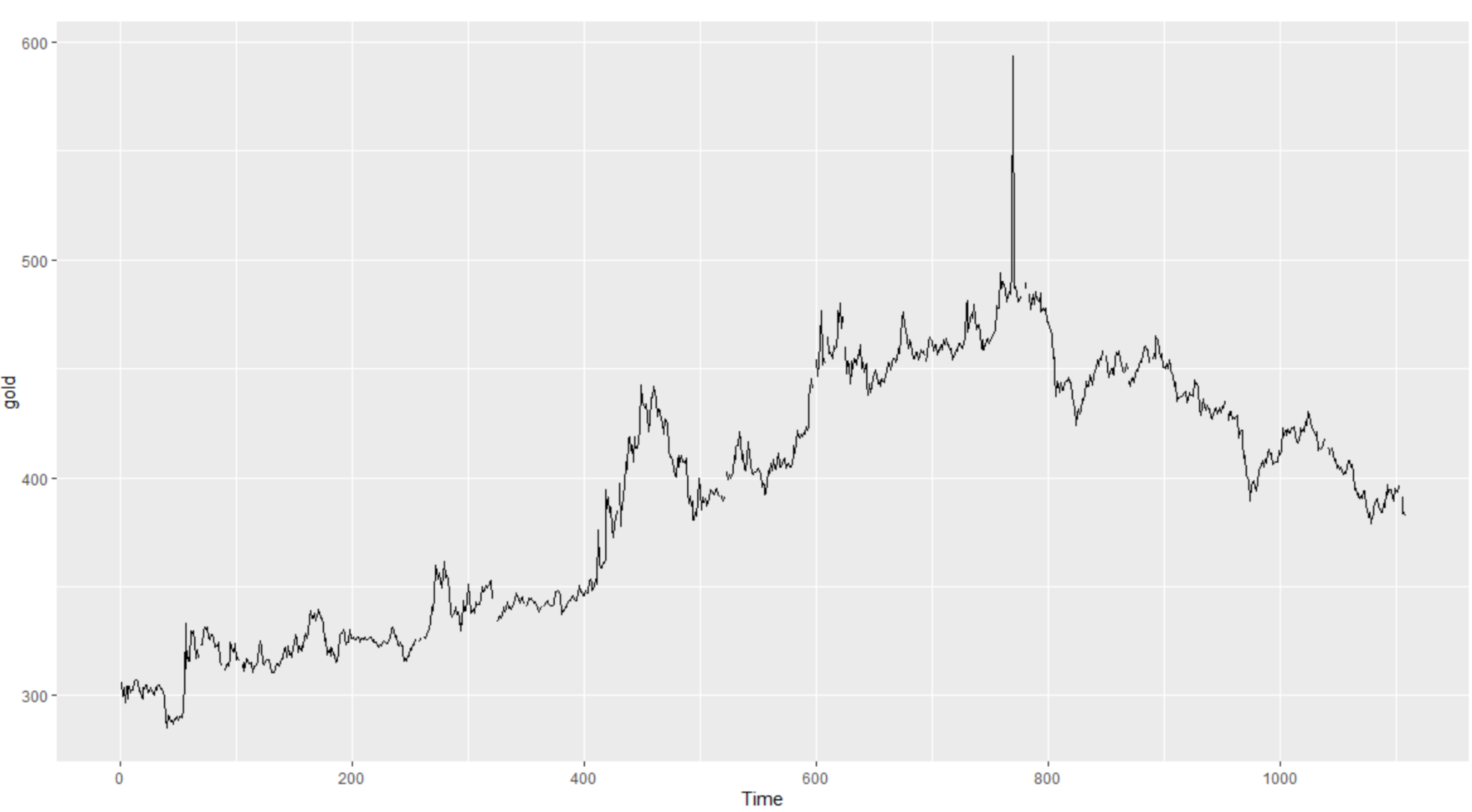
```{r}

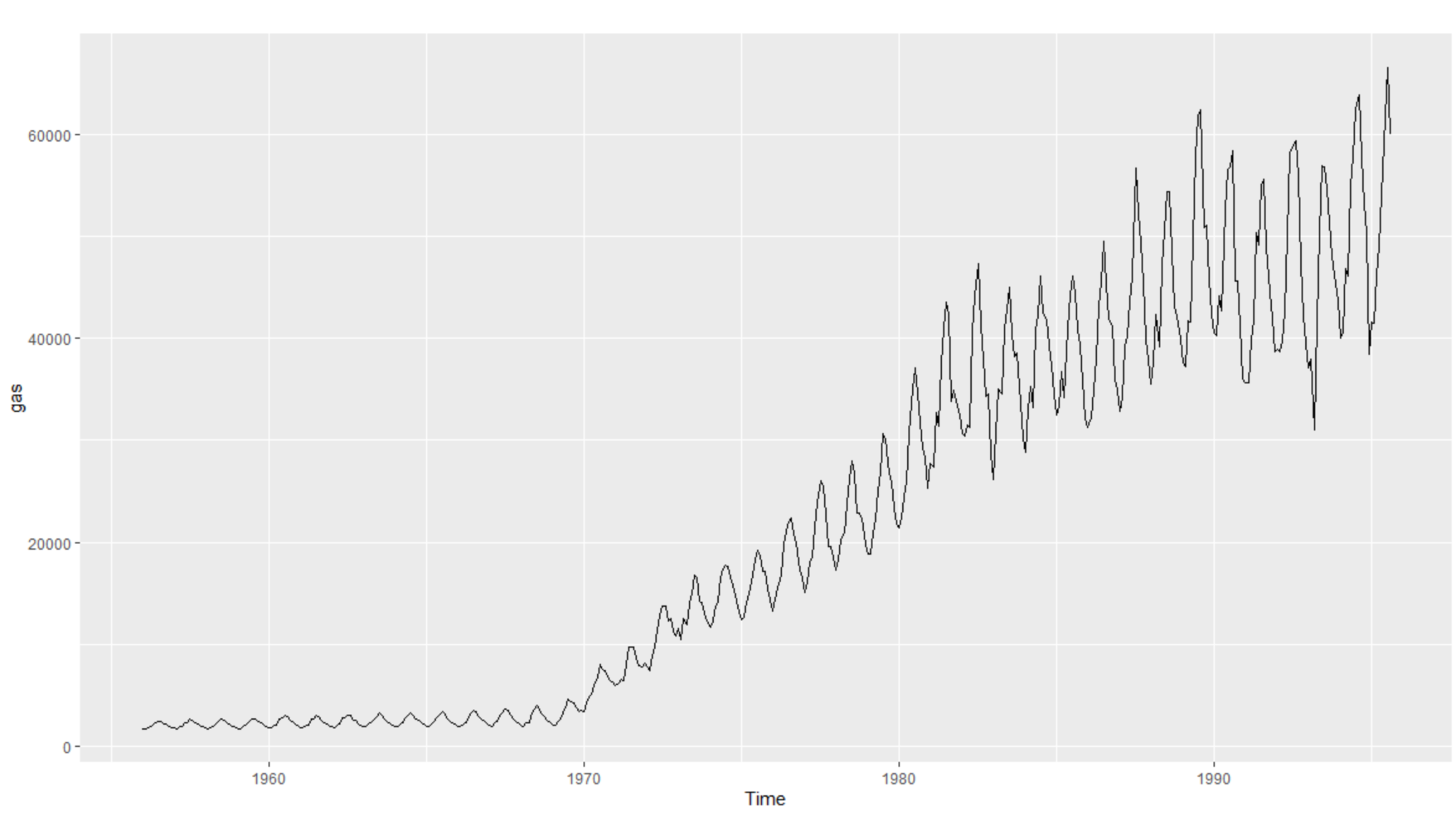
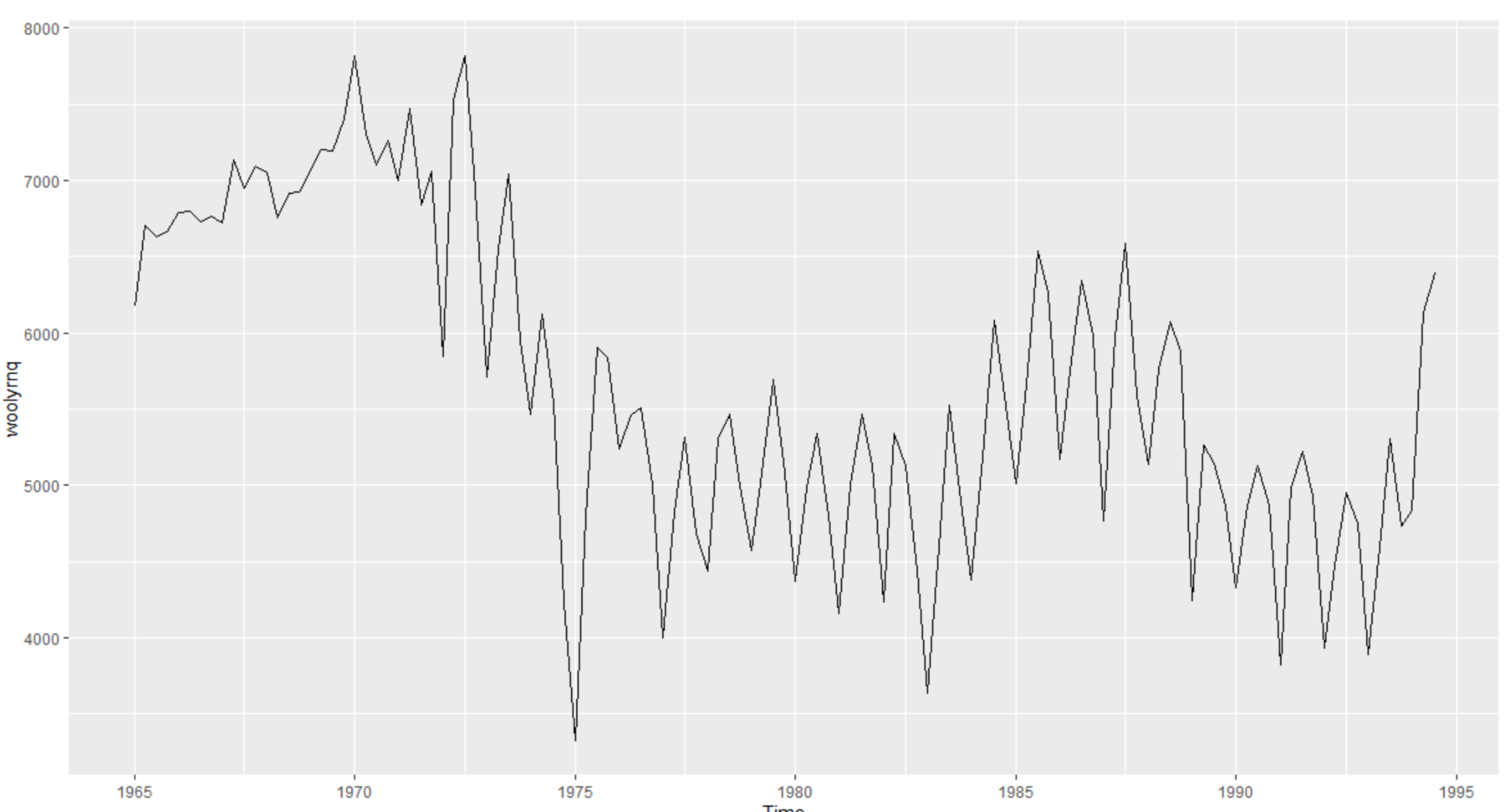
autoplot(gold)

autoplot(woolyrnq)

autoplot(gas)

```





B) What is the frequency of each series?

In this case, we simply apply the frequency function to each data set and record the results.

```{r}

frequency(gold)

frequency(woolyrnq)

frequency(gas)

```

Gold has a frequency of 1, woolyrng of 4, and gas of 16.

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

The function which.max() will display the index value of the maximum value in the gold series. In this case it also happens to be the outlier we are looking for. We can also use this index value to find the price of gold based on this data set.

```{r}

which.max(gold)

gold[770]

```

The observation at 770 is an outlier when the daily gold price was $593.7.

# HA 2.2

Download the file tute1.csv from the book website, open it in Excel (or some other spreadsheet application), and review its contents. You should find four columns of information. Columns B through D each contain a quarterly series, labelled Sales, AdBudget and GDP. Sales contains the quarterly sales for a small company over the period 1981-2005. AdBudget is the advertising budget and GDP is the gross domestic product. All series have been adjusted for inflation.

The following package is used to generate plots.

```{r}

library(forecast)

```

A) You can read the data into R with the following script:

For this, we downloaded the data once then reuploaded it to Github to host it openly. Now, anyone can access it freely through a link.

```{r}

tute1 <- read.csv("https://raw.githubusercontent.com/palmorezm/msds/main/624/tute1.csv")

```

This method works for any computer with internet access.

B) Convert the data to time series

After looking at the data we know the frequency of it is 4. We apply this in the function ts() from the forecast package. We also set the starting date of the time series object to 1981 and remove the index column we do not need.

```{r}

myts <- ts(tute1[,-1], start=1981, frequency=4)

```

This took the tute data set, selected values from 1981 onwards in the series at the same frequency as the original tute data to form a new data set that is stored as a time series object.

C) Construct time series plots of each of the three series. Check what happens when you don't include facets=TRUE.

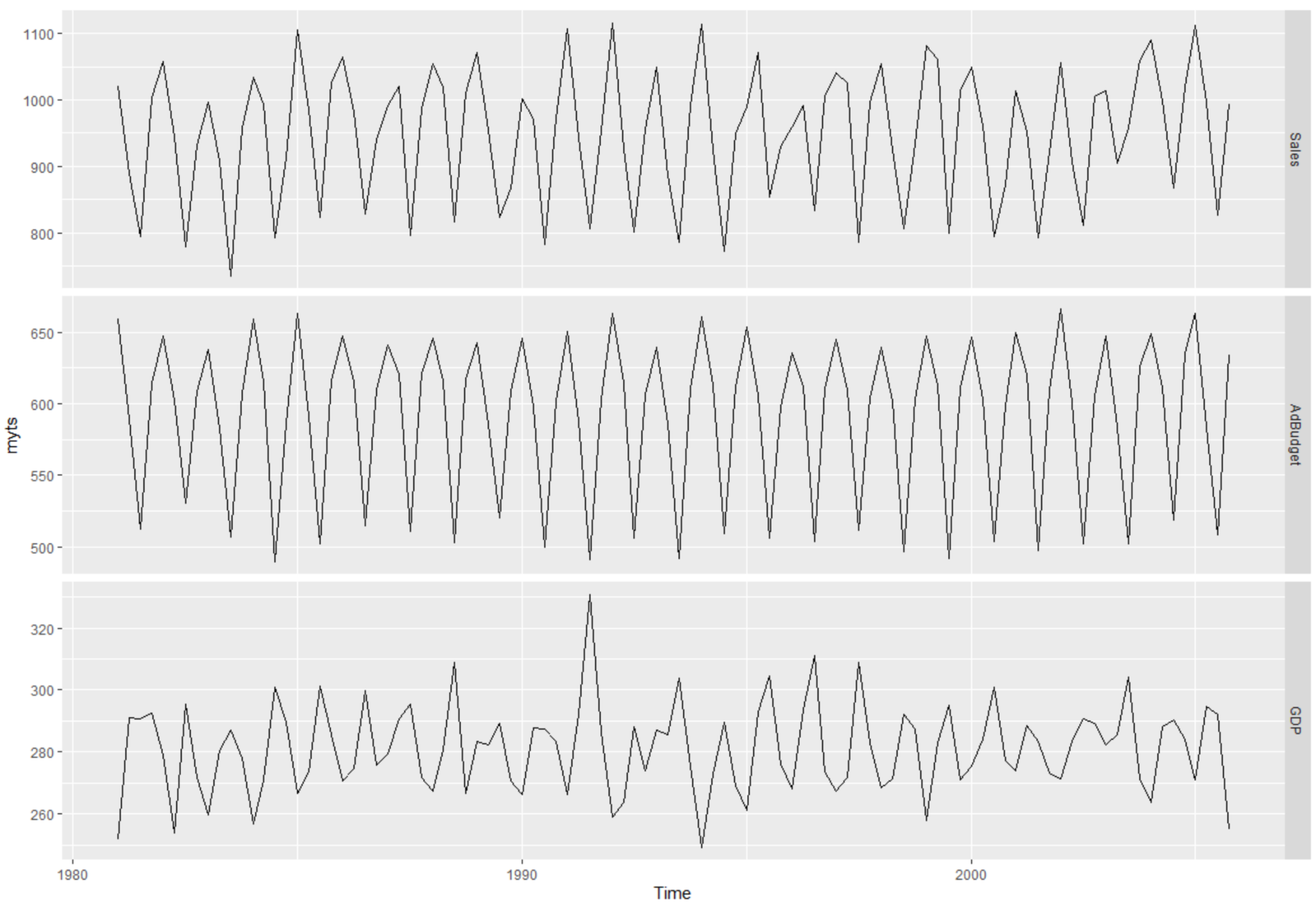
We construct the time series plots with the autoplot() function. We change the facets argument to true or false and compare. Both results are displayed for observation with facets=TRUE appearing first.

