FORECASTING AND PREDICTIVE MODELING

LECTURE 5

Lect. PhD. Oneț-Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2024 - 2025

In Lecture 4

- Time series visualizations
  - Lag plots
  - Autocorrelation
- Stationary time series
- Differencing

Today

- Time series transformations
- Time series decomposition

- Let us look again at the Google stock data that we have used last week.
- The initial data is clearly not stationary, it shows a *wandering behaviour* which is specific for financial and economic data.

```
google_2018 <- gafa_stock |>
  filter(Symbol == "GOOG", year(Date) == 2018) |>
  select(Date, Close)

google_2018
google_2018 |> autoplot() + geom_point()
google_2018 |> ACF() |>
  autoplot() + geom_point()
```

- If we difference it, we get a stationary time series.
- The differenced data has values in the [-57.6, 63.2] interval and its mean value is 0.12.
- Moreover, besides being stationary, the differenced data is also a white noise.

```
google_2018 |> autoplot(difference(Close)) + geom_point()
google_2018 |> features(difference(Close), unitroot_kpss)

google_2018 |> ACF(difference(Close)) |>
  autoplot() + geom_point()

#first value after difference is NA. max, min, mean cannot handle it
diff <- difference(google_2018$Close) |> tail(250)
diff
max(diff)
min(diff)
mean(diff)
```

Random walk model

- A differenced time series is the *change* between consecutive observations: $y'_t = y_t - y_{t-1}$. The resulting series will have one less observation, because we do not have y'_1.
- Since the differenced time series is white noise, we can conclude that the daily change in stock price is a random amount, uncorrelated with the previous day.
- When the differenced time series is white noise with 0 mean, we can write this as: $y_t - y_{t-1} = \epsilon_t$ (where ϵ_t denotes the white noise, assumed to be normally and independently distributed with mean 0)
- If we rearrange the terms, we have the *random walk* model: $y_t = y_{t-1} + \epsilon_t$
- This means that every observation is equal to the previous one plus a random noise.

- In case of the random walk model, the best forecasting method is to simply forecast observation $y_{T+1|T} = y_T$, since the expected value of the error term is zero.

Time series transformations

- Before decomposing a time series it might be useful to perform some form of adjustment on the data.
- These adjustments can help simplify the data, remove variation or the effect of other factors.
- The simpler the time series, the easier to model it and the more accurate the resulting forecast.

Calendar adjustment I

- Calendar adjustment is the method for removing the *calendar effect* from a time series.
- Calendar effect can depend on:
  - the number of working days (in general between 18 and 23 in a month) or simply the number of days in a month
  - the number of trading days in a month (more people shop on Saturday than on Monday, so months with 5 Saturdays have higher retail shopping values)
  - the timing of certain public holidays (ex. Easter, which may be in April or May or partially in both or the Chinese New Year which is in general in February, but can be in January)
  - the occurrence of a leap year
  - on the fact that some public holidays might overlap with non-working days

Calendar adjustment II

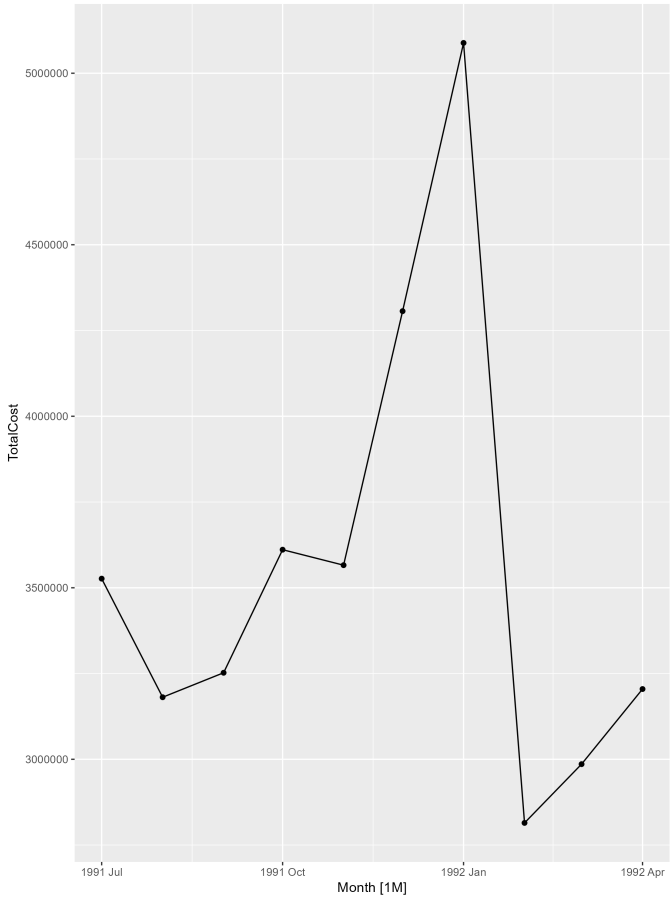
- *Working day adjustment* focuses on the changing number of working days. This can be eliminated by simply dividing the monthly data by the number of working days, to get an average value.
- Other calendar adjustments might be more complicated to remove.

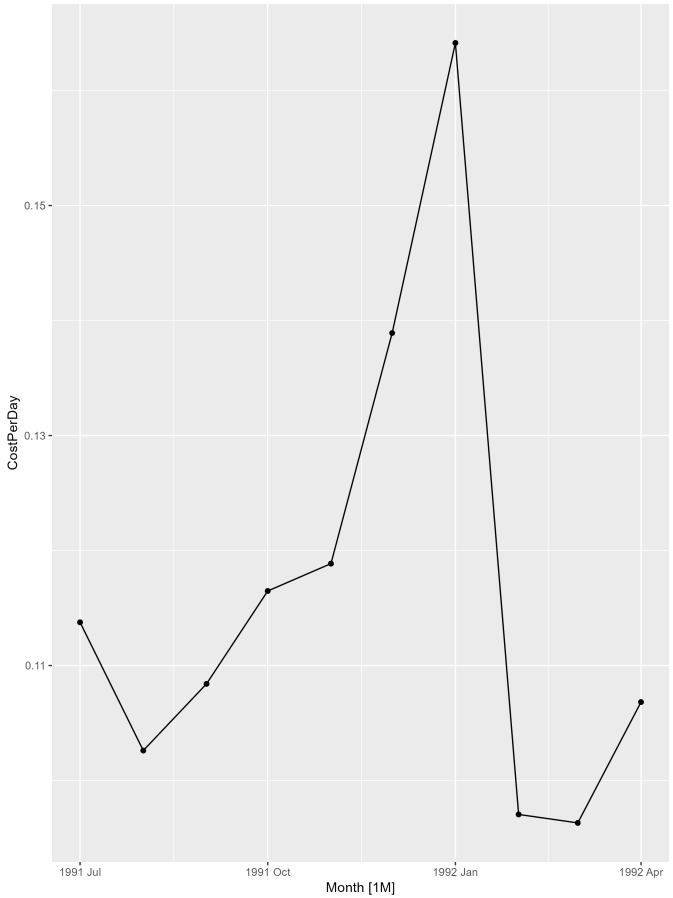
- One reason why time series decomposition is performed, is to do *seasonal adjustment*: remove the seasonal effects from the data (in many cases we are rather interested in the trend, not the seasonal variations).
- This includes removing the seasonal component from the data, but it is often combined with calendar adjustments as well.

- EuroStat's Guidelines on seasonal adjustment can be found here `https://ec.europa.eu/eurostat/documents/3859598/6830795/KS-GQ-15-001-EN-N.pdf`, while the full Handbook on Seasonal Adjustment (2018 edition) can be found here: `https://ec.europa.eu/eurostat/documents/3859598/8939616/KS-GQ-18-001-EN-N.pdf/7c4d120a-4b8a-441b-aefd-6afe81a7cf59?t=1533194231000`

Calendar adjustment example

```
A10 <- PBS |>
  filter(ATC2 == "A10") |>
  select(Month, Concession, Type, Cost) |>
  summarize(TotalCost = sum(Cost)) |>
  mutate(Cost = TotalCost / 1000000)

A10_oneYear = A10 |> head(n= 10)

p1 <- A10_oneYear |> autoplot() + geom_point()
p1
A10_oneYear |>
  mutate(CostPerDay = Cost / days_in_month(Month)) -> A10_oneYear
p2 <- A10_oneYear |> autoplot(CostPerDay) + geom_point()
p2

plot_grid(p1, p2)
```

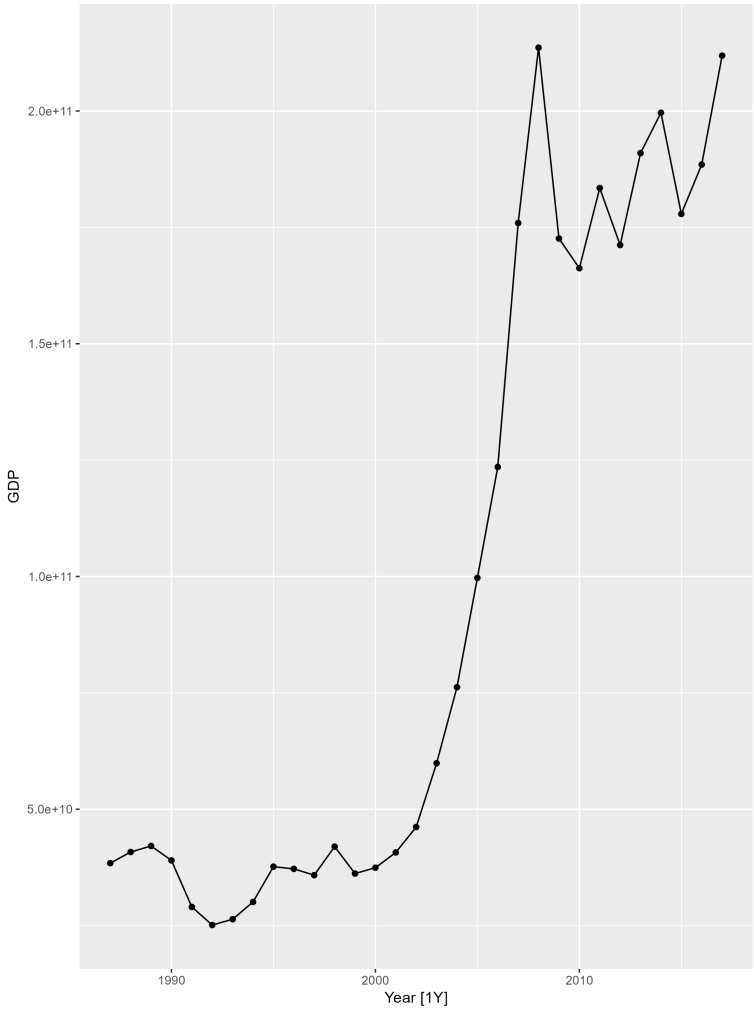

5000000
4500000
4000000
3500000
3000000
TotalCost
1991 Jul
1991 Oct
1992 Jan
1992 Apr
Month [1M]

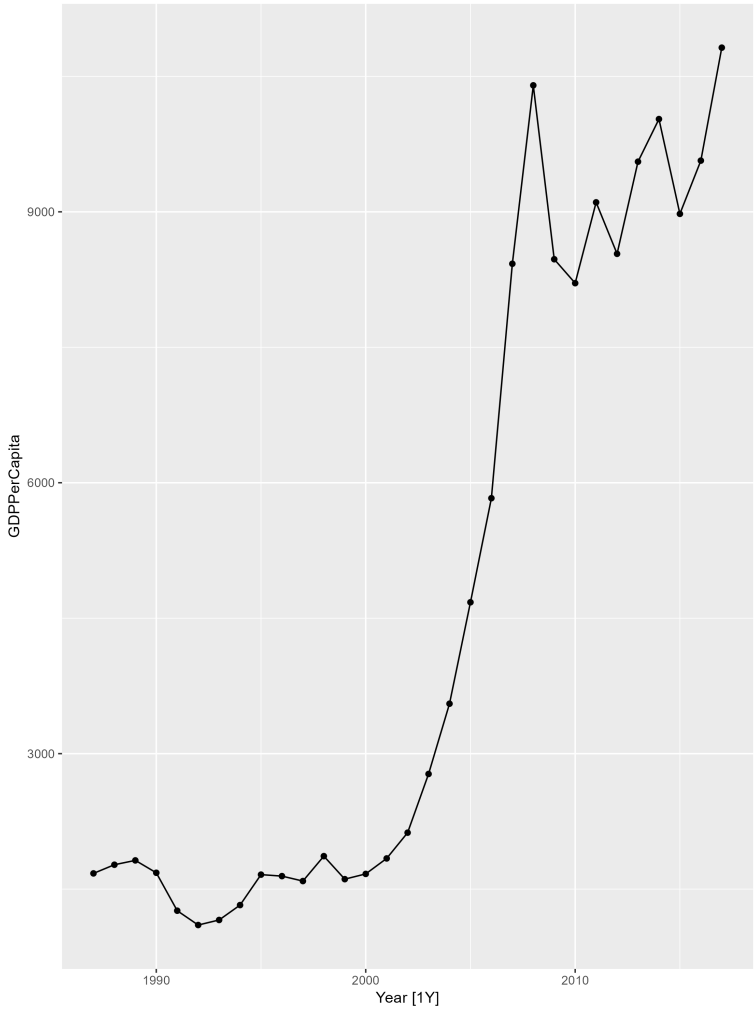

0.15
0.13
0.11
CostPerDay
1991 Jul
1991 Oct
1992 Jan
1992 Apr
Month [1M]

Population adjustment

- Data that can be affected by population changes should be adjusted to provide per-capita values (or per thousand, million, etc. persons).
- For example, an increase in the number of hospital beds, might be simply caused by an increase in the population, not neccessarily an improvement in the health system.

```
global_economy |>
  filter(Country == "Romania") -> ro_economy

ro_economy |> autoplot(GDP) + geom_point()
ro_economy |>
  filter(is.na(GDP) == FALSE, is.na(Population) == FALSE) -> ro_economy
ro_economy
ro_economy |> autoplot(GDP) + geom_point()

ro_economy |>
  mutate(GDPPerCapita = GDP / Population) -> ro_economy

ro_economy |> autoplot(GDPPerCapita) + geom_point()
```


2.0e+11
1.5e+11
1.0e+11
5.0e+10
GDP
1990
2000
2010
Year [1Y]


9000
6000
3000
GDPPerCapita
1990
2000
2010
Year [1Y]

Inflation adjustment

- Data that is affected by the value of the money also has to be adjusted: the price of a house today cannot be directly compared to the price of a house 10 years ago.
- This is why such time series are adjusted so that all values are reported according to the values for a specific year.
- For this, a so-called *price-index* is used. For example, for the price of goods, the *Consumer Price Index (CPI)* is used.
- If we denote the price index for year t by z_t, the original value for year t by y_t and we want the adjusted price, x_t for 2023, we can compute it by:

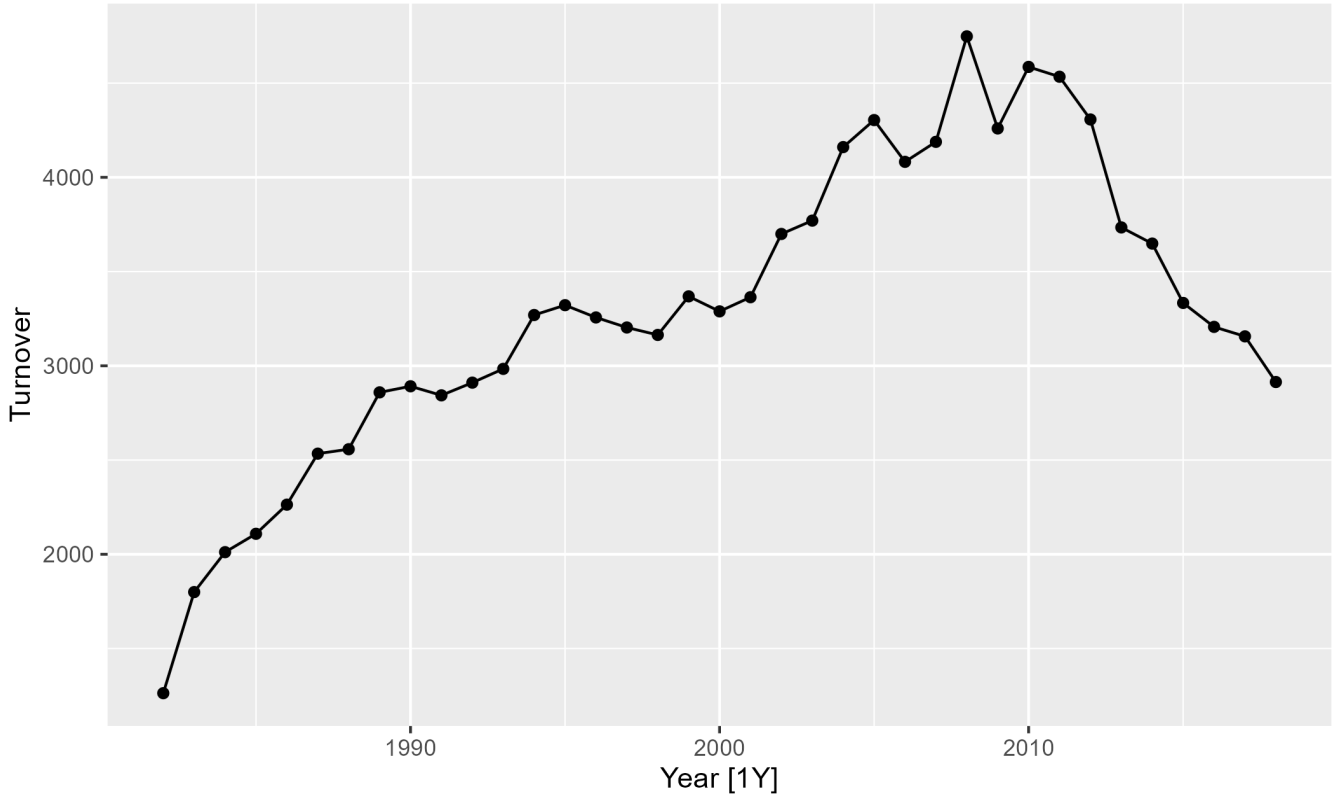
$$x_t = \frac{y_t}{z_t} * z_{2023}$$

Example of inflation adjustment

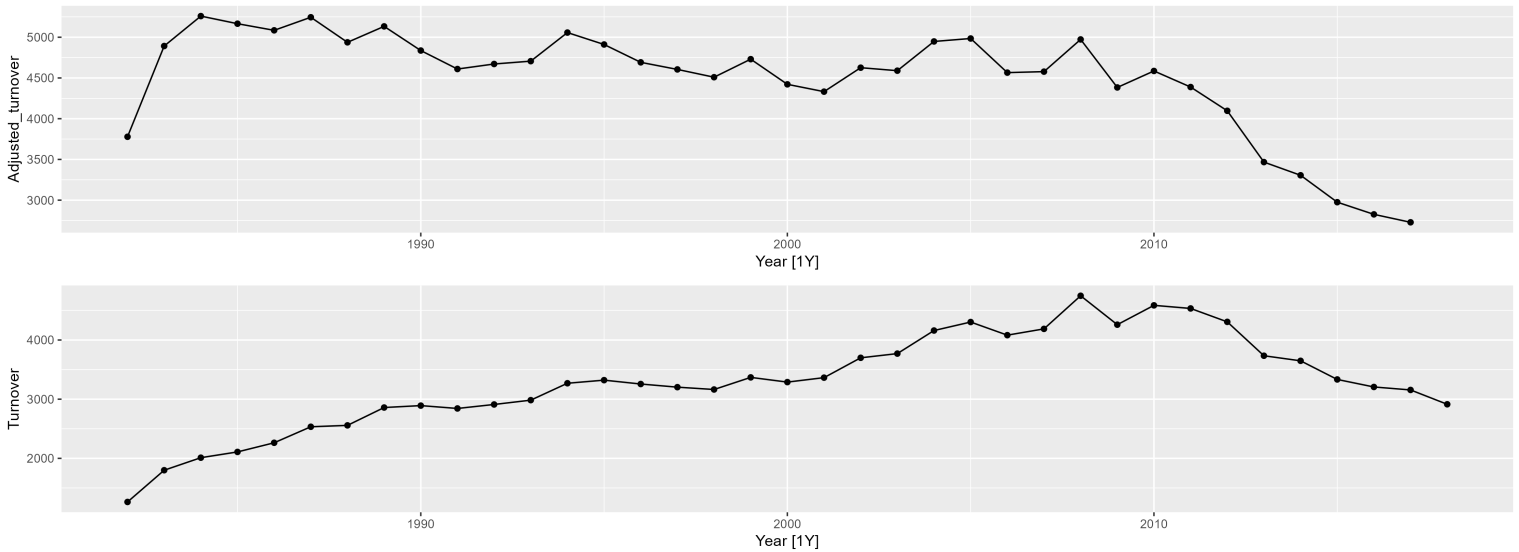
- The *global_economy* time series contains the CPI for each country and year.
- We can use it to see Australia's newspaper and book industry retail over the years.

```
book_retail <- aus_retail |>
  filter(Industry == "Newspaper and book retailing")

book_retail |>
  group_by(Industry) |>
  index_by(Year = year(Month)) |>
  summarise(Turnover = sum(Turnover)) -> book_retail_year
book_retail_year |> autoplot(Turnover) + geom_point()

aus_economy <- global_economy |>
  filter(Country == "Australia")
aus_economy |> autoplot(CPI) + geom_point()

book_retail_CPI <- book_retail_year |>
  left_join(aus_economy, by = "Year")

book_retail_CPI <- book_retail_CPI |>
  mutate(Adjusted_turnover = Turnover / CPI * 100) |>
  select(Year, Industry, Turnover, Adjusted_turnover)

p1 <- book_retail_CPI |> autoplot(Adjusted_turnover) + geom_point()
p2 <- book_retail_CPI |> autoplot(Turnover) + geom_point()
plot_grid(p1, p2, nrows = 2, ncol = 2)
```

