Chapter 3 Review

Anthony Tricarico

Table of contents

1 Time Series Decomposition

The main topic of this section is how we can exploit the information contained in time series data to make reliable and useful forecasts. You might have noticed that thus far some time series exhibited some peculiar behaviors. Those behaviors were highlighted by the plots produced to visualize the data. For reference, you can have a look at section 3.2 of your textbook to have a basic idea of what it is that we are talking about.

1.1 Trend

The first component of a time series is its trend. Put simply, the trend refers to where the data seems to be headed in a specific time window (i.e., if I compare the start and the end of the time series, I check if my variable is increasing or decreasing in value over time). Making an example using the employment data for the US, we have the following.

```r
library(fpp3)
```

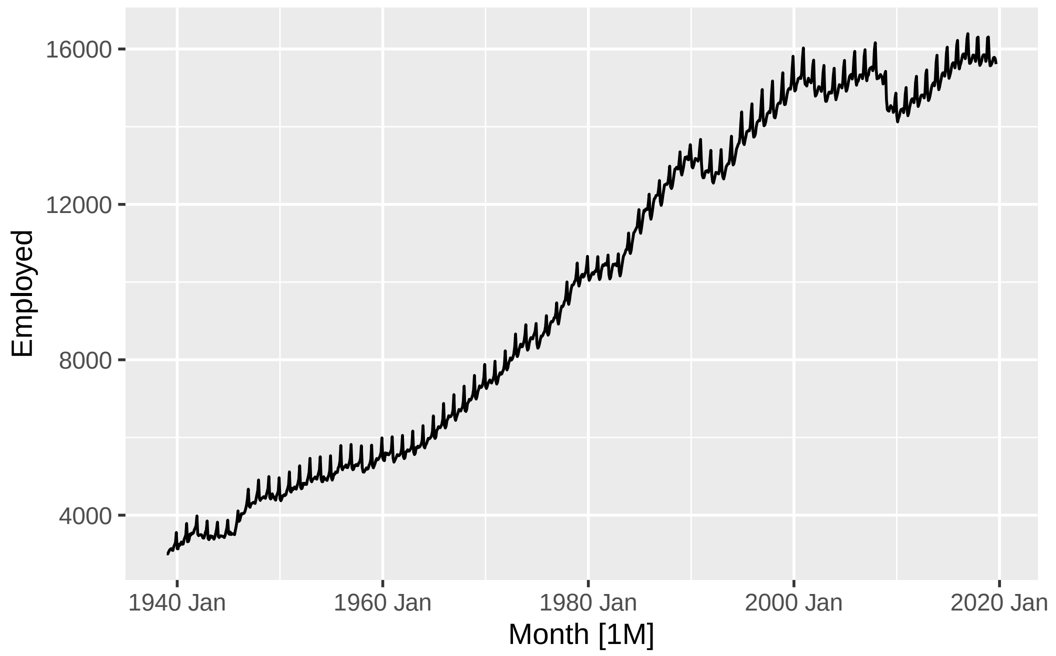
① filtering only for retail sector data
② plotting the time series

```
Registered S3 method overwritten by 'tsibble':
  method               from
  as_tibble.grouped_df dplyr

-- Attaching packages -------------------------------------------- fpp3 1.0.1 --

v tibble      3.2.1     v tsibble     1.1.5
v dplyr       1.1.4     v tsibbledata 0.4.1
v tidyr       1.3.1     v feasts      0.4.1
v lubridate   1.9.3     v fable       0.4.1
v ggplot2     3.5.1

-- Conflicts ------------------------------------------------- fpp3_conflicts --
x lubridate::date()    masks base::date()
x dplyr::filter()      masks stats::filter()
x tsibble::intersect() masks base::intersect()
x tsibble::interval()  masks lubridate::interval()
x dplyr::lag()         masks stats::lag()
x tsibble::setdiff()   masks base::setdiff()
x tsibble::union()     masks base::union()
```

```r
library(latex2exp)
library(slider)
library(gridExtra)
```

```
Attaching package: 'gridExtra'


The following object is masked from 'package:dplyr':

    combine
```

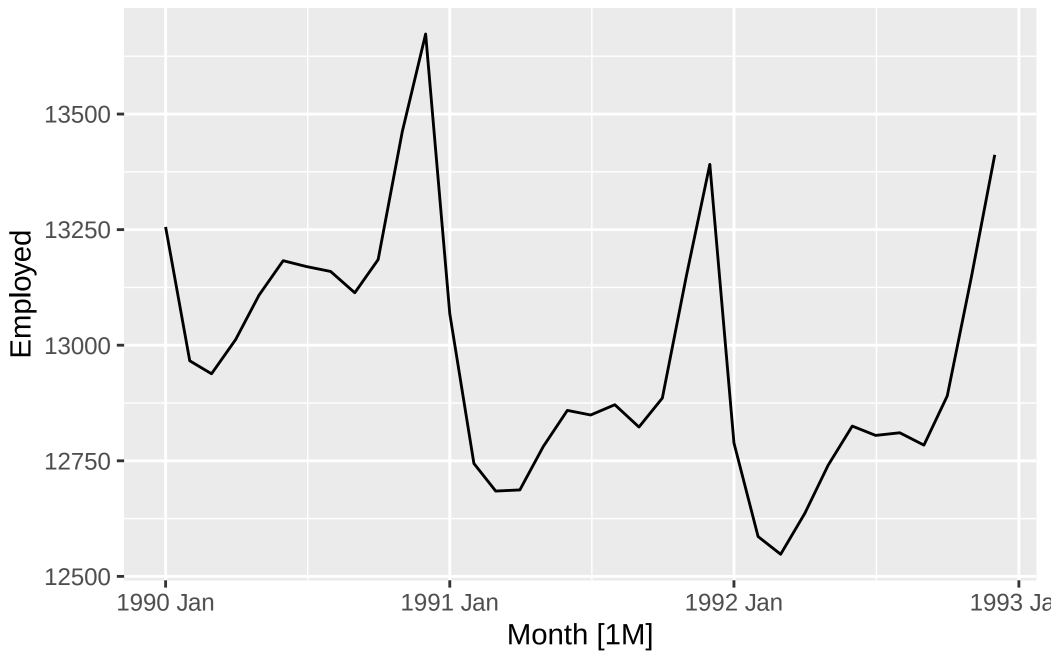
```r
us_employment %>%
  filter(Title == "Retail Trade") %>%                                       ①
  autoplot(Employed)                                                        ②
```



Overall, we can see that the trend from 1940 to 2020 is positive (i.e., there has been a significant growth in people employed in the retail sector from 1940 to 2020). However, since we said that the trend is dependent on the time window that is analyzed we can show this by considering only data from 1990 to 1992.

```
us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1990, year(Month) <= 1992) %>%   ①
  autoplot(Employed)                                                               ②
```

① filtering for retail sector data between 1990 and 1992.
② plotting the number of people employed in the retail sector throughout the period.

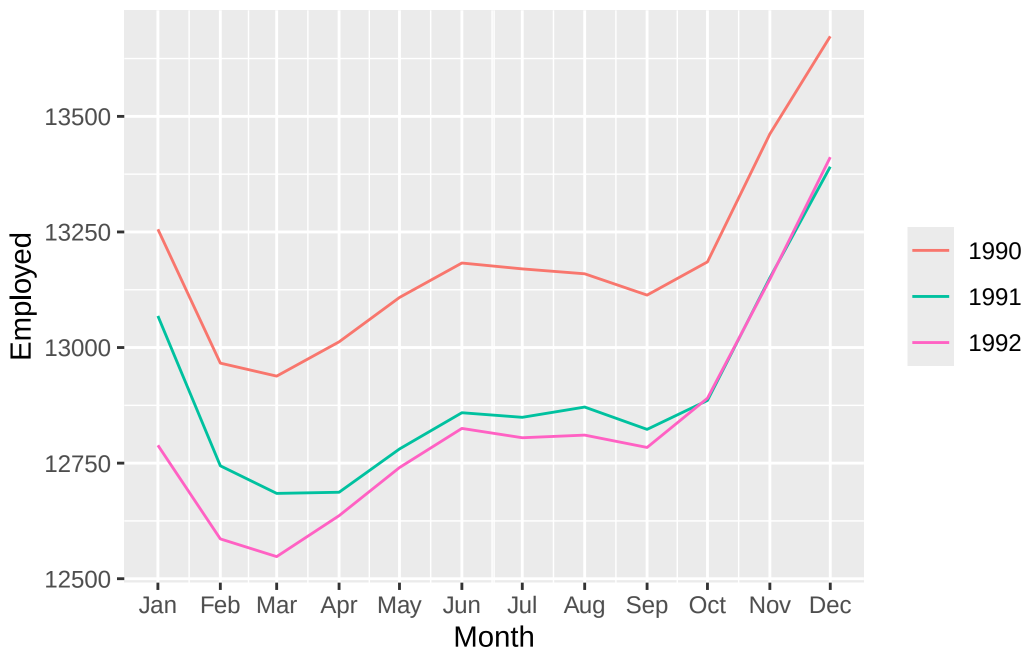


Here, we see how it is much more difficult to determine a specific trend for this time period as the beginning of the series seems to be almost at the same level as the end value of the series (i.e., number of people employed in Jan 1990 is almost the same as the number of people employed in Dec 1992). Keeping in mind this concept of trend is fundamental since we will use it to determine the portion of variation of the time series that can be attributed to it.

1.2 Seasonality

The second component of a time series is its seasonality. Seasonality refers basically to the cyclical trends that are present in the data. For instance, one might notice that in the months of October, November and December, each year from 1990 to 1992 there is a recurring spike in employment. We can visualize this using a seasonal plot produced with `gg_season()`.

```
us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1990, year(Month) <= 1992) %>%
  gg_season(Employed)
```



Notice how every year there is a larger number of employed people in the last months of the year as noticed above.

Up to this point, we only know how to visualize trend and seasonality, but now we will learn how to split the time series into its different trend and seasonal components to be able to pinpoint exactly what portion of variation in our time series can be attributed to each one.

1.3 Moving averages

Before we learn more about decomposition techniques, it is first important to define what a moving average is. A moving average is simply the arithmetic mean computed on a subset of the original data. This average is *moving* in the sense that the time window we consider *slides* around and considers sequentially each different subset. How many time periods should be averaged together is determined by the *order* (m) of the moving average. An order of 5, for instance, means that the moving average is being computed on the 5 adjacent observations. This means that I am only considering the observation at the current time (Y_t) along with the 2 previous observations (Y_{t-2} and Y_{t-1}) and the following 2 observations in time (Y_{t+1} and Y_{t+2}). Putting this into a formula:

$$\frac{Y_{t-2} + Y_{t-1} + Y_t + Y_{t+1} + Y_{t+2}}{m} \tag{1}$$

! Important

Notice that the order m is always computed as the number of observations considered before the current observation and the number of observations considered after the current observation $+1$.

We notice that as t changes the time window *slides* allowing for different observations to enter the computation. In general, the formula to compute moving averages is given by:

$$\frac{1}{m}\sum_{j=-k}^{k} Y_{t+j} \tag{2}$$

Equation 2 effectively generalizes what observed in Equation 1

Moving averages are used to smooth out the time series so as to highlight an overarching trend in the data and that's why they are so important!