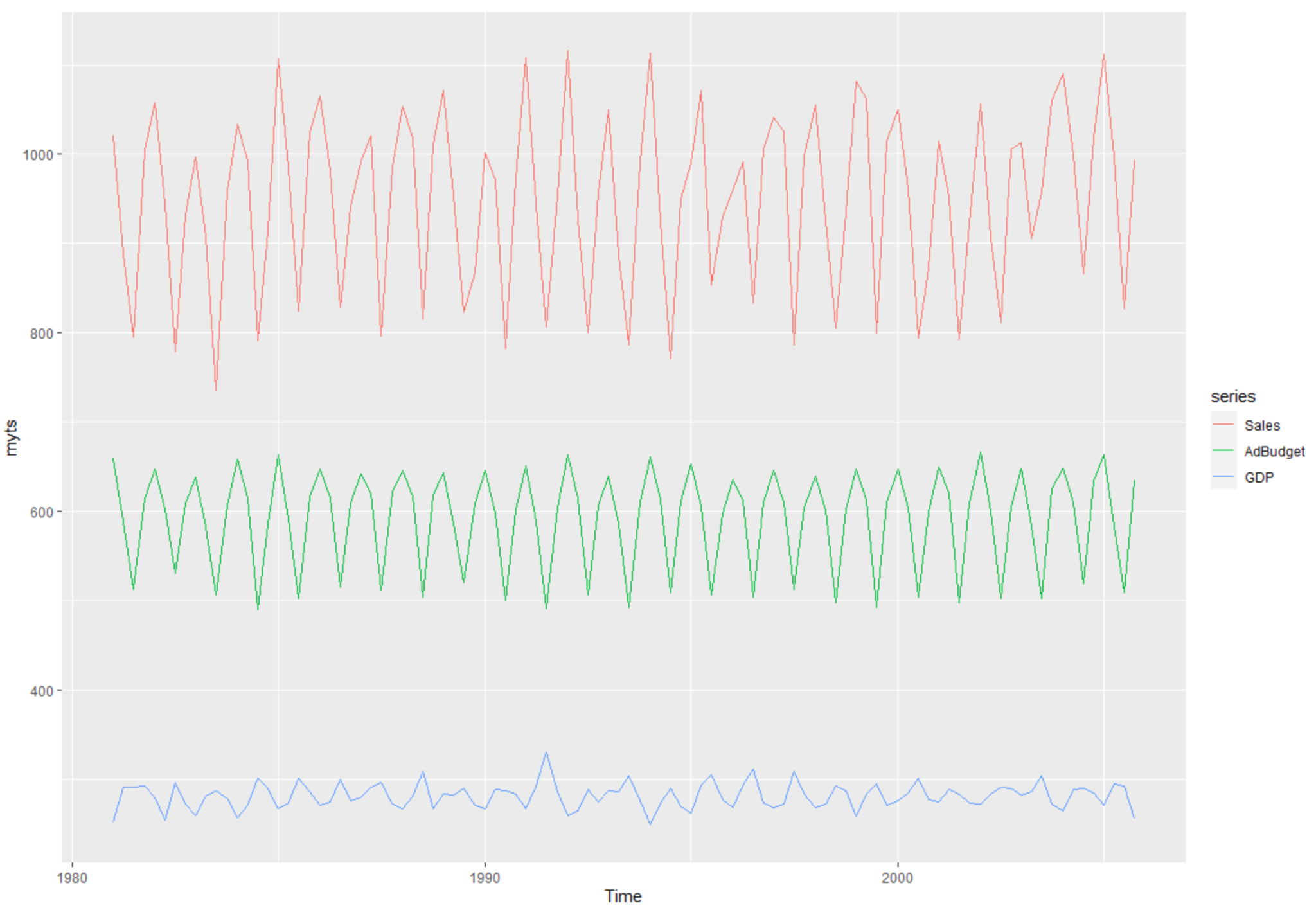
```{r}

autoplot(myts, facets=TRUE)

autoplot(myts, facets=FALSE)

```





Each series is plotted on the same plot with the same axes when facets=FALSE. Otherwise, facets=TRUE displays each series as its own plot with its own axis.

# HA 6.2

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

The following packages are used.

```{r}

library(forecast)

library(fma)

library(dplyr)

library(seasonal)

```

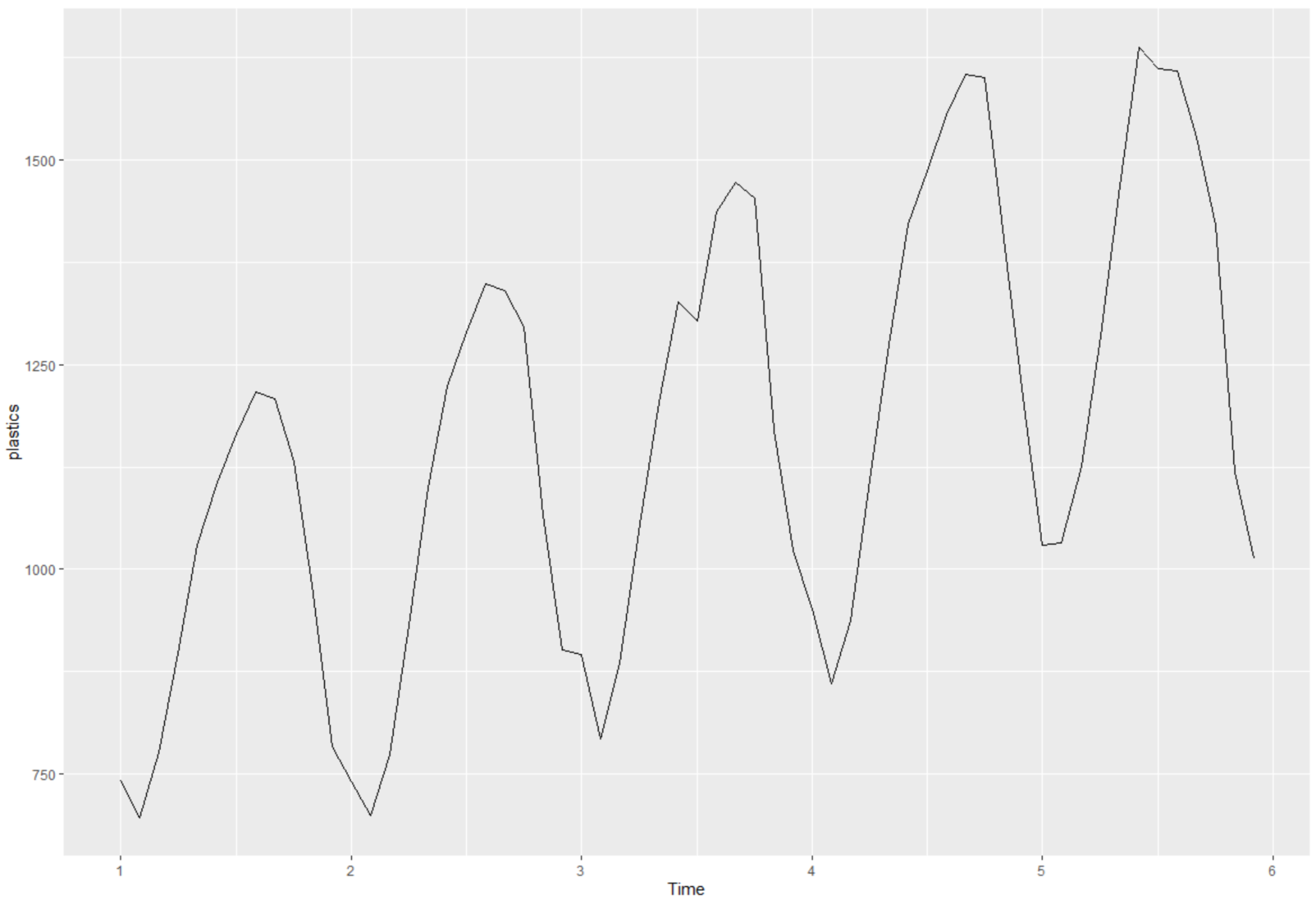
A) Plot the time series of sales of product A. Can you identify seasonal fluctuations and/or a trend-cycle?

We use the autoplot() function of the to display the time series of sales of product A. If there are seasonal fluctuations we would notice regular intervals of some kind where sales tend to increase or decrease at the same times within each interval.

```{r}

autoplot(plastics)

```



Yes there are clear seasonal variations with regular intervals on an annual cycle. There is also a clear upward trend from cycle start to cycle end.

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

This classical multiplicative decomposition can be calculated with the decomposition() function. We will then plot the results to observe the trends and seasonal changes.

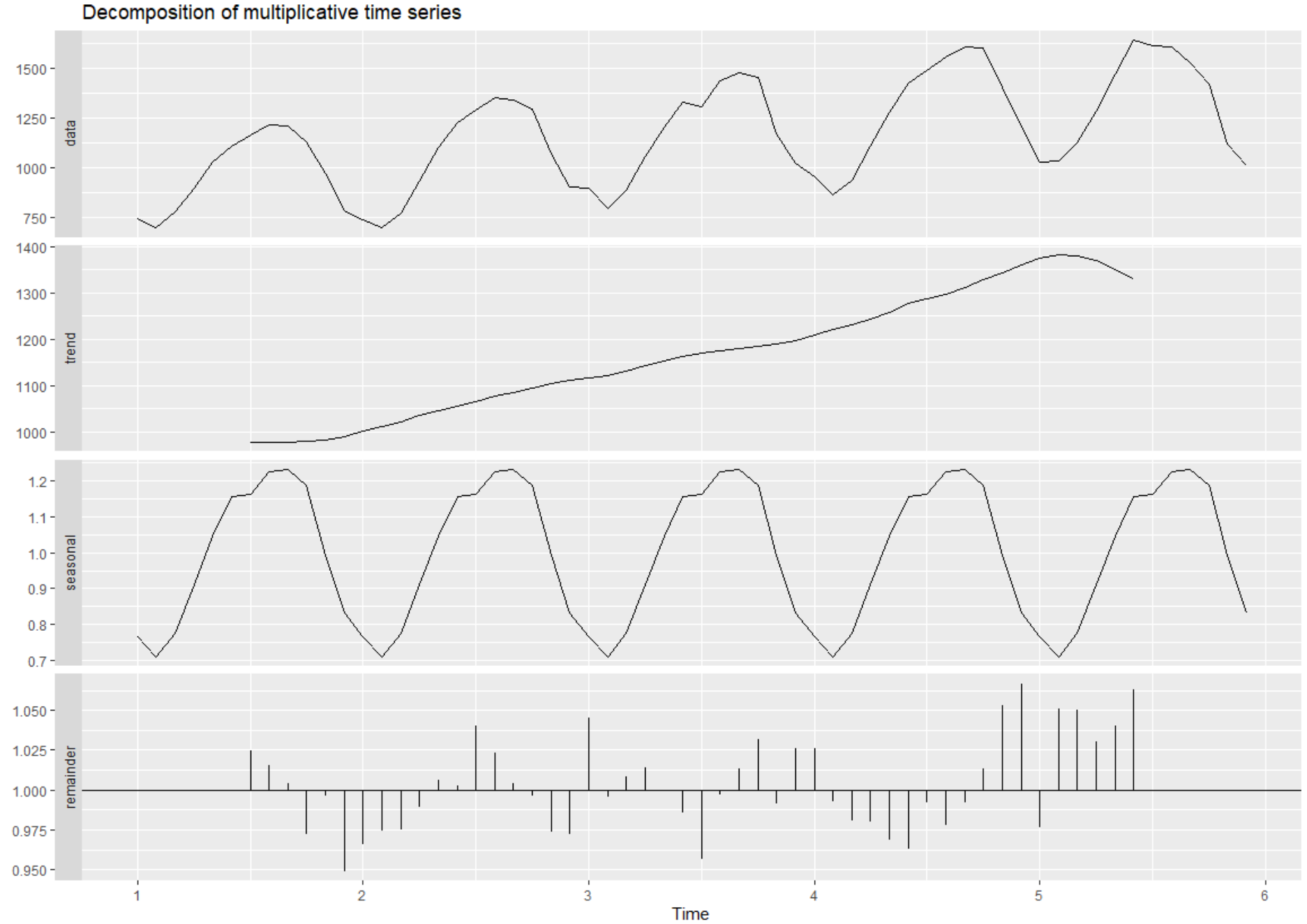
```{r}

plastics %>%

decompose(type='multiplicative') %>%

autoplot()

```



By using decomposition, we can see the breakdown of the original data compared with its overall trend, seasonality, and remainder patterns. There is a clear positive upward trend in sales and near constant seasonality.

C) Do the results support the graphical interpretation from part a?

We will create another seasonal plot of the sales made by month with axes labeled accordingly. We then analyze the results visually to determine if this plot, made with the seasonplot() function, supports our graphical interpretation from the autoplot() made in part ‘A’.

```{r}

seasonplot(plastics,

ylab="Sales (Thousands)",

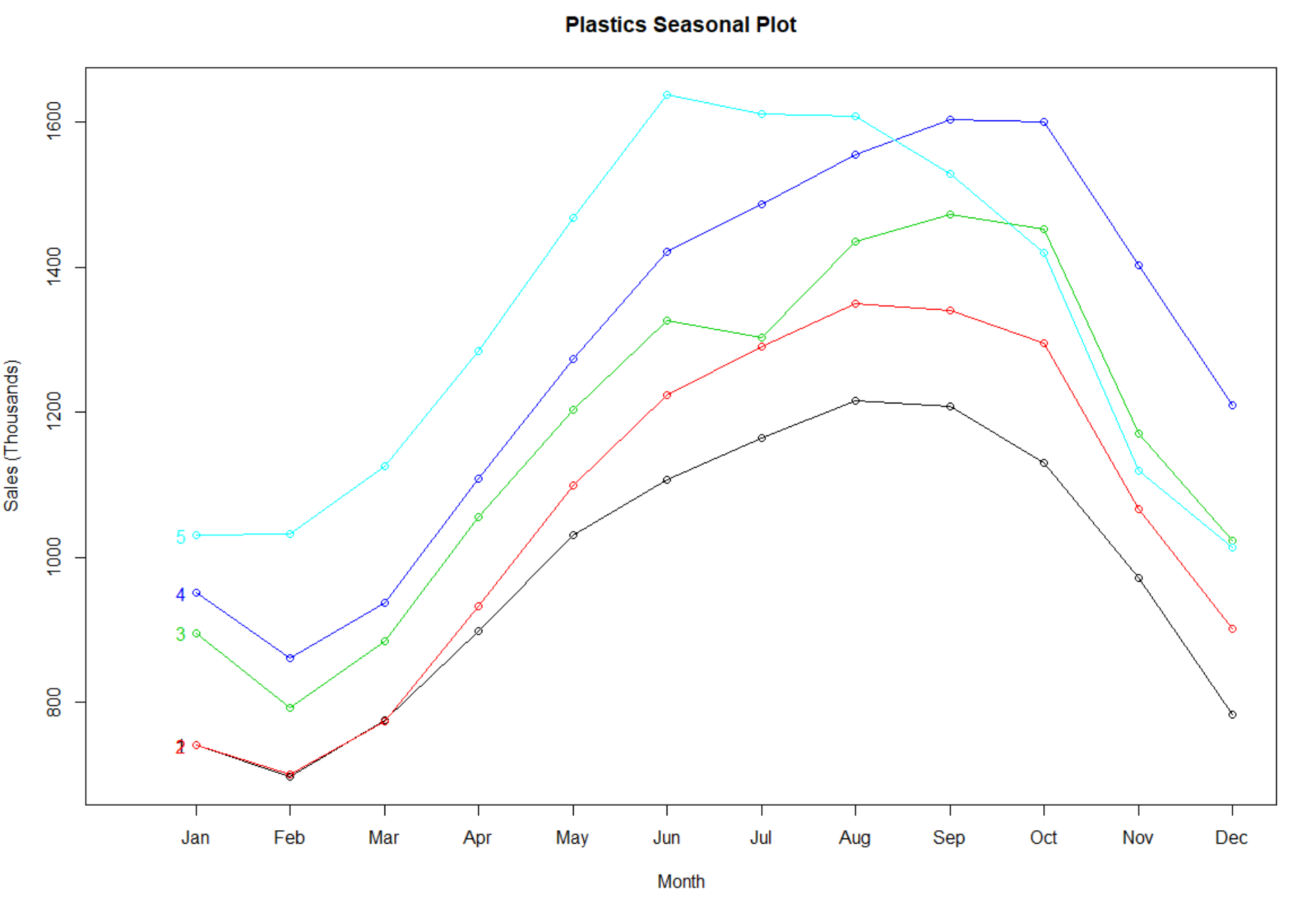
xlab="Month",

year.labels.left=TRUE,

main="Plastics Seasonal Plot",

col = 1:5)

```



Yes, our statement is confirmed. There is a clear positive trend upward in sales with clear seasonal variation over time. The only unknown is the remainder, which we did not describe previously but also seems to loosely follow the same seasonal variations until the most recent seasons.