- Turnover for Newspaper and book retailing industry



- Comparison between turnover and CPI adjusted turnover for Newspaper and book retailing industry



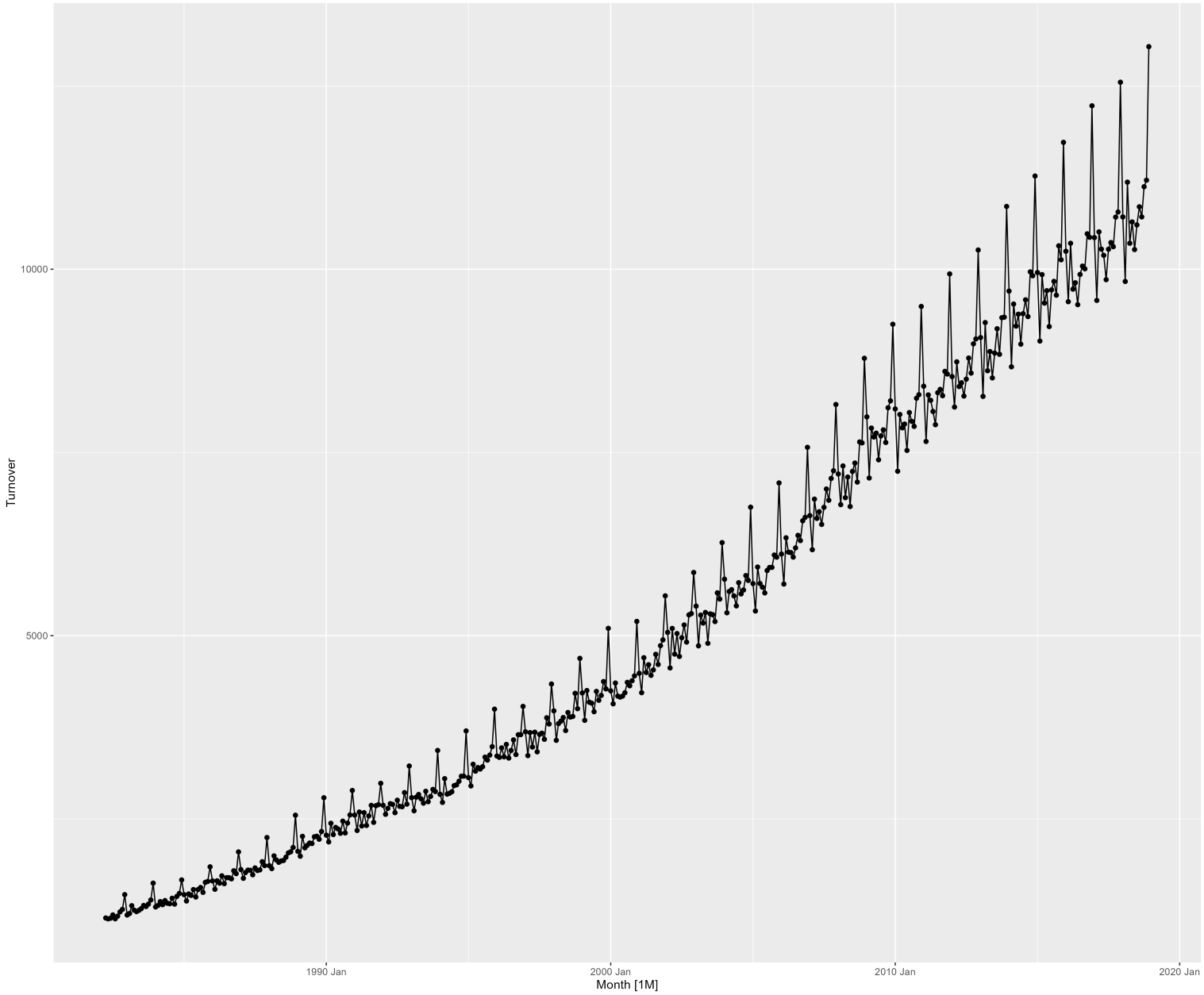
Mathematical transformations

- These are transformations to remove the change (increase and decrease) in the variation of the data.
- The most frequently used mathematical transformations are:
  - Square root
  - Cube root
  - Take the logarithm
- The above order is the order of strength for these transformations.
- Logarithms are the most interpretable: a change in the log value is percentage change in the original data.

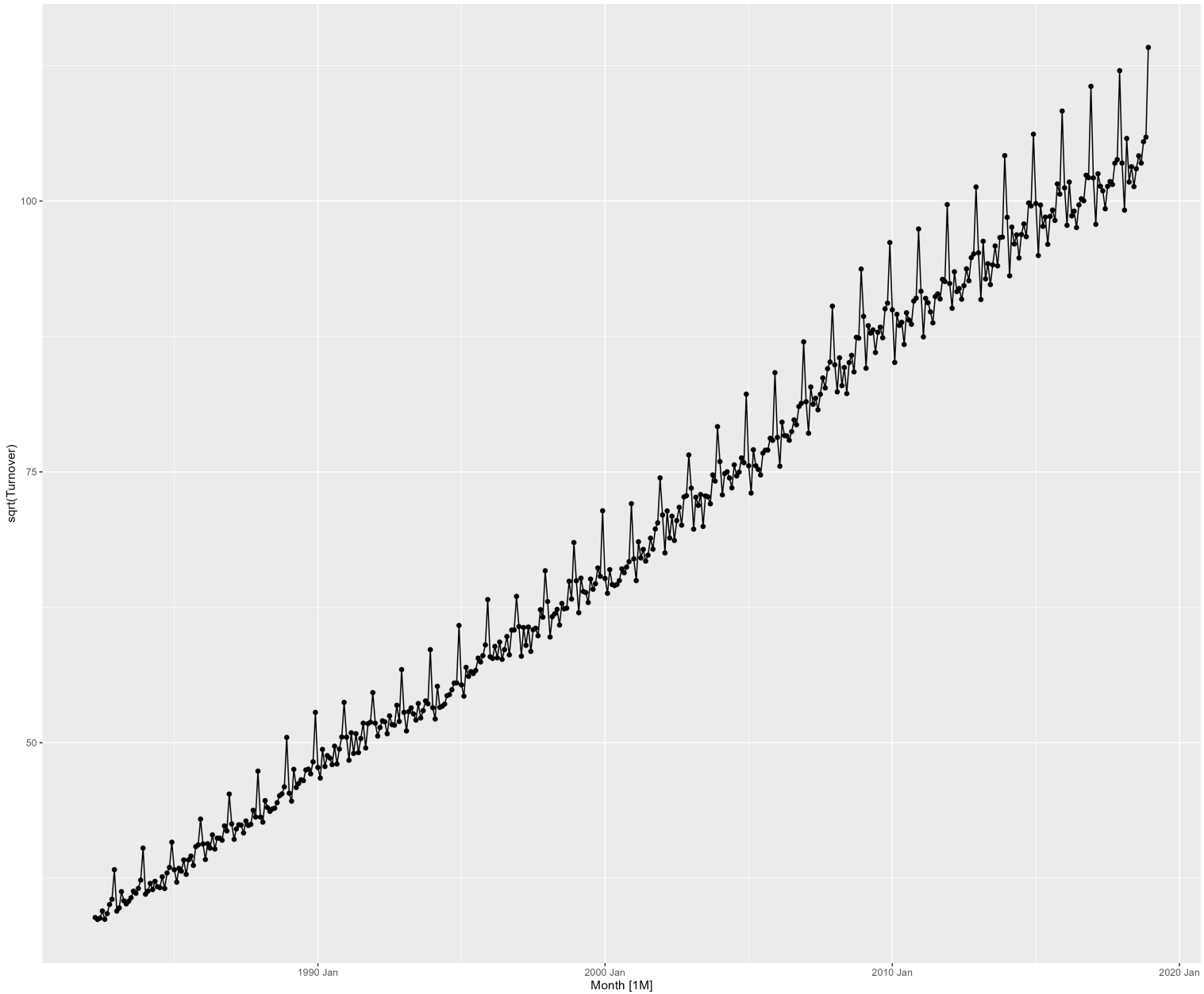
Example

```
food_retail <- aus_retail |>
  filter(Industry == "Food retailing") |>
  summarize(Turnover = sum(Turnover))

food_retail |> autoplot() + geom_point()
food_retail |> autoplot(sqrt(Turnover)) + geom_point()
food_retail |> autoplot(Turnover^(1/3)) + geom_point()
food_retail |> autoplot(log(Turnover)) + geom_point()
food_retail |> autoplot(-1 / Turnover) + geom_point()
```

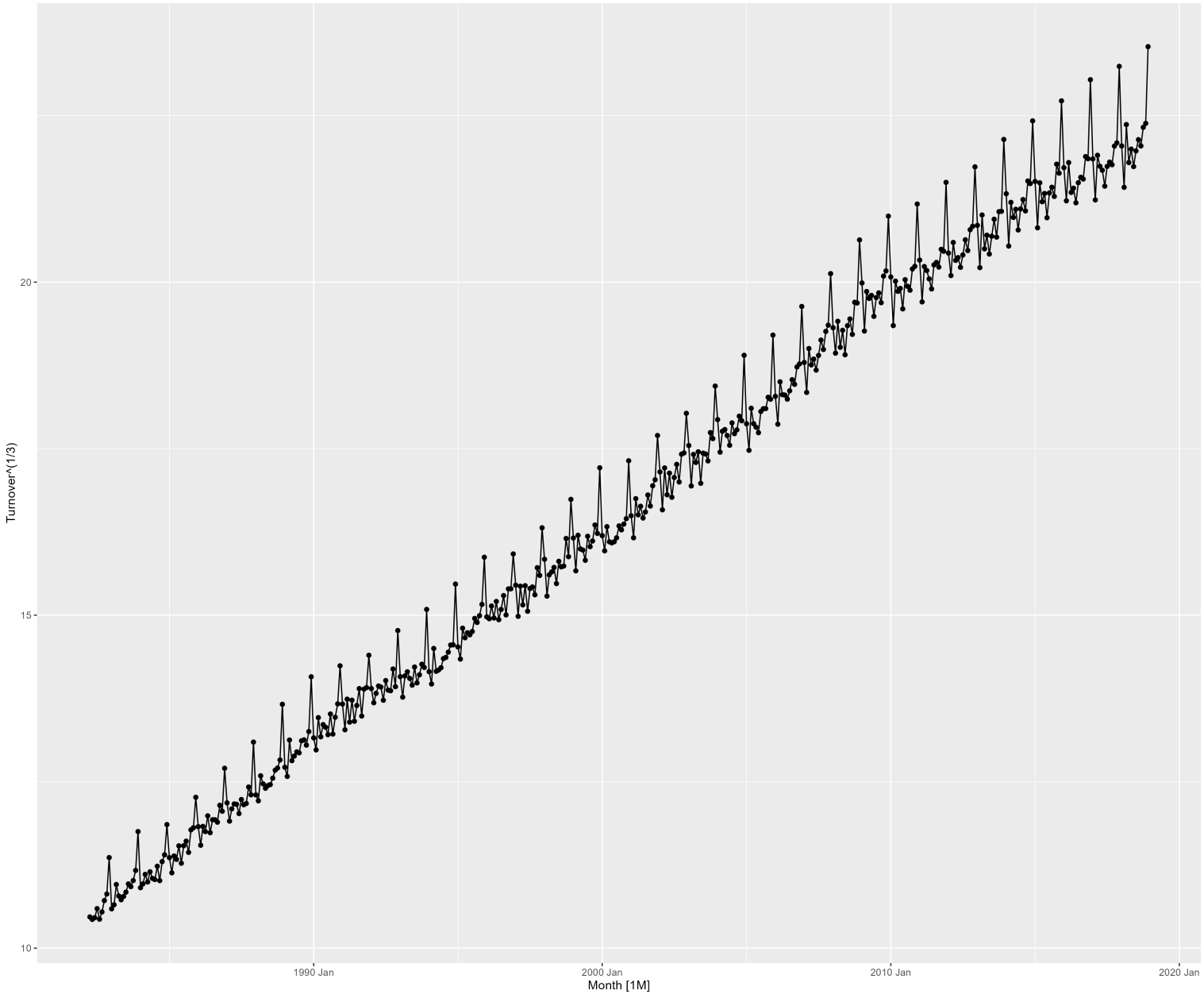
- Turnover for food retail



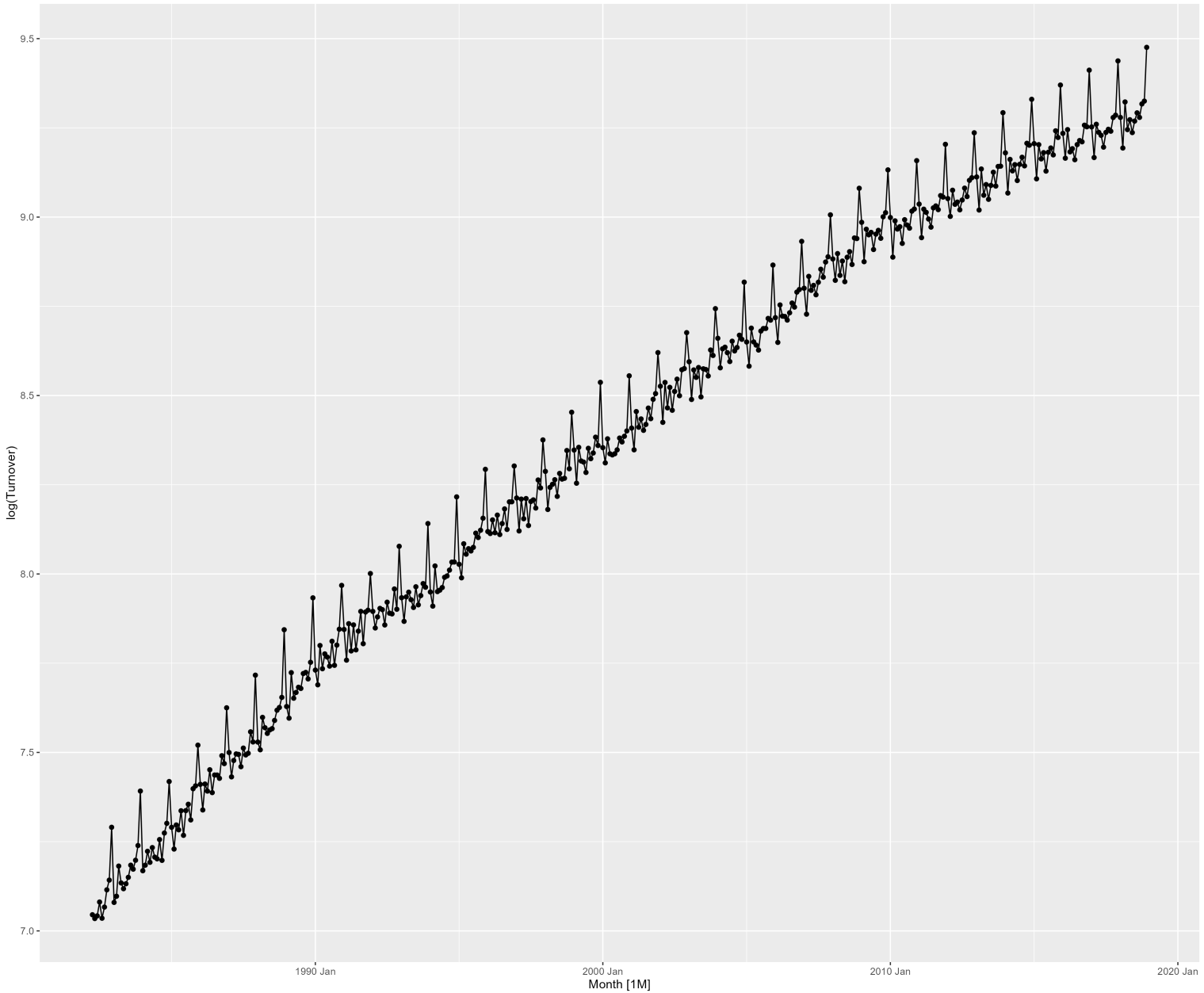
- Square root of turnover for food retail



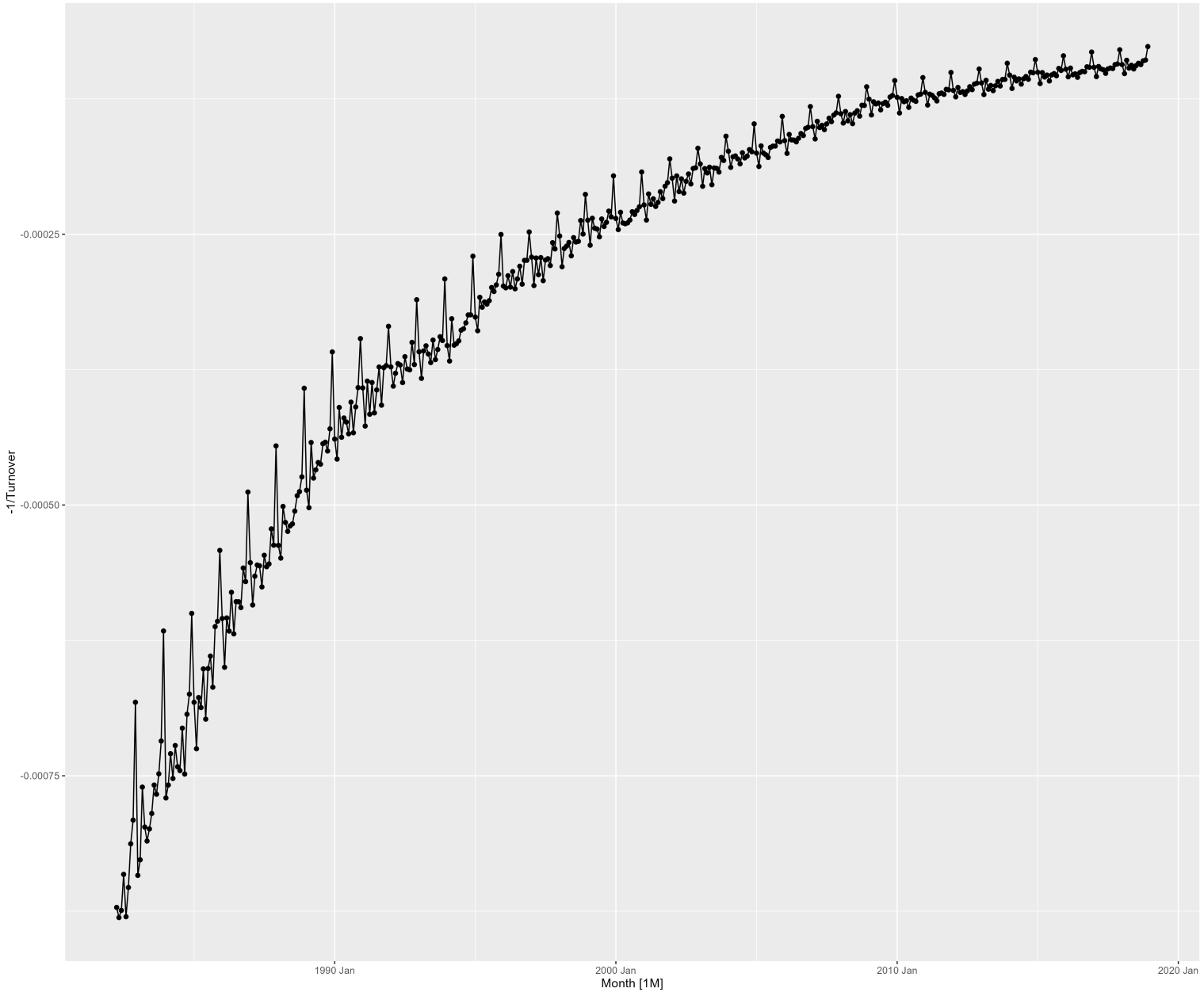
- Cube root of turnover for food retail



- Logarithm of turnover for food retail



- Inverse of turnover for food retail



Box-Cox transformations I

- Notation: the original time series is denoted by $y_1 .... y_t$ and that the transformed time series will be denoted by $w_1 .... w_t$.
- It is a family of transformations, which includes both the logarithmic and the power transformations, through a parameter λ.

$$w_t = \begin{cases} log(y_t), & \text{if } \lambda = 0 \\ \frac{sign(y_t) * |y_t|^{\lambda} - 1}{\lambda} & \text{otherwise} \end{cases}$$

- Actually, this is a modified version of the Box-Cox transformation, which allows for negative values in y_t (assuming that λ is not 0).

