Chapter 3: Time series decomposition

Ankit Gupta

30/11/2023

Time series data can exhibit a variety of patterns, and it is often helpful to split a time series into several components, each representing an underlying pattern category.

When we decompose a time series into components, we usually combine the trend and cycle into a single trend-cycle component (often just called the trend for simplicity). Thus we can think of a time series as comprising three components: a trend-cycle component, a seasonal component, and a remainder component (containing anything else in the time series). For some time series (e.g., those that are observed at least daily), there can be more than one seasonal component, corresponding to the different seasonal periods.

In this chapter, we consider the most common methods for extracting these components from a time series. Often this is done to help improve understanding of the time series, but it can also be used to improve forecast accuracy.

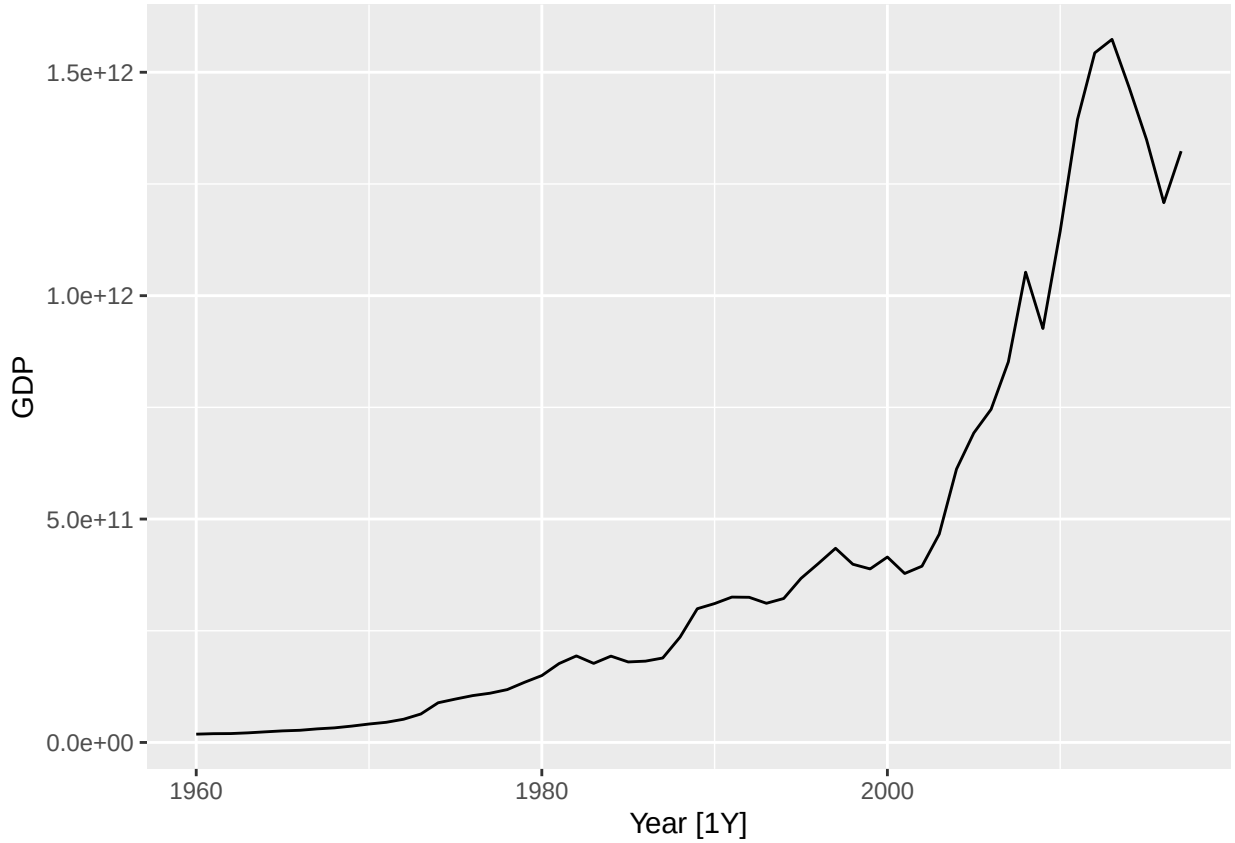
3.1 Transformations and adjustments

Example: Per capita Adjustments

```
library(fpp3)
```

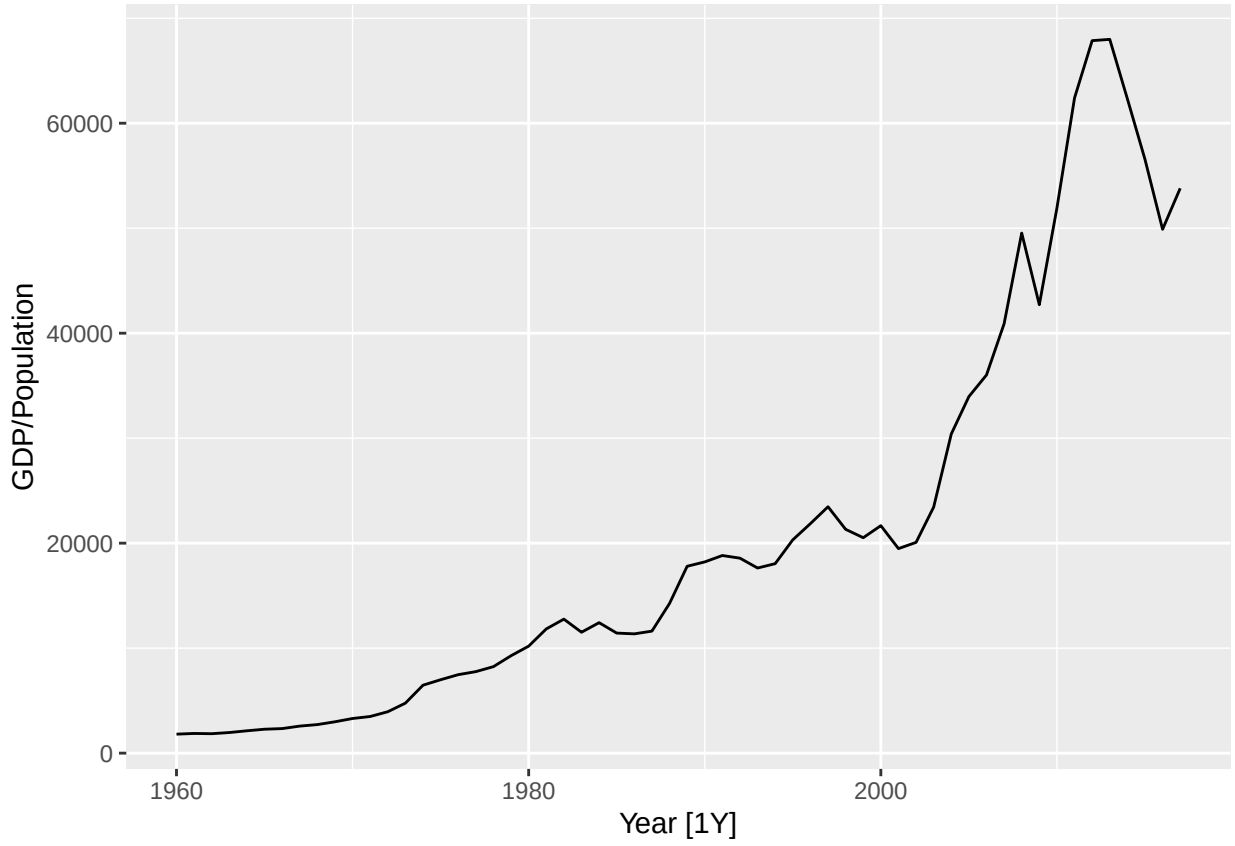
```
## -- Attaching packages --------------------------------------------- fpp3 0.5 --

## v tibble      3.2.1     v tsibble     1.1.3
## v dplyr       1.1.3     v tsibbledata 0.4.1
## v tidyr       1.3.0     v feasts      0.3.1
## v lubridate   1.9.3     v fable       0.3.3
## v ggplot2     3.4.4     v fabletools  0.3.4

## -- 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()
```

```
global_economy |>
  filter(Country== 'Australia') |>
  autoplot(GDP)
```