1.3.1 Moving averages in R

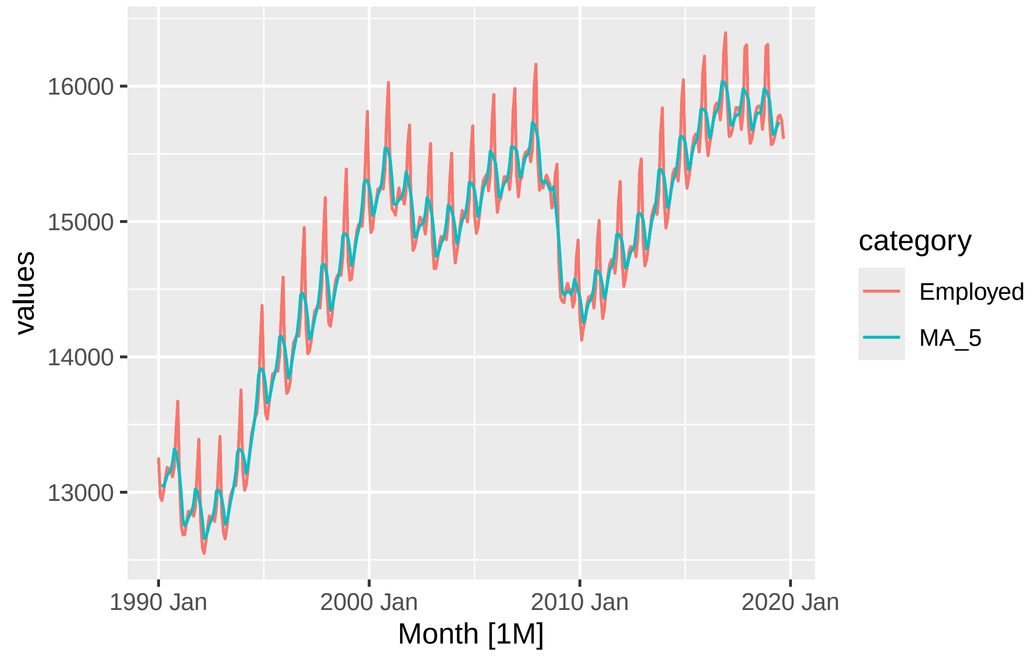
To compute the moving average of a variable in your time series, you can follow these steps:

1. use the `slide_dbl()` function from the `slider` package
2. Now you specify the following arguments inside the function:
    1. the name of the dataset (which can be passed into the function also through a pipe operator)
    2. the name of the column inside the dataset to use for computing the moving average
    3. specify that you want to compute the mean
    4. set .before and .after to the number of observations you want to be in each average (the k in Equation 2)
    5. set .complete = TRUE so that the function only computes the average for observations that have no values outside the specified range given by $[-k, k]$.

The function `slide_dbl()` will return a vector (i.e., a column) containing the computed moving average of the column specified. You can then add the result of the function to the dataset using `mutate()`. Here follows an example:

```r
employment_avg <- us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1990) %>%
  select(-Series_ID) %>%
  mutate(MA_5 = slide_dbl(Employed, mean, .before = 2, .after = 2, .complete = TRUE))
```

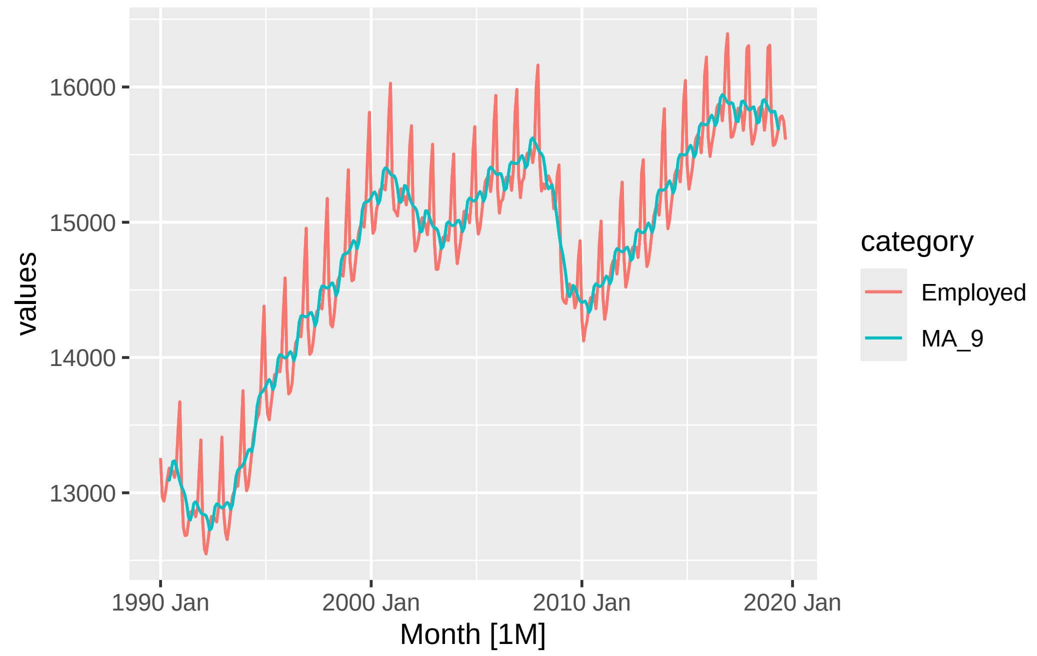
```
employment_long <- pivot_longer(employment_avg, c(Employed, MA_5), names_to = "category", values_

autoplot(employment_long, values) +
  geom_line(aes(y=values, color = category))
```



```
#autoplot(employment_avg, Employed) +
 # geom_line(aes(y=MA_5))
```

From the plot above we see how the trend (light blue line) approximates the original data but does not take values as extreme as those in the original series. This is exactly the meaning of *smoothing* the time series.

```
employment_avg <- us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1990) %>%
  select(-Series_ID) %>%
  mutate(MA_9 = slide_dbl(Employed, mean, .before = 4, .after = 4, .complete = TRUE))

employment_long <- pivot_longer(employment_avg, c(Employed, MA_9),
                                names_to = "category", values_to = "values")

autoplot(employment_long, values) +
  geom_line(aes(y=values, color = category))
```



In the plot above we see the effect of using higher order moving averages. In the specific case, using a moving average with $m = 9$ we see that the peaks and troughs of the moving average are much much smaller compared to those of the original time series. In fact those are even smaller than those of the moving average of order 5.

1.4 STL Decomposition

As explained in Section 1.1 and in Section 1.2 every time series can be decomposed in its trend and seasonality components, along with a remainder component (i.e., the variability that is neither due to seasonality nor trend). Summing these components together, we get the original values of the starting time series.

You can use different statistical tools to decompose a time series, but the STL decomposition is one of the most flexible methods to work with additive decompositions.

Note 1: Note

Even multiplicative decompositions can be turned into additive decompositions if the scale of the data is changed through a log transformation. This derives from the basic properties of logarithms. Specifically:

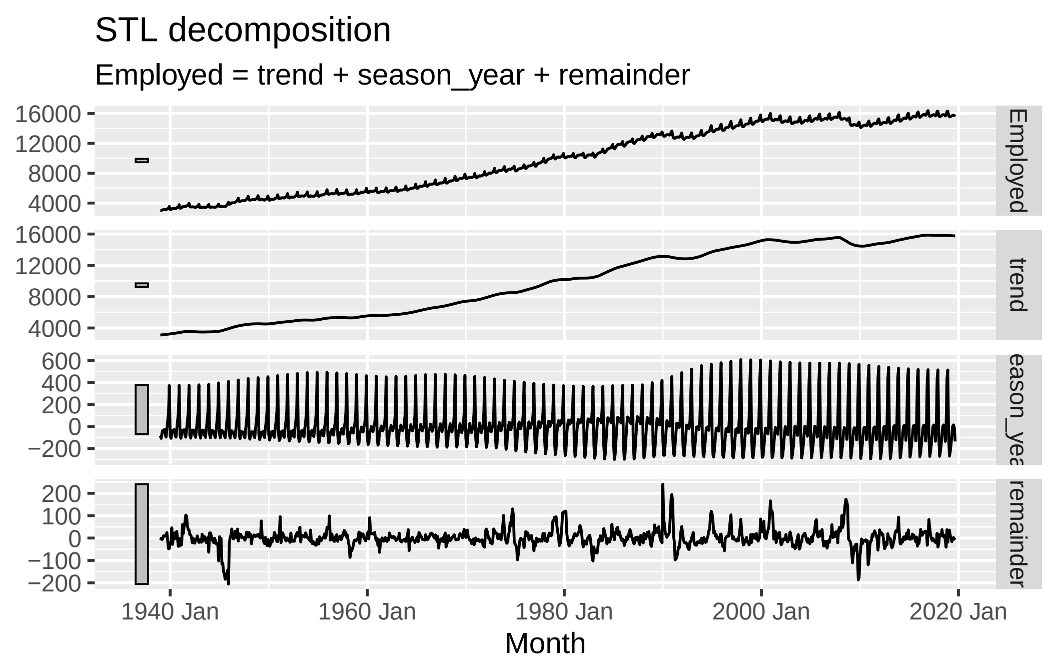
$$\log(a * b) = \log(a) + \log(b)$$

What presented in Note 1 is a good justification of why the STL decomposition is so flexible.

1.4.1 STL in R

```r
retail_employment <- us_employment %>%
  filter(Title == "Retail Trade")
```

```r
stl_model <- retail_employment %>%
  model(STL(Employed~trend()+season(),
            robust = TRUE))

components(stl_model) %>%
  autoplot()
```



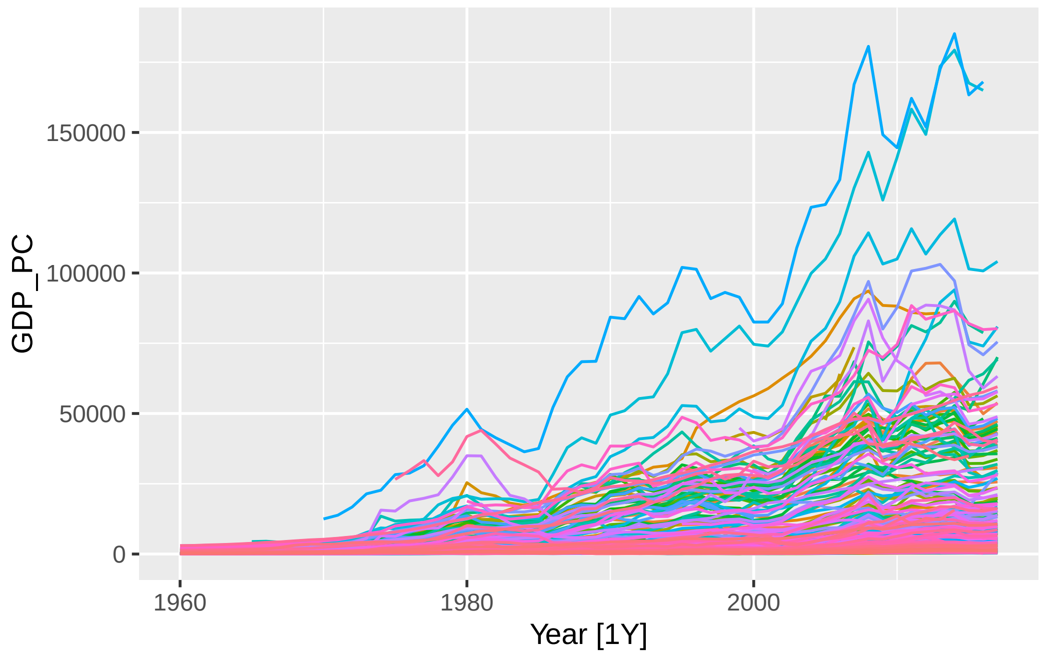
Inside the trend() and season() parameters you can adjust the window argument but by default this will be set to a specific value based on the interval in the time series. To know more about default values refer to the textbook.