D) Compute and plot the seasonally adjusted data.

To compute we will decompose the plastics data set using the same classical multiplicative process and store the results. This data will then be plotted using the autoplot() function on the stored data with an overlay of seasonally adjusted data using the autolayer() function to compare results.

```{r}

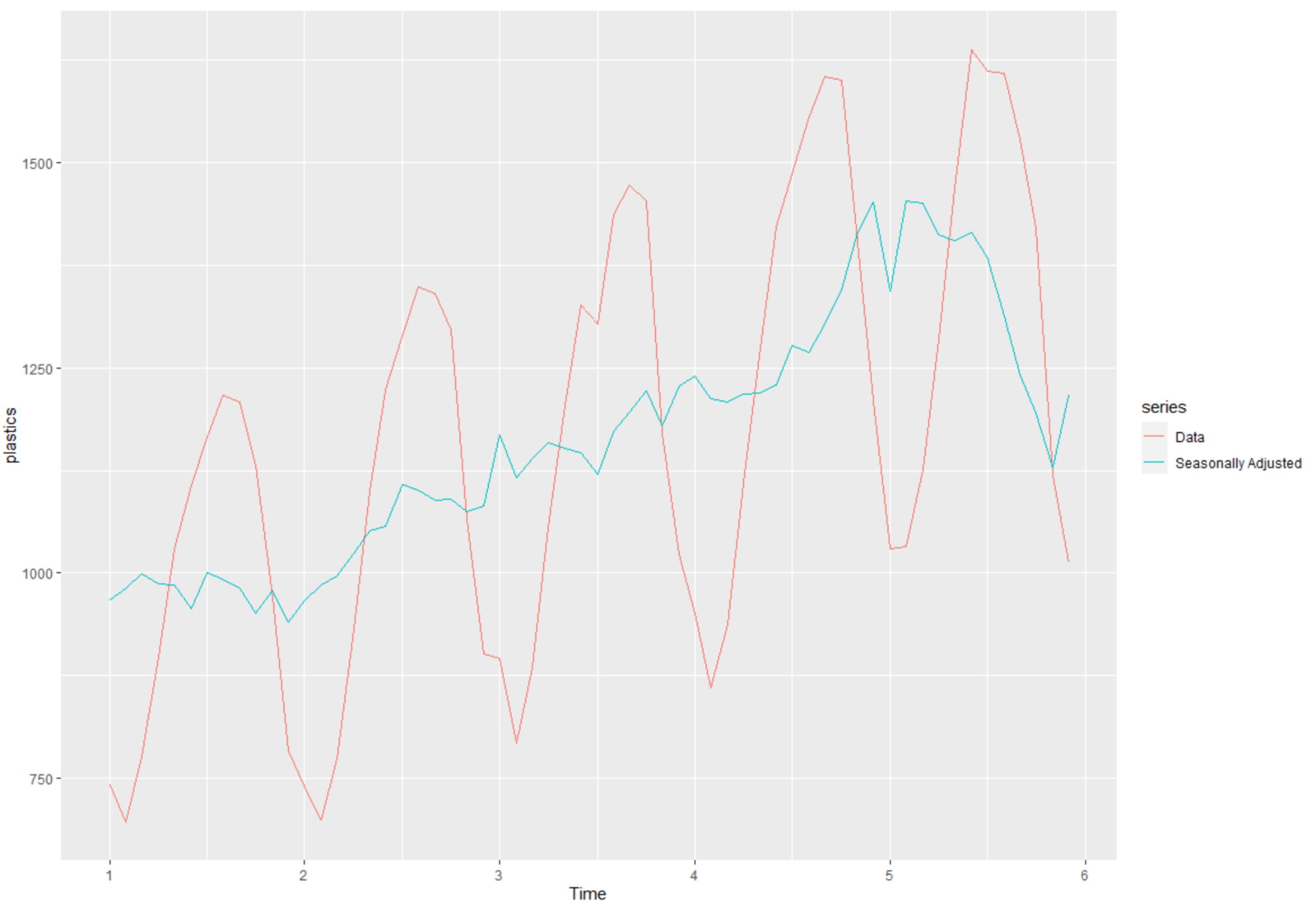
seasons <- plastics %>%

decompose(type='multiplicative')

autoplot(plastics, series='Data') +

autolayer(seasadj(seasons), series='Seasonally Adjusted')

```



Seasonal adjustments temper the variation in the data and help show the overall trend.

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

To keep it simple, we use a randomly selected index value near the middle of the data set and add 500 to it to make it an outlier. Then we reassign the outlier value to the plastics data set and plot for visualization. We observe the effect of the outlier through an autoplot().

```{r}

plastics[26] <- plastics[26] + 500

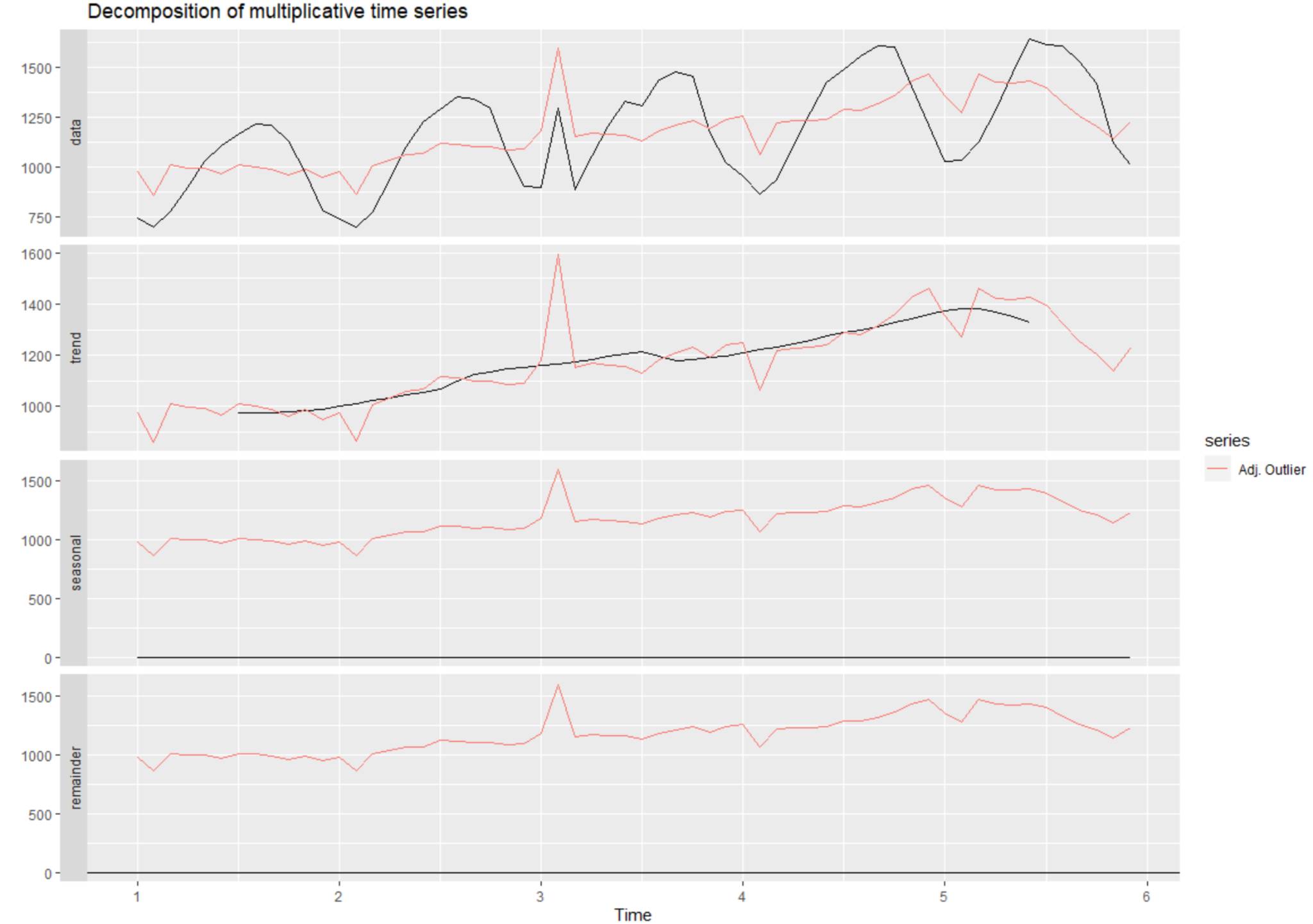
dec2 <- plastics %>%

decompose(type='multiplicative')

autoplot(dec2, series='Data') +

autolayer(seasadj(dec2), series='Adj. Outlier')

```



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

To determine how much influence an outlier at the ends compared to the middle of series we create two additional plots with the autoplot() and classical multiplicative decomposition. In the first plot we will add 500 to a value at the beginning of the series. Then, in the next plot, we will 500 to a random value near the end of the series. We complete this in two chunks starting with an outlier placed at the beginning of the series.

```{r}

plastics[2] <- plastics[2] + 500

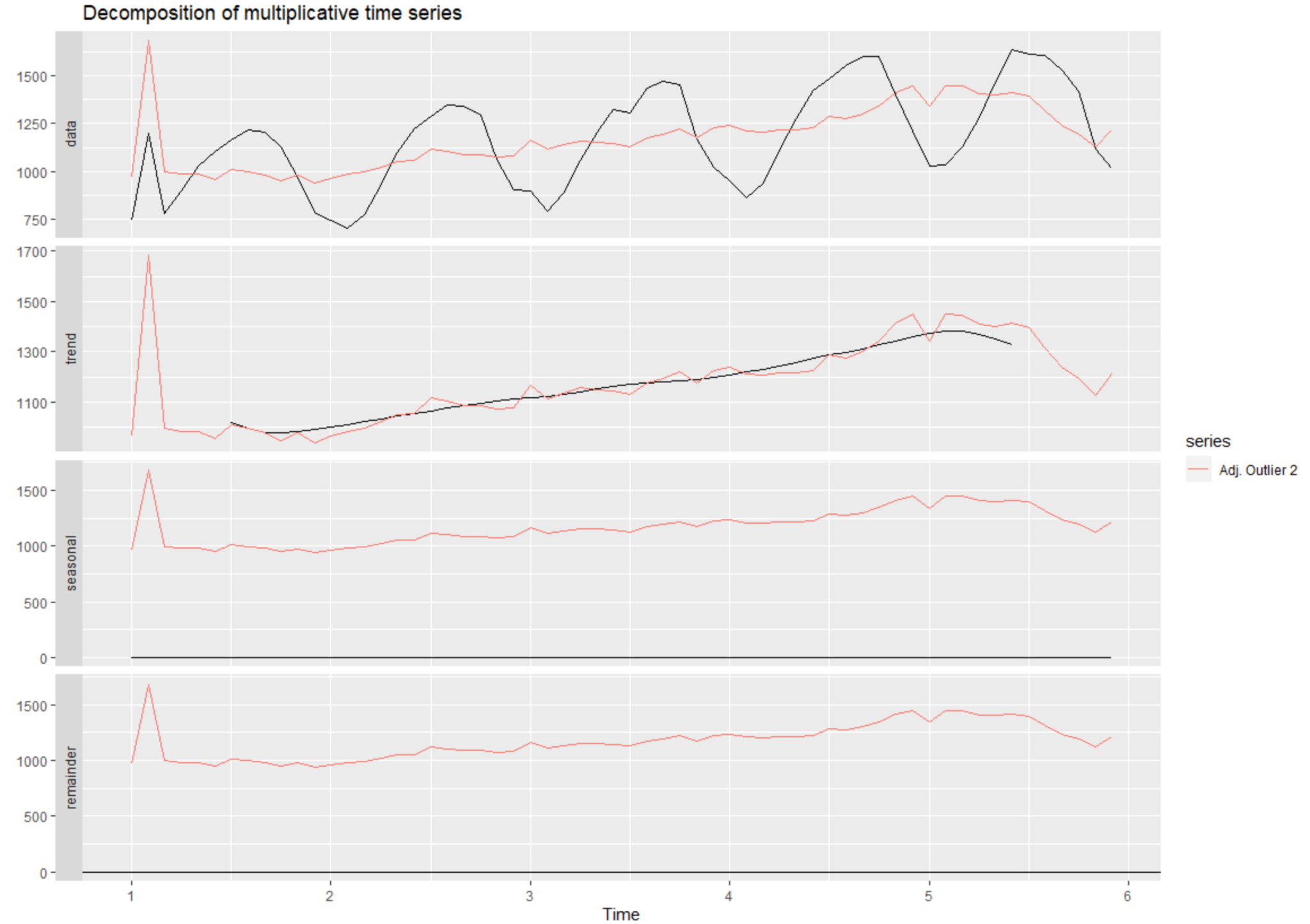
dec3 <- plastics %>%

decompose(type='multiplicative')

autoplot(dec3, series='Data') +

autolayer(seasadj(dec3), series='Adj. Outlier 2')