Here, we divide the data by the population so that instead of looking at total we look at total per capita. Above example shows the annual GDP data since 1960 for the Australia. The population has also grown over time so if we are interested to understand the growth of the economy, we need to think about the economy relative to the size of the population so we can simply divide GDP through by population and we get a data set which is per capita and then it's better to model the per capita data. Also, if we need to forecast GDP then we multiply the forecast forecast to the per capita forecase and get the forecast for the GDP.

```
global_economy |>
  filter(Country== 'Australia') |>
  autoplot(GDP/Population)
```

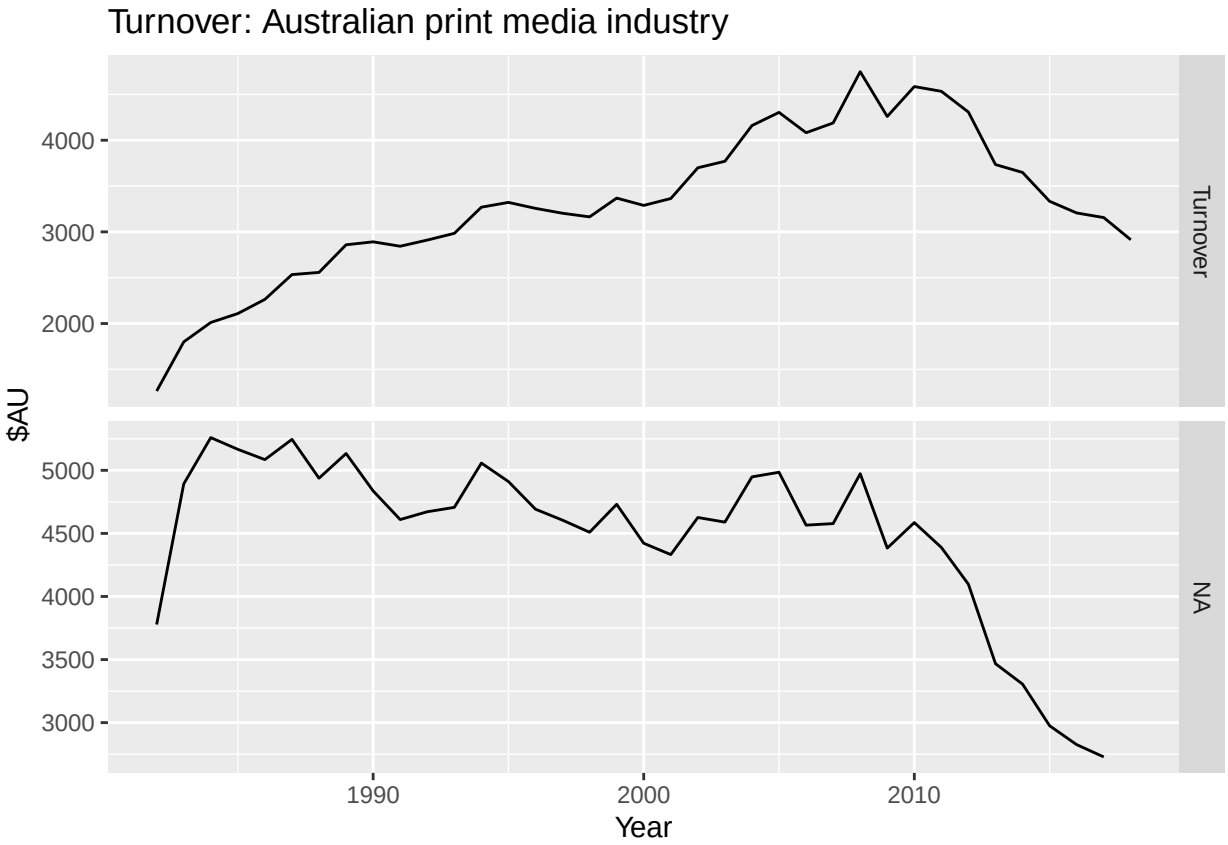


That's the meaning of adjustment here.