2 Exercises

2.1 First

```r
gdp_pc <- global_economy %>%
  mutate(GDP_PC = GDP / Population)

autoplot(gdp_pc, GDP_PC) +
  theme(legend.position = "none")
```

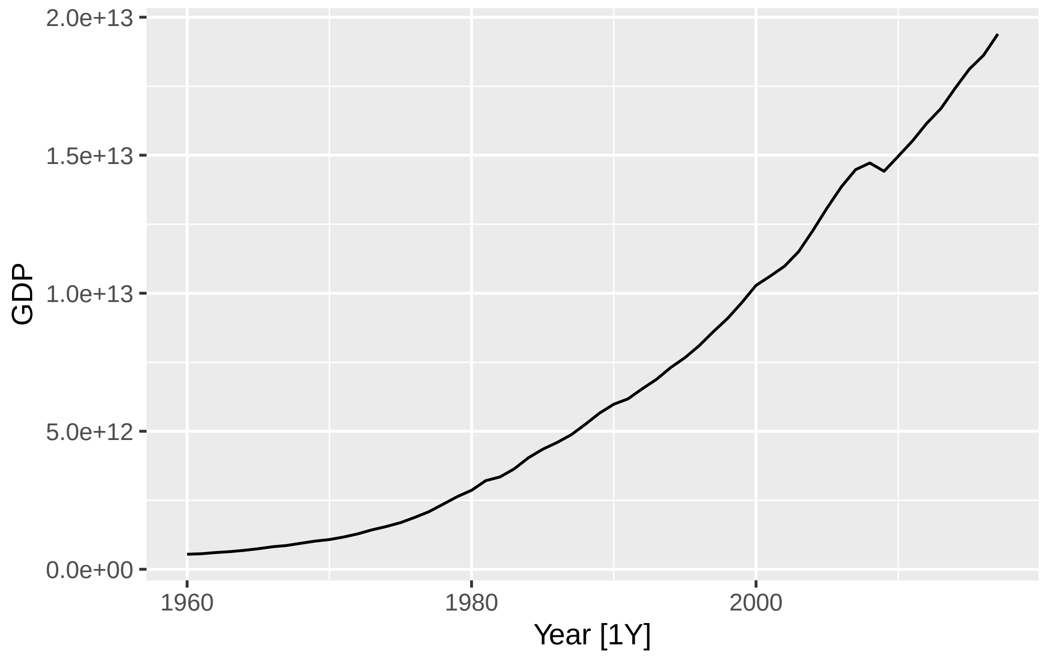
```
Warning: Removed 3242 rows containing missing values or values outside the scale range
(`geom_line()`).
```



2.2 Second

2.2.1 Global Economy

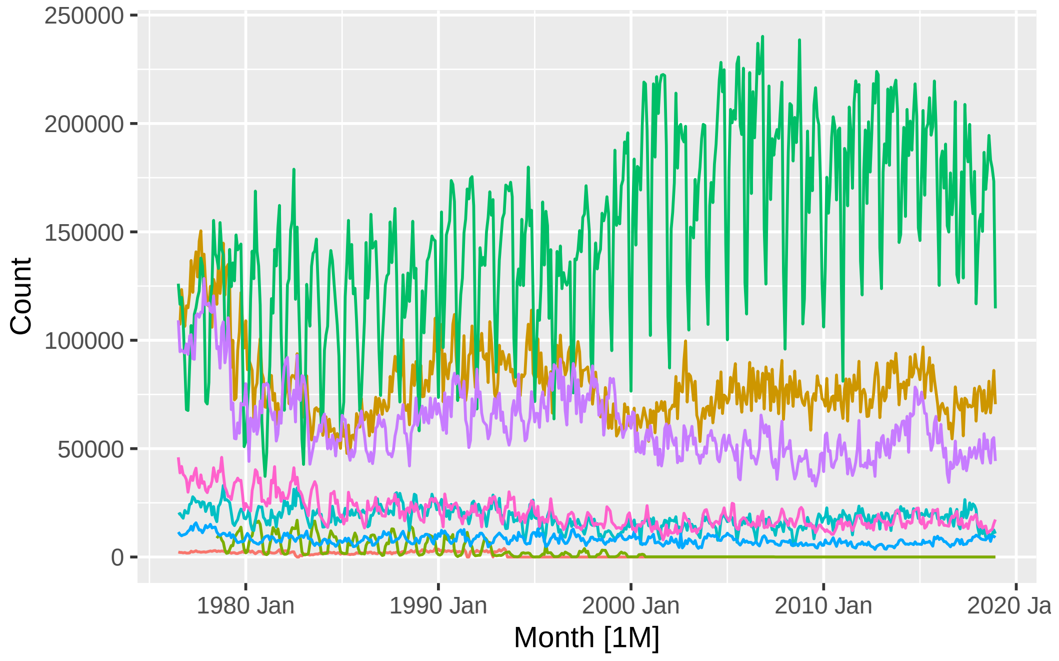
```r
us_gdp <- global_economy %>%
  filter(Code == "USA")

autoplot(us_gdp, GDP)
```



In this case, there is no need to transform the data due to the absence of seasonal patterns.

2.2.2 Livestock

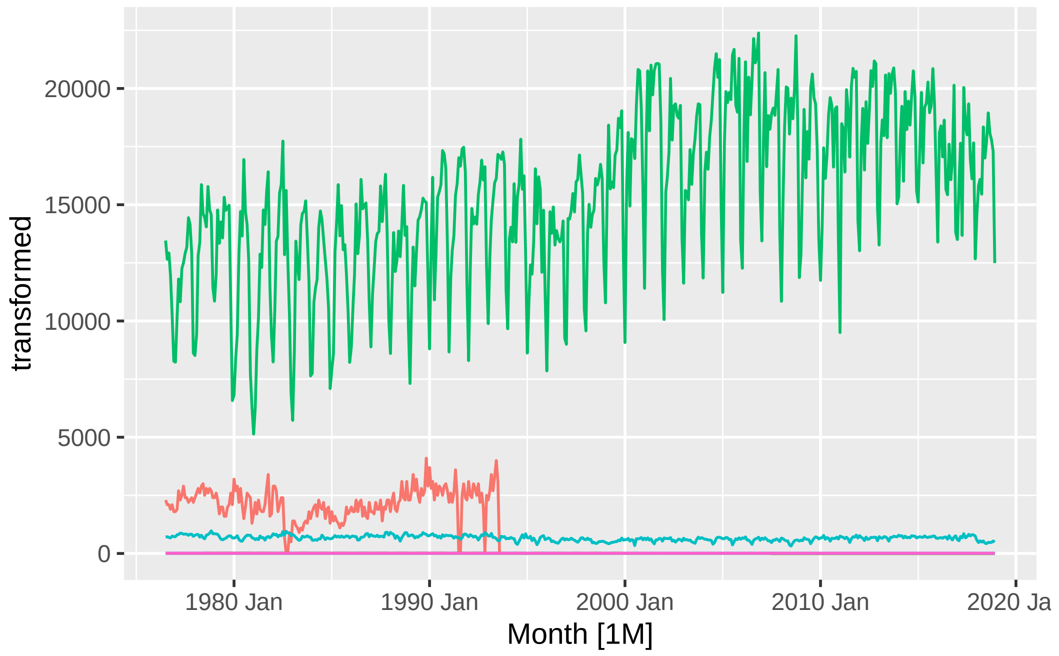
```
bbs <- aus_livestock %>%
  filter(Animal == "Bulls, bullocks and steers")

autoplot(bbs, Count) +
  theme(legend.position = "none")
```



Given the difference in the interval at which the seasonal pattern emerges, it would be useful to use a box_cox transformation. First we estimate lambda using the Guerrero method.

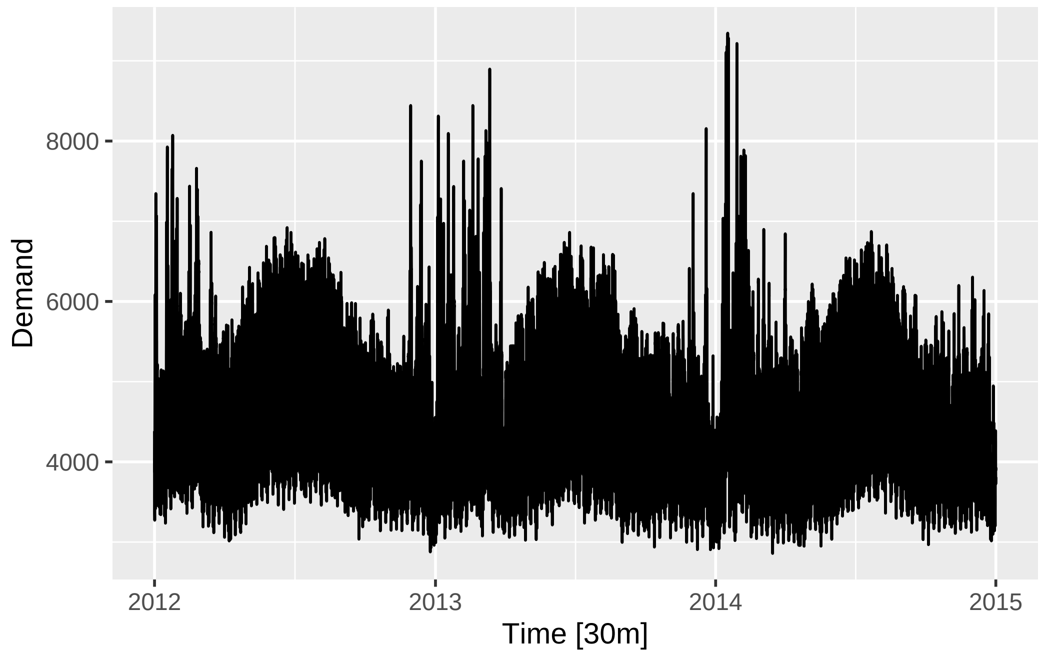
```
# Check the structure of lambda to ensure it contains the required columns
lambda <- features(bbs, Count, features = guerrero)

# Initialize transformed column in bbs
bbs <- bbs %>% mutate(transformed = Count)

# Loop through states and apply the transformation
for (state in lambda$State) {
  lambda_value <- filter(lambda, State == state) %>% pull(lambda_guerrero)
  bbs <- bbs %>%
    mutate(transformed = ifelse(State == state, box_cox(Count, lambda_value), transformed))
}
```

```
autoplot(bbs, transformed) +
  theme(legend.position = "none")
```

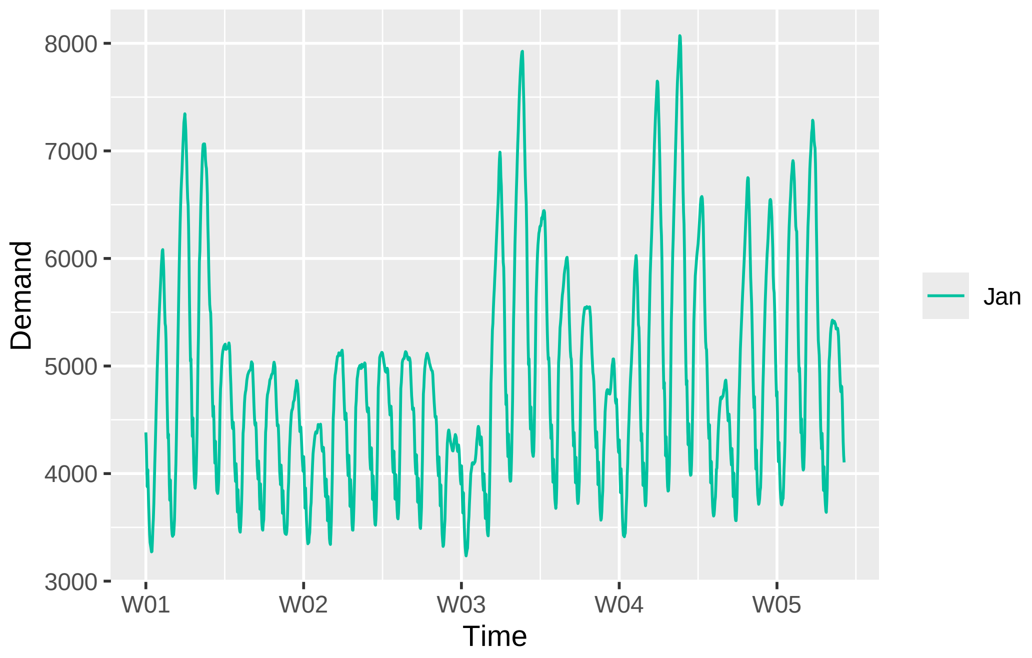


2.2.3 Victoria Electricity Demand

```
autoplot(vic_elec, Demand)
```



```
vic_elec %>%
  filter(year(Time) == 2012, month(Time)==1) %>%
  gg_season(Demand)
```