```



Then we add an outlier near the end of the series and plot again.

```{r}

plastics[58] <- plastics[58] + 500

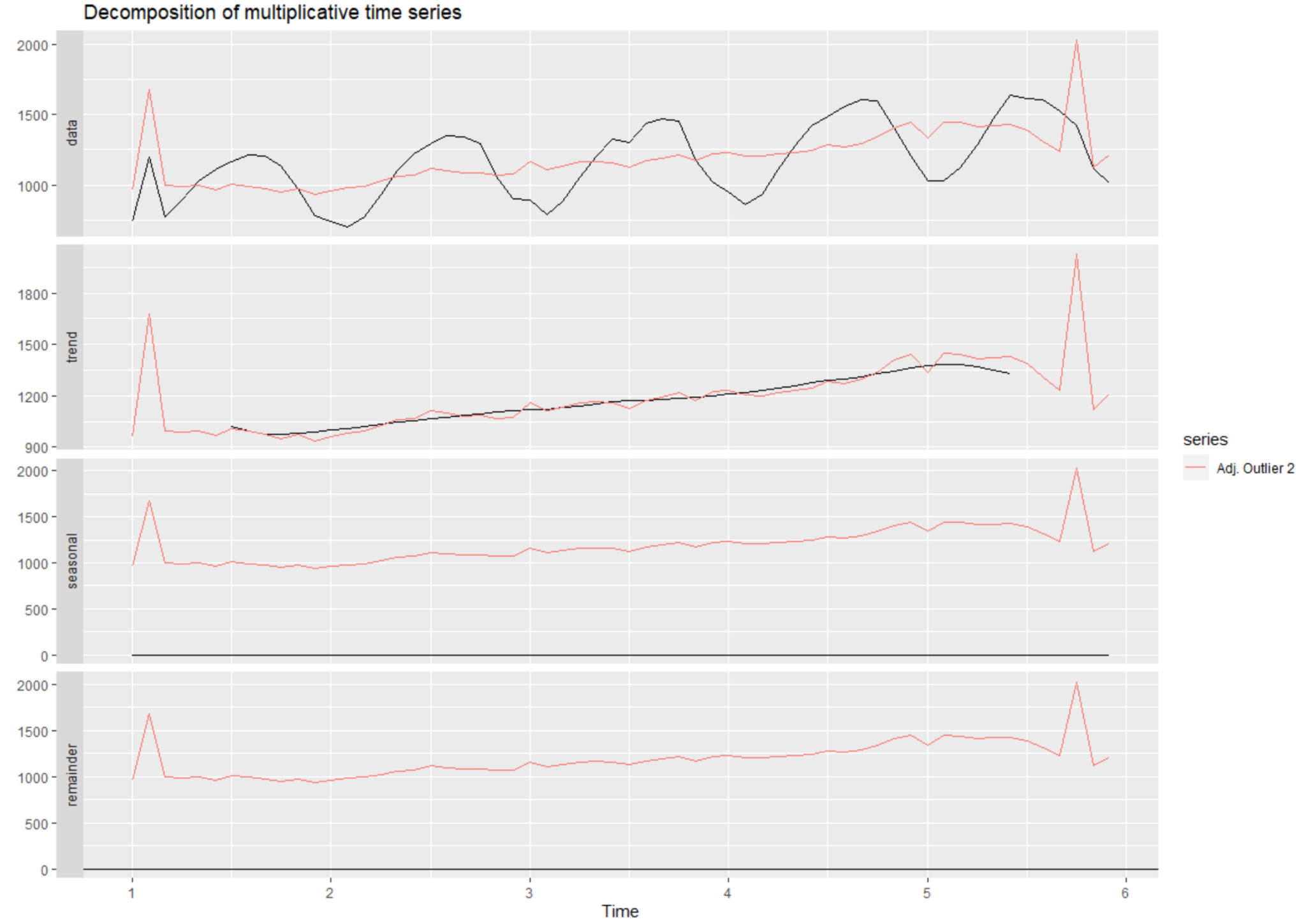
dec4 <- plastics %>%

decompose(type='multiplicative')

autoplot(dec3, series='Data') +

autolayer(seasadj(dec4), series='Adj. Outlier 2')

```



Outliers near the beginning and end of the series appear to have a greater influence on the overall trend and seasonality of the data. When adjusting for seasonality this is particularly apparent since an outlier towards either end of the data set causes the data to skew wildly far off from its original value. However, it is also clear that this data is sensitive to extreme values. Any outlier appears to skew the trend far from its expectation and nearby data points.

# HA 7.1

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

These packages are used.

```{r}

library(forecast)

library(fpp2)

```

With the help() function, we can find out that this is a series of Monthly total number of pigs slaughtered in Victoria, Australia from Jan 1980 to Aug 1995. We plot the series as below to have a better understanding of the series. According to the plot, it seems to have some seasonality, but no trend.

```{r}

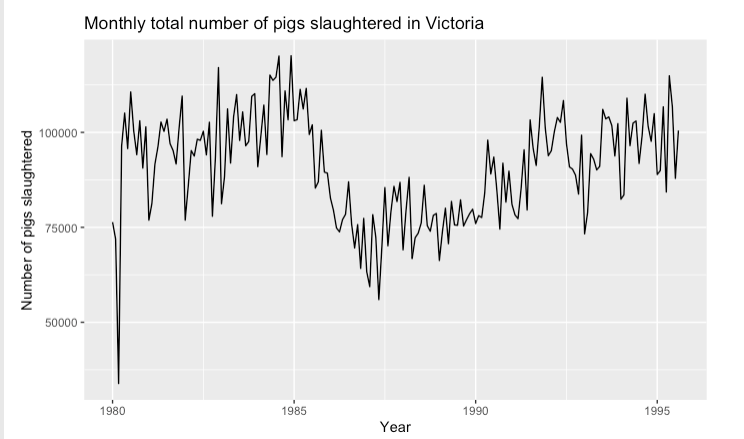
autoplot(pigs)+

xlab("Year")+

ylab("Number of pigs slaughtered")+

ggtitle("Monthly total number of pigs slaughtered in Victoria")

```



## A) Use the ses() function in R to find the optimal values of α and ℓ0, and generate forecasts for the next four months.

```{r}

fc\_pigs<-ses(pigs, h=4)

summary(fc\_pigs)

```

With the summary of simple exponential smoothing, alpha = 0.2971, and ℓ0 =77260.0561

The forecast from September 1995 to December 1995, the forecast gives the flat number of 98816.41 pigs for the next 4 months.

# 

## B) Compute a 95% prediction interval for the first forecast using y ± 1.96s where ‘s’ is the standard deviation of the residuals. Compare your interval with the interval produced by R.

```{r}

fc\_pigs

sdr<-sd(fc\_pigs$residuals)\*1.96

manual\_lo95<-fc\_pigs$mean[1] - sdr

manual\_hi95<-fc\_pigs$mean[1] + sdr

lo\_hi95<-c(manual\_lo95, manual\_hi95)

lo\_hi95

autoplot(fc\_pigs)+

autolayer(fc\_pigs$fitted, series = "Fitted",PI=F)+

ylab("Number of pigs slaughtered")+

xlab("Year")

```

To compute the 95% of intervals, we use the formula: . With manual calculation, the results are 78679.97 and 118952.84 which are very close to the output from the ses() function of 78611.97 and 119020.8.

TRY

upper = pigs\_ses$mean[1] + pigs\_resd\*1.96  
lower = pigs\_ses$mean[1] - pigs\_resd\*1.96  
  
print(paste("95% CI: ", lower , " --- ", upper))

## [1] "95% CI: 78679.9672534162 --- 118952.844969765"

# HA 7.2

Write your own function to implement simple exponential smoothing. The function should take arguments y (the time series), alpha (the smoothing parameter α) and level (the initial levelℓ0 ). It should return the forecast of the next observation in the series. Does it give the same forecast as ses() ?

According to the math formula, yt+1|t = αyt + (1 − α)yt|t−1, we can build a function as below.

We already found the Alpha and the initial level by summarizing the ses() function. Alpha is 0.2791 and the initial level is 77260.05. Therefore, we can plug these two numbers into the below function to forecast manually.

```{r}

manual\_ses <- function(y, alpha, initiate){

y\_hat <- initiate

for(index in 1:length(y)){

y\_hat <- alpha\*y[index] + (1 - alpha)\*y\_hat

}

return(as.character(y\_hat))

}

summary(fc\_pigs)

manual\_ses(pigs,0.2971,77260.0561)

```

Finally, our manually calculated forecast is 98816.45 from above function which is almost the same as the ses() forecasted of 98816.41.

# HA 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 ℓ0 . Do you get the same values as the ses() function?

We can define the sum of squared errors as subtracting the predicted value from the observed value, square them, then sum up these values/residuals. The function is modified to return the SSE value. Finally, using the optim() function to compare the results of alpha and initial levels with ses() outputs. The results are tested with pigs and a10 serieses to verify whether the modification works.

```{r}

fc\_a10<-ses(a10)

SES <- function(pars = c(alpha, l0), y){

error <- 0

SSE <- 0

alpha <- pars[1]

l0 <- pars[2]

y\_hat <- l0

for(index in 1:length(y)){

error <- y[index] - y\_hat

SSE <- SSE + error^2

y\_hat <- alpha\*y[index] + (1 - alpha)\*y\_hat

}

return(SSE)

}

opt\_pigs <- optim(par = c(0.5, pigs[1]), y = pigs, fn = SES)

as.character(opt\_pigs$par[1]) #alpha = 0.299

as.character(opt\_pigs$par[2]) # initial level = 76379.365

summary(fc\_pigs) # alpha = 0.297, initial level= 77260.056

opt\_a10 <- optim(par = c(0.5, a10[1]), y = a10, fn = SES)

as.character(opt\_a10$par[1]) # alpha = 0.359

as.character(opt\_a10$par[2]) # initial level = 3.464

summary(fc\_a10) # alpha = 0.358, initial = 3.465

```

The results from the modified function are very similar to the outputs from the ses() function. For the pigs series, the modified function gives alpha as 0.299 and initial level as 76379.36, compared to the ses() output of 0.297 and 76379.36. For the a10 series, the modified function gives alpha as 0.359 and initial level as 3.464 compared to the ses() output of 0.358 and 3.465.

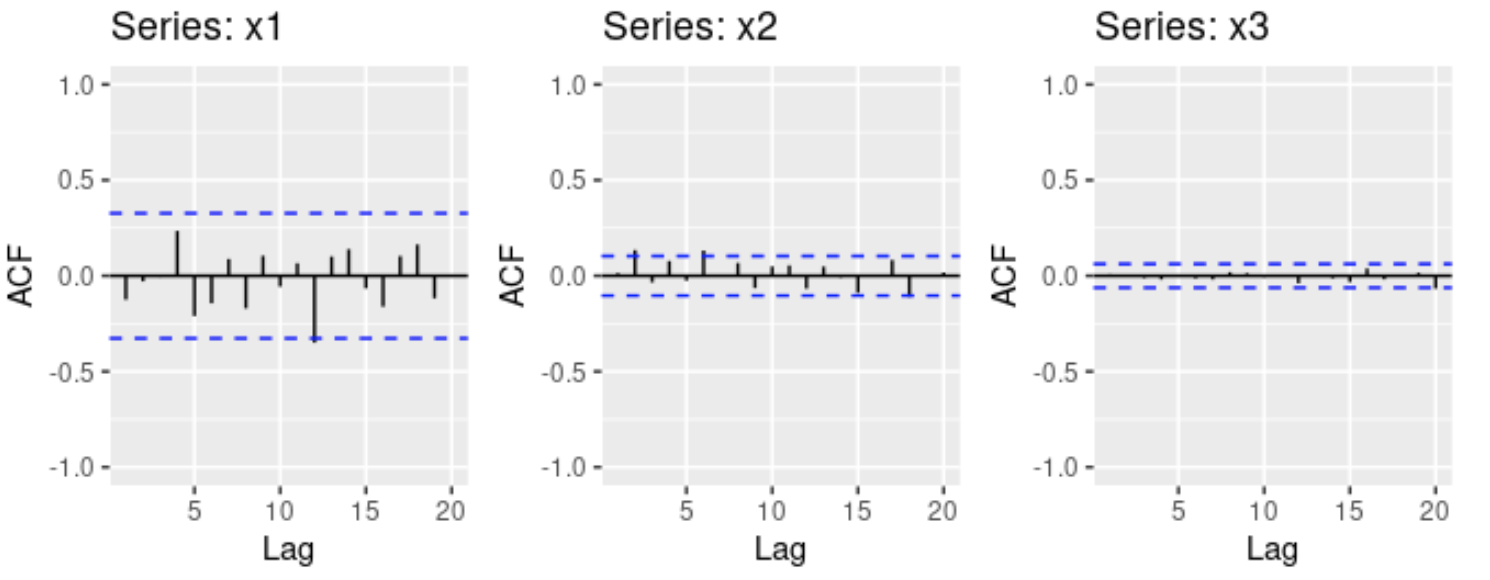
Overall, the modification works, and it provides almost identical results of alpha and initial levels.

# HA 8.1

Figure [8.31](https://otexts.com/fpp2/arima-exercises.html#fig:wnacfplus) 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?

Each plot shows the white noise autocorrelation for randomly generated numbers with a lag of 20. The correlations at each point are both positive and negative and appear to be randomly distributed which is representative of white noise. We also notice as the sample size increases from x1 → x3 we see the values of ACF approach zero and the critical value interval become tighter.



## 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 value interval decreases in size as the sample size increases because the equation for calculating the critical values is . As the length of the time series T increases the interval will get smaller. White noise should be completely random and have no correlation from point to point. When the sample size is small like 36 (x1), the confidence interval is greater and so the ACF plot may show some correlation that is not actually present in the true distribution. As the sample size increases to 1000 (x3), the confidence level is much tighter and can see the white noise is truly random with no correlation.