Box-Cox transformations II

- If $\lambda = 0$, we have the logarithmic transformation (natural logarithm).
- If $\lambda = 0.5$, we have the square root transformation.
- If $\lambda = 1$, $w_t = y_t - 1$ (lower values, but exactly the same shape for the series). For other values, the shape of the series will change.
- If $\lambda = -1$, we have the reciprocal transformation and for -0.5 we have the reciprocal square root transformation.
- In R, we can compute the Box Cox transformation using the *box_cox* function, which takes as parameter the data and the value of λ.

```
food_retail |>
    autoplot(box_cox(Turnover, 0.8)) + geom_point()
```

Box-Cox transformations III

- The goal is to find a value for λ for which the seasonal variation is the same across the entire time series. This might be complicated, since you need to try different values until you find a good one.
- An alternative is to use the *guerrero* feature, which finds the right value for λ automatically.

```
food_retail |> features(Turnover, guerrero)
```

```
# A tibble: 1 x 1
  lambda_guerrero
            <dbl>
1          0.0895
```

Time series decomposition

- We have discussed that in case of a time series we can identify a *trend*, *season* and *cycle*. When we talk about decomposing a time series, we consider the cycle and trend together, in a component called *trend-cycle* (or simply just trend).

- Consequently, a time series y_t can be considered as being made of three components:
  - trend (or trend-cycle) - T,
  - seasonality - S
  - remainder - R.

- We can assume an additive decomposition and consider that:

$$y_t = S_t + T_t + R_t$$

where y are the data, S is the seasonal component, T is the trend component, and R is the remainder, all in period t.

- Alternatively, we can assume a multiplicative decomposition, and consider that:

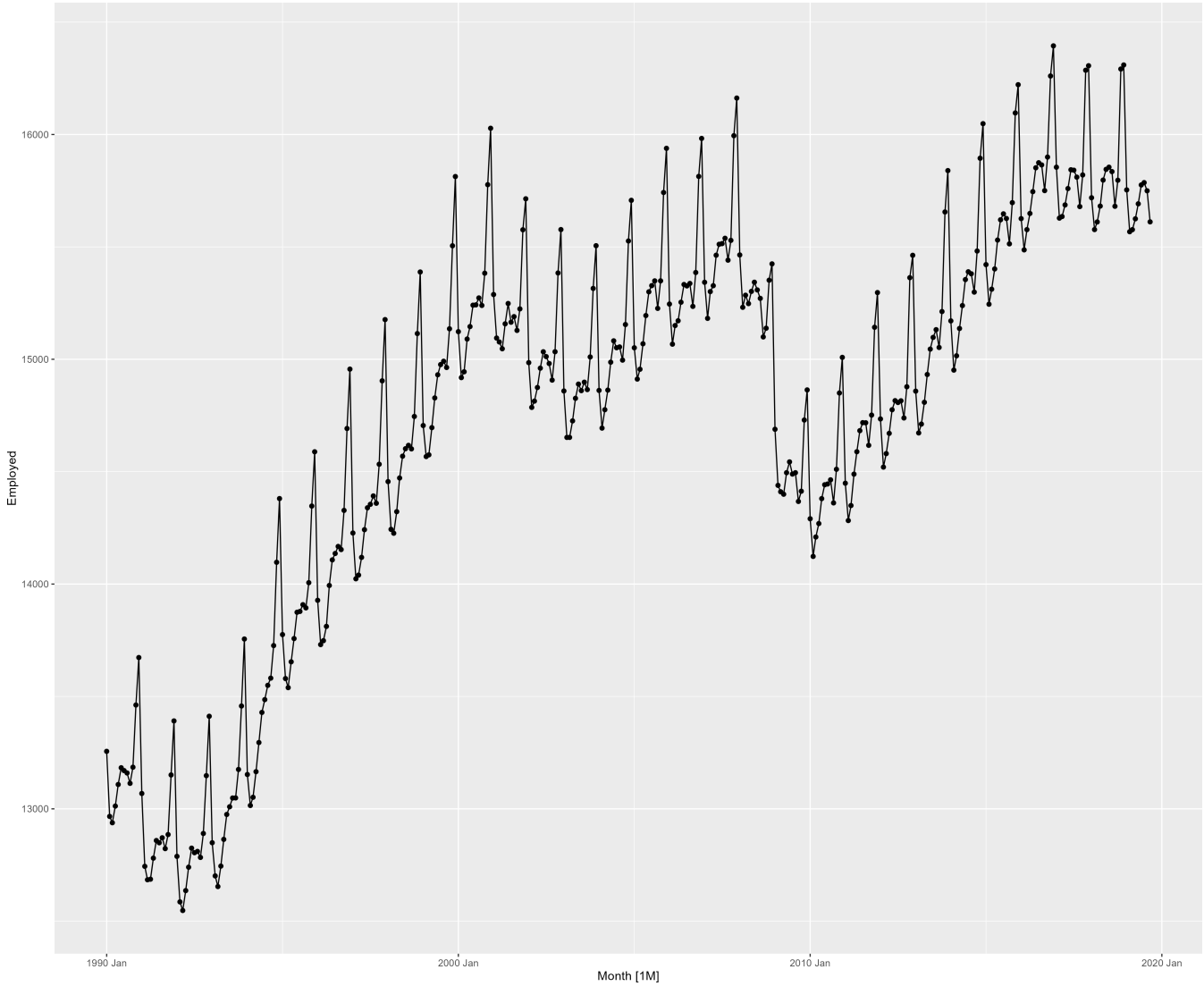
$$y_t = S_t * T_t * R_t$$

- Multiplicative decomposition is a better alternative when the seasonal or cyclic fluctuations vary with the level of the time series (common in case of economic series). Alternatively, we can use a Box-Cox transformation on it, and use an additive decomposition.
- When there is no such variation, additive decomposition is appropriate.

Examples

- Let us look at a first example of US retail trade unemployment rates.

```
us_retail_employment <- us_employment |>
  filter(Year(Month) >= 1990, Title = "Retail Trade") |>
  select(-Series_ID)

us_retail_employment
us_retail_employment |> autoplot() + geom_point()
```


16000
15000
14000
13000
Employed
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month [1M]

Decomposition example

- We will talk about how to the decomposition is done, but let's see first some examples of how a decomposition looks like.
- One function to decompose a time series is the *STL* function. It can have many parameters to customize the decomposition, but currently, we will only focus on the default values (so no parameters passed).
- On the other hand, STL can only do additive decomposition.
- Let's see how we can decompose the US retail unemployment data.

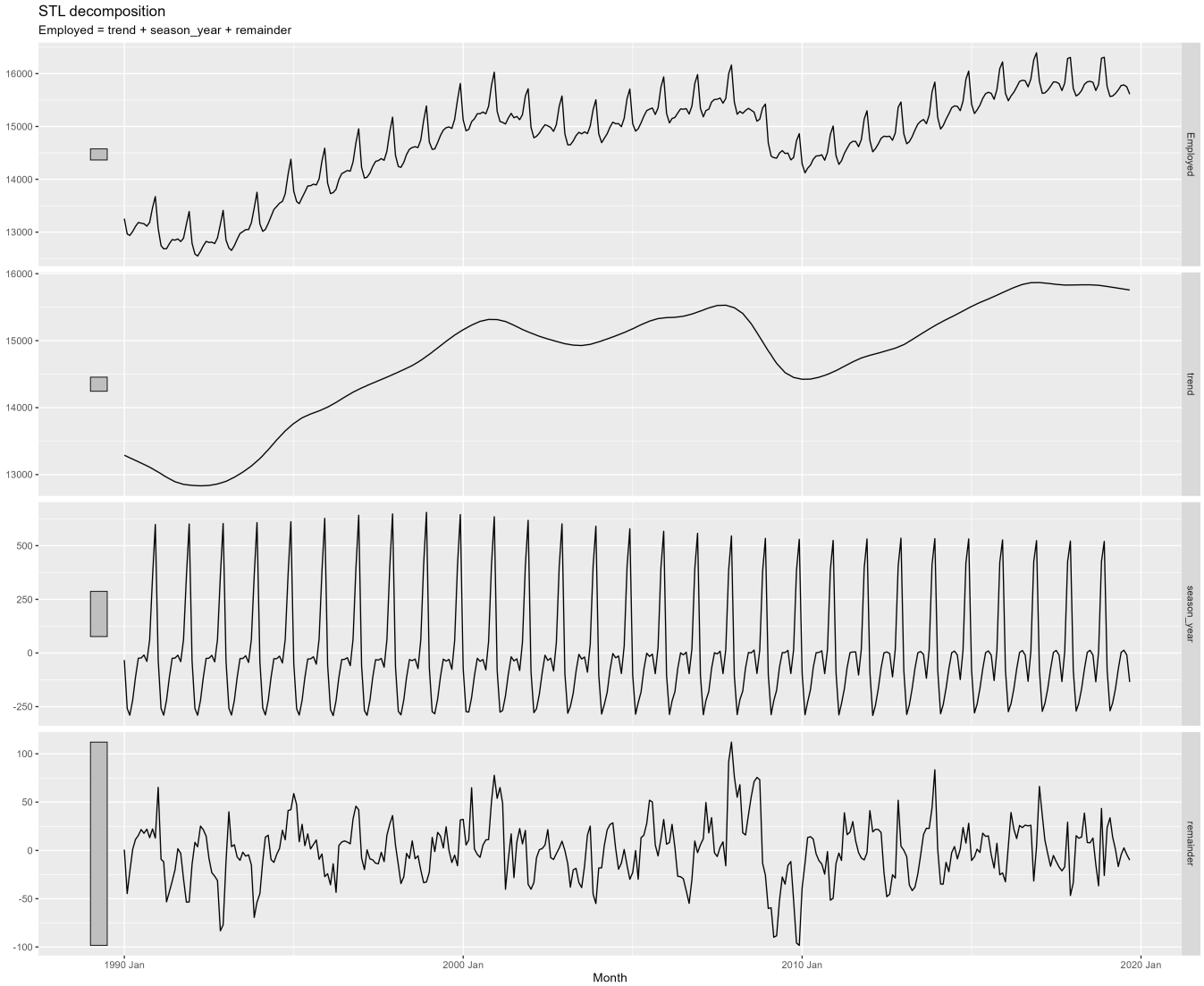
```
dcmp <- us_retail_employment |>
  model(stl = STL(Employed))
dcmp
components(dcmp)
```

- We can see that the result of the decomposition is a *mable*, a *model table*.
- Calling the *components* function on *dcmp* we will get a *dabble*, a *decomposition table*, which contains both the initial data and the three components in which it was separated. The third comment line shows that indeed an additive decomposition was performed.
- Column *Month* is the timestamp of the observation.
- Column *Employed* contains the initial values for the Employed attribute for each observation.
- Column *trend* shows the trend component from the decomposition for each observation.
- Column *season_year* shows the seasonal component from the decomposition for each observation.
- Column *remainder* shows the remainder component from the decomposition for each observation.

- Components of the decomposition

```
> components(dcmp) |> print(n = 15)
# A dable: 357 x 7 [1M]
# Key:     .model [1]
# :        Employed = trend + season_year + remainder
   .model    Month Employed  trend season_year remainder season_adjust
   <chr>     <mth>    <dbl>  <dbl>       <dbl>     <dbl>          <dbl>
 1 stl    1990 Jan   13256. 13288.      -33.0      0.836         13289.
 2 stl    1990 Feb   12966. 13269.     -258.      -44.6           13224.
 3 stl    1990 Mar   12938. 13250.     -290.      -22.1           13228.
 4 stl    1990 Apr   13012. 13231.     -220.       1.05           13232.
 5 stl    1990 May   13108. 13211.     -114.      11.3            13223.
 6 stl    1990 Jun   13183. 13192.      -24.3     15.5            13207.
 7 stl    1990 Jul   13170. 13172.      -23.2     21.6            13193.
 8 stl    1990 Aug   13160. 13151.       -9.52    17.8            13169.
 9 stl    1990 Sep   13113. 13131.      -39.5     22.0            13153.
10 stl    1990 Oct   13185. 13110.       61.6     13.2            13124.
11 stl    1990 Nov   13462. 13086.      354.      22.2            13108.
12 stl    1990 Dec   13673. 13062.      599.      12.8            13074.
13 stl    1991 Jan   13068. 13037.      -34.2     65.3            13103.
14 stl    1991 Feb   12744. 13012.     -258.      -9.15           13003.
15 stl    1991 Mar   12684. 12986.     -290.     -11.7            12974.
# i 342 more rows
```

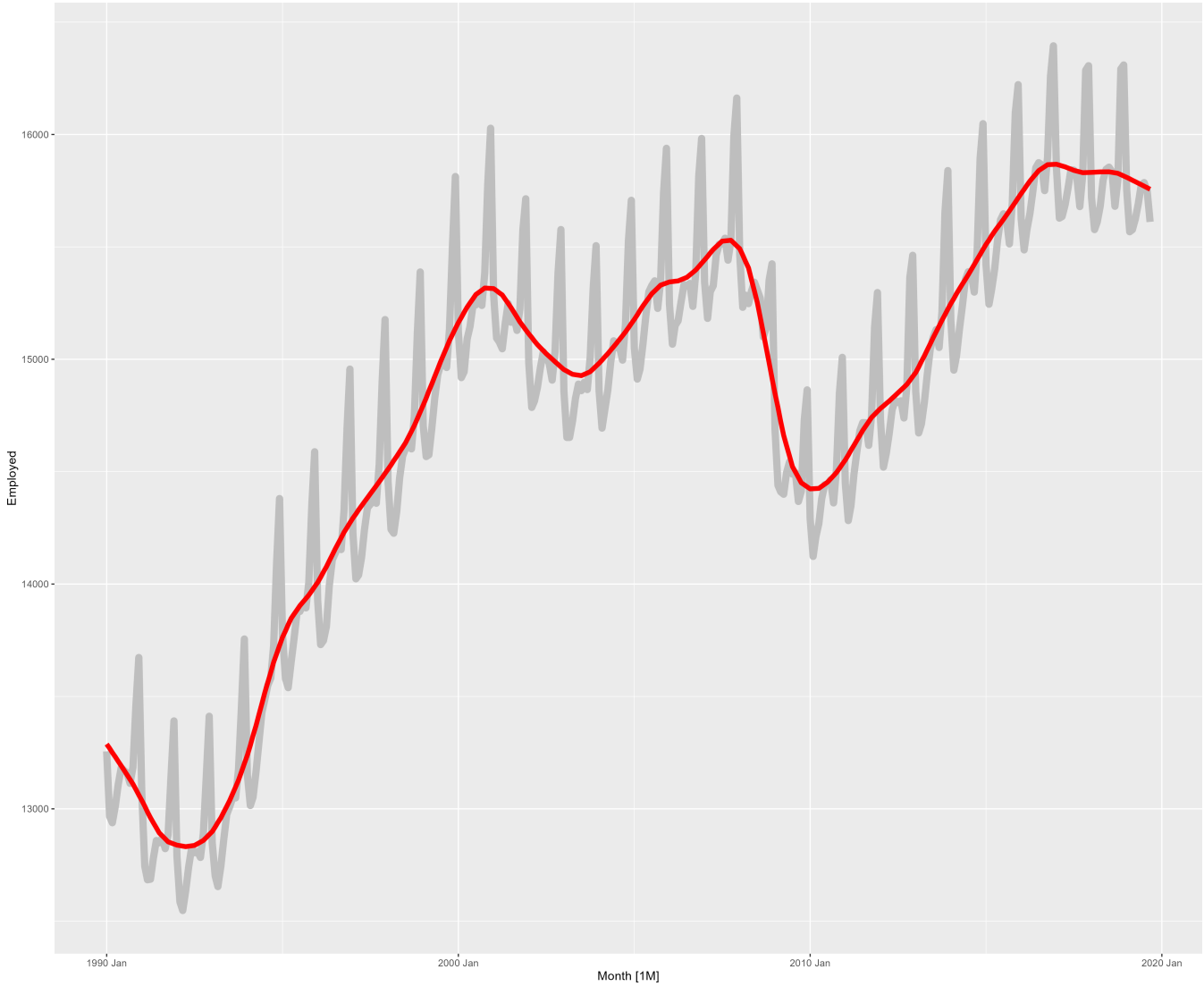
- We can use *autoplot* to plot the components of the decomposition

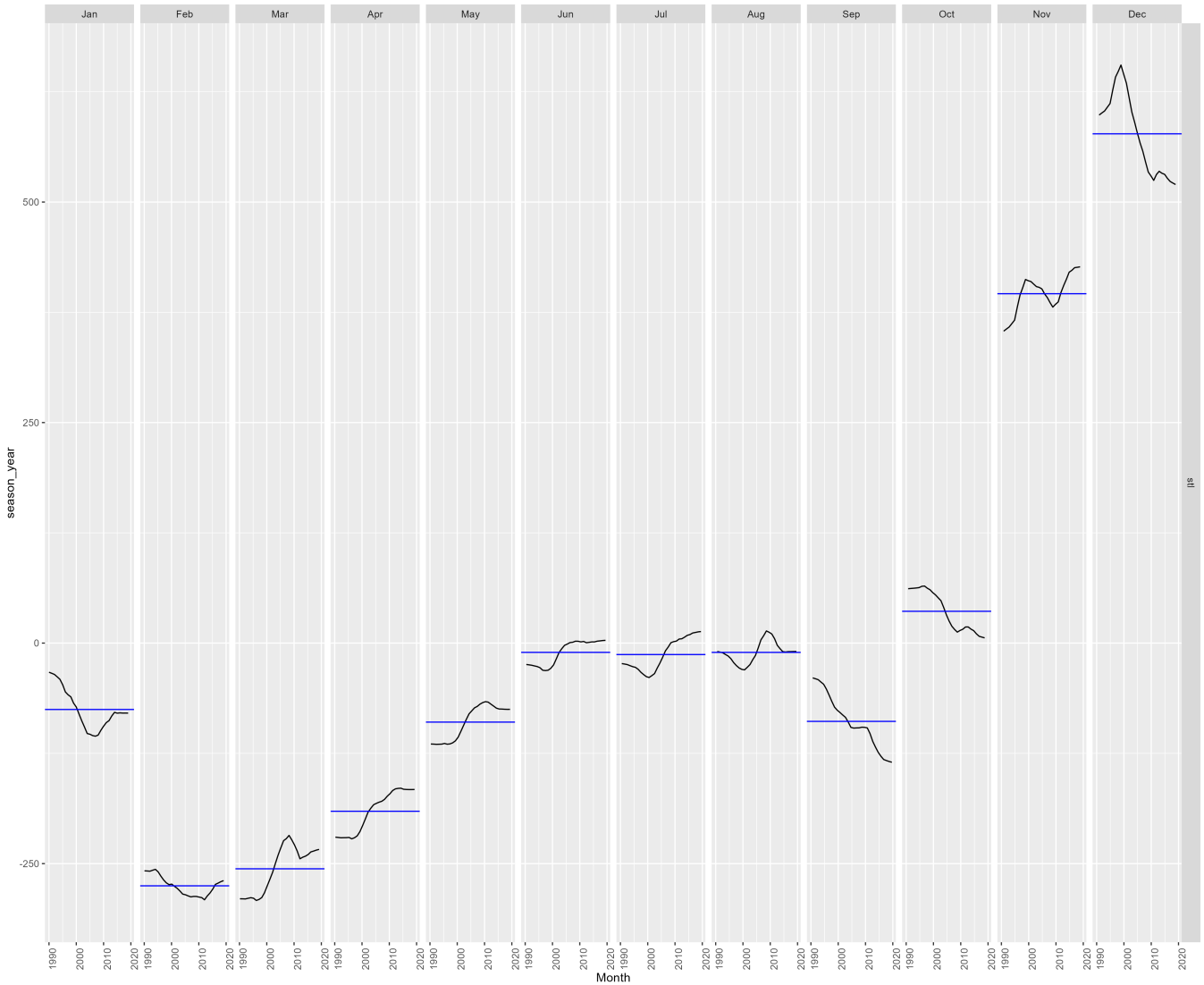


- The first subplot contains the original data.
- The second subplot contains the trend component, which is a smoother line, following the general direction of the time series.
- The third subplot is the seasonal component which looks very regular.
- The last subplot is the remainder (what remains when the subtract from the original data the trend and the season).
- All 4 panels have the same size, but the data they contain is on different scales. The remained varies between -100 and 100, while the trend varies between aprox. 12500 and 16000. This variation is denoted by the size of the grey bars on the left.