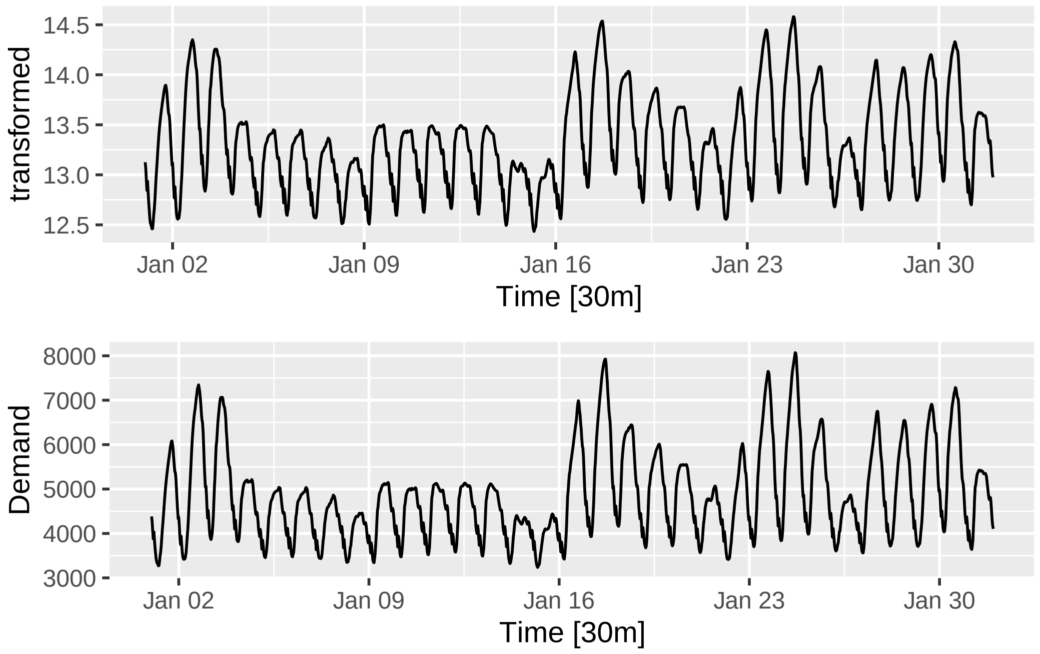


Overall, we can see that it would be useful to use a box cox transformation to make the seasonal variation approximately equal throughout the periods. We follow the usual steps.

```
lambda <- features(vic_elec, Demand, features = guerrero)$lambda_guerrero
#lambda is very close to 0, so close to a simple log transformation
(vic_elec_transformed <- vic_elec %>%
  mutate(transformed = box_cox(Demand, lambda)))
```

```
# A tsibble: 52,608 x 6 [30m] <Australia/Melbourne>
   Time                Demand Temperature Date       Holiday transformed
   <dttm>               <dbl>       <dbl> <date>     <lgl>         <dbl>
 1 2012-01-01 00:00:00  4383.        21.4 2012-01-01 TRUE           13.1
 2 2012-01-01 00:30:00  4263.        21.0 2012-01-01 TRUE           13.1
 3 2012-01-01 01:00:00  4049.        20.7 2012-01-01 TRUE           12.9
 4 2012-01-01 01:30:00  3878.        20.6 2012-01-01 TRUE           12.8
 5 2012-01-01 02:00:00  4036.        20.4 2012-01-01 TRUE           12.9
 6 2012-01-01 02:30:00  3866.        20.2 2012-01-01 TRUE           12.8
 7 2012-01-01 03:00:00  3694.        20.1 2012-01-01 TRUE           12.7
 8 2012-01-01 03:30:00  3562.        19.6 2012-01-01 TRUE           12.7
 9 2012-01-01 04:00:00  3433.        19.1 2012-01-01 TRUE           12.6
10 2012-01-01 04:30:00  3359.        19.0 2012-01-01 TRUE           12.5
# i 52,598 more rows
```

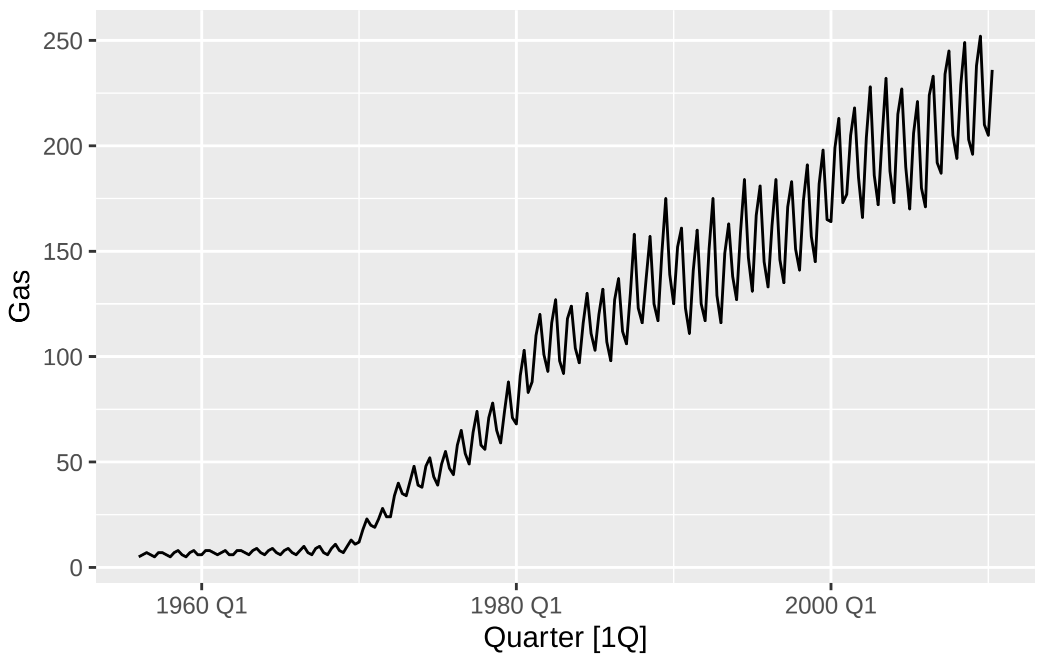
```r
p1 <- vic_elec_transformed %>%
  filter(year(Time) == 2012, month(Time) == 1) %>%
  autoplot(transformed)

p2 <- vic_elec_transformed %>%
  filter(year(Time) == 2012, month(Time) == 1) %>%
  autoplot(Demand)

grid.arrange(p1,p2,nrow=2)
```



Having the data rescaled using box cox, does not help us solve the issue with different seasonality variations. So, contrary to what expected, the box cox transformation is useless in this case.

2.2.4 Gas production

```r
autoplot(aus_production, Gas)
```



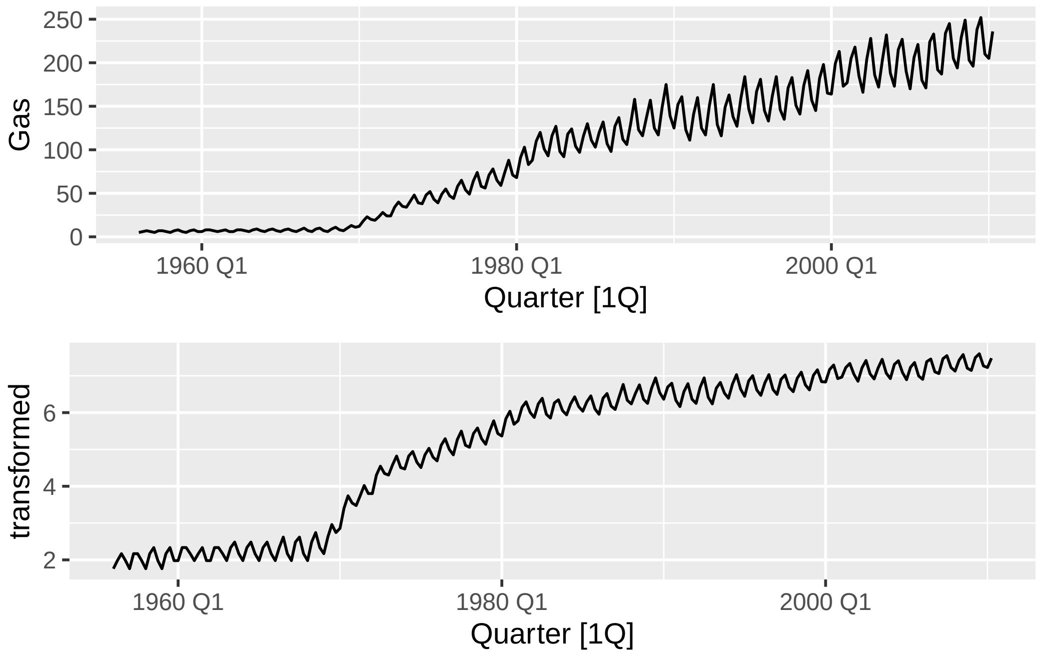
We notice that the incidence of the seasonal variation increases by a lot throughout the years. Therefore, to try and make this variation approximately equal across years, we can use a box_cox transformation. We follow the usual steps.

```
lambda <- features(aus_production, Gas, features = guerrero)$lambda_guerrero

(transformed_aus_prod <- aus_production %>%
  mutate(transformed = box_cox(Gas, lambda)))
```

```
# A tsibble: 218 x 8 [1Q]
   Quarter  Beer Tobacco Bricks Cement Electricity   Gas transformed
     <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>       <dbl>
 1 1956 Q1   284    5225    189    465        3923     5        1.76
 2 1956 Q2   213    5178    204    532        4436     6        1.98
 3 1956 Q3   227    5297    208    561        4806     7        2.17
 4 1956 Q4   308    5681    197    570        4418     6        1.98
 5 1957 Q1   262    5577    187    529        4339     5        1.76
 6 1957 Q2   228    5651    214    604        4811     7        2.17
 7 1957 Q3   236    5317    227    603        5259     7        2.17
 8 1957 Q4   320    6152    222    582        4735     6        1.98
 9 1958 Q1   272    5758    199    554        4608     5        1.76
10 1958 Q2   233    5641    229    620        5196     7        2.17
# i 208 more rows
```

Now we plot this and compare to the original data

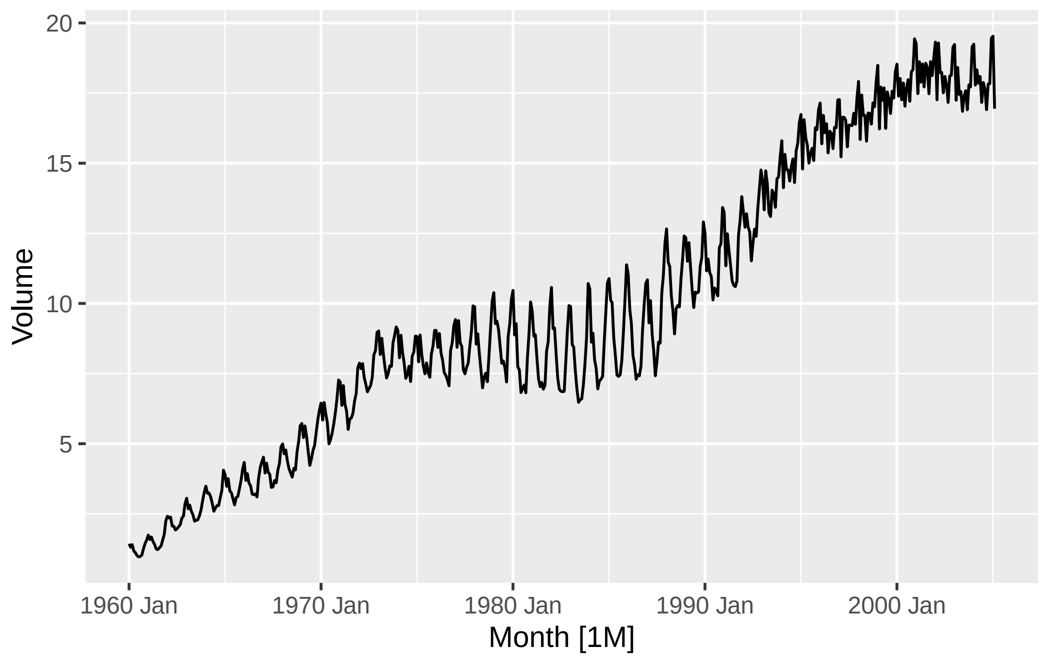
```
p1 <- autoplot(aus_production, Gas)
p2 <- autoplot(transformed_aus_prod, transformed)

grid.arrange(p1,p2,nrow=2)
```



We now see that the variation due to seasonality is approximately equal throughout the years, showing that the time series benefits from a box cox transformation.