# 

# HA 8.2

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

First we need to install and import the fma and forecast libraries. The fma library contains the ibmclose dataset we are using in this problem and the forecast library is used to create the difference, ACF, and PACF plots. The image in the right shows the plots before differencing the data, the image in the right are plots after the raw data has been differenced.

```{r}

library(fma)

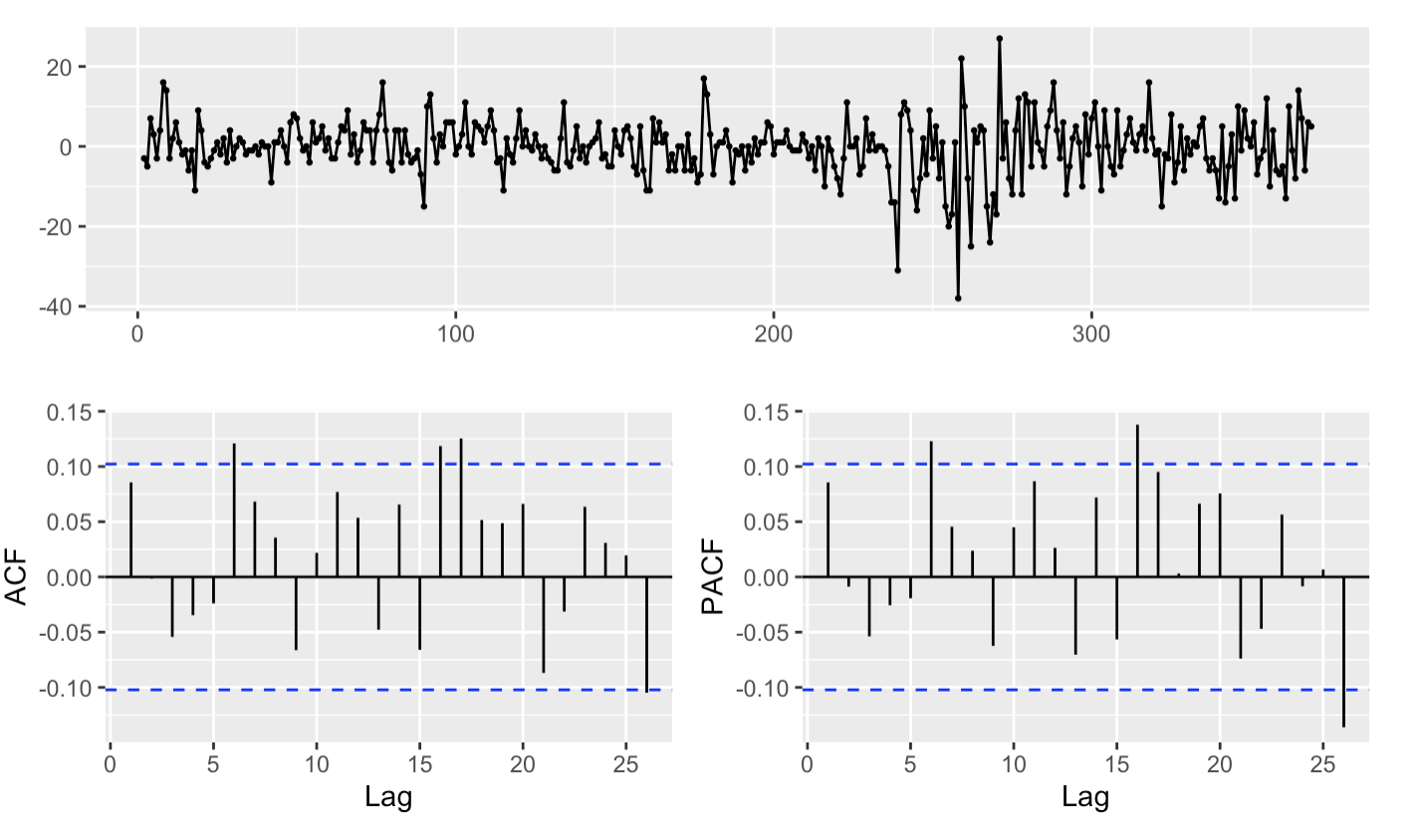
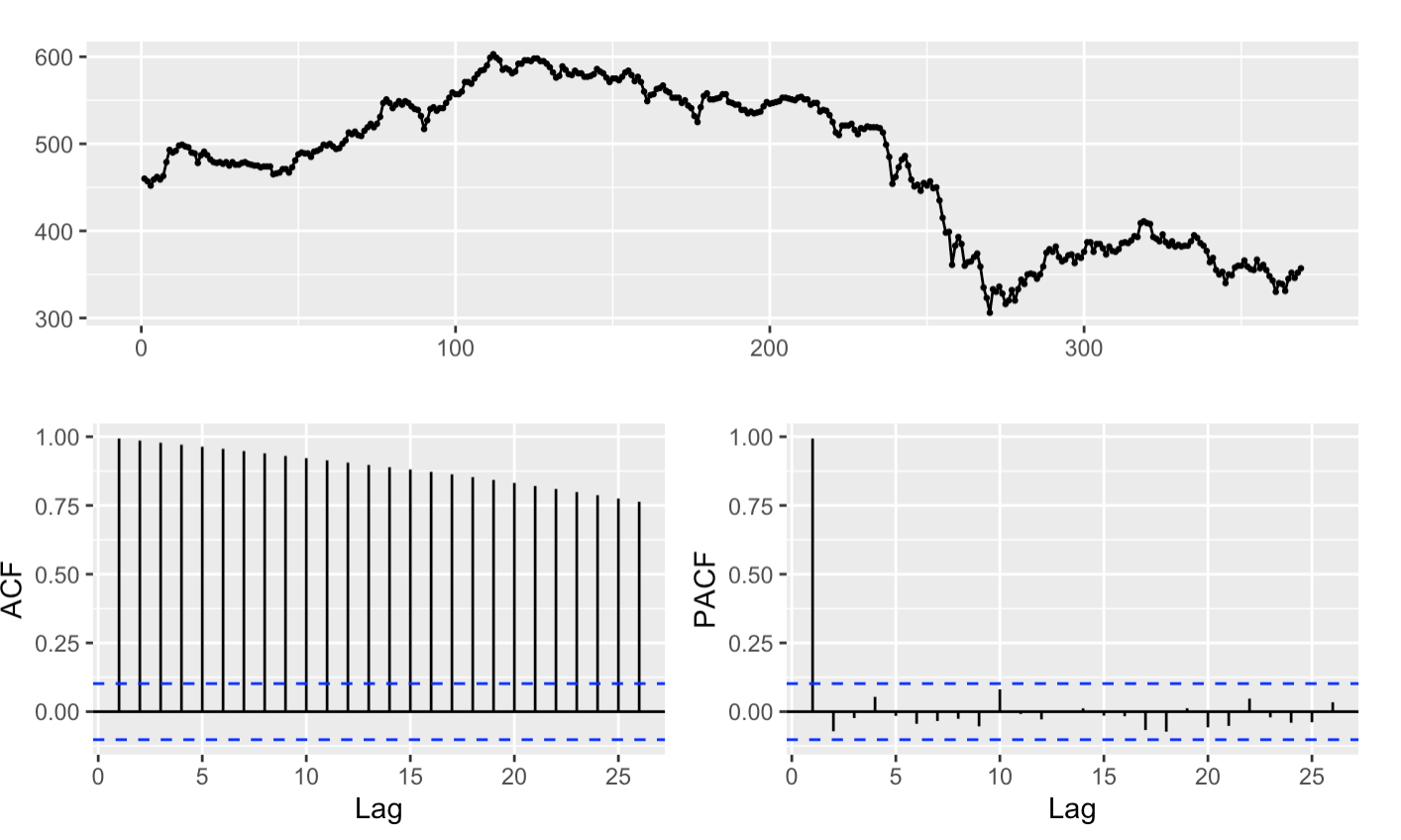
library(forecast)

plot(ibmclose)

ggtsdisplay(diff(ibmclose))

```

**Raw IBM Closing Price Data**  **Differenced IBM Closing Price Data**



It’s easy to see the plot on the left before differencing has a definite trend and is not stationary. The ACF values are all positive and slowly decreasing. After differencing the data it becomes stationary. The values in the line and ACF plots are both positive and negative with a mean hovering around zero. The differenced ACF and PACF plots show a couple points outside of the of the critical point limits which indicate there may be some seasonality to the data. You would want to take this into account when creating an ARIMA model for forecasting the change in IBM closing stock price.

# HA 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 and . The process starts with .

The below code creates some sample time series with 100 values and a normally distributed (noise) term of randomly generated numbers. We set the seed to ensure the plot looks the same each time. We then replace each item in the time series with the value of multiplied by .6 and added in the error term.

```{r}

set.seed(8)

y <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

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

plot(y, main=expression(paste('Line plot with ', phi, " = .6")))

```

## B) Produce a time plot for the series. How does the plot change as you change ?

We use the code below to create three line plots with the following values , , and to see the impact it has on the data. We set the seed to replicate the output then use the provided function and change the value of and plot the results.

```{r}

set.seed(8)

y <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

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

y1 <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

y1[i] <- 0.01\*y1[i-1] + e[i]

y2 <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

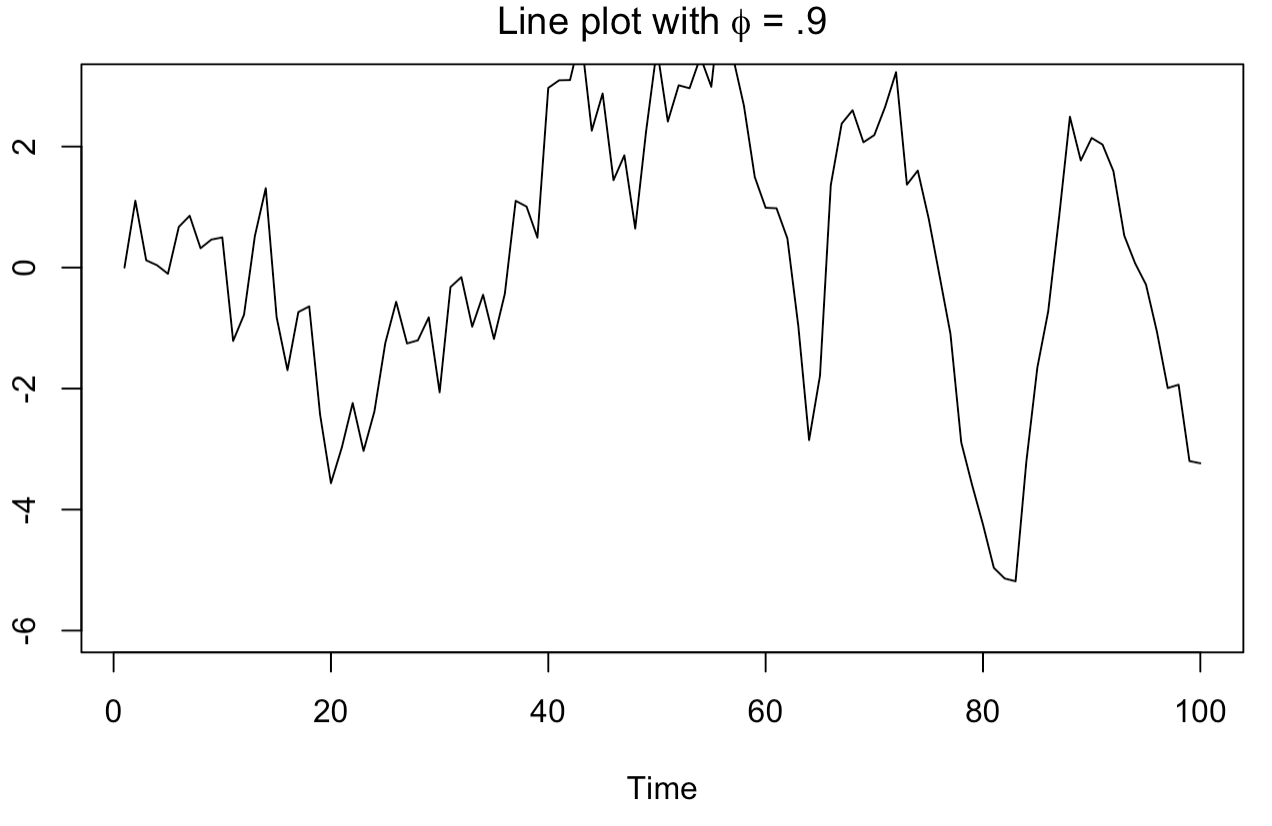
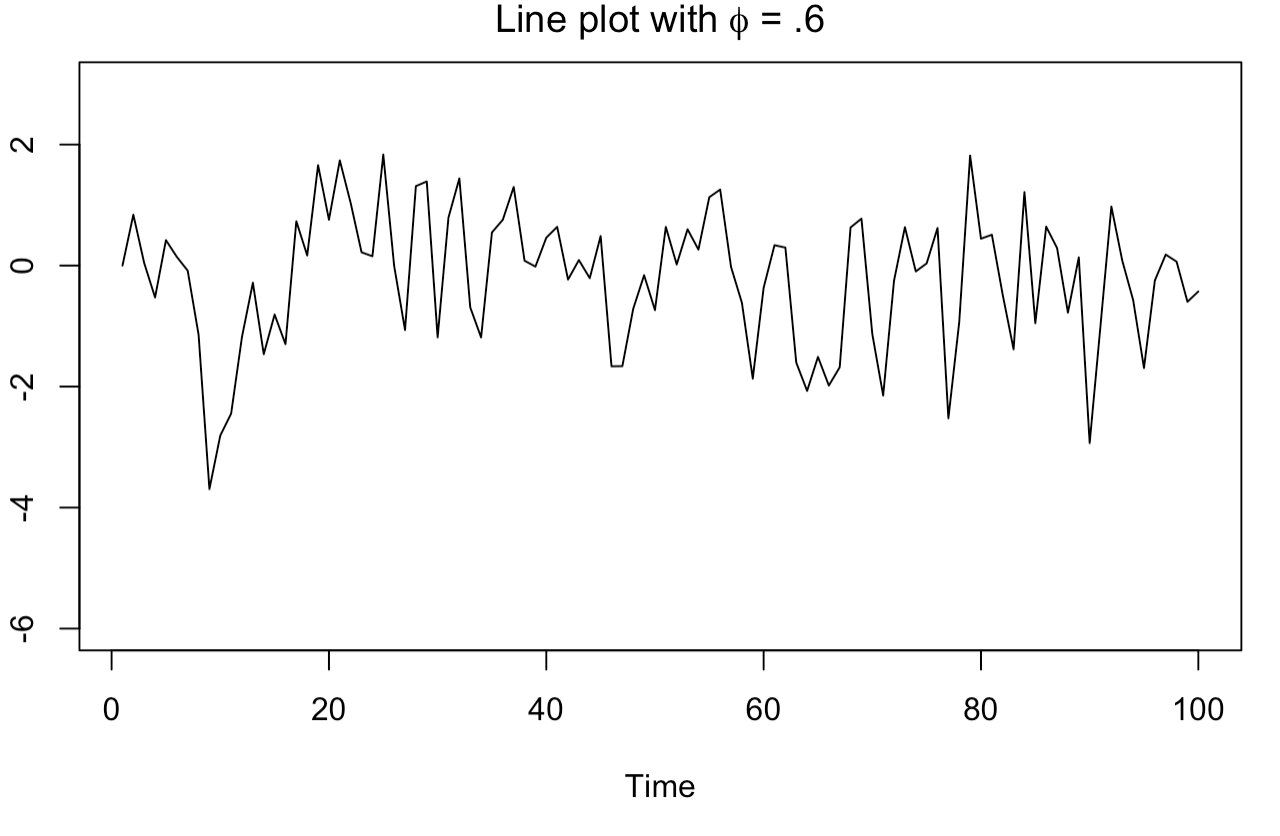
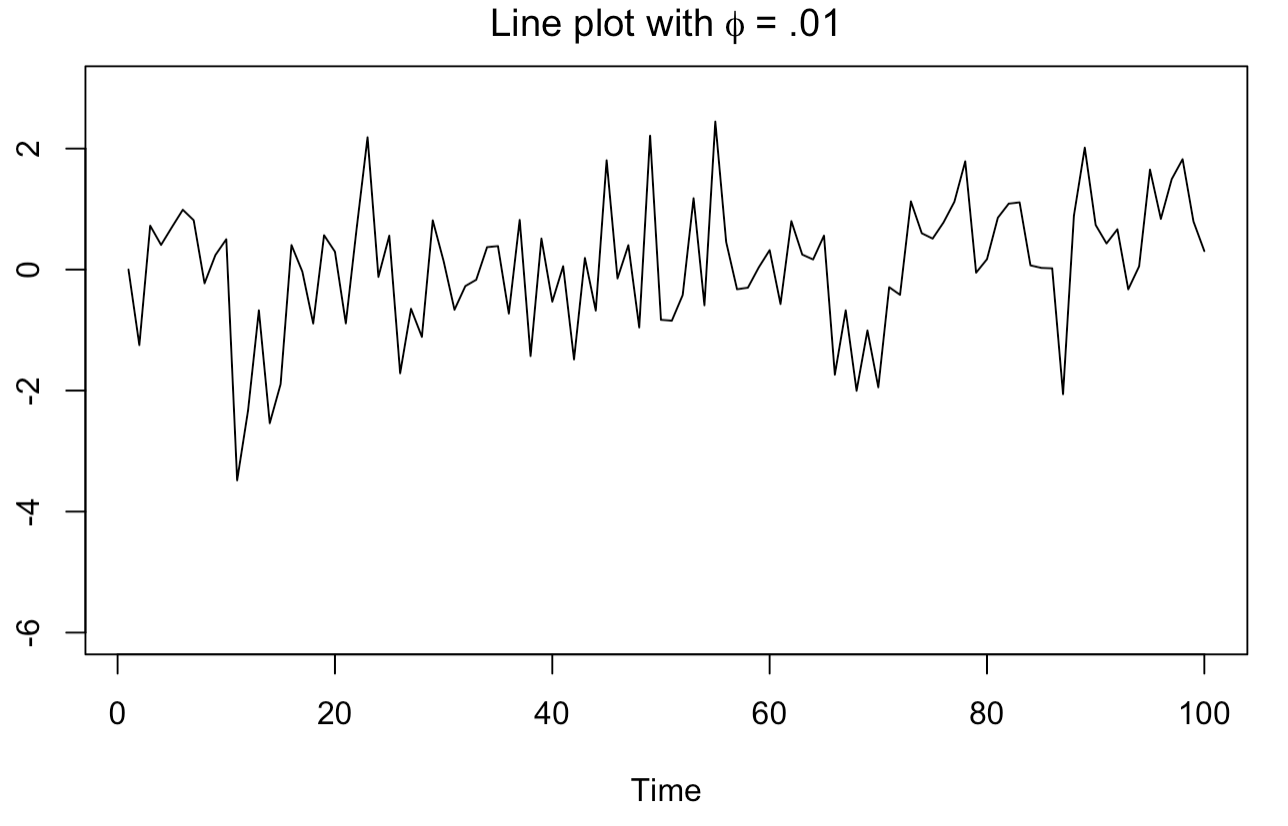
y2[i] <- .9\*y2[i-1] + e[i]

plot(y1,main=expression(paste('Line plot with ', phi, " = .01")), ylim=c(-6, 3))

plot(y,main=expression(paste('Line plot with ', phi, " = .6")), ylim=c(-6, 3))

plot(y2, main= expression(paste('Line plot with ', phi, " = .9")), ylim=c(-6, 3))

```



Increasing the value of increases the variability of standard deviation of the sample.When we start to see definite trends and seasonality starting to form in the data. It becomes less stationary.