- We can create different plots using the decomposition data:
  - We can plot the original data and the trend component
  - We can create seasonal subseries plots for the seasonal component to see how the seasons change.

```
us_retail_employment |>
  autoplot(Employed, color = "gray", size = 3) +
  autolayer(components(dcmp), trend, color = "red", size = 2)

components(dcmp) |> gg_subseries(season_year)
```

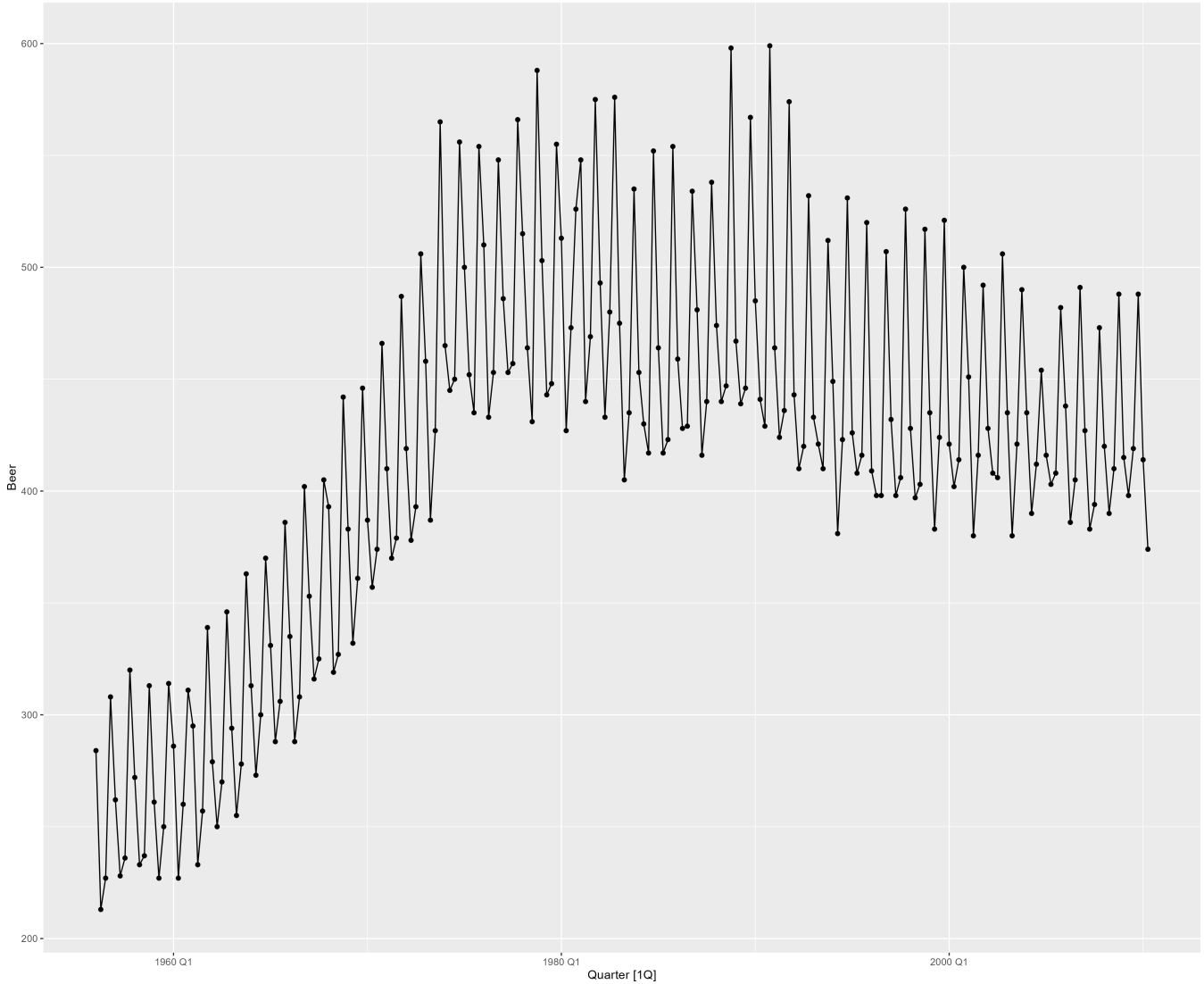

16000
15000
14000
13000
Employed
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month [1M]

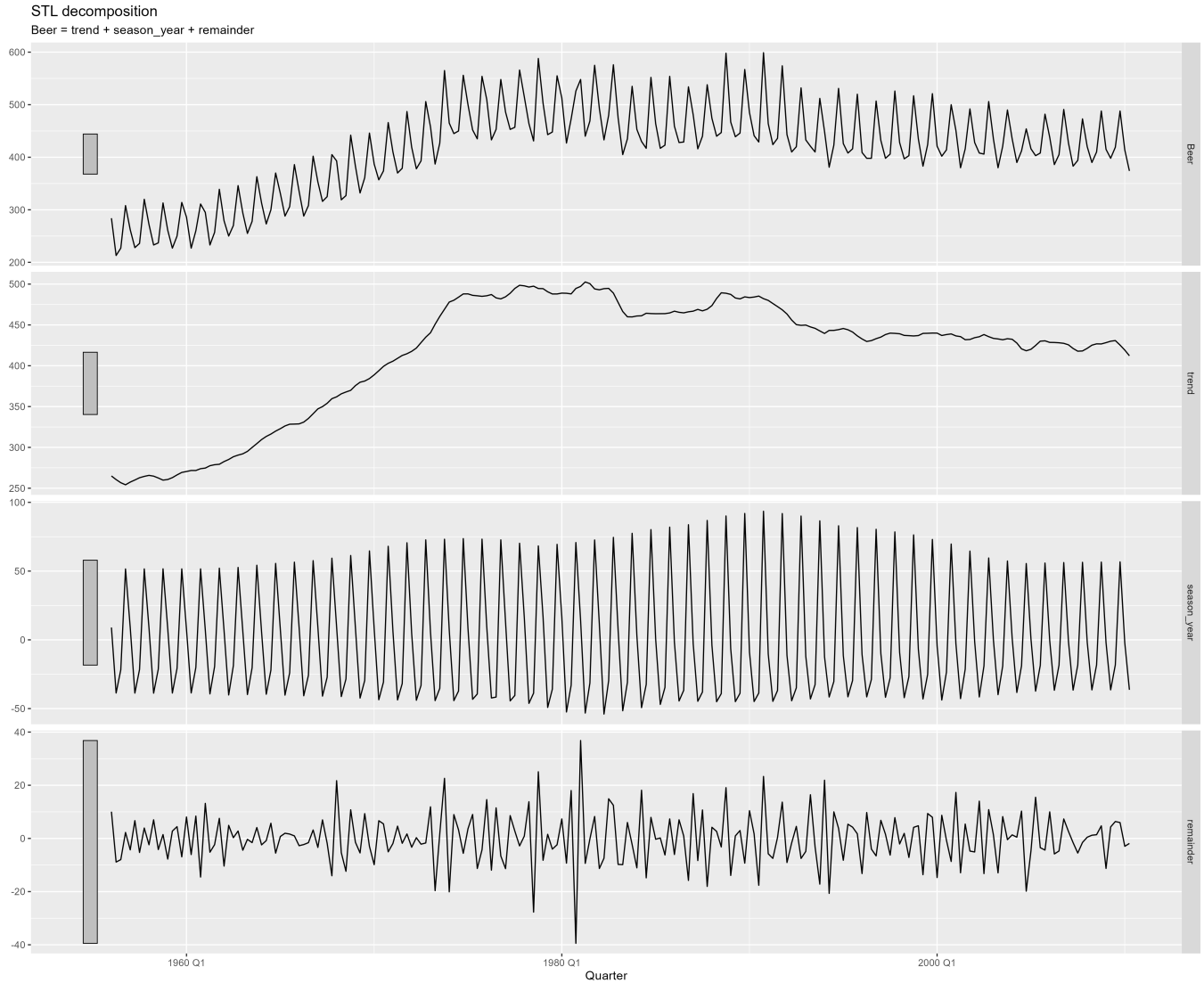

Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
season_year
500
250
0
-250
1990
2000
2010
2020
Month
stl

Other examples

- Quarterly Australian beer production

```
beer <- aus_production |>
  select(Quarter, Beer)

beer |> autoplot() + geom_point()

dcmpBeer <- beer |>
  model(stl = STL(Beer))

components(dcmpBeer)
```

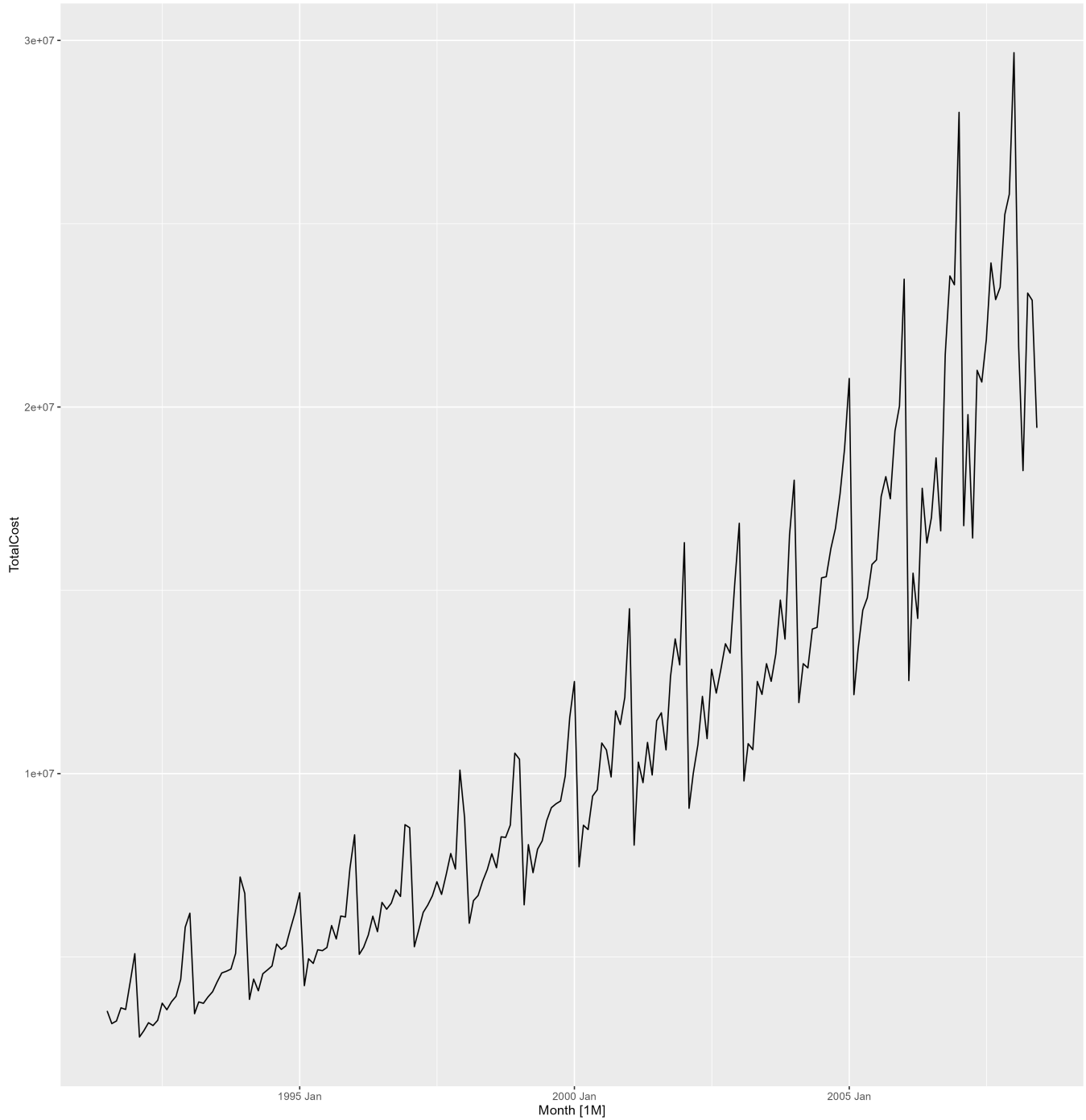

600
500
400
300
200
Beer
1960 Q1
1980 Q1
2000 Q1
Quarter [1Q]

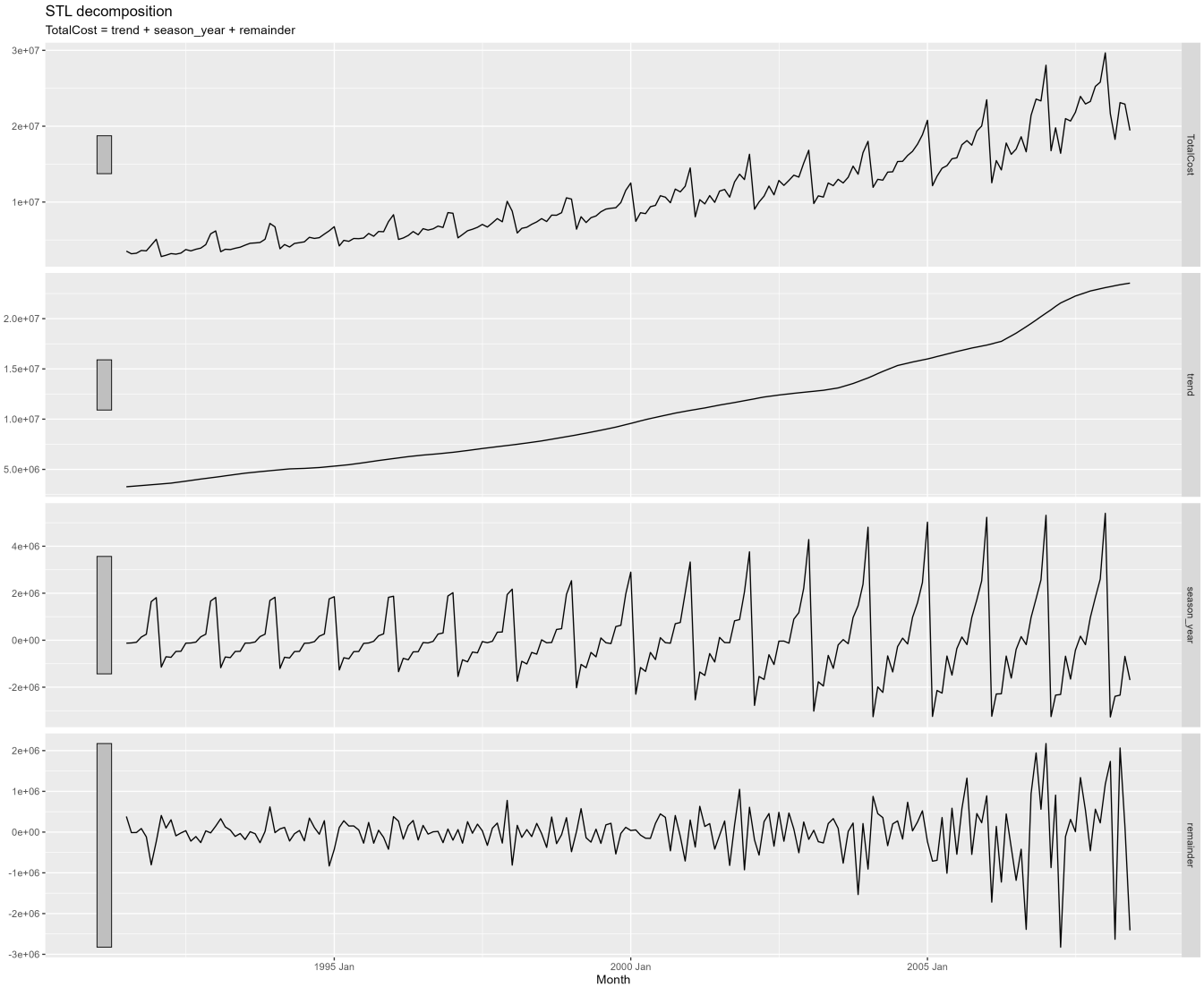

STL decomposition
Beer = trend + season_year + remainder
600
500
400
300
200
Beer
500
450
400
350
300
250
trend
100
50
0
-50
season_year
40
20
0
-20
-40
remainder
1960 Q1
1980 Q1
2000 Q1
Quarter