2.3 Third

```
autoplot(canadian_gas, Volume)
```



We notice that specifically throughout the time series there is a lot of noise during every years' end. Since the seasonal pattern that we are trying to smooth out is not evolving at a constant rate throughout the time series a box cox transformation would not help in this case. You can compare the plot above with the plot produced for Section 2.2.4 to notice the differences.

2.4 Fourth

Already answered in the 2nd Homework. Here is what was reported there.

2.4.1 Data visualization

We just scratched the surface of what data visualization can do, so in Exercise 3 of HW2 we delve a little deeper into that.

```
labour <- fma::labour
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

```
dates <- seq.Date(from = as.Date("1978-02-01"), to = as.Date("1995-08-01"), by = "month")

labour <- as.data.frame(labour)
labour$date <- dates
names(labour) <- c("labor_force", "month_year")
```

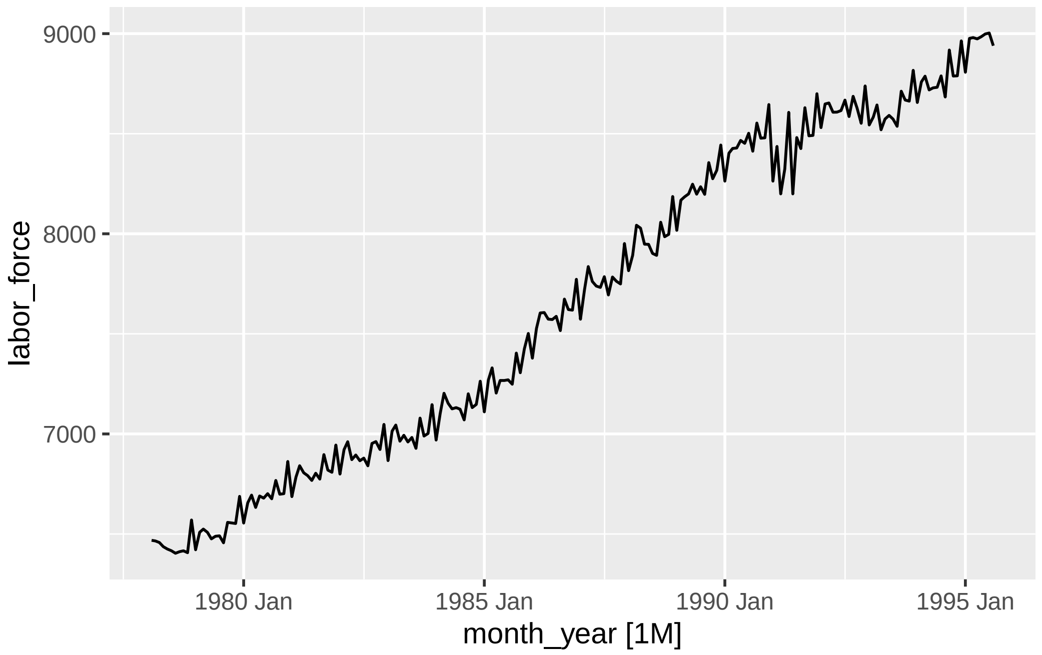
```
labour <- labour %>%
  mutate(month_year = yearmonth(month_year))

(labour_ts <- as_tsibble(labour, index = month_year))
```

```
# A tsibble: 211 x 2 [1M]
   labor_force month_year
         <dbl>      <mth>
 1       6468.   1978 Feb
 2       6465.   1978 Mar
 3       6458.   1978 Apr
 4       6437.   1978 May
 5       6425.   1978 Jun
 6       6417.   1978 Jul
 7       6404.   1978 Aug
 8       6412.   1978 Sep
 9       6416.   1978 Oct
10       6407.   1978 Nov
# i 201 more rows
```

As usual, we import the dataset and perform the routine steps to organize it into a nice tsibble complete with a column for dates. Now, we are ready to produce some plots to explore some features of this time series.

2.4.2 Time Plot

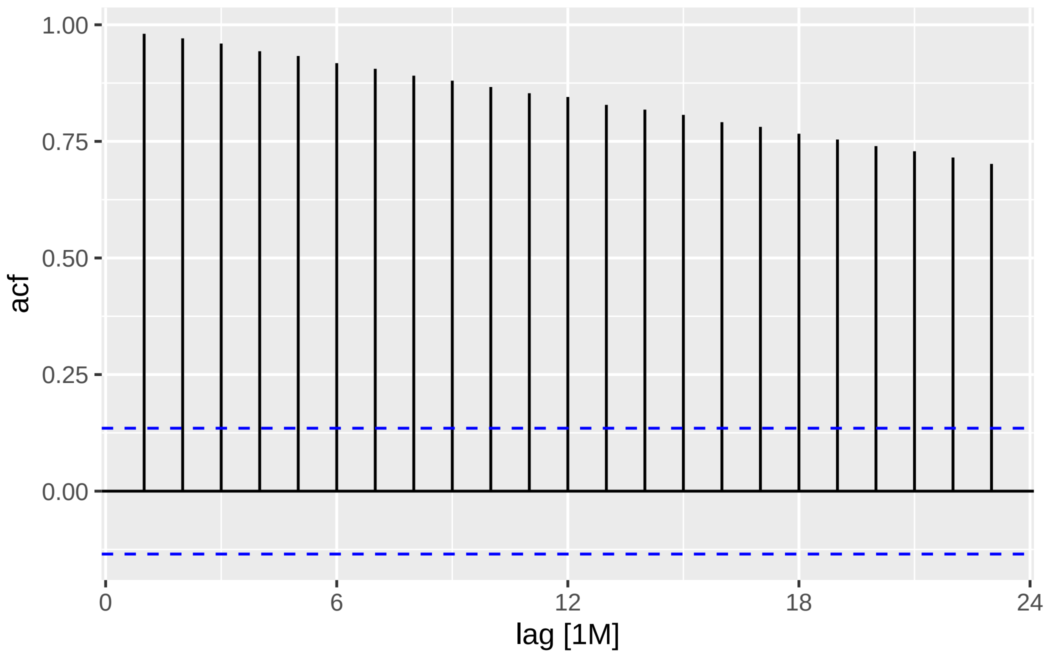
```
autoplot(labour_ts, .vars = labor_force) #time plot
```



You can plot a time plot very easily with `autoplot()`. This allows you to see trends in data.

2.4.3 Correlogram

```
## correlogram
autocorr <- ACF(labour_ts, labor_force)
autoplot(autocorr) #huge autocorrelation across months
```



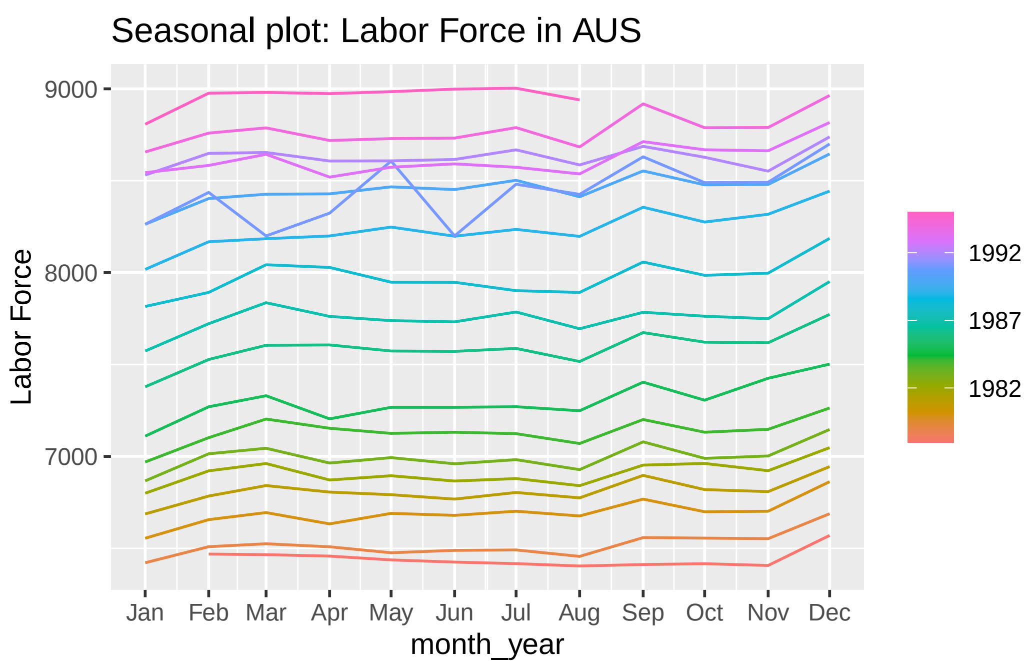
To compute a correlogram you follow these steps:

1. you estimate the auto correlation function with `ACF(dataset, name_of_column_in_dataset)` and assign it to a variable (autocorr in my example above)
2. then you pass that variable as an argument in `autoplot()` and that will do the trick.

2.4.4 Seasonality Plots

They allow you to check for the presence of seasonality (trends based on time periods) in your data.