## C) Write your own code to generate data from an MA(1) model with and .

To create a moving average model we multiply by our normally distributed noise term at instead of the value of y. We add that number to and replace that number in the time series.

```{r}

ma <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

ma[i] <- .6\*e[i-1] + e[i]

```

## D) Produce a time plot for the series. How does the plot change as you change ?

The below code creates three MA models with values of , , & to be able to visualize the impact of the change. The below code sets the seed to ensure replication of the plots and uses the above function changing the value of and plotting the results.

```{r}

set.seed(8)

ma <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

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

ma1 <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

ma1[i] <- 0.01\*e[i-1] + e[i]

ma2 <- ts(numeric(100))

e <- rnorm(100)

for(i in 2:100)

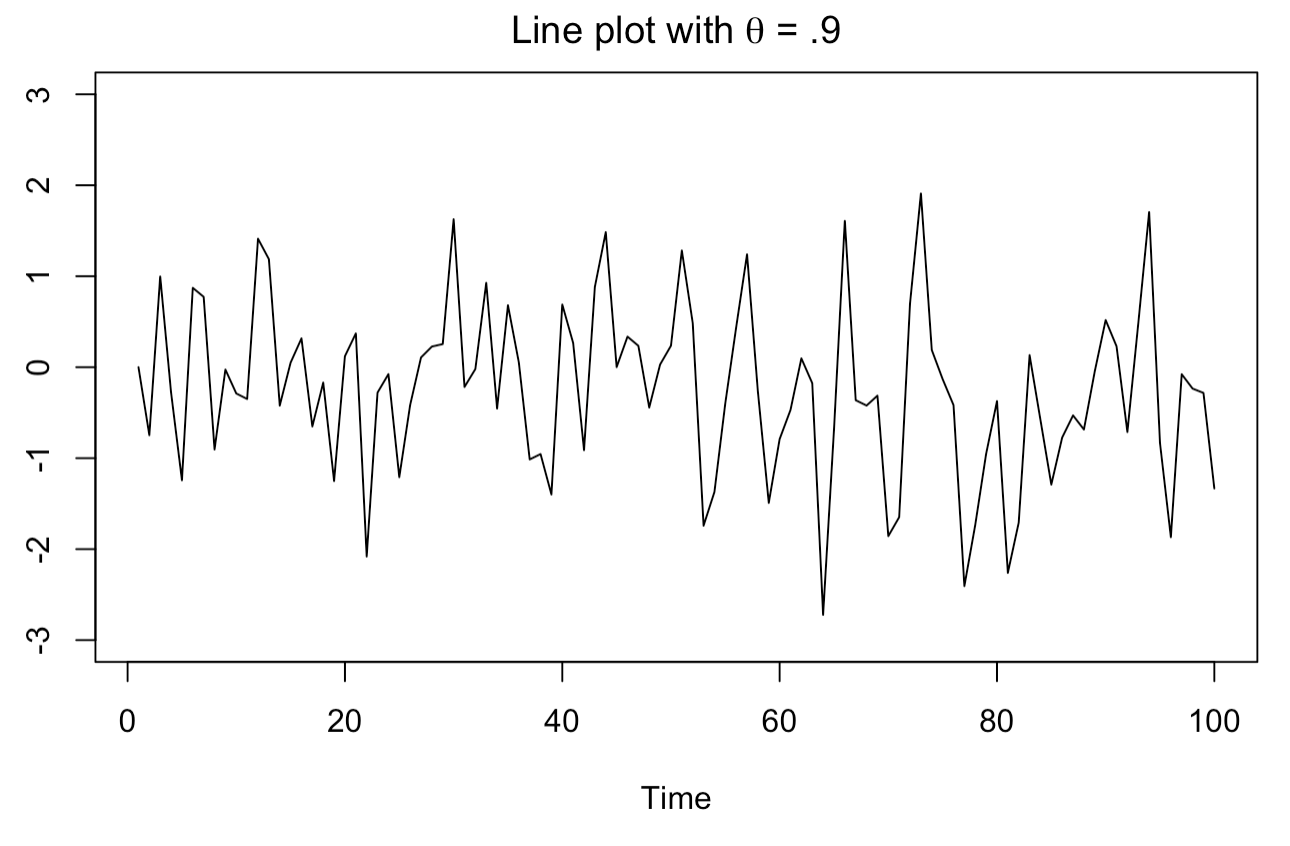
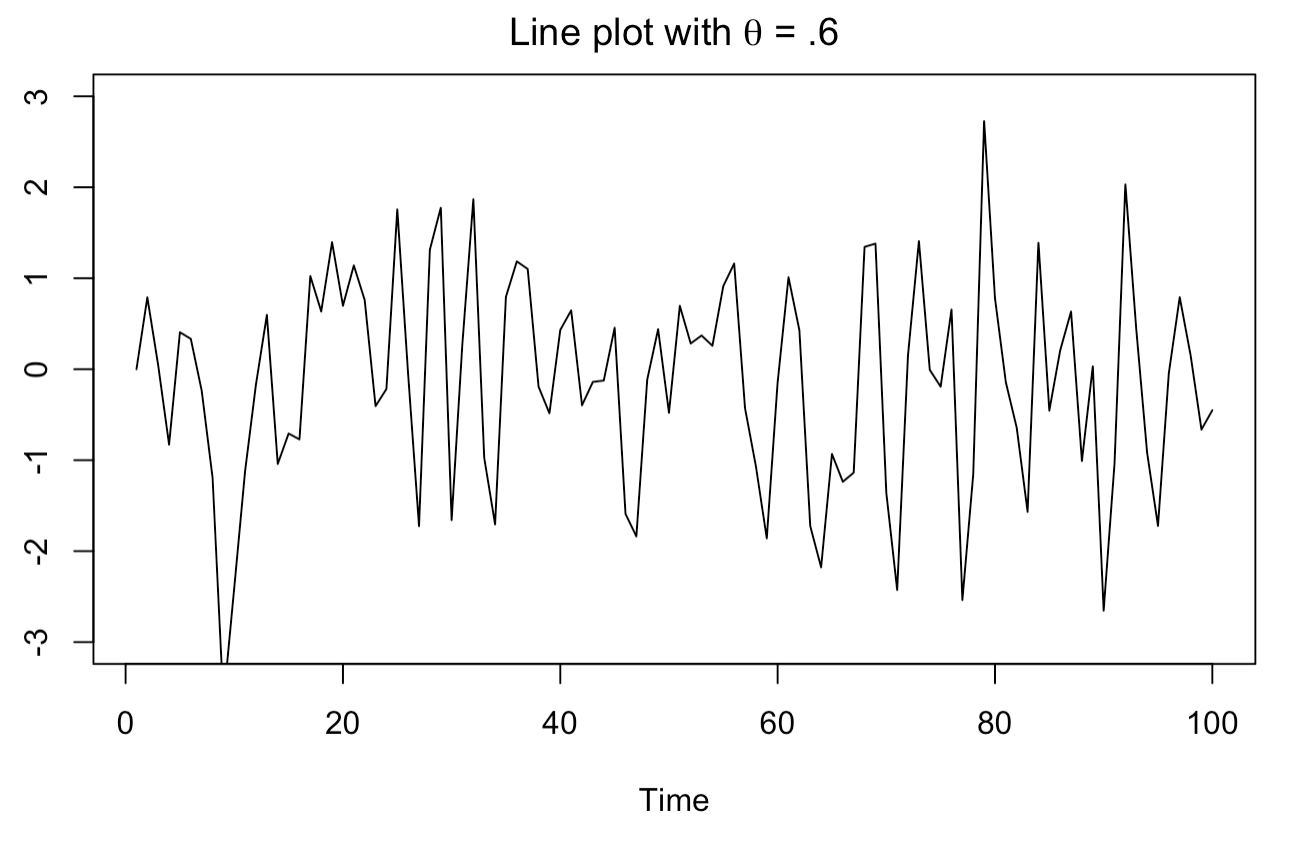
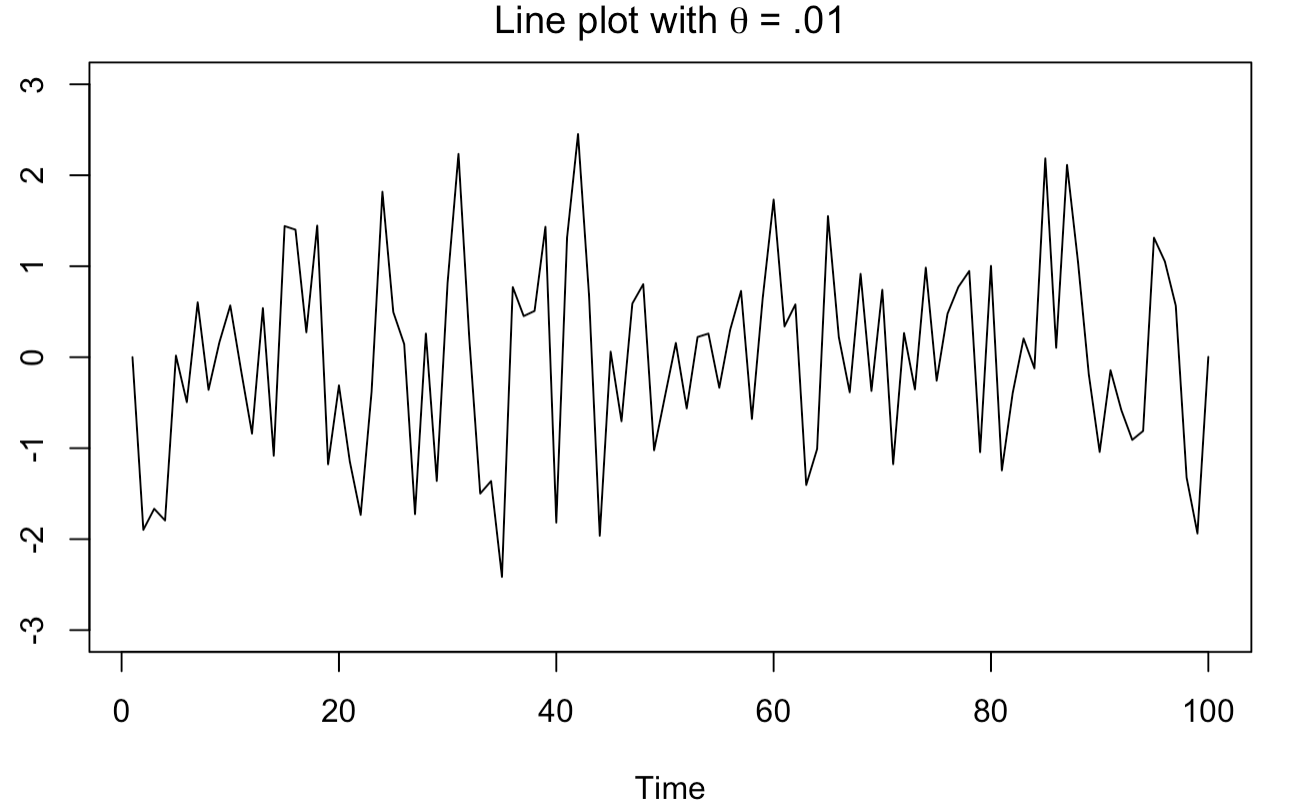
ma2[i] <- .9\*e[i-1] + e[i]

plot(ma,main=expression(paste('Line plot with ', theta, " = .01")), ylim=c(-3, 3))

plot(ma1,main=expression(paste('Line plot with ', theta, " = .6")), ylim=c(-3, 3))

plot(ma2, main= expression(paste('Line plot with ', theta, " = .9")), ylim=c(-3, 3))

```



By looking at the graphs we can see that as we increase the value of there is less fluctuation in the data. It has a smoothing effect on the sample.

## E) Generate data from an ARMA(1,1) model with , and .

To create an ARIMA(1,1) model we multiply the above weights to the time series data and noise term at then add them to the error term at . We take the sum and replace the time series data. The below code loops over the sample and applies this change.

```{r}

arima <- ts(numeric(100))

e <- rnorm(10000)

for(i in 3:100)

arima[i] <- .6\*e[i-1] + .6\*arima[i-1] + e[i]

```

## F) Generate data from an AR(2) model with , and . (Note that these parameters will give a non-stationary series.)

To create an AR(2) model we multiply the above weights to the time series data at and respectively. We then replace the time series data with the sum of those two numbers with the addition of the normally distributed random noise term. The below code loops over the sample and applies this change.

```{r}

ar2 <- ts(numeric(100))

e <- rnorm(10000)

for(i in 3:100)

ar2[i] <- -.8\*y[i-1] + .3\*ar2[i-2] + e[i]

```

## G) Graph the latter two series and compare them.

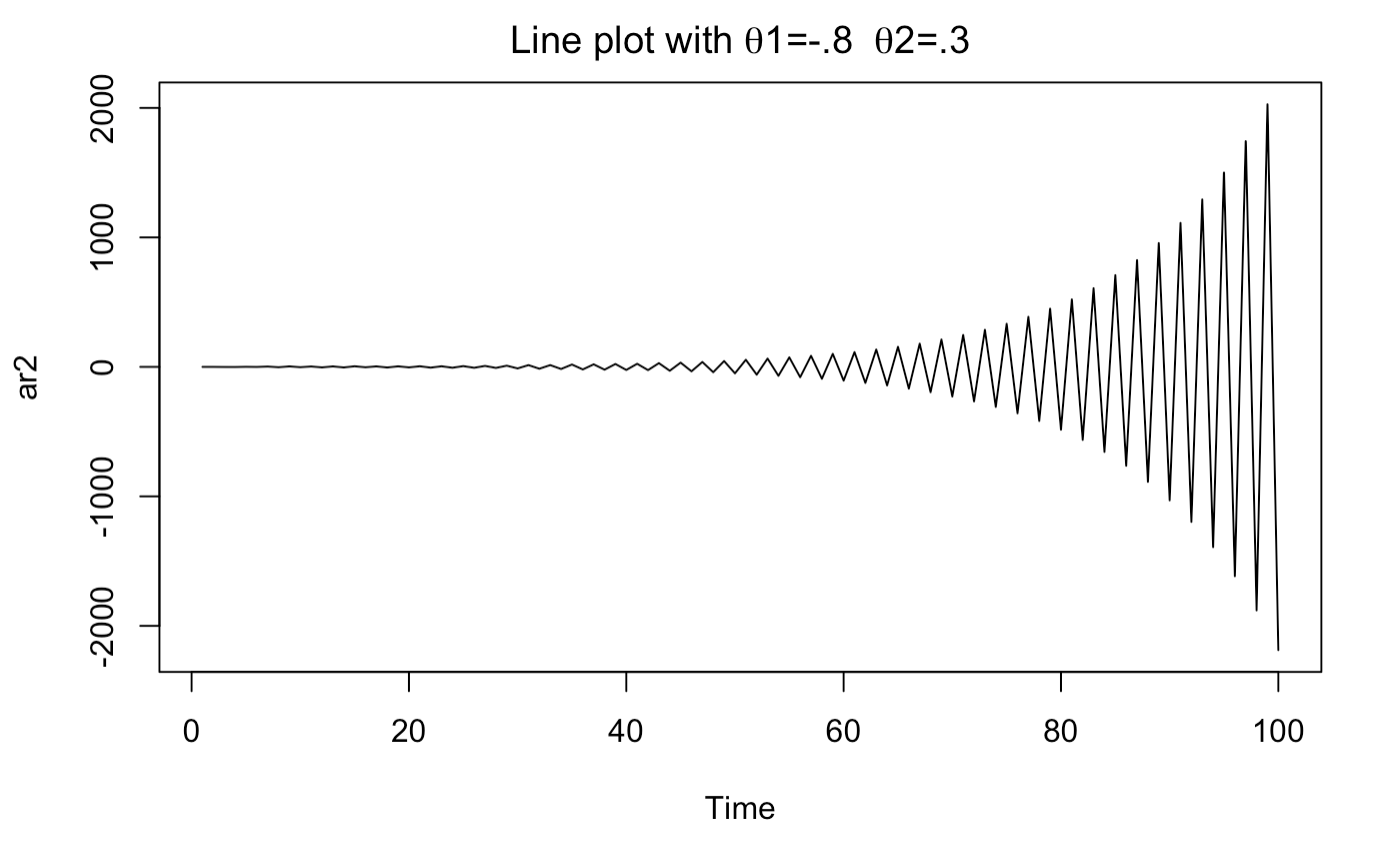
The below code creates line plots of the datasets we created above.

```{r}

plot(arima, main= expression(paste('Line plot with ', theta, "=.6 ", phi, "= 6")))

plot(ar2, main= expression(paste('Line plot with ', theta, "1=-.8 ", theta, "2=.3")))

```



The first plot appears to be stationary with both positive and negative values with a mean of slightly below zero. There appears to be some seasonality but likely occurring randomly in the small sample size. The plot on the right is not stationary even though the data has a mean of roughly zero with both positive and negative values. For each increase in value there is an equal and opposite decrease in value the next period. This effect slowly increases in magnitude each period creating a wave like effect in the graph.