Other examples

- Antidiabetic drug sales

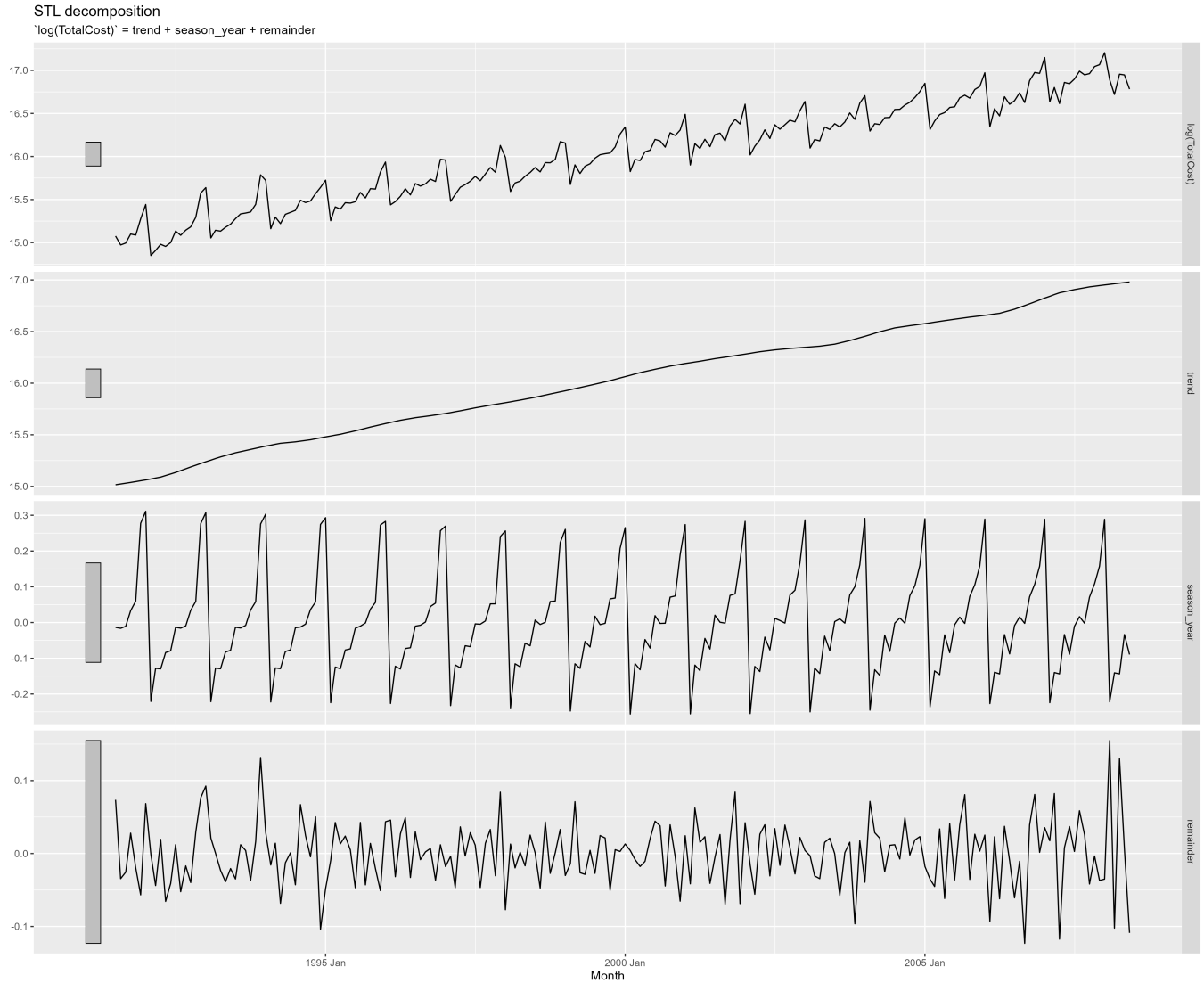
```
A10 |> autoplot()

dcmpA10 <- A10 |> model(stl = STL(TotalCost))
components(dcmpA10)
components(dcmpA10) |> autoplot()
```


3e+07
2e+07
1e+07
TotalCost
1995 Jan
2000 Jan
2005 Jan
Month [1M]


STL decomposition
TotalCost = trend + season_year + remainder
3e+07
2e+07
1e+07
TotalCost
2.0e+07
1.5e+07
1.0e+07
5.0e+06
trend
4e+06
2e+06
0e+00
-2e+06
season_year
2e+06
1e+06
0e+00
-1e+06
-2e+06
-3e+06
remainder
1995 Jan
2000 Jan
2005 Jan
Month

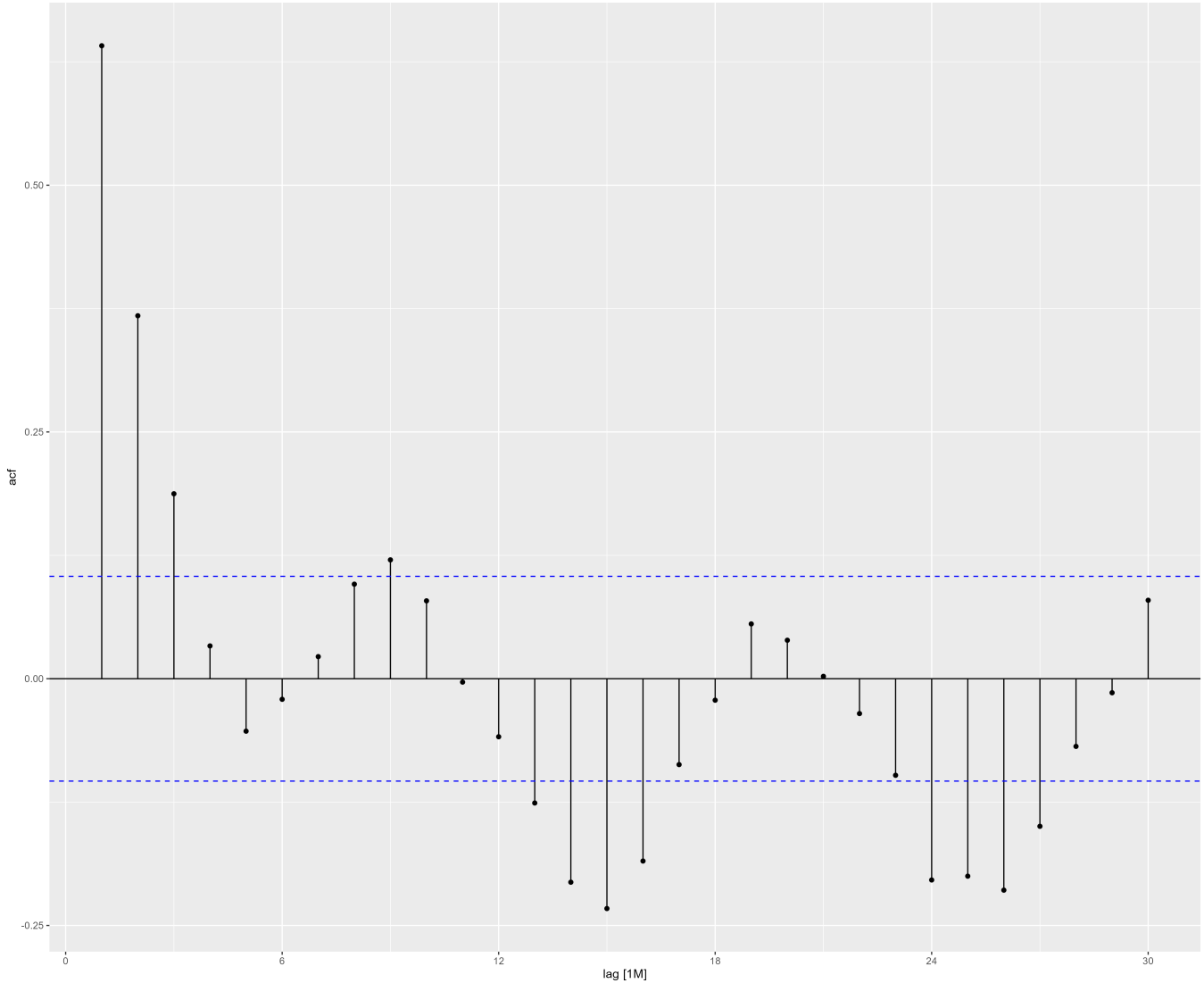
- The antidiabetic drug sales time series was one with multiplicative seasonality.
- We cannot do multiplicative decomposition with STL, and on the seasonal component of the additive decomposition we can see that something is not right.
- We can transform the data applying a log transformation and then do the decomposition.

```
dcmpA10L <- A10 |> model(stl = STL(log(TotalCost)))
components(dcmpA10L)
components(dcmpA10L) |> autoplot()
```


STL decomposition
`log(TotalCost)` = trend + season_year + remainder
log(TotalCost)
trend
season_year
remainder
17.0
16.5
16.0
15.5
15.0
0.3
0.2
0.1
0.0
-0.1
-0.2
1995 Jan
2000 Jan
2005 Jan
Month

- A decomposition is said to be a good one if the remainder is white noise (meaning that all patterns in data were captured by the trend and season components).
- We can check for white noise using the ACF plot.

```
components(dcmp) |>
 ACF(remainder, lag_max = 30) |>
 autoplot() + geom_point()
```

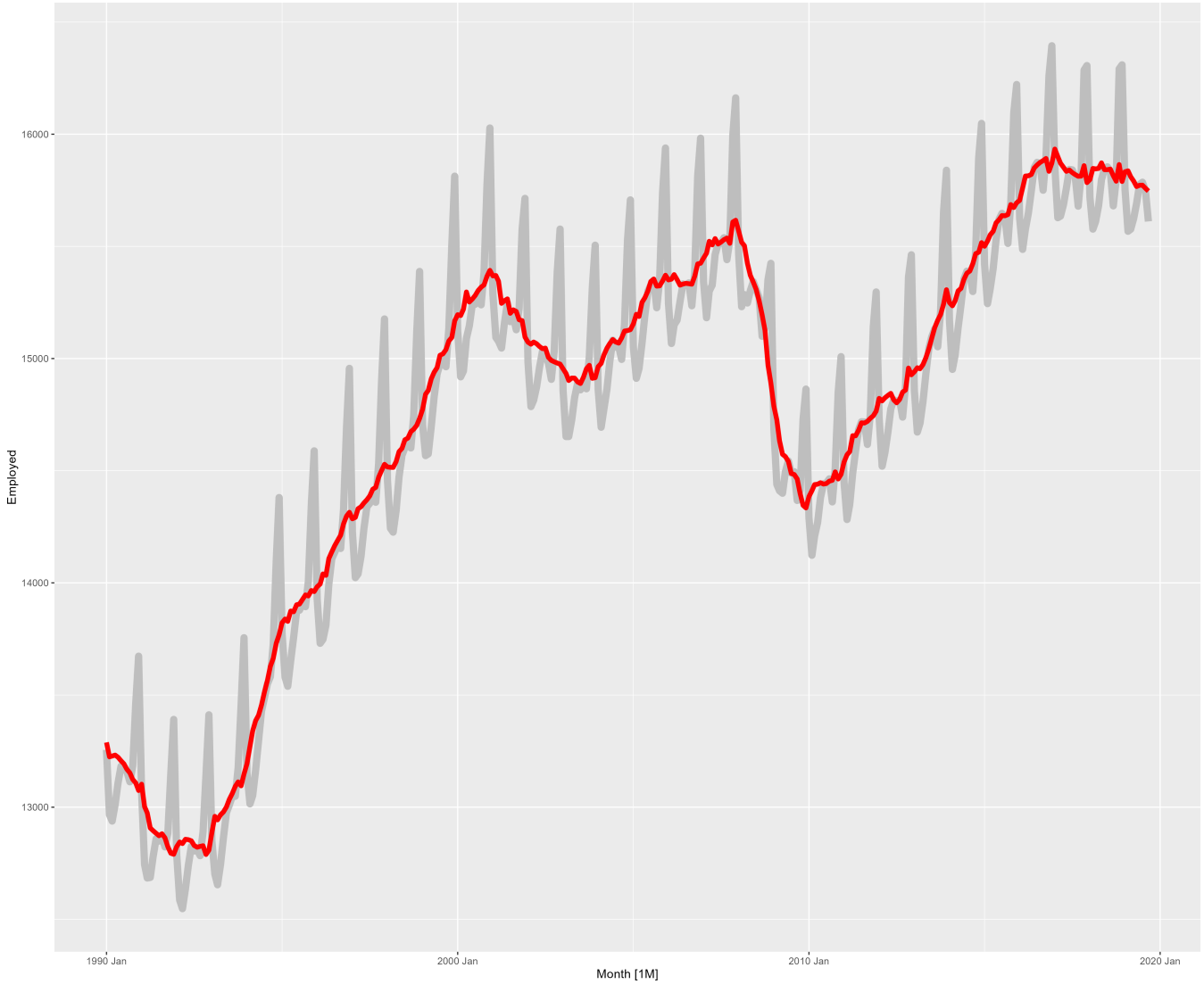


- Definitely not white noise

Seasonally adjusted data

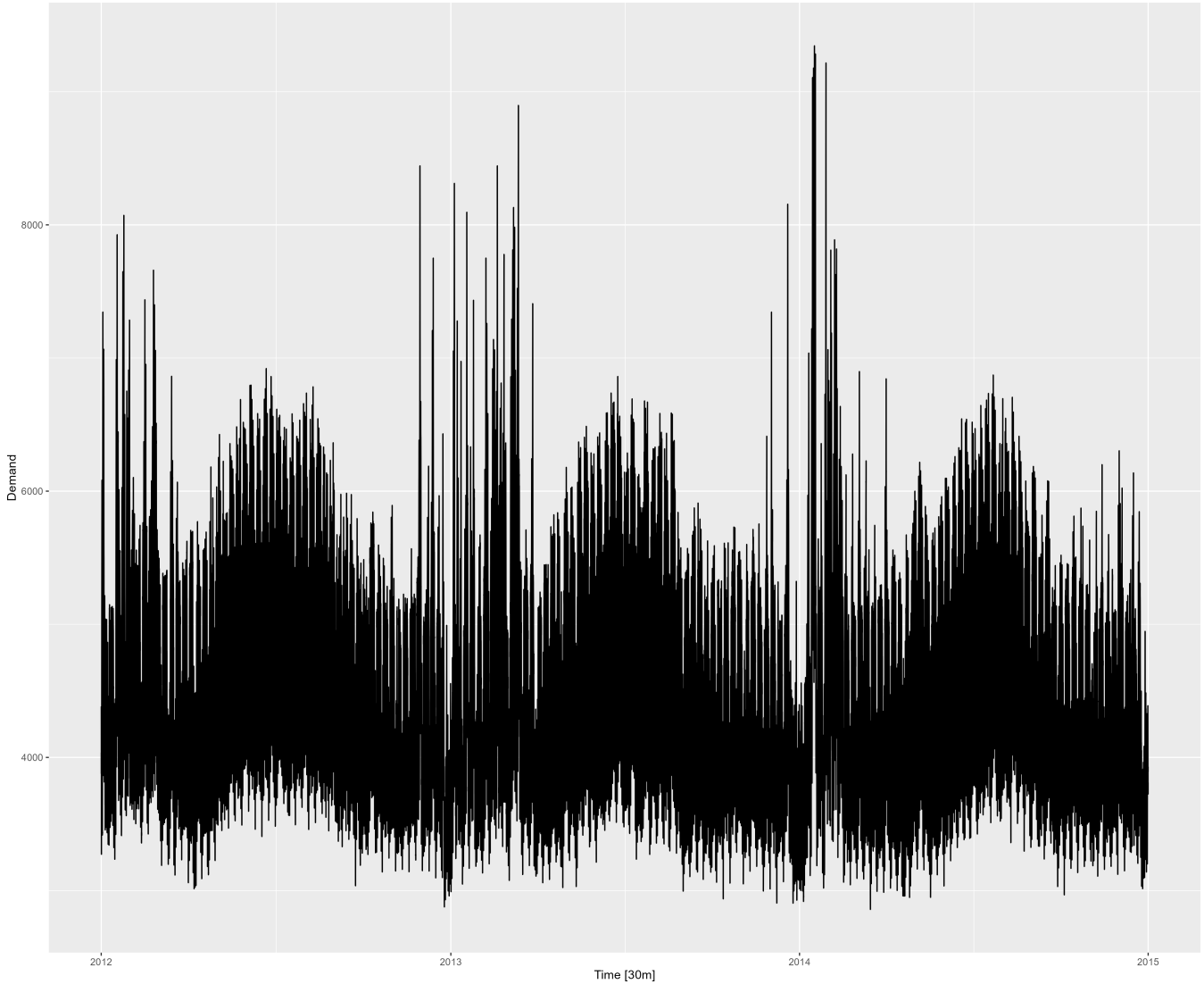
- The last column in the dabble was called *season_adjust* and it contains seasonally adjusted data, i.e. data from which the seasonal component was removed.
- In our example, where an additive decomposition was used, the seasonally adjusted data is just the trend + noise component. In case of a multiplicative decomposition the seasonally adjusted data is the trend * noise.
- Let us plot the original Employment data and the seasonally adjusted component together

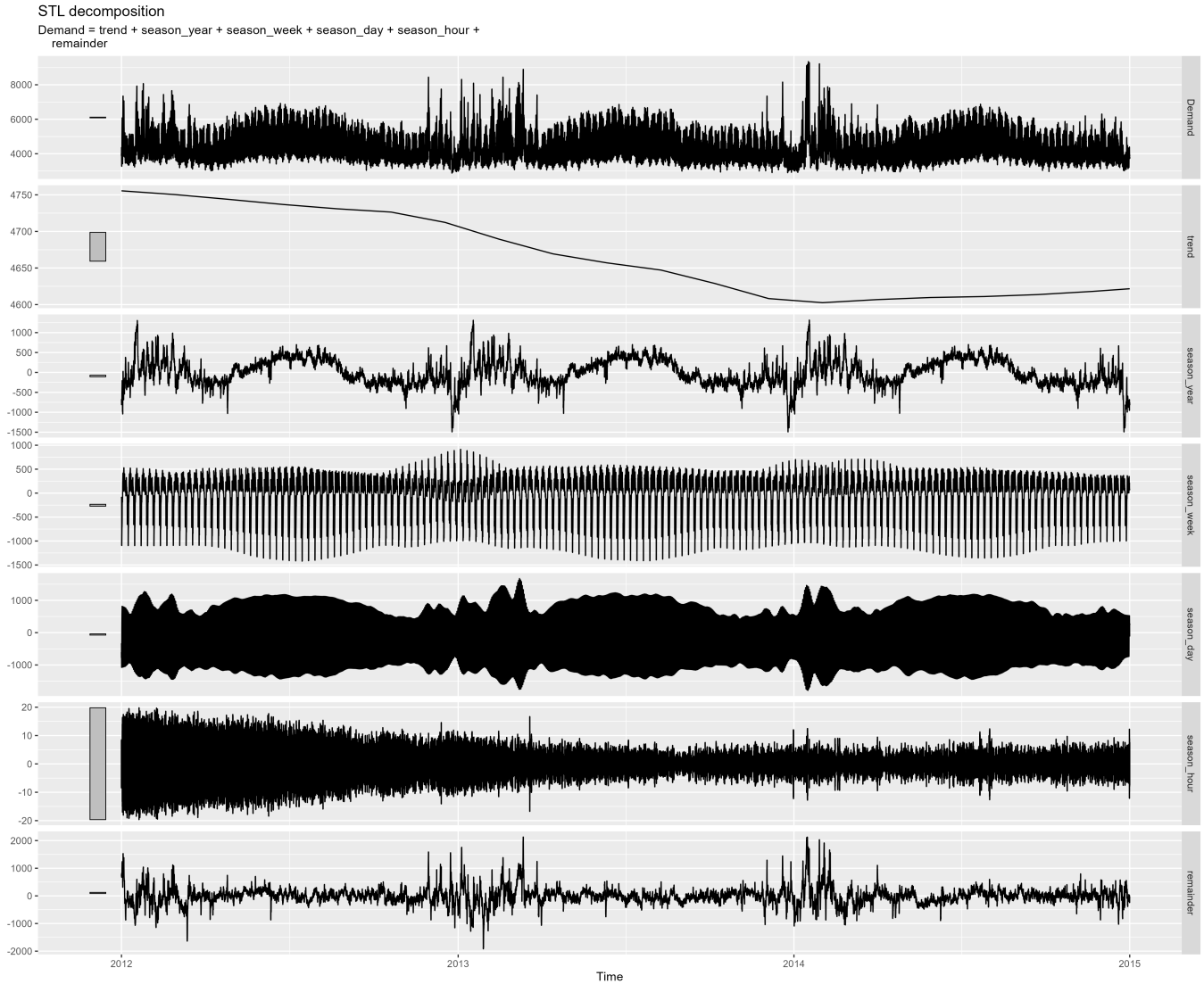
```
us_retail_employment |>
  autoplot(Employed, color = "gray", size = 3) +
  autolayer(components(dcmp), season_adjust, color = "red", size= 2)
```


16000
15000
14000
13000
Employed
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month [1M]

- Let us see another example with STL.
- Consider the *Demand* from the *vic_elec* time series.