```
## seasonality plot
gg_season(labour_ts, labor_force) +
  labs(y = "Labor Force",
       title = "Seasonal plot: Labor Force in AUS")
```

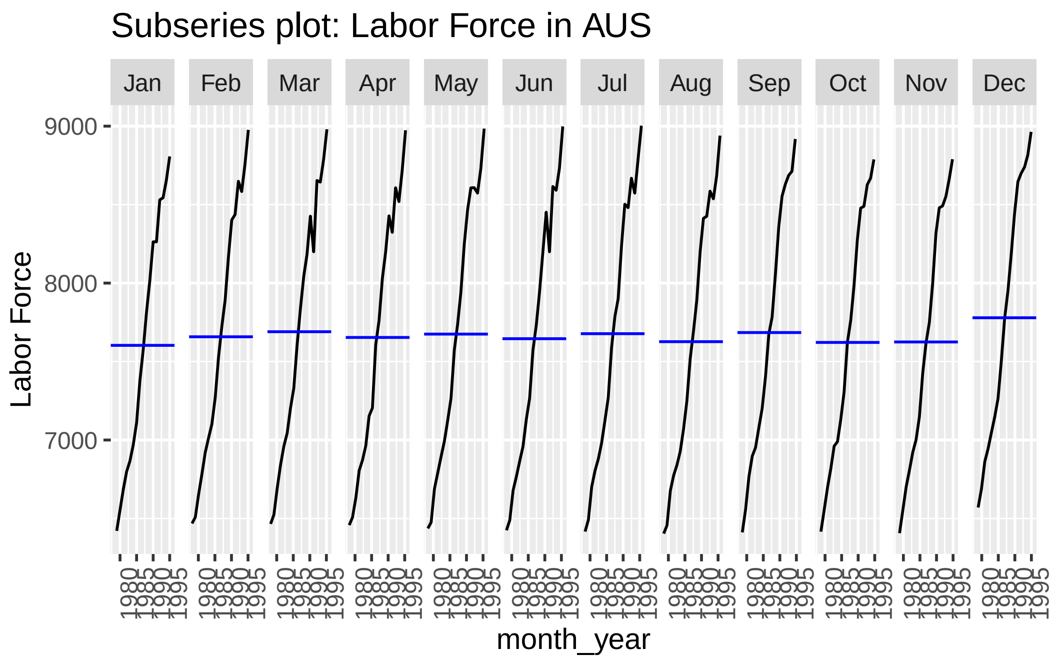


For these plots you use `gg_season()` and specify the dataset and the column to plot. You can add (literally with a + sign) other layers to your plot. In my example I added y labels and a title. If you learn this syntax you can be sure that you can use it on any other ggplot you'll ever do in your life.

From this seasonal plot we learn that every year around December more people join the labor force.

2.4.5 Subseries Plots

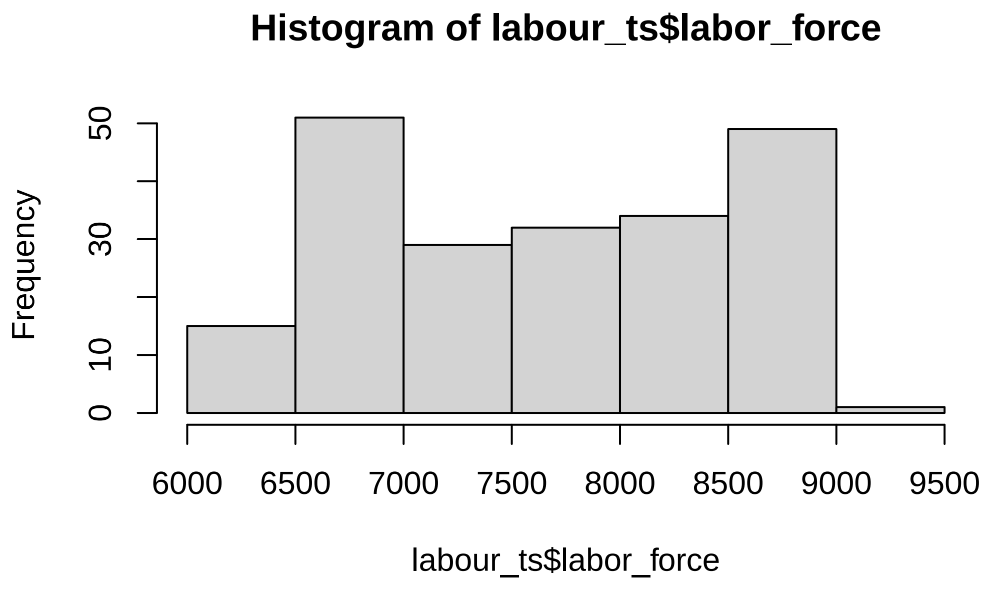
```
gg_subseries(labour_ts, labor_force) +
  labs(y = "Labor Force",
       title = "Subseries plot: Labor Force in AUS")
```



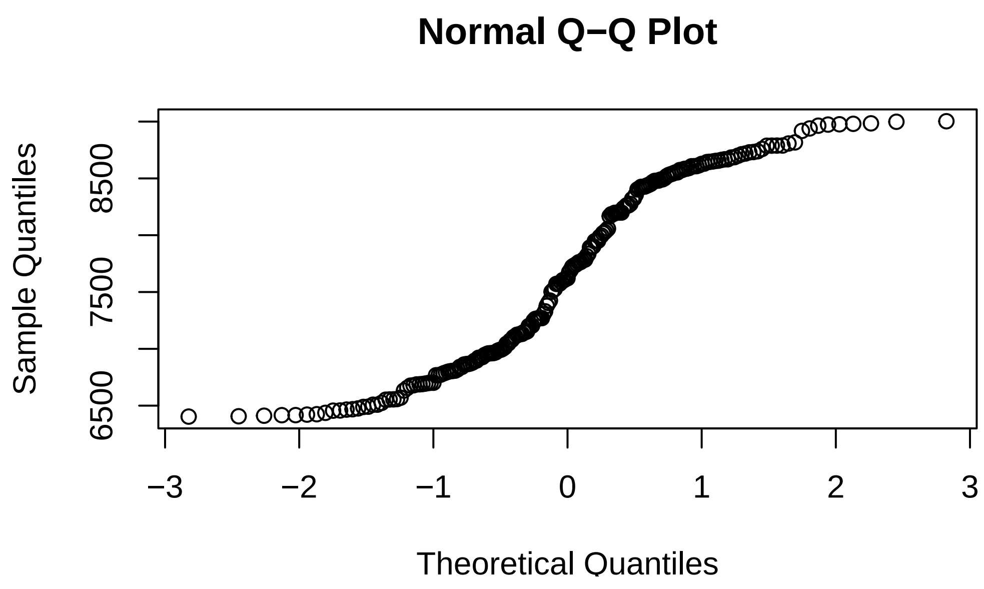
In these plots we can see how the series evolves across months in different years. The syntax is the same as the one for seasonality plots, except for the function `gg_subseries()` of course.

```r
hist(labour_ts$labor_force)                                                    (1)
```

(1) the distribution appears to be bimodal, therefore a smoothing through box-cox would make sense



```r
qqnorm(y = labour_ts$labor_force)
```

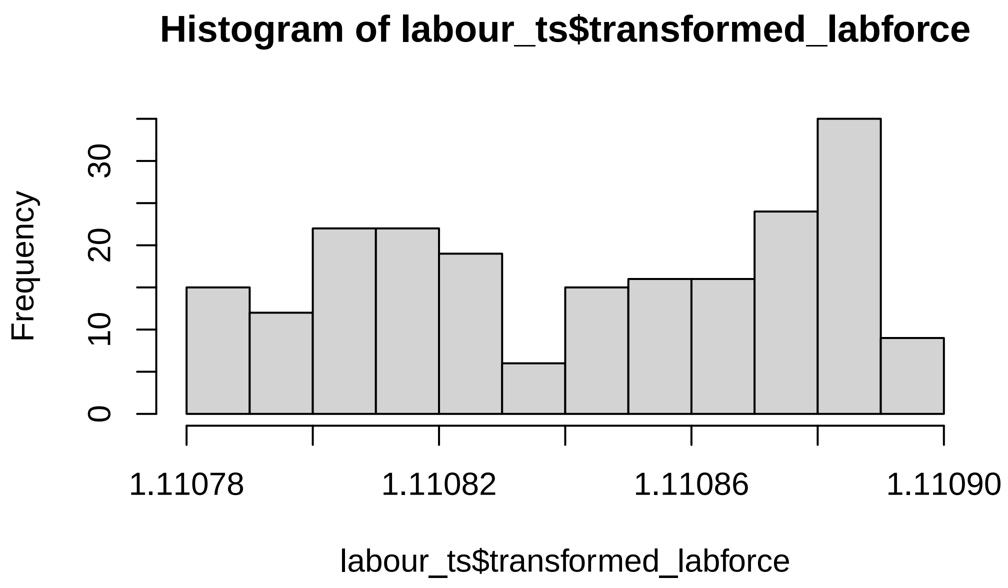


```
lambda <- features(labour_ts, .var = labor_force, features = guerrero)$lambda_guerrero ①
transformed_labforce <- box_cox(labour_ts$labor_force, lambda)                         ②

labour_ts$transformed_labforce <- transformed_labforce                                 ③
labour_ts
```

① estimate the value of lambda using the Guerrero method through the `features()` extraction
② perform the `box_cox()` transformation on the `labor_force` column of the data using the `lambda` estimated at point 1
③ create a new column in `labour_ts` called `transformed_labforce` which contains the values of the box_cox transformation

```
# A tsibble: 211 x 3 [1M]
   labor_force month_year transformed_labforce
         <dbl>      <mth>                <dbl>
 1       6468.   1978 Feb                 1.11
 2       6465.   1978 Mar                 1.11
 3       6458.   1978 Apr                 1.11
 4       6437.   1978 May                 1.11
 5       6425.   1978 Jun                 1.11
 6       6417.   1978 Jul                 1.11
 7       6404.   1978 Aug                 1.11
 8       6412.   1978 Sep                 1.11
 9       6416.   1978 Oct                 1.11
10       6407.   1978 Nov                 1.11
# i 201 more rows
```

```
hist(labour_ts$transformed_labforce)
```



```
#however the distribution remains highly bimodal and normality
#does not seem to be improved by much
```

For the last step of the exercise we compute again the Box-Cox transformation for this time series, but it does not do much to improve upon the normality of this specific time series.

2.5 Fifth

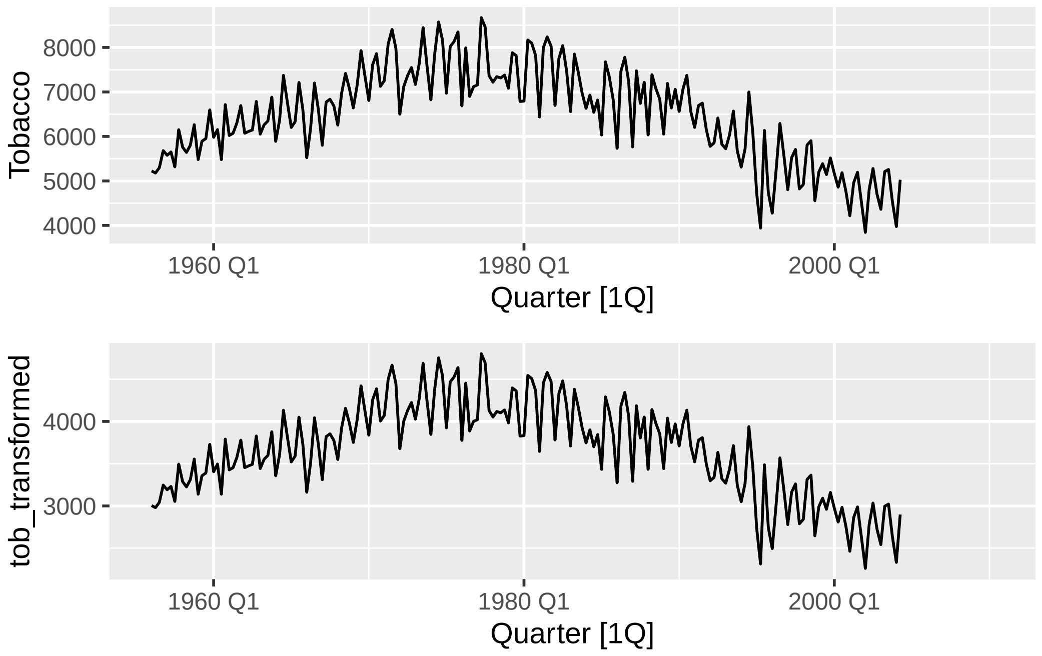
2.5.1 Tobacco

```
lambda <- features(aus_production, Tobacco, features = guerrero)$lambda_guerrero

(trasnformed_tob <- aus_production %>%
  mutate(tob_transformed = box_cox(Tobacco, lambda)))
```