Example 2 Inflation Adjustments:

```
print_retail <- aus_retail |>
  filter(Industry == 'Newspaper and book retailing') |>
  group_by(Industry) |>
  index_by(Year = year(Month)) |>
  summarise(Turnover=sum(Turnover))
aus_economy <- global_economy |>
  filter(Code == 'AUS')
print_retail |>
  left_join(aus_economy, by= 'Year') |>
  mutate(Adjusted_turnover = Turnover/CPI*100) |>
  pivot_longer(c(Turnover,Adjusted_turnover),values_to = 'Turnover') |>
  mutate(name=factor(name,levels=c('Turnover','Adjusted_Turnover'))) |>
  ggplot(aes(x=Year, y= Turnover))+
  geom_line()+
  facet_grid(name ~ ., scales= "free_y")+
  labs(title = "Turnover: Australian print media industry",y="$AU")
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



Here, we are combining the two datasets based on year because due to CPI, price retail data is going down when we do autoplot on price retail data after getting sum over turnover, this downfall might due to CPI, so we have to include the factor of CPI in the model that's why we are combining two datasets. NOw, to adjust the turnover, we divide the turnover by CPI and multiply it by 100 (because CPI has a 100 index) so that it will be on the same scale as the original data.

For the above two plots, in the 2nd plot, declining happens back from 1990 once we adjusted the turnover and declining from 2010 is much faster. In the first plot, increment happens initially due to the increase in the value of the dollar and it is not a real increase in retailing of books and newspapers. So when we do the forecast, we often adjust the CPI.

Mathematical Transformations

If the data show different variation at different levels of the series, then a transformation can be useful.

Denote the original observations as $y_1, y_2, ..., y_T$ and the transformed observtions as $w_1, ..., w_T$

Some mathematical transformations for stabilizing variations:

- *Square Root* $w_t = \sqrt{y_t}$ (decreasing)
- *Cube Root* $w_t = y_t^{1/3}$ (increasing)
- *Logarithm* $w_t = \log y_t$ (strength)

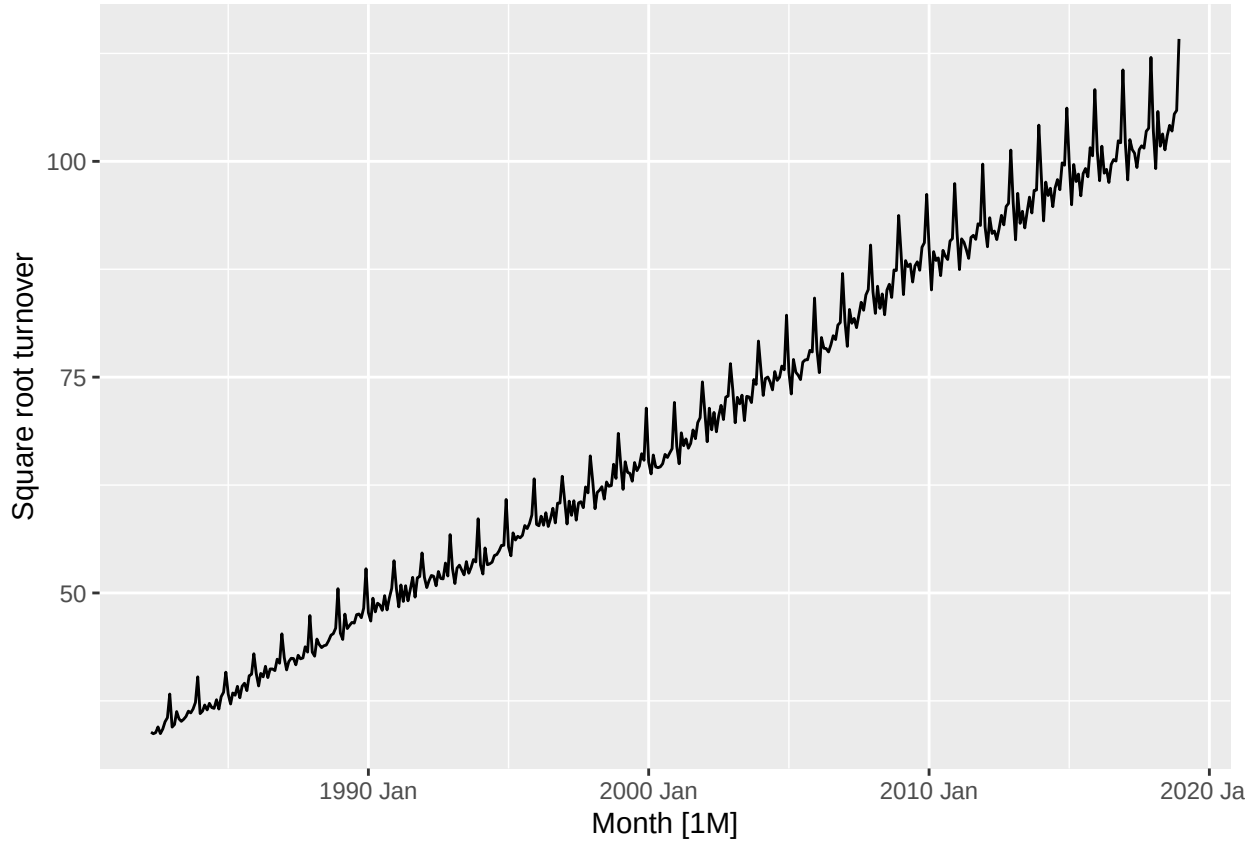
Logarithms, in particular, are useful because they are more interpretable: changes in a log value are relative(percent) changes on original scale. Square roots and cube roots are little harder to interpret.

Example