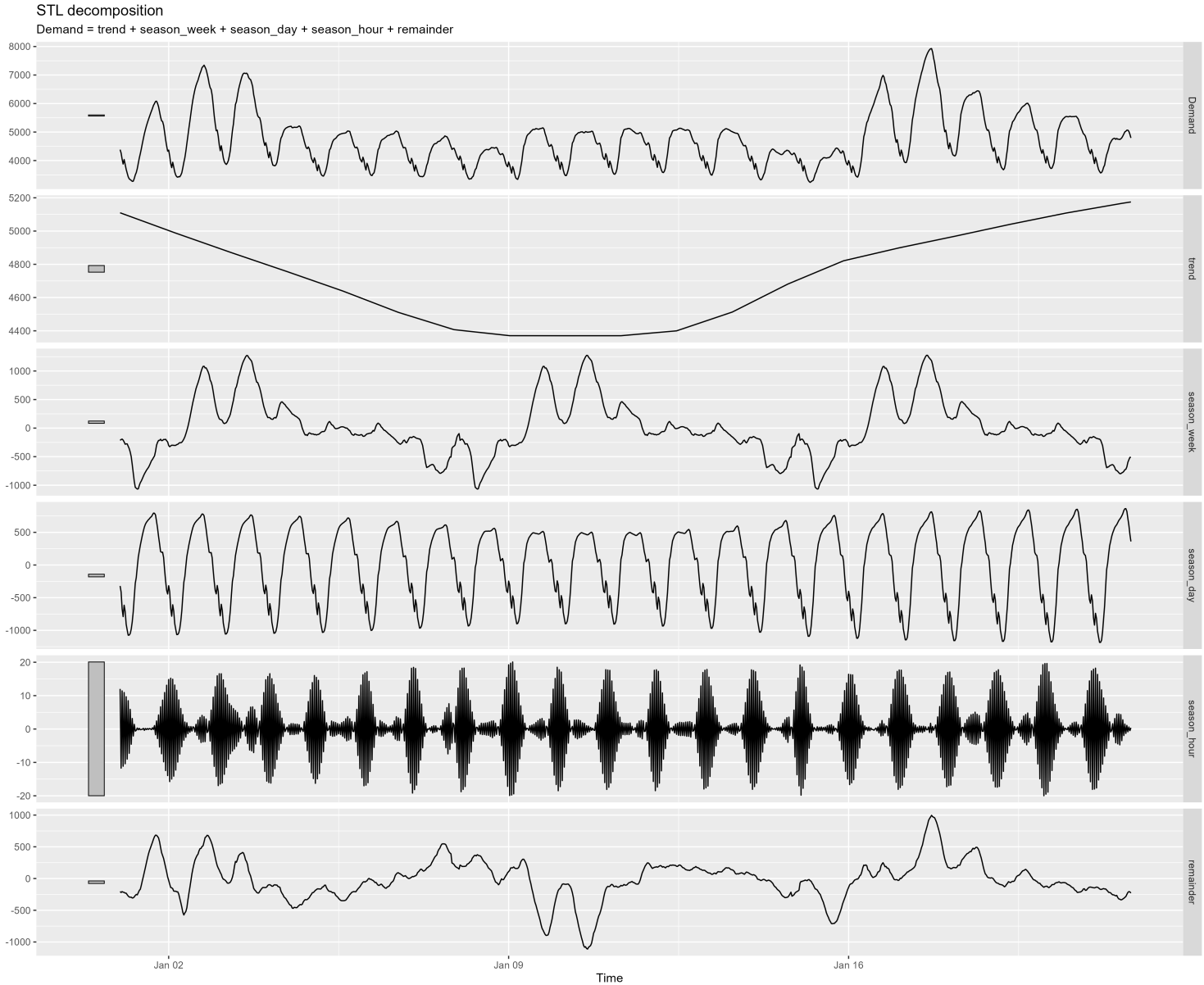
```
vic_elec |> autoplot(Demand)

dcmpVicElec <- vic_elec |>
  model(stl = STL(Demand))

dcmpVicElec |> components()

dcmpVicElec |> components() |> autoplot()
```


8000
6000
4000
Demand
2012
2013
2014
2015
Time [30m]


STL decomposition
Demand = trend + season_year + season_week + season_day + season_hour + remainder
Demand
trend
season_year
season_week
season_day
season_hour
remainder
2012
2013
2014
2015
Time

- We can see that STL identified several seasons in the data: hourly, daily, weekly and yearly.

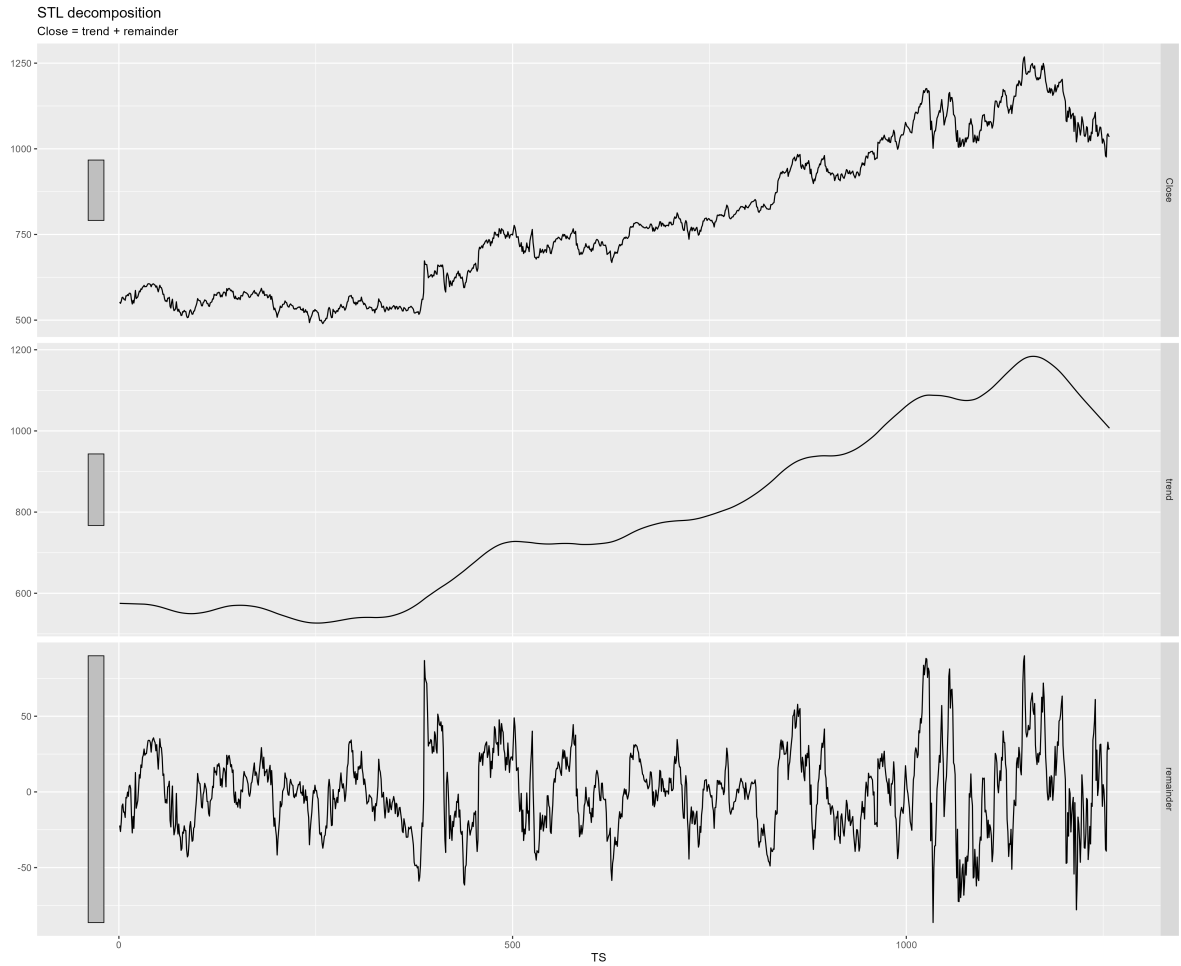
- Since we have so many observations, the seasons are not clearly visible. Let's do the same only for part of the data (first 1000 observations, approximately 3 weeks).


STL decomposition
Demand = trend + season_week + season_day + season_hour + remainder
Demand
trend
season_week
season_day
season_hour
remainder
Jan 02
Jan 09
Jan 16
Time

- What happens if we decompose the Google stock prices time series?

```
google <- gafa_stock |>
  filter(Symbol == "GOOG")
google |> autoplot(Close)

dcmpGoogle <- google |>
  model(stl = STL(Close))

google <- google |>
  mutate(TS = seq(from = 1, to = 1258, by = 1))

google <- tsibble(Close = google$Close, TS = google$TS, index = TS)

dcmpGoogle <- google |> model(stl = STL(Close))
dcmpGoogle |> components()
dcmpGoogle |> components() |> autoplot()
```

- Since the Google data is not a regular one (it considers trading days, not days or weeks), we will have an error.
- To make the time series regular, we will reindex the observations.



- We can see that there is no seasonal component.

Moving averages

- Moving averages are needed in order to estimate the trend-cycle component in the classical decomposition.
- A moving average of order m, denoted by m-**MA** can be computed as:

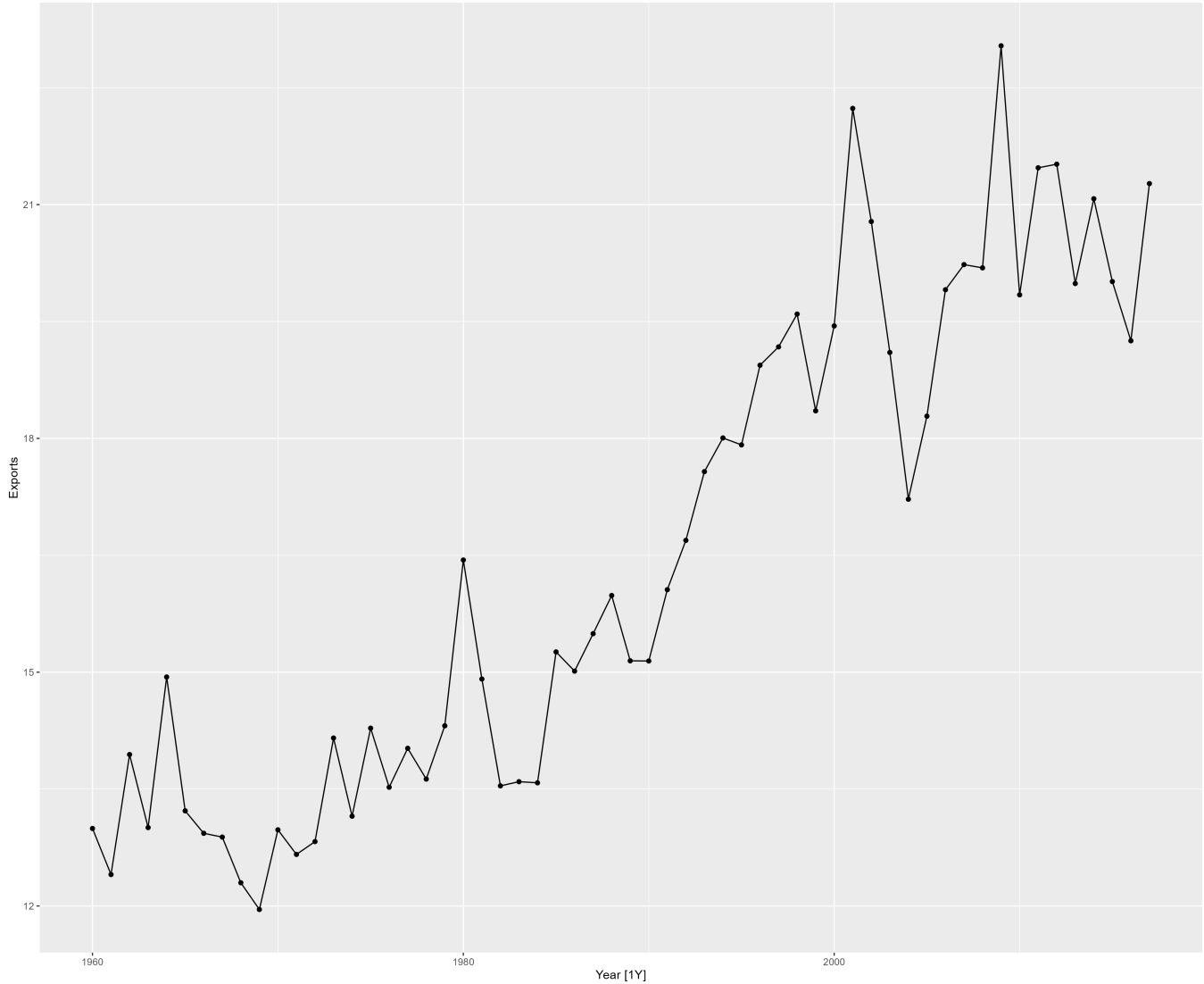
$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}$$

- where $m = 2k + 1$.
- In short, for estimating the moving average at a given time t, we compute the average of that value and the values for the previous and the next k observations. Assuming that observations close in time are close in value as well, the moving average will eliminate some of the randomness (will generate a smoother series).

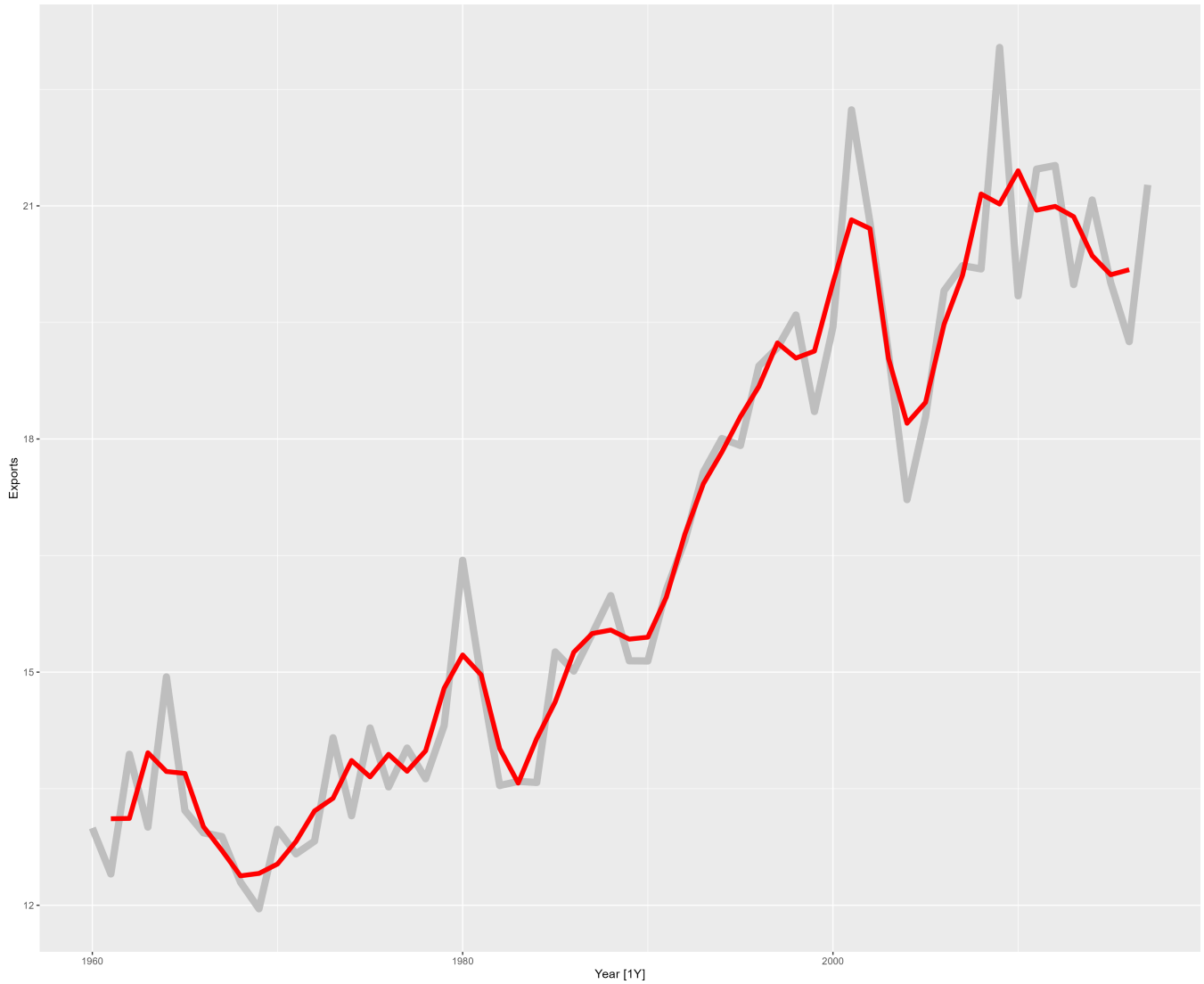
- The moving average can be computed in R with the *slide_dbl()* function, from the *slider* package, where you need to specify the data to work on, what function to apply (in our case it is the *mean* function), how many observations to consider before and after the center one.

- Let us look at the yearly exports for Australia.