# KJ 3.1

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

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

Some interesting points based on the boxplots of the features, histogram of the Type label, and the correlation heatmap are…

1. All of the features, except Mg have outliers.

2. The Ba features has a unusual distribution, with a mean near 0, and a large number of outliers

3. The widely varied measures for each feature (as shown on the y-axis) makes this dataset a good candidate to apply scalarization to.

4. Histograms reveal that RI, Na, Ai, Si have an almost normal distribution.

5. K, Ba, Fe are all skewed to the right, while Mg is skewed left.

6. The problem statement mentions 7 class categories, but the histogram of the Type shows only 6 categories in this dataset.

7. The Type categories are skewed (around 130 out of 213) categories 1 and 2.

8. Categories 3,4,5,6 are reasonably close to each other in frequency.

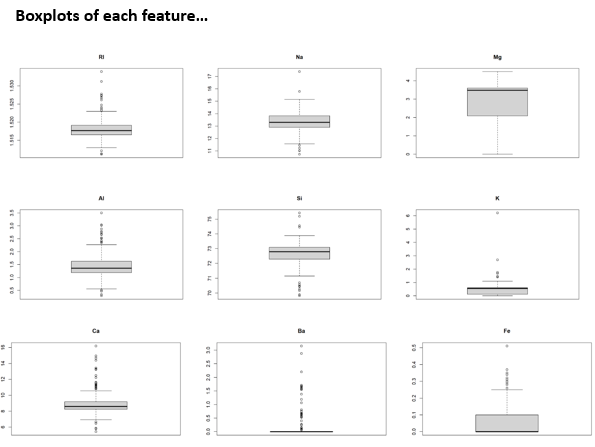
*Correlations to the Type label…*

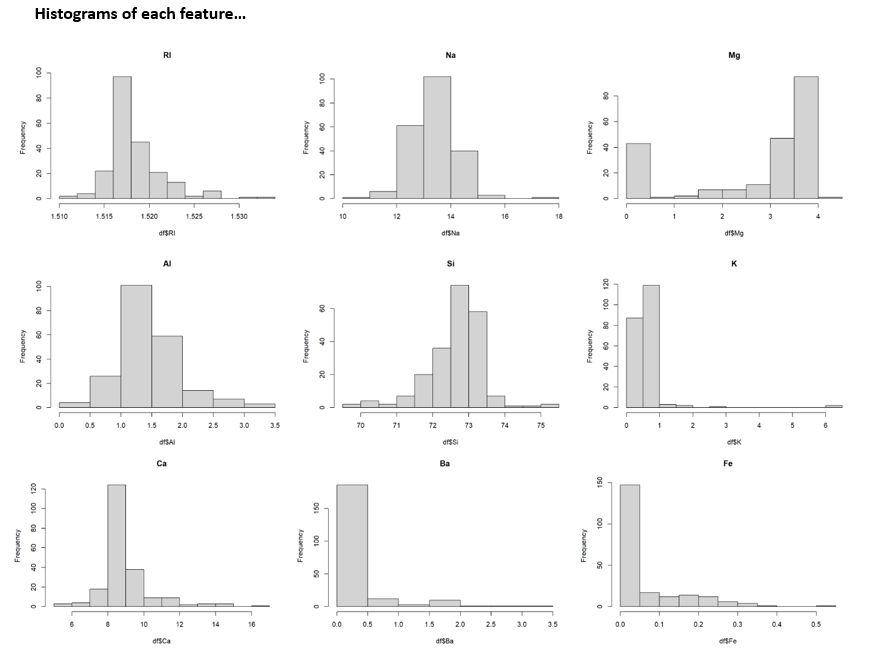
9. Ba, Ai, and Na have the strongest positive correlations with a value of over 50%

10. Mg has the strongest negative correlation at over -80%

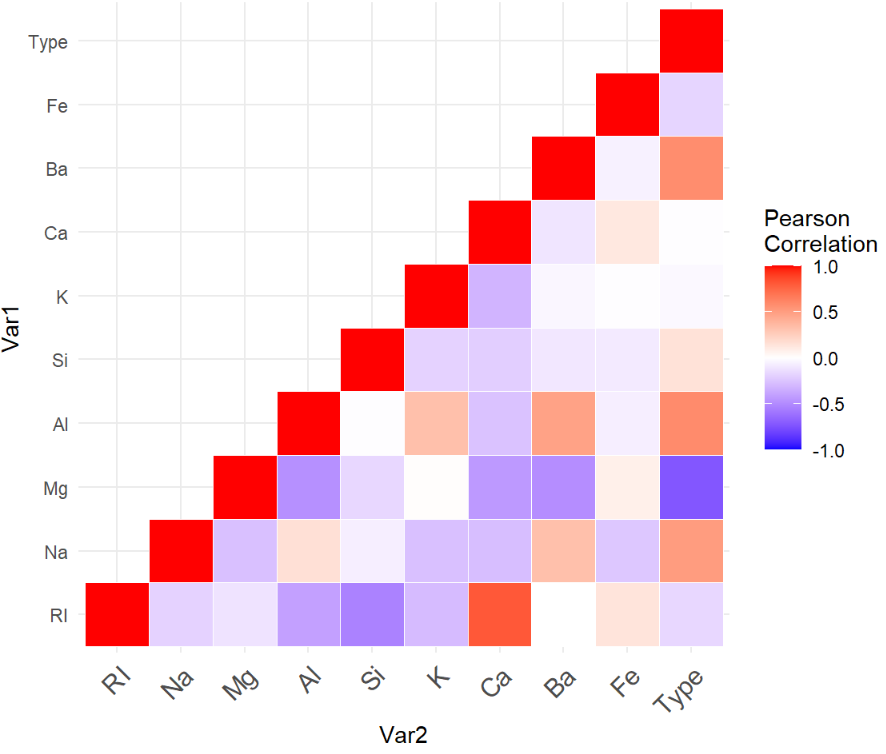
11. Si and Ca have close to 0 correlation, so will probably be dropped

12. All in all the heatmap reveals some strong correlations that make these features high potential as predictors. But, given that we are working with a multi-class label with 6 categories I anticipate this will be a challenging prediction problem.









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

Yes there are outliers and yes some predictors are skewed. See observations made under point A above for detail.

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

Scalarization is a transformation to consider applying across the features.

# KJ 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.

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

By running this code, I was able to generate histograms for all the predictors. Below is a sample of the first 3 of the 35 predictors.

In looking through these distributions, there are some that are degenerate. For example, “Hail” is over 80% one of two categories, “Precipitation” is over 50% one of three categories, and “Leaves” is a 6:1 ratio in one of two categories.

```{r}

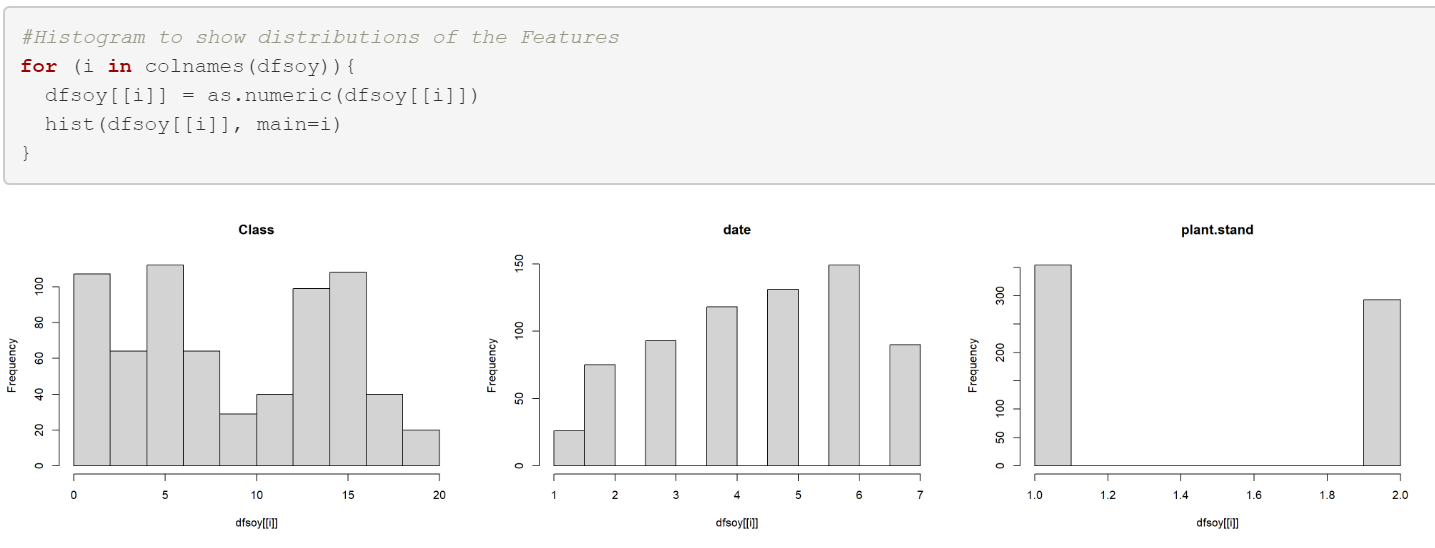
for (i in colnames(dfsoy)){

dfsoy[[i]] = as.numeric(dfsoy[[i]])

hist(dfsoy[[i]], main=i)

}

```



## 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?

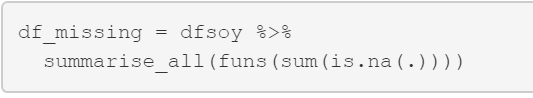
Running the code below, provided a count of missing values in each column.

```{r}

df\_missing = dfsoy %>%

summarise \_all(funs(sum(is.na(.))))

```



In reviewing the results, the six features that start with “leaf.” have a range of 84 to 108 missing values. It does appear based on this, that the missing data is related to the classes.

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

Here are a few approaches to consider taking…

1. Remove all rows with any missing values

2. Sum all the leaf features and use the new column as a feature (N/A would count as 0)

3. Evaluate correlation of the leaf features. If it’s low, then just eliminate these features.

# HA 8.8

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

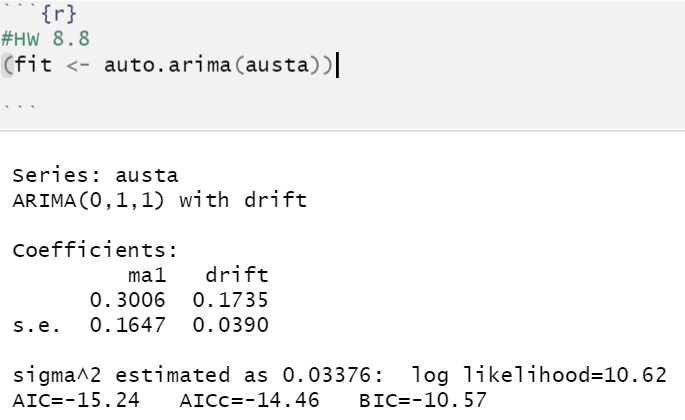
## A) Use auto.arima() to find an appropriate ARIMA model. What model was selected…

The model selected is ARIMA(0,1,1) with drift which is a moving average model with one round of stationary adjustments made.

```{r}

(fit <- auto.arima(austa))

```



## Check that the residuals look like white noise.

The plot below of the residuals, residuals’ ACF, and residuals’ PACF supports that the model is white noise as none of the ACF and PACF (except lag 1) are significant.

```{r}

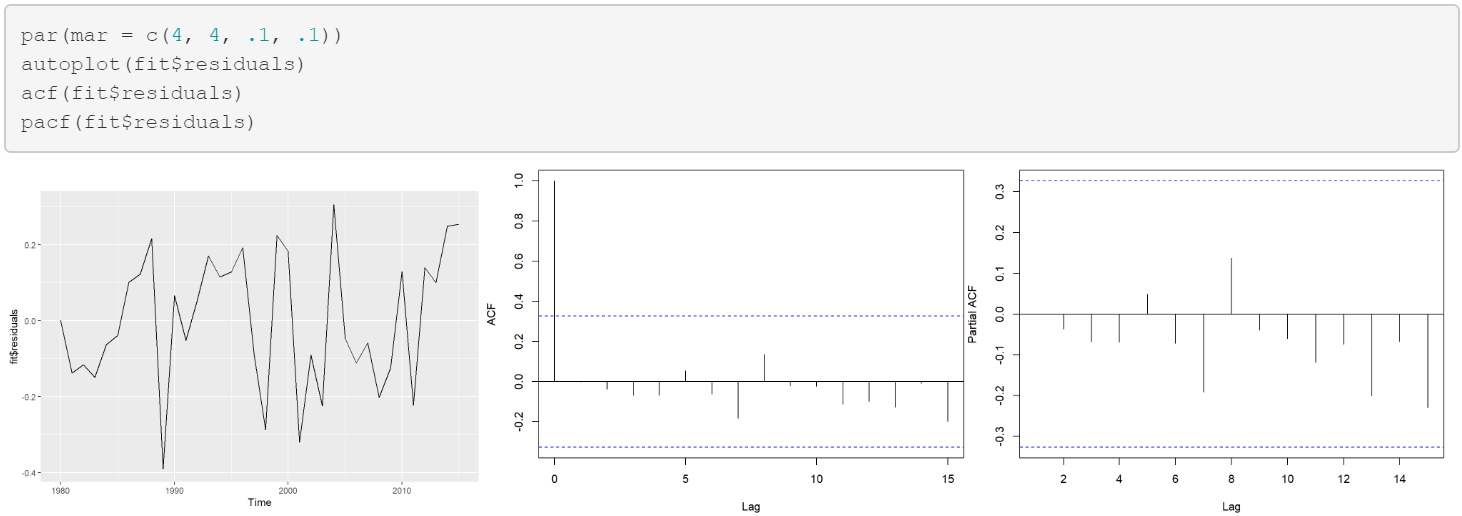
par(mar = c(4,4,.1,.1))

autoplot(fit$residuals)

acf(fit$residuals)

pacf(fit$residuals)

```



## Plot forecasts for the next 10 periods.

Here is a forecast of the next 10 periods…

```{r}

fit %>% forecast(h=10) %>% autoplot(include=80)