```
# A tsibble: 218 x 8 [1Q]
   Quarter  Beer Tobacco Bricks Cement Electricity   Gas tob_transformed
     <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>           <dbl>
 1 1956 Q1   284    5225    189    465        3923     5           3004.
 2 1956 Q2   213    5178    204    532        4436     6           2979.
 3 1956 Q3   227    5297    208    561        4806     7           3042.
 4 1956 Q4   308    5681    197    570        4418     6           3246.
 5 1957 Q1   262    5577    187    529        4339     5           3191.
 6 1957 Q2   228    5651    214    604        4811     7           3230.
 7 1957 Q3   236    5317    227    603        5259     7           3053.
 8 1957 Q4   320    6152    222    582        4735     6           3495.
 9 1958 Q1   272    5758    199    554        4608     5           3287.
10 1958 Q2   233    5641    229    620        5196     7           3225.
# i 208 more rows
```

```r
p1 <- autoplot(aus_production, Tobacco)
p2 <- autoplot(trasnformed_tob, tob_transformed)

grid.arrange(p1,p2,nrow=2)
```

```
Warning: Removed 24 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 24 rows containing missing values or values outside the scale range
(`geom_line()`).
```

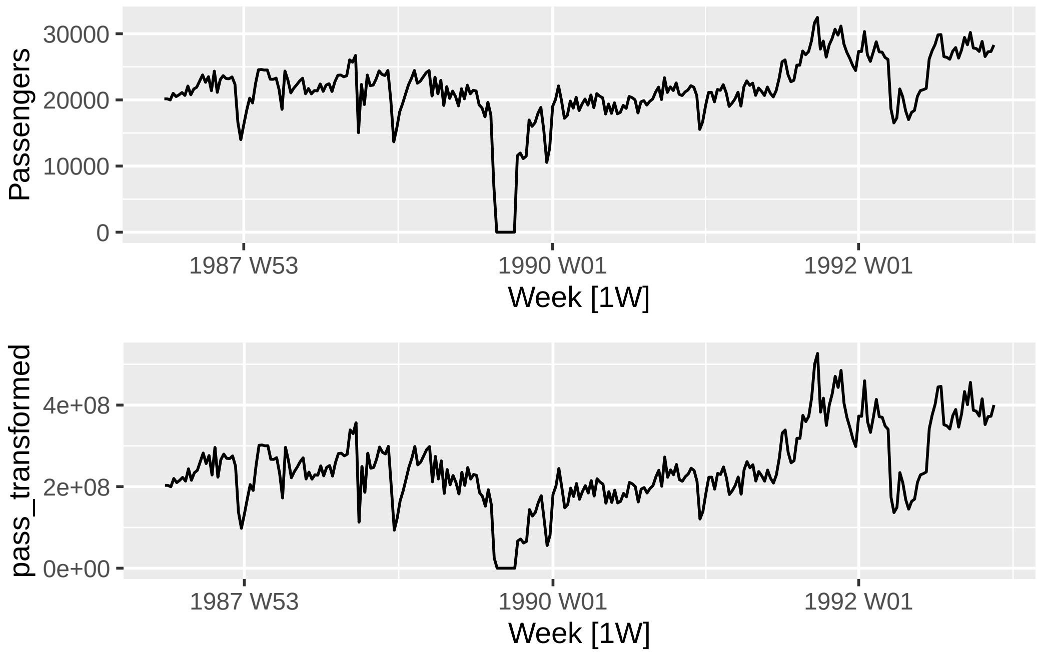


2.5.2 Ansett passengers

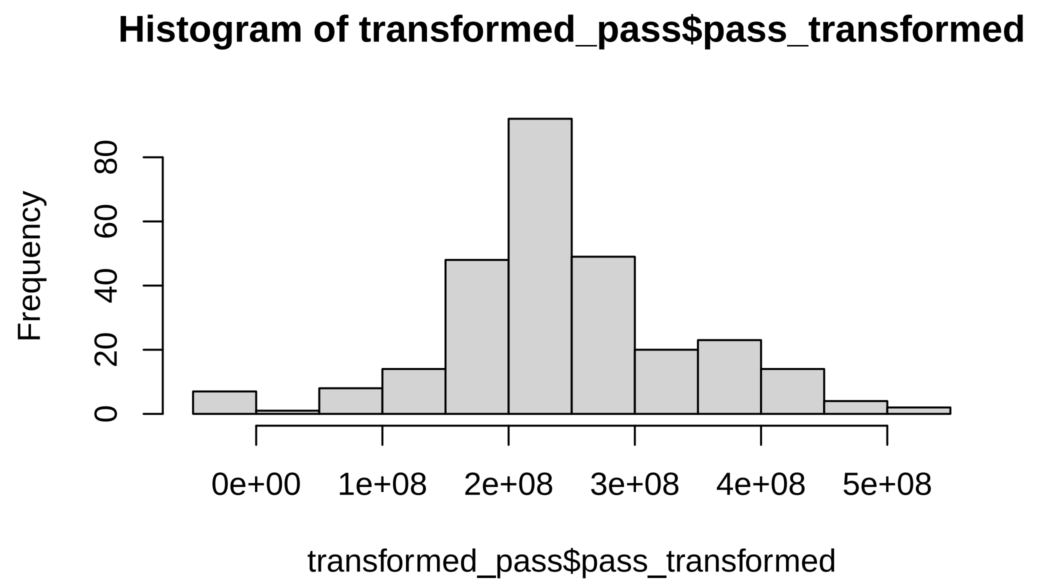
```r
filtered_ansett <- filter(ansett, Airports == "MEL-SYD", Class == "Economy")
lambda <- features(filtered_ansett, Passengers, features = guerrero)$lambda_guerrero

(transformed_pass <- filtered_ansett %>%
  mutate(pass_transformed = box_cox(Passengers, lambda)))
```

```
# A tsibble: 282 x 5 [1W]
# Key:       Airports, Class [1]
       Week Airports Class   Passengers pass_transformed
     <week> <chr>    <chr>        <dbl>            <dbl>
 1 1987 W26 MEL-SYD  Economy      20167       203213842.
 2 1987 W27 MEL-SYD  Economy      20161       203092945.
 3 1987 W28 MEL-SYD  Economy      19993       199722456.
 4 1987 W29 MEL-SYD  Economy      20986       220053743.
 5 1987 W30 MEL-SYD  Economy      20497       209918530.
 6 1987 W31 MEL-SYD  Economy      20770       215547379.
 7 1987 W32 MEL-SYD  Economy      21111       222682889.
 8 1987 W33 MEL-SYD  Economy      20675       213580173.
 9 1987 W34 MEL-SYD  Economy      22092       243858478.
10 1987 W35 MEL-SYD  Economy      20772       215588890.
# i 272 more rows
```

```
p1 <- autoplot(filtered_ansett, Passengers)
p2 <- autoplot(transformed_pass, pass_transformed)

grid.arrange(p1,p2,nrow=2)
```



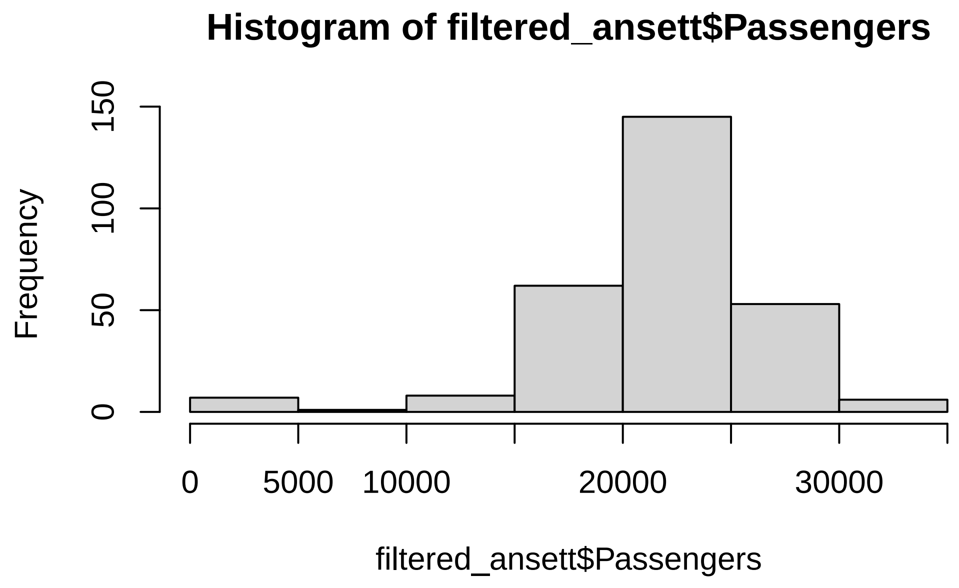
```
hist(transformed_pass$pass_transformed)
```



```r
shapiro.test(transformed_pass$pass_transformed)
```

```
    Shapiro-Wilk normality test

data:  transformed_pass$pass_transformed
W = 0.97042, p-value = 1.459e-05
```

```r
hist(filtered_ansett$Passengers)
```



```r
shapiro.test(filtered_ansett$Passengers)
```

```
    Shapiro-Wilk normality test
```

```
data:  filtered_ansett$Passengers
W = 0.86005, p-value = 2.698e-15
```

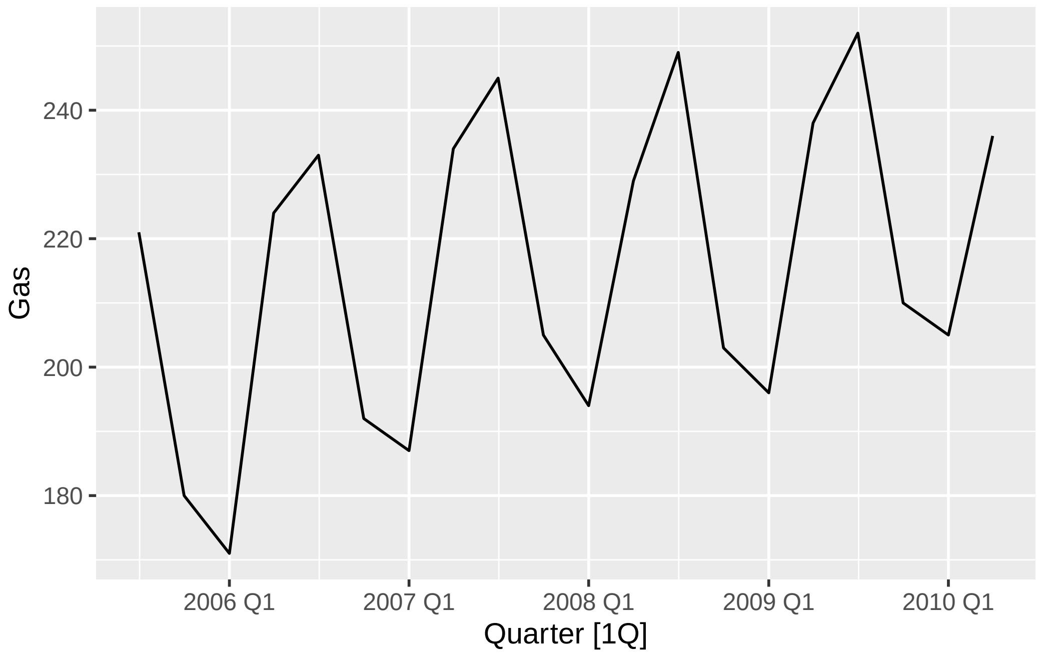
The box cox transformation seems to improve on the normality of the distribution of passengers but it fails to produce a non-significant result for the Shapiro-Wilk normality test. Therefore, the series remains not normally distributed and the distribution is not very useful.

2.6 Sixth

This can be proven following the procedure described in the textbook under the "Moving averages of moving averages" section.

2.7 Seventh