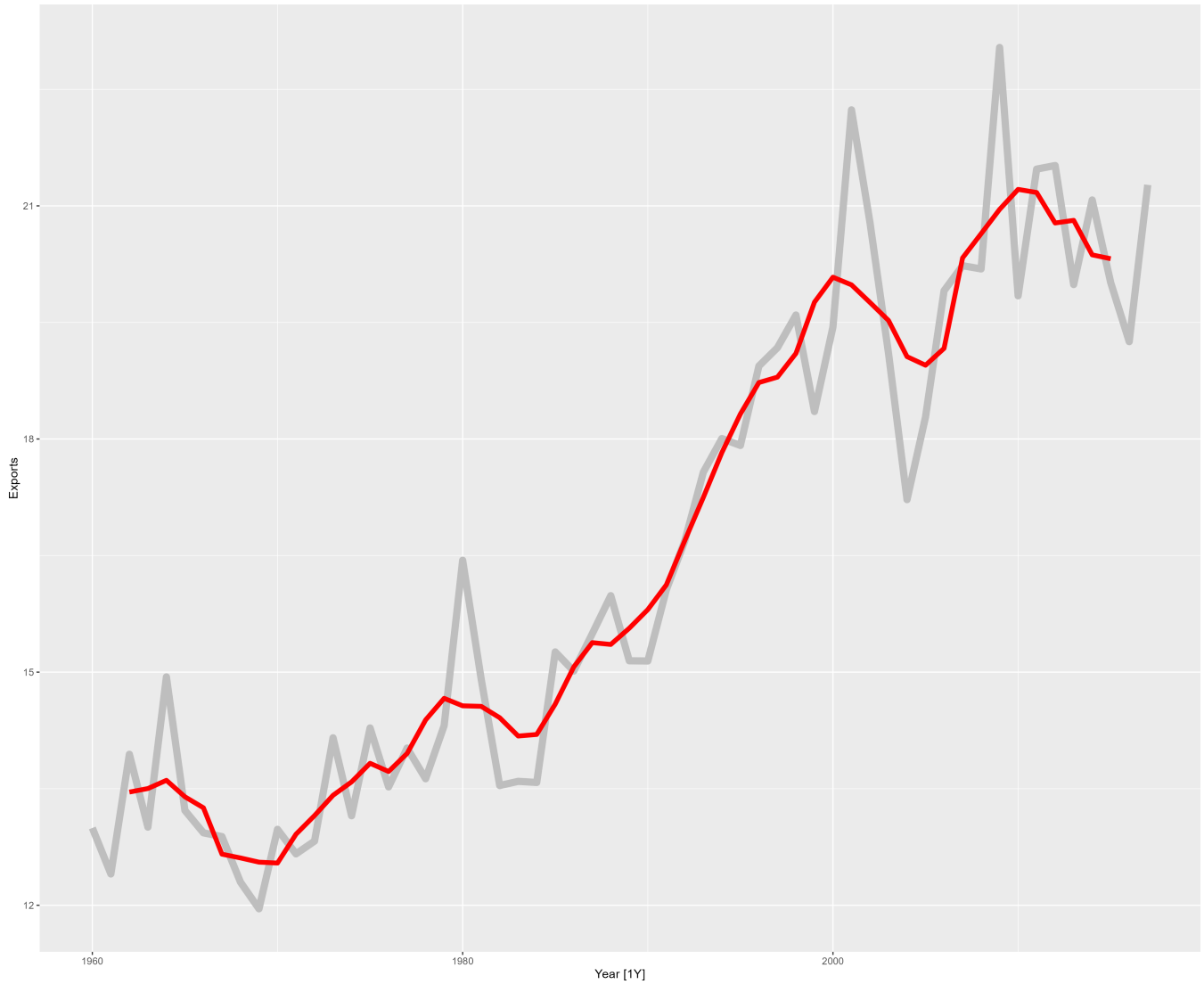
```
global_economy |> filter(Country == "Australia") |> select(Exports) -> aus_
    exports
aus_exports |> autoplot() + geom_point()

aus_exports_withMA <- aus_exports |>
  mutate(MA3 = slider::slide_dbl(Exports, mean, .before=1, .after = 1, .complete
      = TRUE)) |>
  mutate(MA5 = slider::slide_dbl(Exports, mean, .before=2, .after = 2, .complete
      = TRUE)) |>
  mutate(MA7 = slider::slide_dbl(Exports, mean, .before=3, .after = 3, .complete
      = TRUE))
```


Exports
21
18
15
12
1960
1980
2000
Year [1Y]

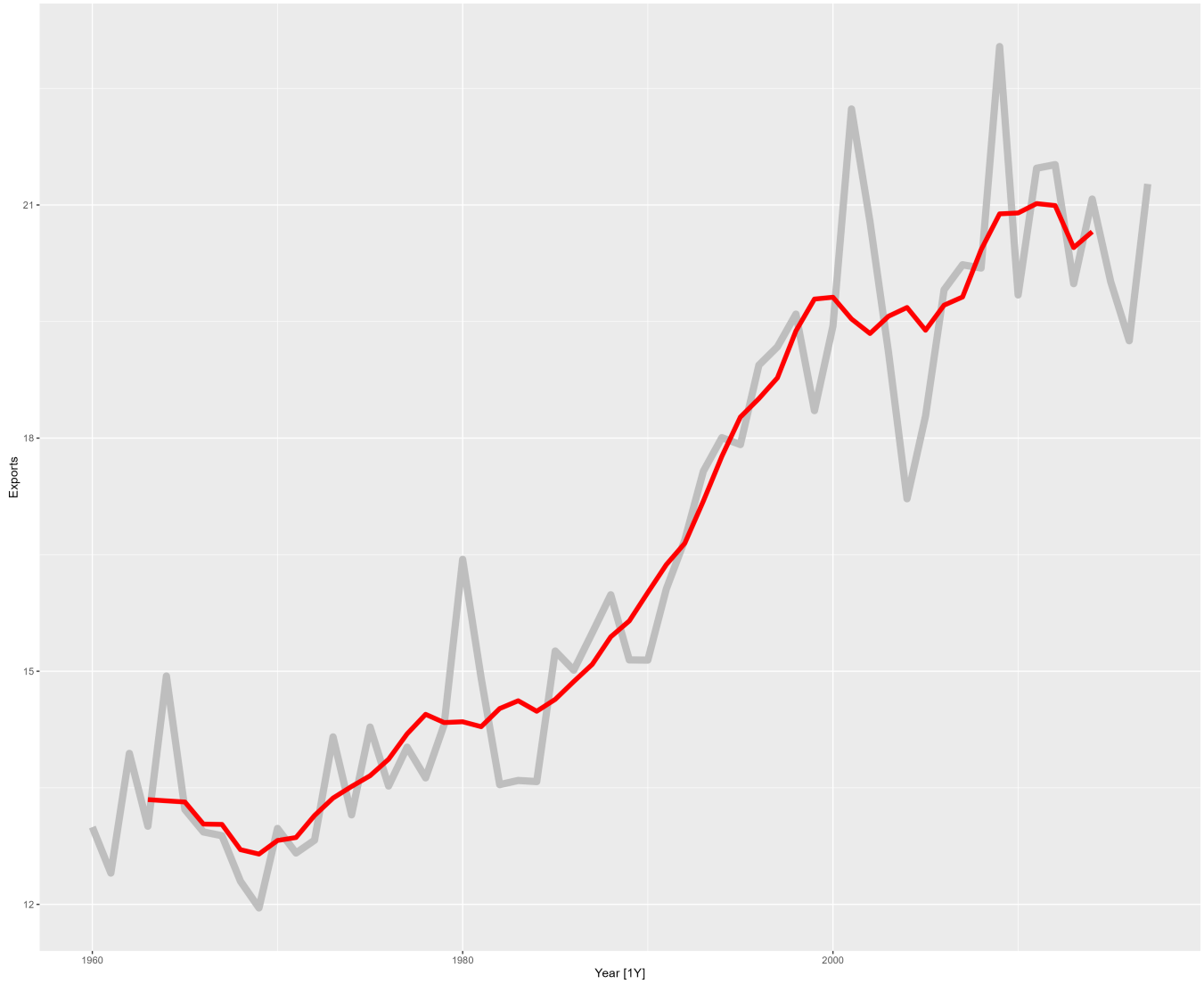
- 3-MA and original data



- 5-MA and original data



- 7-MA and original data



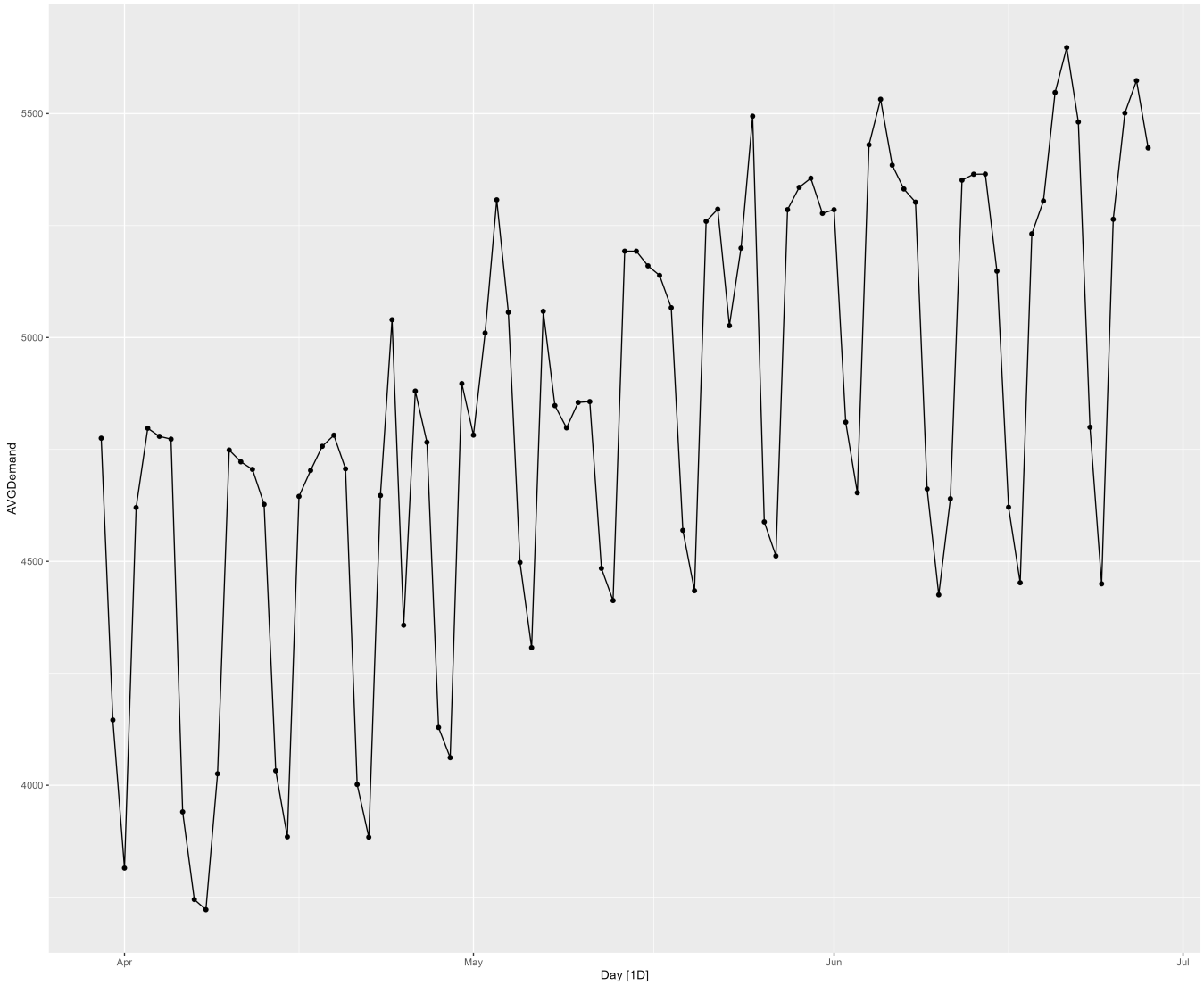
- What do you notice on this plots? How does the moving average line change, when we increase the value of m?

- Observe how the red line is shorter on the ends than our series. This is due to the fact that we did not calculate the moving average for incomplete periods. For a *5-MA* you need 2 observations before and 2 after your observation, so for the first and last two observations we cannot compute the average. This is set in *slide_dbl* with the *.complete* parameter, which flags if we want to compute the average for complete windows of observations only.
- The moving average is a smoother line than the original time series. How smooth the line is, depends on the value of m, i.e. how many observations we consider before and after. The larger the number of observations, the smoother the line.
- Simple moving averages are in general of an odd-order, so that they are symmetric.

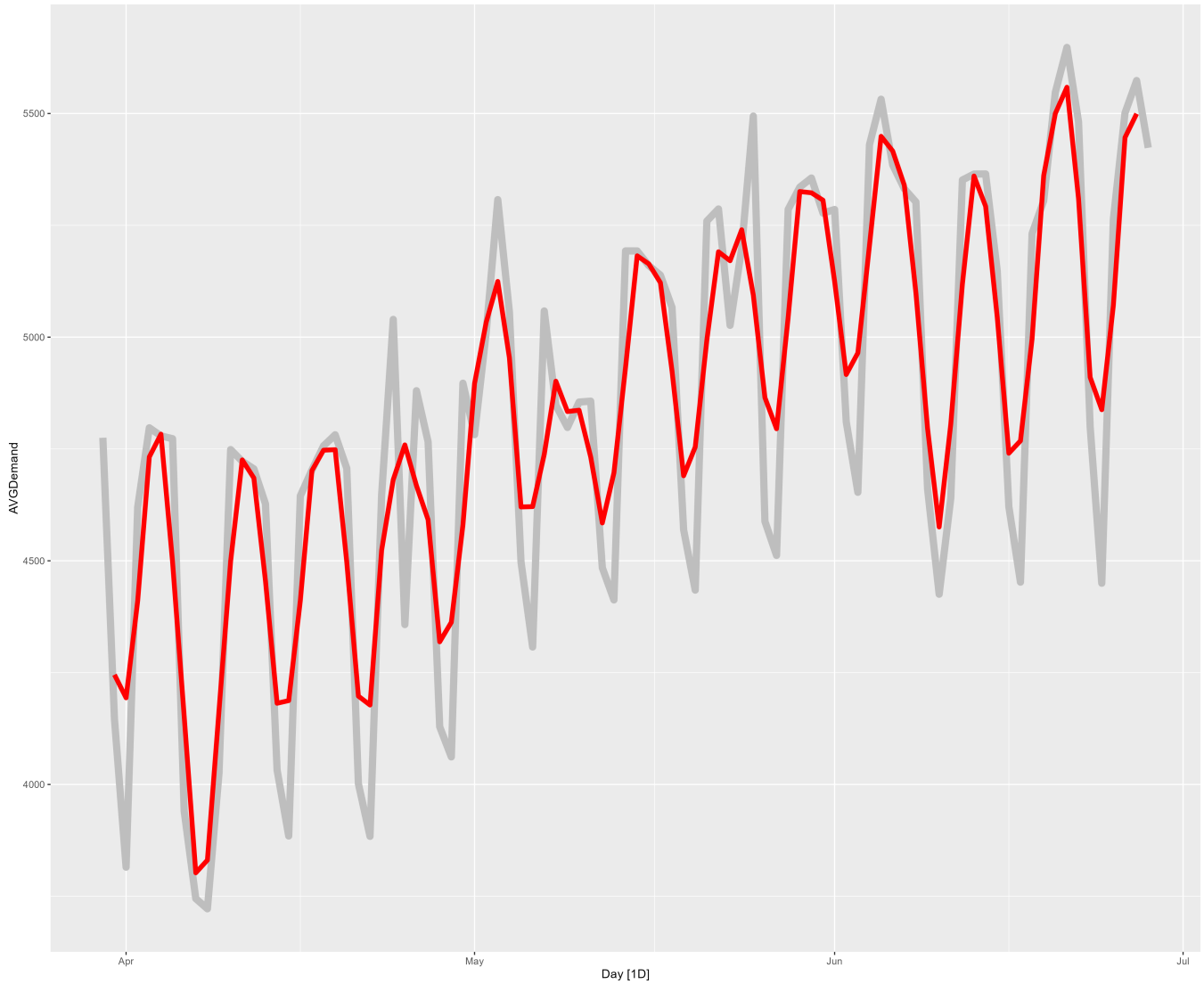
- Let us take another example, the *vic_elec* time series, but let us aggregate its data to average daily data (instead of 30 minutes frequency).
- Also, for better visibility of the plots, we will only consider approximately 3 months data, for which we will look at 3-MA, 5-MA, 7-MA, 9-MA, 11-MA.

```
vic_elec
vic_elec_daily <- vic_elec |>
  mutate(Day = as_date(Time))
vic_elec_daily |>
  index_by(Day) |>
  summarize(AVGDemand = mean(Demand)) -> vic_elec_daily
vic_elec_daily |> head(n = 366) ->vic_elec_daily
vic_elec_daily |> slice(90:180) -> vic_elec_daily
vic_elec_daily |> autoplot() + geom_point()

vic_elec_daily |>
  mutate(MA3 = slider::slide_dbl(AVGDemand, mean, .before=1, .after = 1, .
      complete = TRUE)) |>
  mutate(MA5 = slider::slide_dbl(AVGDemand, mean, .before=2, .after = 2, .
      complete = TRUE)) |>
  mutate(MA7 = slider::slide_dbl(AVGDemand, mean, .before=3, .after = 3, .
      complete = TRUE)) |>
  mutate(MA9 = slider::slide_dbl(AVGDemand, mean, .before=4, .after = 4, .
      complete = TRUE)) |>
  mutate(MA11 = slider::slide_dbl(AVGDemand, mean, .before=5, .after = 5, .
      complete = TRUE))  ->vic_elec_daily_withMA
```