```



## B) Plot forecasts from an ARIMA(0,1,1) model with no drift and compare these to part a.

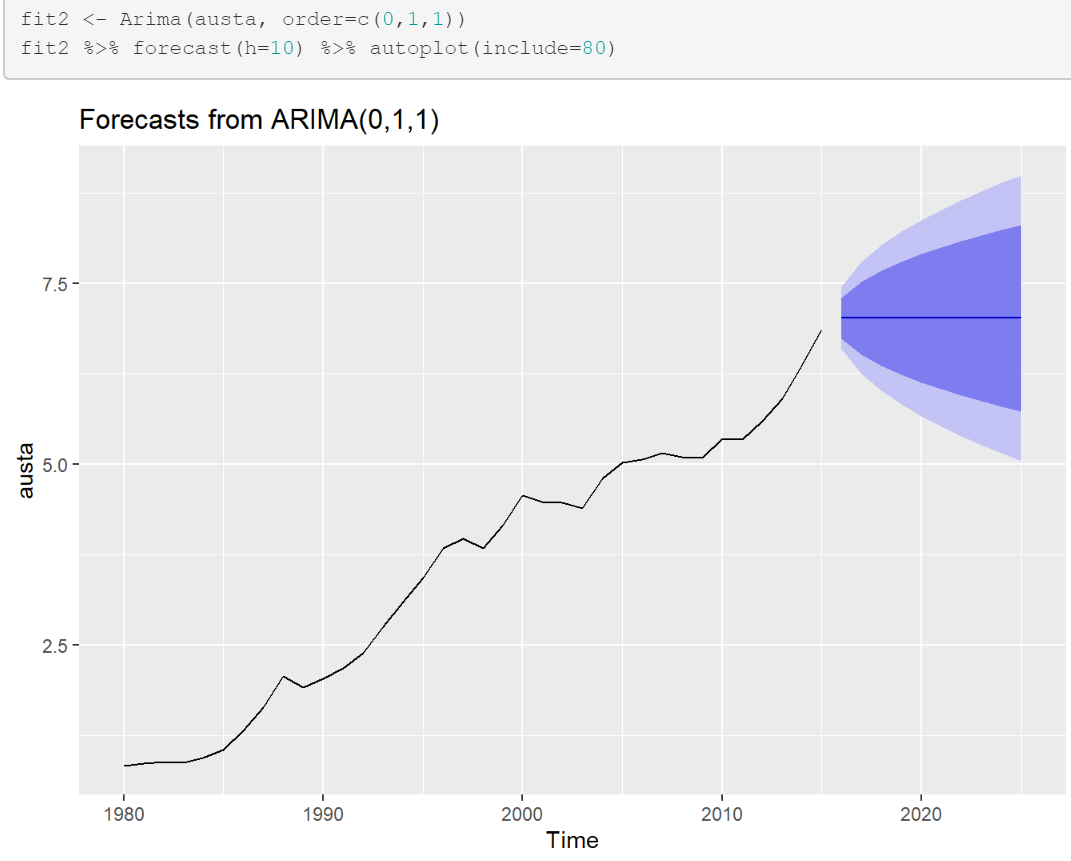
In comparison to the forecast in Part a, this model projects a flat line (same value) forecast for the next 10 periods and a wider confidence interval (so less confidence). All in all, part a’s forecast appears superior.

```{r}

fit2 <- Arima(austa, order=c(0,1,1))

fit2 %>% forecast(h=10) %>% autoplot(include=80)

```



## Remove the MA term and plot again.

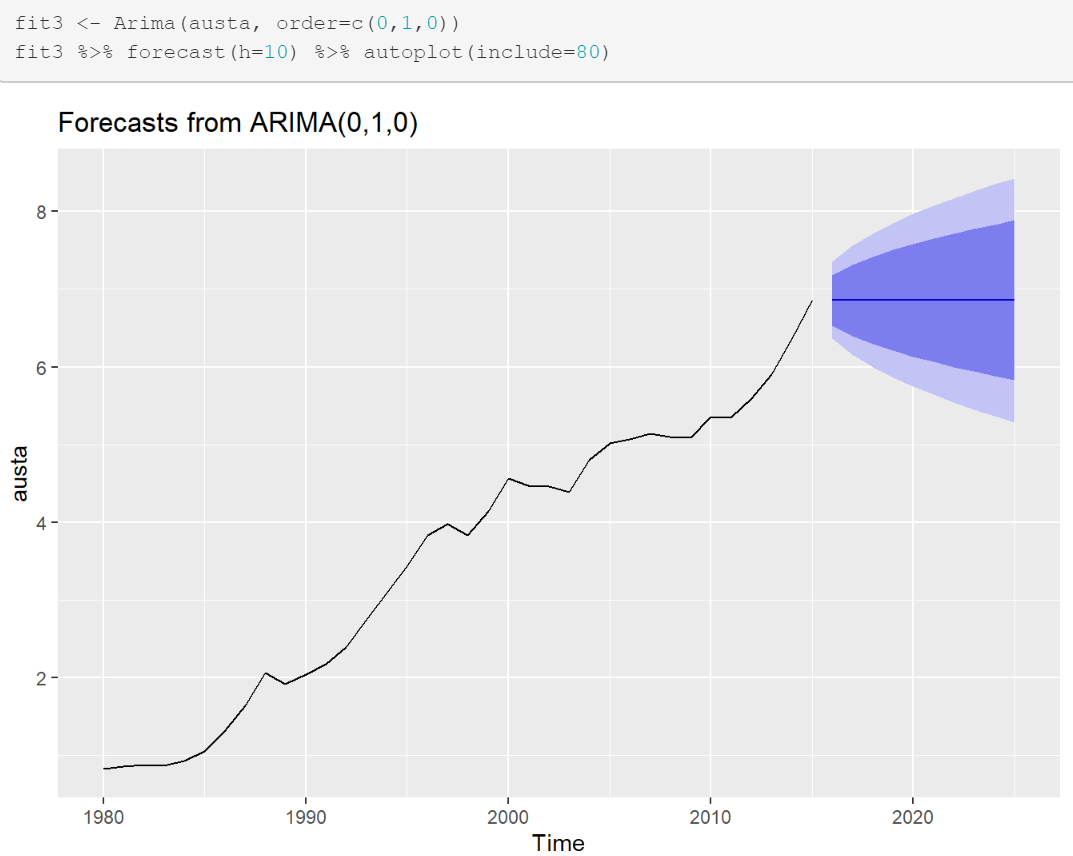
Removing the MA term generates a forecast that looks very similar to the forecast generated in Part B.

```{r}

fit3 <- Arima(austa, order=c(0,1,0))

fit3 %>% forecast(h=10) %>% autoplot(include=80)

```



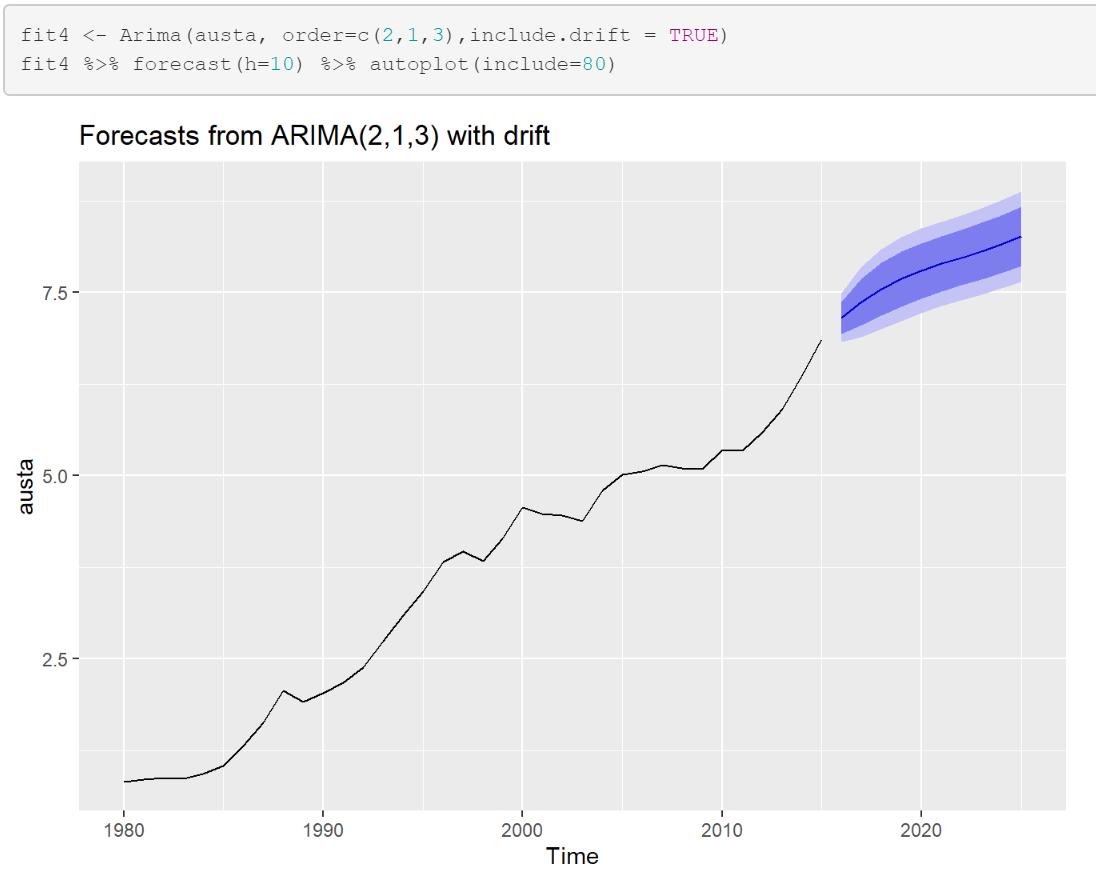
## C) Plot forecasts from an ARIMA(2,1,3) model with drift.

```{r}

fit4 <- Arima(austa, order=c(2,1,3), include.drift = TRUE)

fit4 %>% forecast(h=10) %>% autoplot(include=80))

```



## Remove the constant and see what happens.

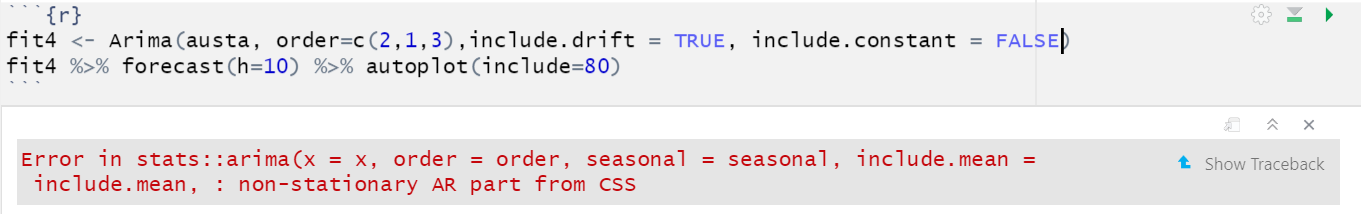
Removing the constant results in an error message as shown below…

```{r}

fit4 <- Arima(austa, order=c(2,1,3), include.drift = TRUE, include.constant = FALSE)

fit4 %>% forecast(h=10 %>% autoplot(include=80)

```



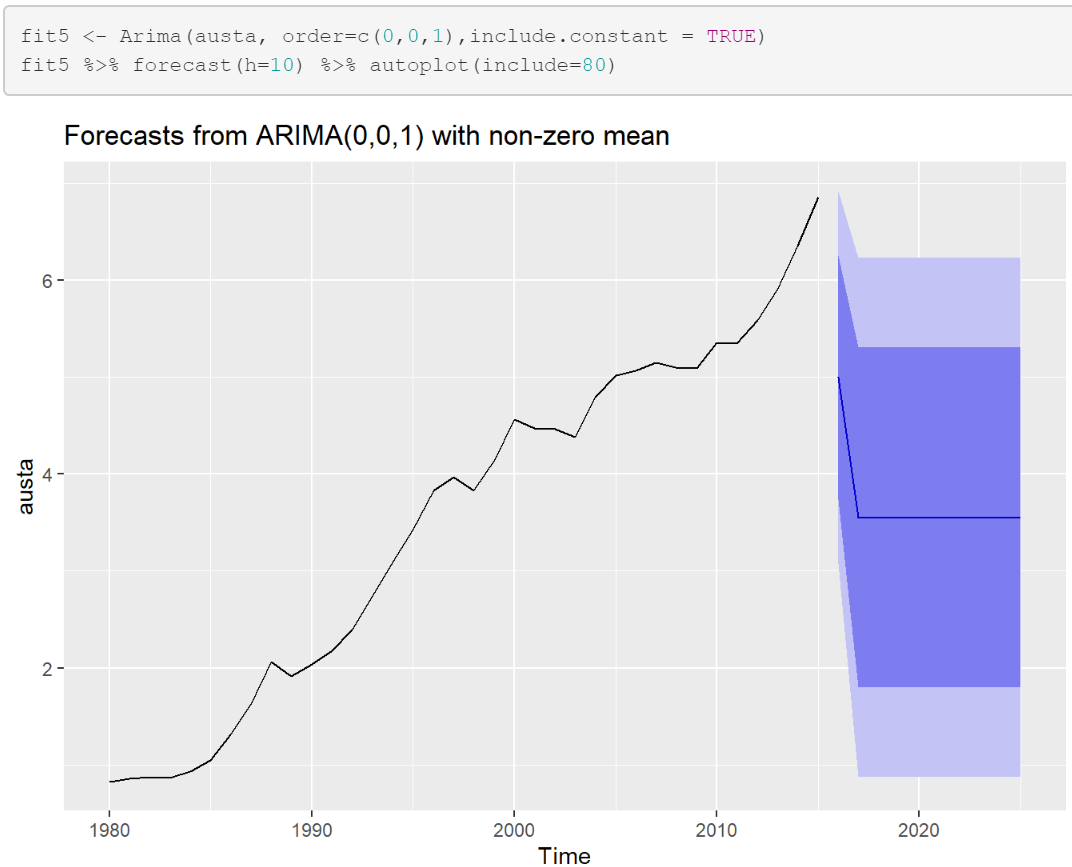
## D) Plot forecasts from an ARIMA(0,0,1) model with a constant.

```{r}

fit5 <- Arima(austa, order=c(0,0,1), include.constant = TRUE)

fit5 %>% forecast(h=10) %>% autoplot(include=80)

```



## Remove the MA term and plot again.

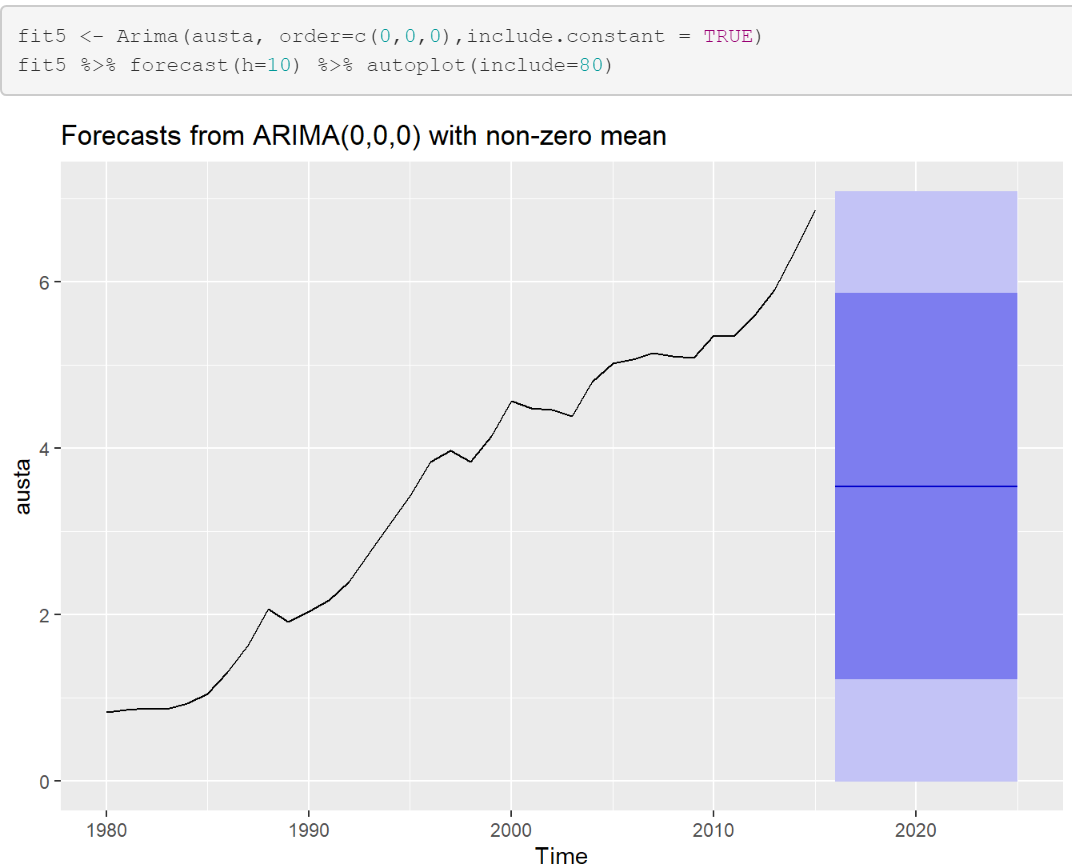
White noise is the result.

```{r]

fit5 <- Arima(austa, order=c(0,0,0), include.constant = TRUE)

fit5 %>% forecast(h=10) %>% autoplot(include=80)

```



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

The model forecast below projects an upward trend with a tight confidence interval.

```{r}

fit5 <- Arima(austa, order=c(0,2,1), include.constant=TRUE)

fit5 %>% forecast(h=10) %>% autoplot(include=80)

```