```r
gas <- tail(aus_production, 5*4) |> select(Gas)
autoplot(gas, Gas)
```



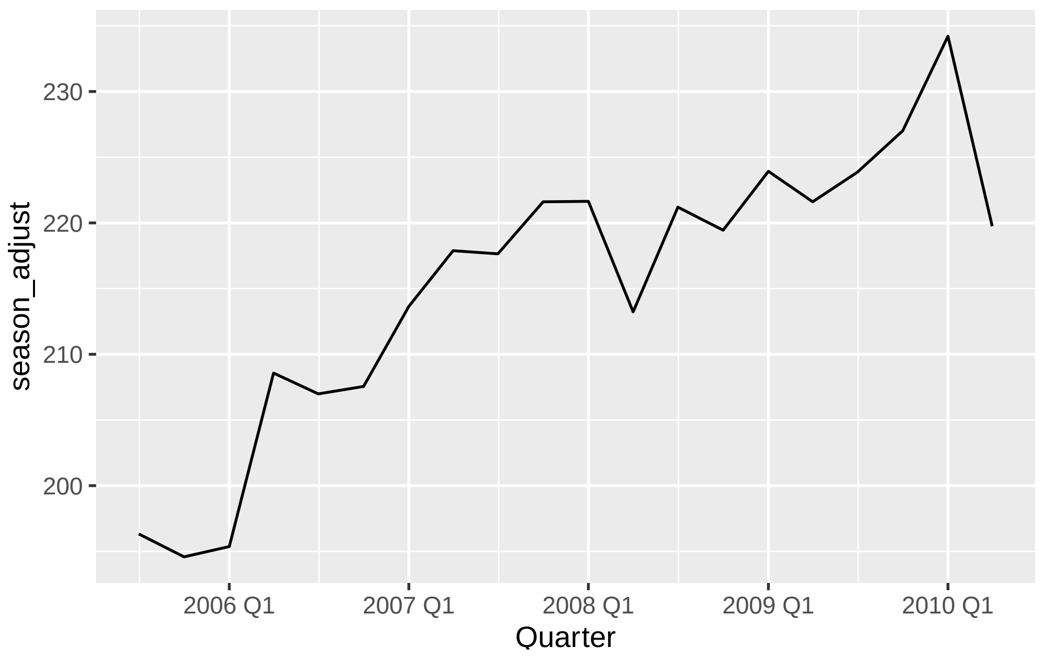
There are seasonal fluctuations and each year seems to exhibit the same pattern with a lower quantity in Q4 and Q1 followed by an increase in the other two quarters.

```r
dcmp <- model(gas, classical_decomposition(Gas, type = "multiplicative"))
(cmp <- components(dcmp))
```

```
# A dable: 20 x 7 [1Q]
# Key:     .model [1]
# :        Gas = trend * seasonal * random
   .model                       Quarter   Gas trend seasonal random season_adjust
   <chr>                          <qtr> <dbl> <dbl>    <dbl>  <dbl>         <dbl>
 1 "classical_decomposition(G~ 2005 Q3   221   NA      1.13  NA             196.
 2 "classical_decomposition(G~ 2005 Q4   180   NA      0.925 NA             195.
 3 "classical_decomposition(G~ 2006 Q1   171  200.     0.875  0.974         195.
 4 "classical_decomposition(G~ 2006 Q2   224  204.     1.07   1.02          209.
 5 "classical_decomposition(G~ 2006 Q3   233  207      1.13   1.00          207.
 6 "classical_decomposition(G~ 2006 Q4   192  210.     0.925  0.987         208.
 7 "classical_decomposition(G~ 2007 Q1   187  213      0.875  1.00          214.
 8 "classical_decomposition(G~ 2007 Q2   234  216.     1.07   1.01          218.
 9 "classical_decomposition(G~ 2007 Q3   245  219.     1.13   0.996         218.
10 "classical_decomposition(G~ 2007 Q4   205  219.     0.925  1.01          222.
11 "classical_decomposition(G~ 2008 Q1   194  219.     0.875  1.01          222.
12 "classical_decomposition(G~ 2008 Q2   229  219      1.07   0.974         213.
13 "classical_decomposition(G~ 2008 Q3   249  219      1.13   1.01          221.
14 "classical_decomposition(G~ 2008 Q4   203  220.     0.925  0.996         219.
15 "classical_decomposition(G~ 2009 Q1   196  222.     0.875  1.01          224.
16 "classical_decomposition(G~ 2009 Q2   238  223.     1.07   0.993         222.
17 "classical_decomposition(G~ 2009 Q3   252  225.     1.13   0.994         224.
18 "classical_decomposition(G~ 2009 Q4   210  226      0.925  1.00          227.
19 "classical_decomposition(G~ 2010 Q1   205   NA      0.875 NA             234.
20 "classical_decomposition(G~ 2010 Q2   236   NA      1.07  NA             220.
```

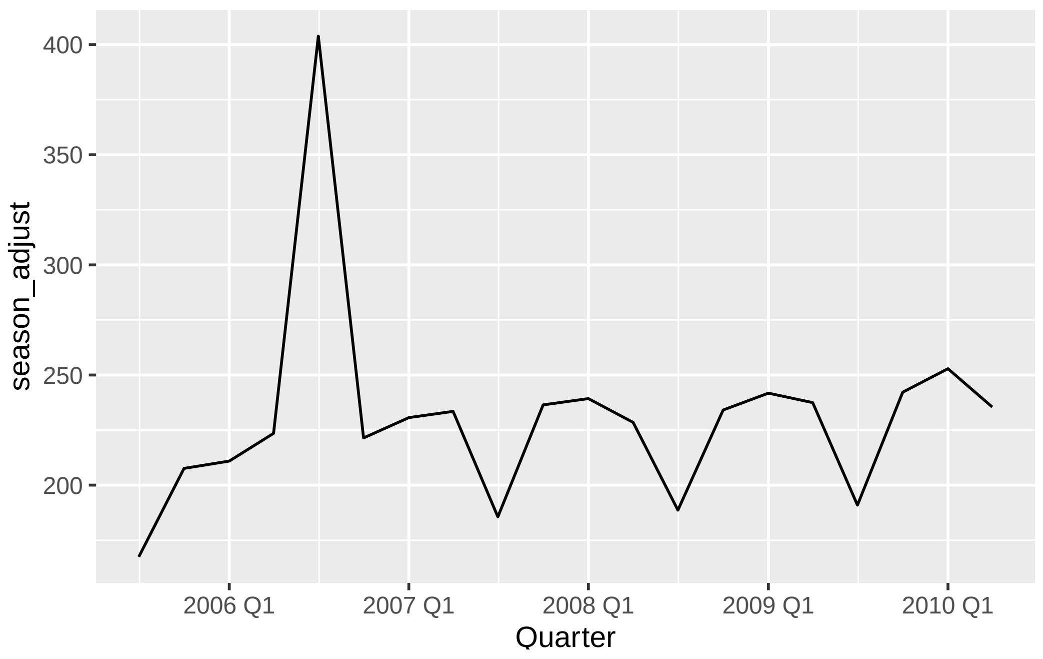
Looking at the seasonal component, we see that it is indeed lower in Q1 and Q4 and higher in the remaining quarters. These results support what observed through the visualization of the data.

```r
ggplot(cmp, aes(x=Quarter, y=season_adjust)) +
  geom_line()
```



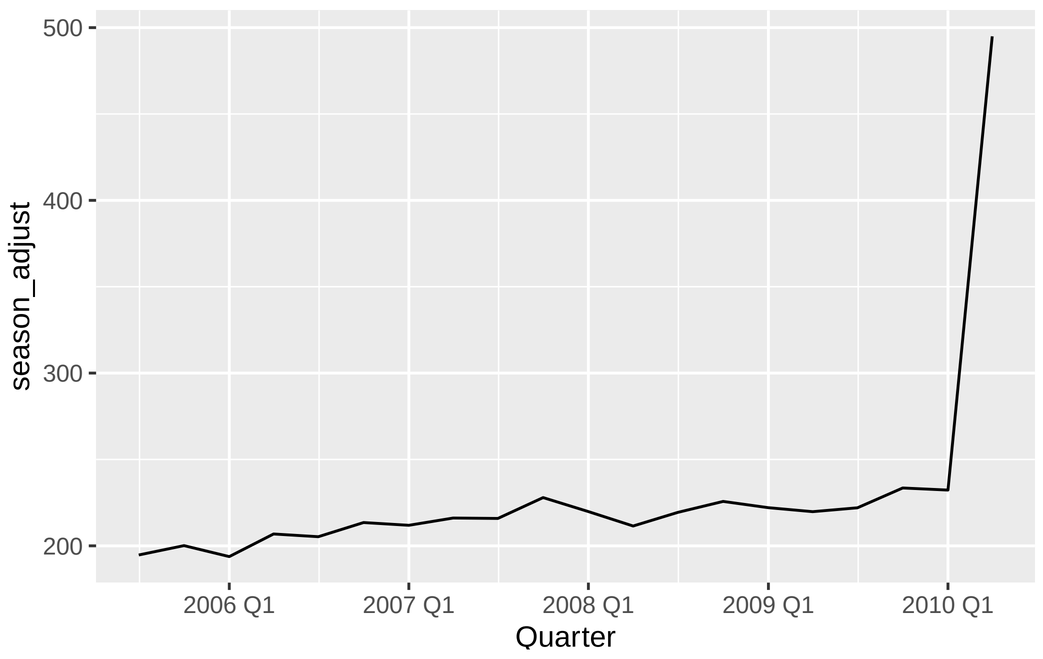
The seasonally adjusted time series is plotted above as it is already computed when the decomposition is applied. However, if one were to compute it by hand the process is simple. Since we want to get rid of the seasonal component, we divide the variable at time t by the estimated seasonal component at that same time. By doing this, we obtain a seasonally adjusted time series. This follows by the way in which the multiplicative decomposition is specified.

```r
gas[5,1] <- 533

cmp_gas <- components(model(.data = gas, classical_decomposition(Gas,"multiplicative")))

ggplot(cmp_gas, aes(x=Quarter, y=season_adjust))+
  geom_line()
```



When placing it in the middle, the outlier seems to not do too much harm to our seasonally adjusted time series

```r
gas <- tail(aus_production, 5*4) |> select(Gas)

gas[dim(gas)[1], 1] <- 536

cmp_gas <- components(model(.data = gas, classical_decomposition(Gas,"multiplicative")))

ggplot(cmp_gas, aes(x=Quarter, y=season_adjust))+
  geom_line()
```



However, we notice that when placing the outlier at the end of the time series we get a very strange result. Here are some explanations for why we observe this.

1. Influence on Seasonal Component Estimation

- Seasonal adjustment methods, such as those based on **moving averages** or decomposition techniques (e.g., X-13ARIMA-SEATS, STL), estimate the seasonal pattern by averaging across multiple periods.
- An outlier at the end of the series can distort these averages, leading to an inaccurate estimation of the seasonal component. This affects how much of the observed value is attributed to seasonality versus the actual trend or irregular components.

2. Trend Distortion

- Outliers can pull the estimated trend upwards or downwards, especially at the series' boundary. This happens because smoothing methods used in trend extraction often struggle to properly differentiate between genuine trend changes and one-off anomalies.
- Since the seasonal adjustment process typically involves removing the trend before isolating seasonality, a distorted trend can lead to incorrect seasonally adjusted results.

3. Boundary Effects

- Statistical smoothing methods (e.g., moving averages) are less reliable at the edges of a time series because they lack enough surrounding data for accurate calculations.
- If an outlier is present at the end of the series, the adjustment algorithms often "overreact" because they can't use future data points to balance the smoothing. This results in exaggerated effects in the seasonally adjusted data near the boundary.

4. Irregular Component Interference

- Seasonally adjusted results include both the trend and any irregular (non-seasonal, non-trend) fluctuations. An outlier is treated as part of the irregular component, but if not properly handled, it can introduce noise that disrupts the seasonal adjustment process, particularly when the irregular component becomes disproportionately large.