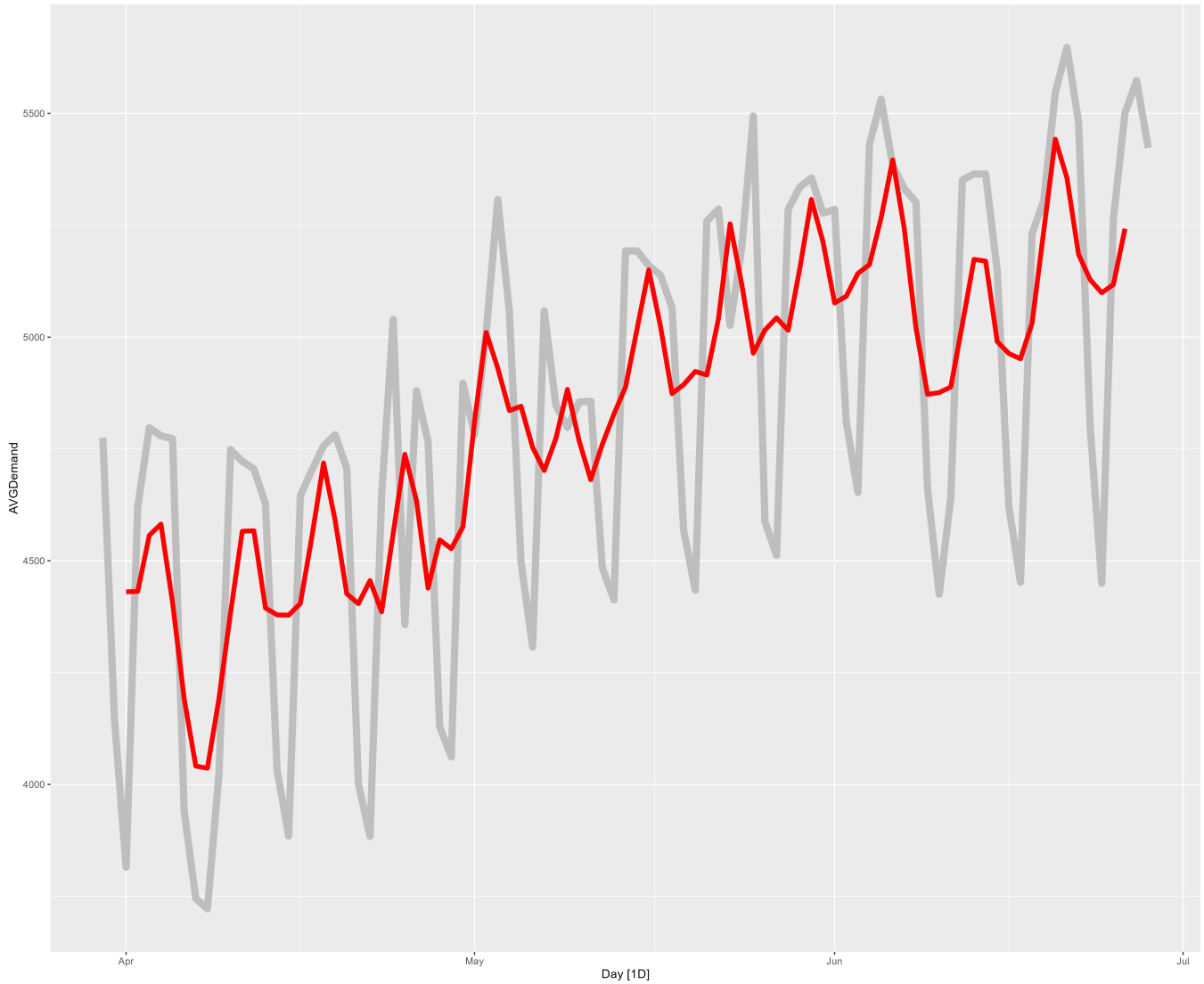
- Original data



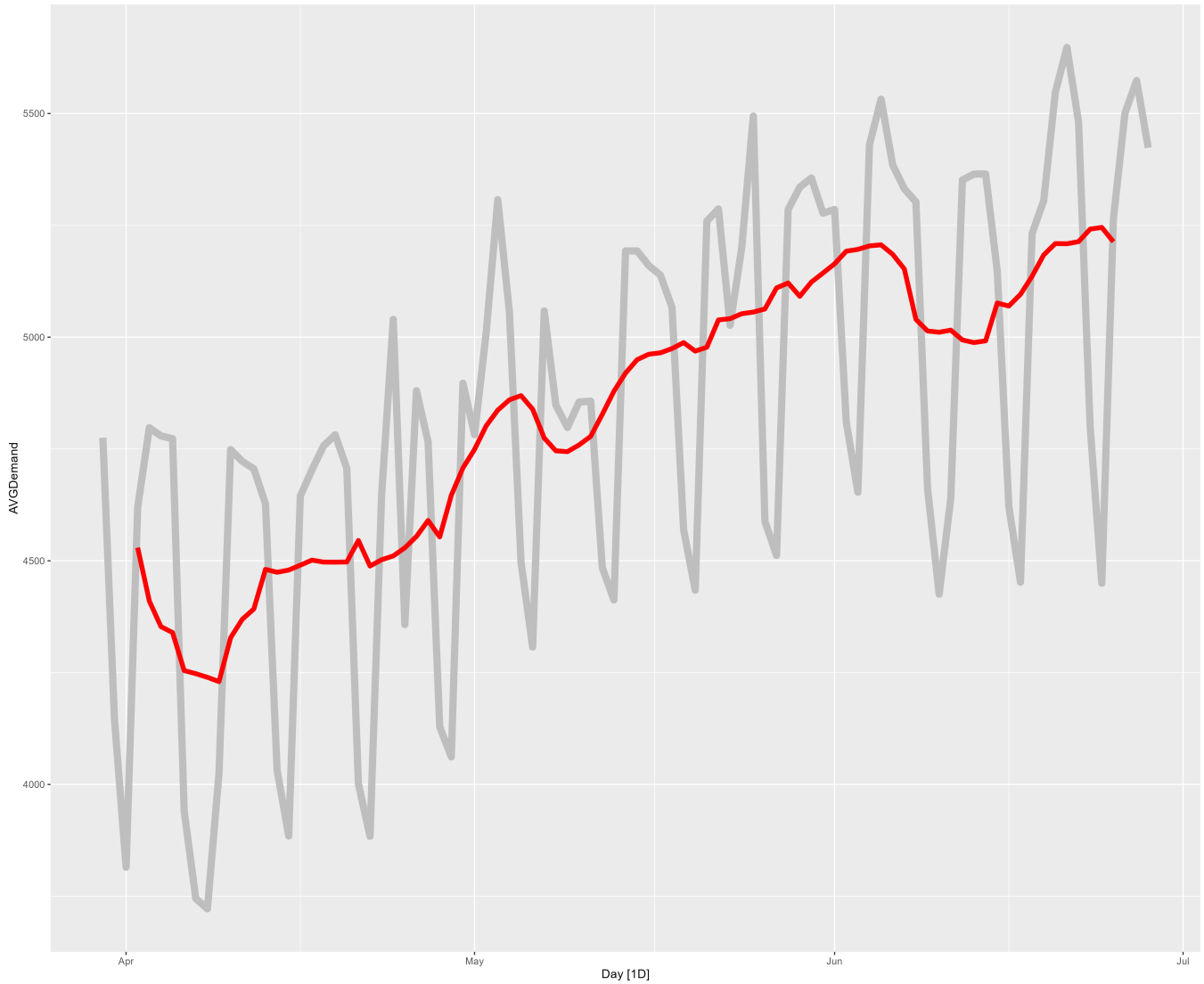
- 3-MA and original data



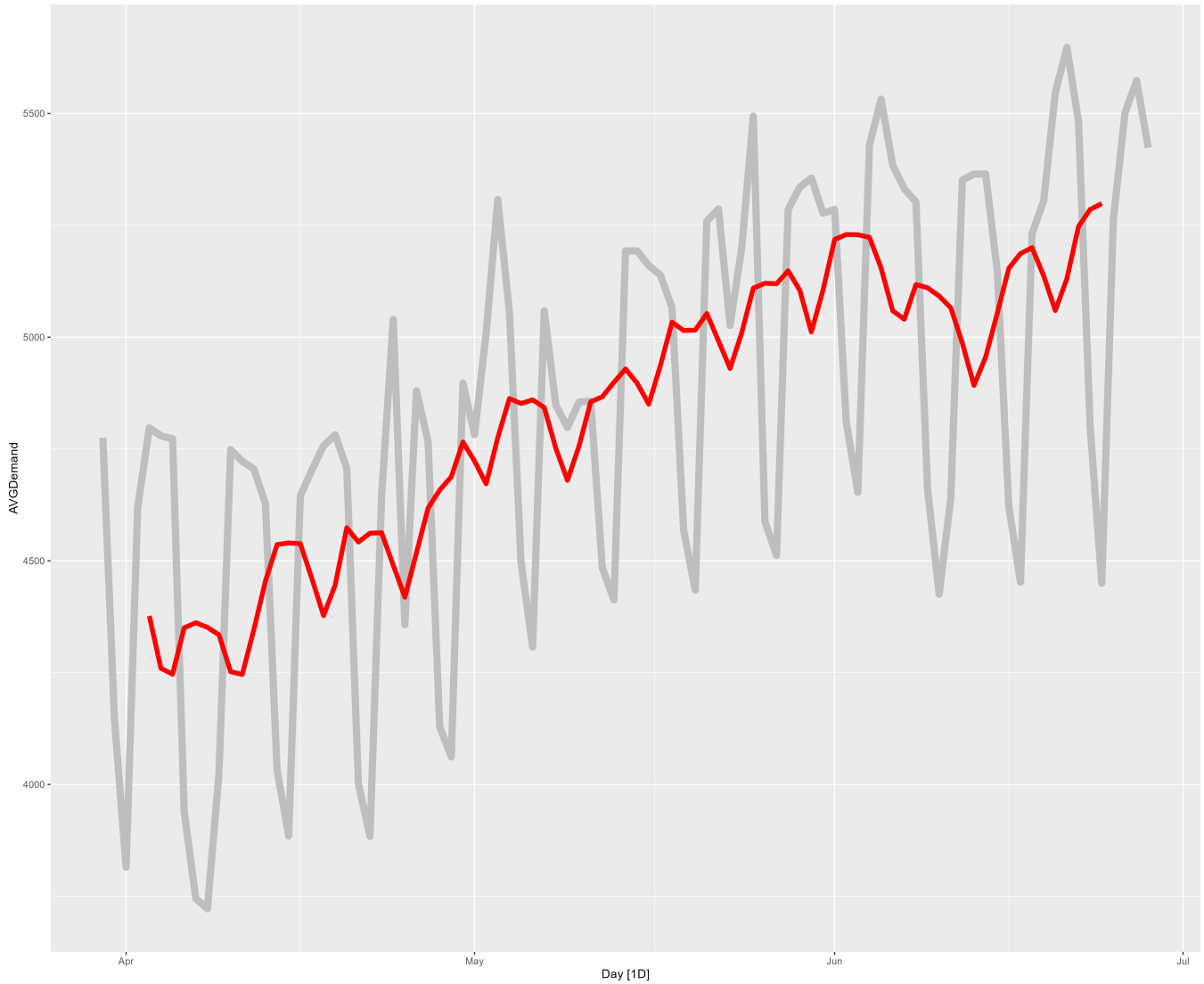
- 5-MA and original data



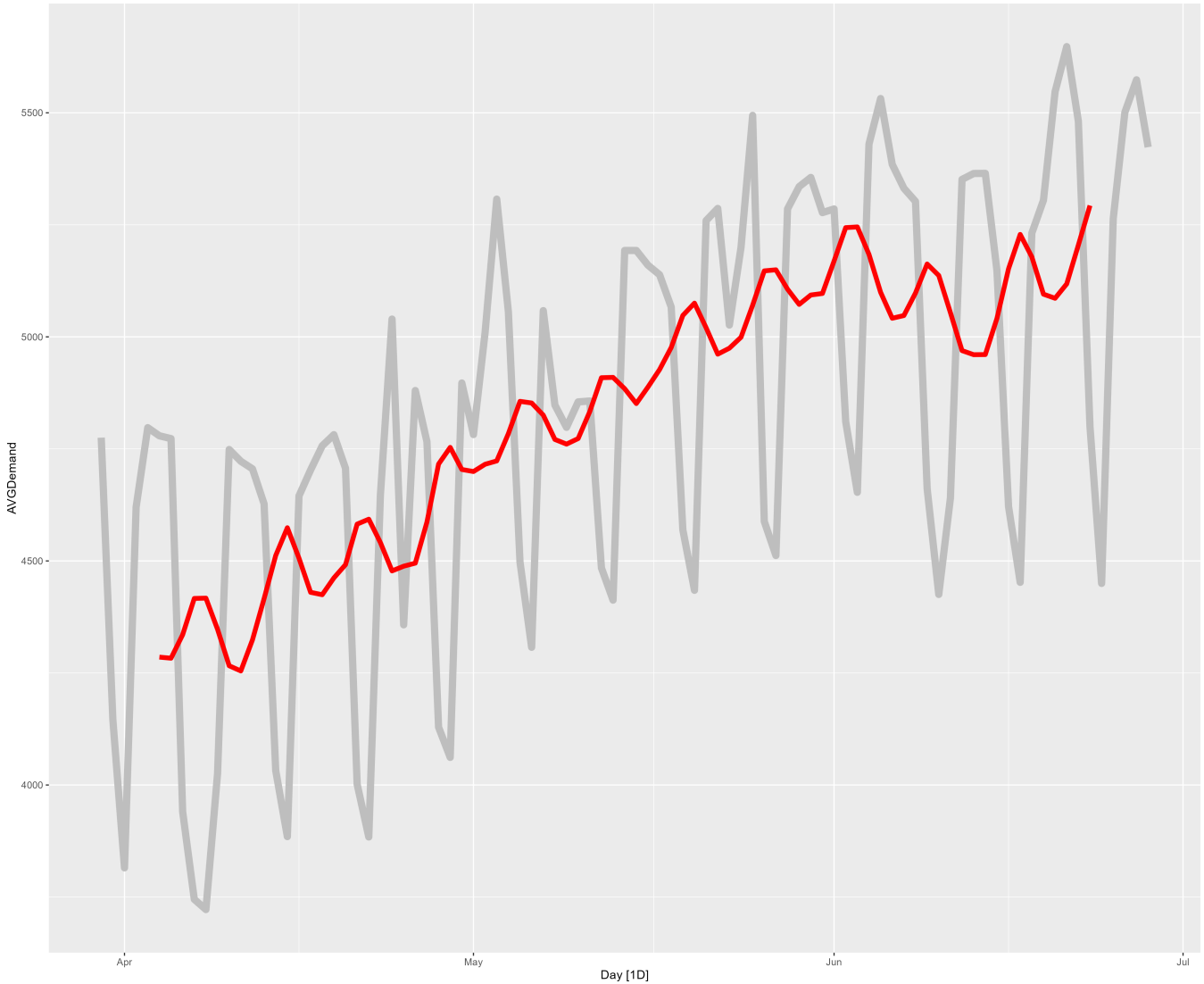
- 7-MA and original data



- 9-MA and original data

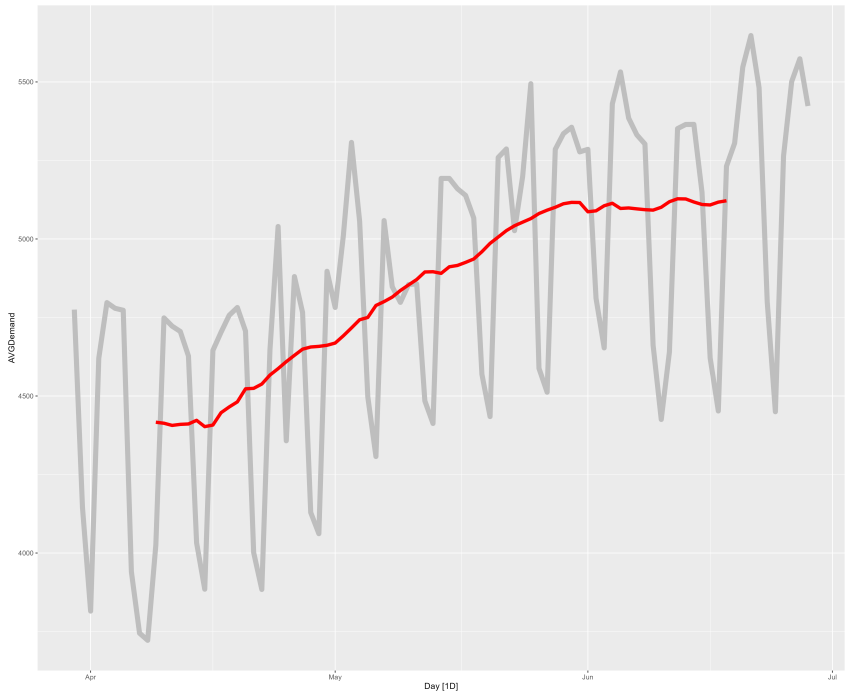


- 11-MA and original data



- Out of the MAs, which one seemed to be the best (smoothest) to you? Do you have any idea why that was the smoothest?

- If the order of the MA is equal to the seasonal length, the MA actually eliminates the effect of seasonality. This is why the 7-MA was looks the smoothest out of them. And if we look at the 21-MA that is even smoother.



- But what do we do if the seasonal length is an even number?

Moving averages of moving averages

- Moving averages of an even order are not symmetric. For example, if you take a 4-MA, you need to decide whether to consider 1 observation before and 2 after your observation, or the other way around.
- However, a moving average of an even order can be made symmetric, if we compute the moving average of the moving average.

- For example, we can compute a *4-MA* and then compute a *2-MA* on the result. In general this is denoted by $2 \times 4 - MA$.
- As an example, consider a simple time series, t_1, t_2, t_3, t_4, etc. In order to compute the *4-MA* we will have (I choose 2 observations before and 1 after):

$$\hat{T}_3 = \frac{1}{4} * (y_1 + y_2 + y_3 + y_4)$$

$$\hat{T}_4 = \frac{1}{4} * (y_2 + y_3 + y_4 + y_5)$$

$$\hat{T}_5 = \frac{1}{4} * (y_3 + y_4 + y_5 + y_6)$$

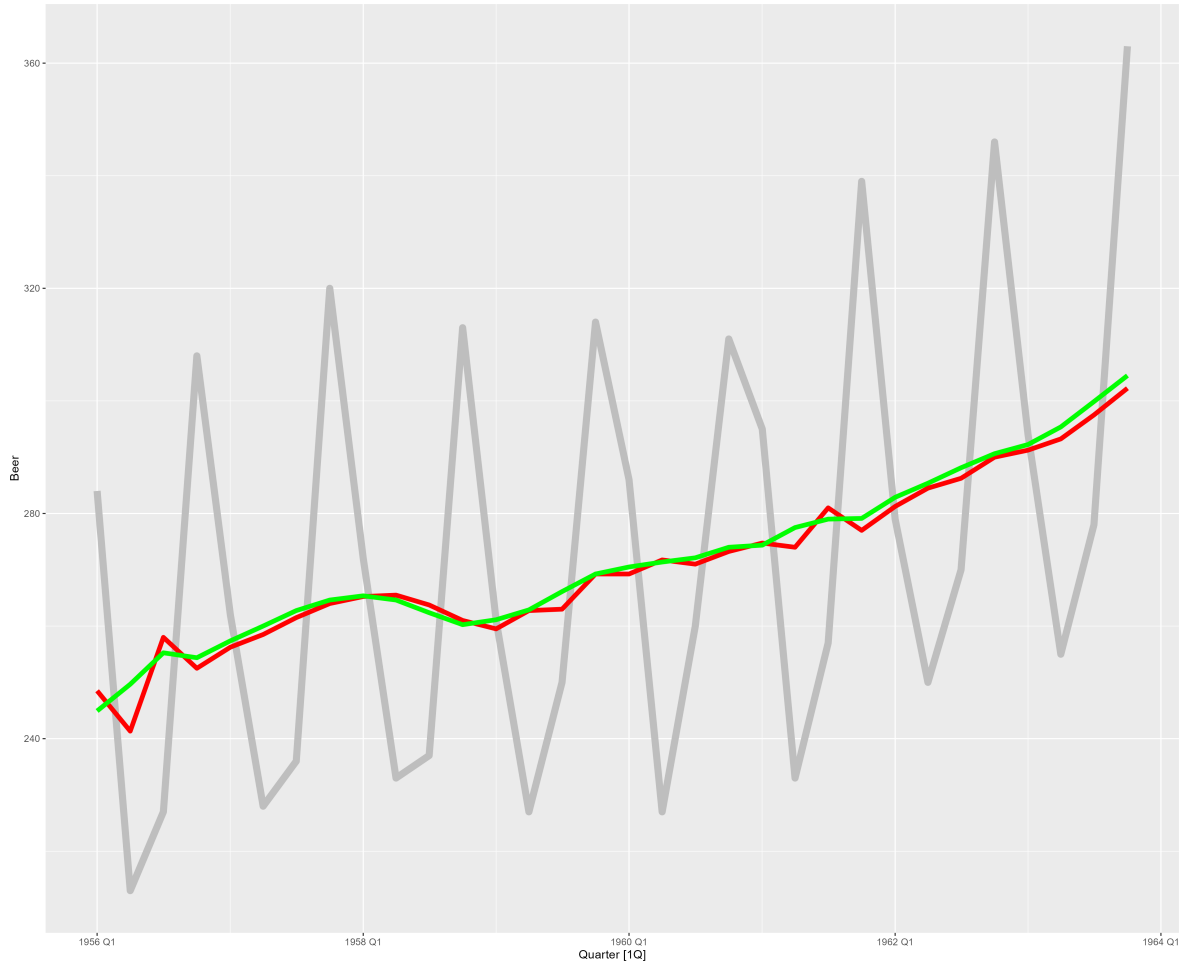
$$\dots$$

- Then, to compute the *2-MA* on this series, we will have

$$\frac{1}{2} * (\hat{T}_3 + \hat{T}_4) = \frac{1}{8}y_1 + \frac{1}{4}y_2 + \frac{1}{4}y_3 + \frac{1}{4}y_4 + \frac{1}{8}y_5$$

- which is a symmetric weighted average of observations.

```
beer |> autoplot() + geom_point()
beer |> mutate(MA4 = slider::slide_dbl(Beer, mean, .before = 2, .after = 1,
      complete = TRUE)) |>
  mutate(MA2x4 = slider::slide_dbl(MA4, mean, .before = 0, .after = 1, complete
     = TRUE)) ->beerMA

beerMA |> autoplot(Beer, color = "grey", size = 3) +
  autolayer(beerMA, MA4, color = "red", size = 2) +
  autolayer(beerMA, MA2x4, color = "green", size = 2)

beerMA_Short <- beerMA |> head(n = 32)
beerMA_Short |> autoplot(Beer, color = "grey", size = 3) +
  autolayer(beerMA_Short, MA4, color = "red", size = 2) +
  autolayer(beerMA_Short, MA2x4, color = "green", size = 2)
```


360
320
280
240
Beer
1956 Q1
1958 Q1
1960 Q1
1962 Q1
1964 Q1
Quarter [1Q]

- Different combinations of moving averages are possible, but the general rule is that a moving average of an odd order should be followed by another one of an odd order, while a moving average of an even order should be followed by a moving average of an even order.

- When a moving average of an even order, m is followed by a moving average of order 2, we call that a *centered moving average of order* $m + 1$, where all observations have a weight of $\frac{1}{m}$ except for the first and last which have a weight of $\frac{1}{2m}$.

- Moving averages can be used to estimate the trend-cycle component of a time series. Consider the *2x4-MA* (the general formula derived from the example above):

$$\hat{T}_t = \frac{1}{8}y_{t-2} + \frac{1}{4}y_{t-1} + \frac{1}{4}y_t + \frac{1}{4}y_{t+1} + \frac{1}{8}y_{t+2}$$

- If the data on which we compute the *2x4-MA* is quarterly, then each quarter will have an equal weight in the moving average, since three quarters have a weight of $\frac{1}{4}$ and the two quarters with weight $\frac{1}{8}$ are the same quarter, but in different years. Since every quarter is included with equal weight, the result will average out (smooth out) the seasonal variation. We can get a similar result if the do a *2X8-MA* or *2X12-MA* on quarterly data.

- The trend-cycle component of a time series can be estimated by moving-averages in the following way:
  - If the seasonal period is even and of order m, we use a *2xm-MA*
  - If the seasonal period is odd and of order m, we use a *m-MA*
  - If there are no seasons in the data, just choose an odd value.
- We can choose multiples of m in both cases, but other values will lead to an estimate which probably contains seasonal variation.

Weighted moving average

- Combining moving averages will lead to weighted moving averages.

$$\hat{T}_t = \sum_{j=-k}^{k} a_j * y_{t+j}$$

- where $k = \frac{m-1}{2}$, a_j is the weight of observation y_j and it is important that the weights are symmetric and their sum is 1.

- The already mentioned *2x4-MA* will lead to a weighted moving average which is the same as the *5-MA* with the following weights: $[\frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}]$.