```
food <- aus_retail |>
  filter(Industry == 'Food retailing') |>
  summarise(Turnover = sum(Turnover))
```

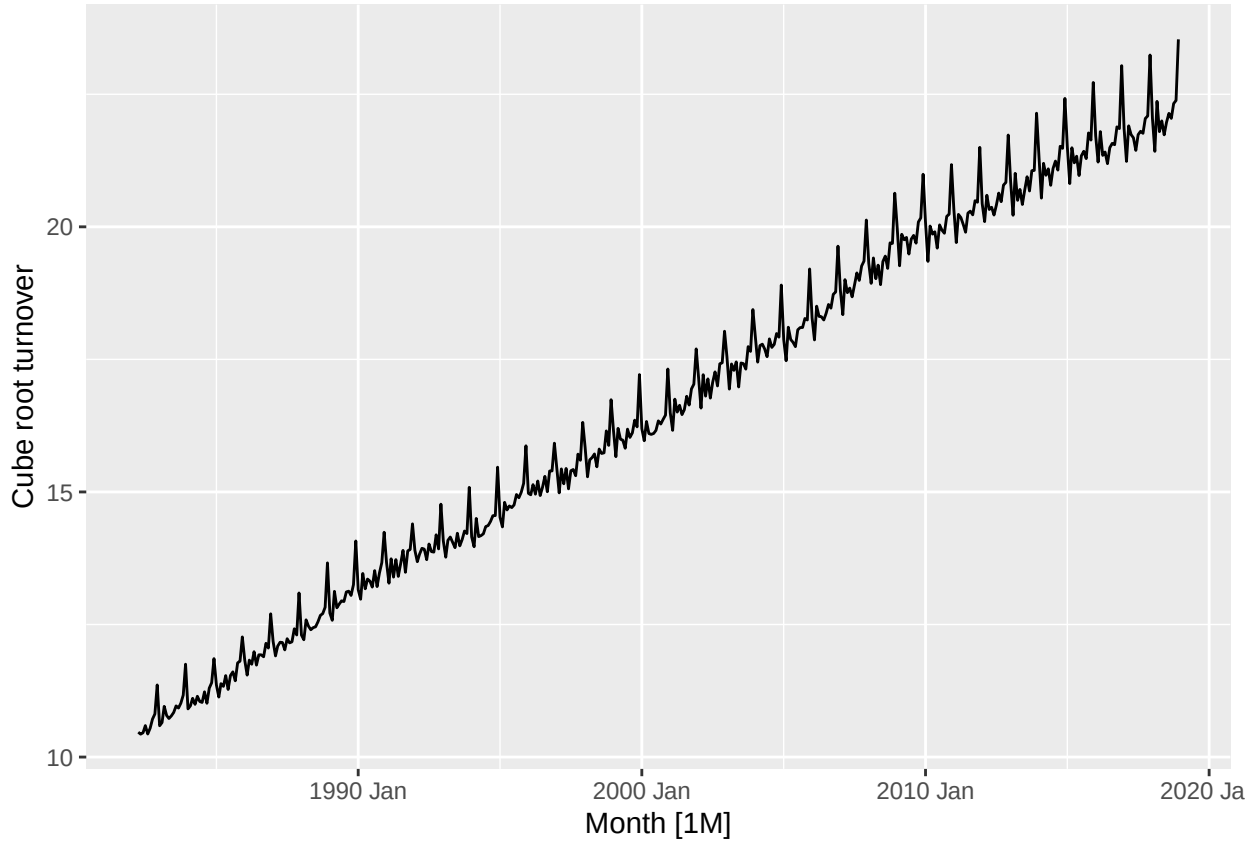
Now, after the transformation, we get:

```
food |> autoplot(sqrt(Turnover)) +
  labs(y = 'Square root turnover')
```



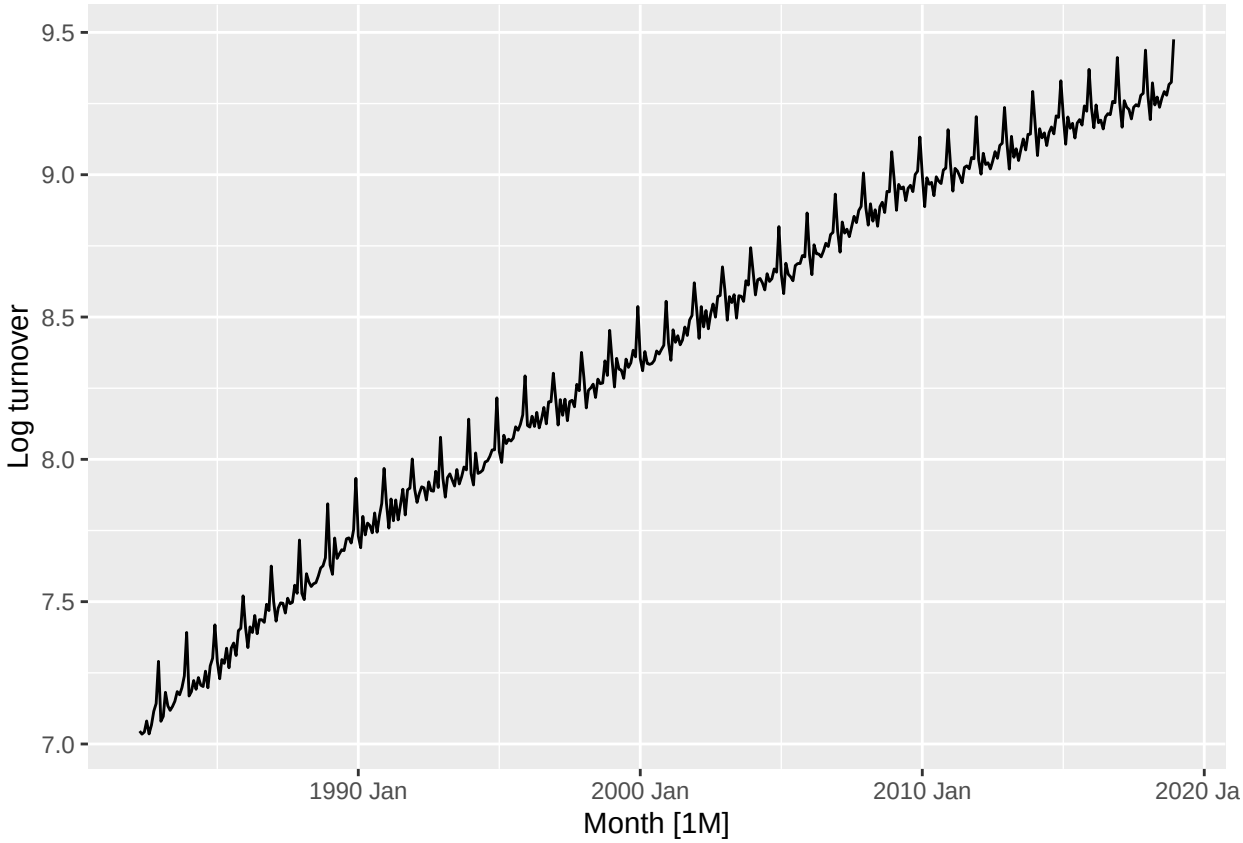
Now, cube root is more stronger because variation is less at around 2020.

```
food |> autoplot(Turnover^(1/3)) +
  labs(y = 'Cube root turnover')
```



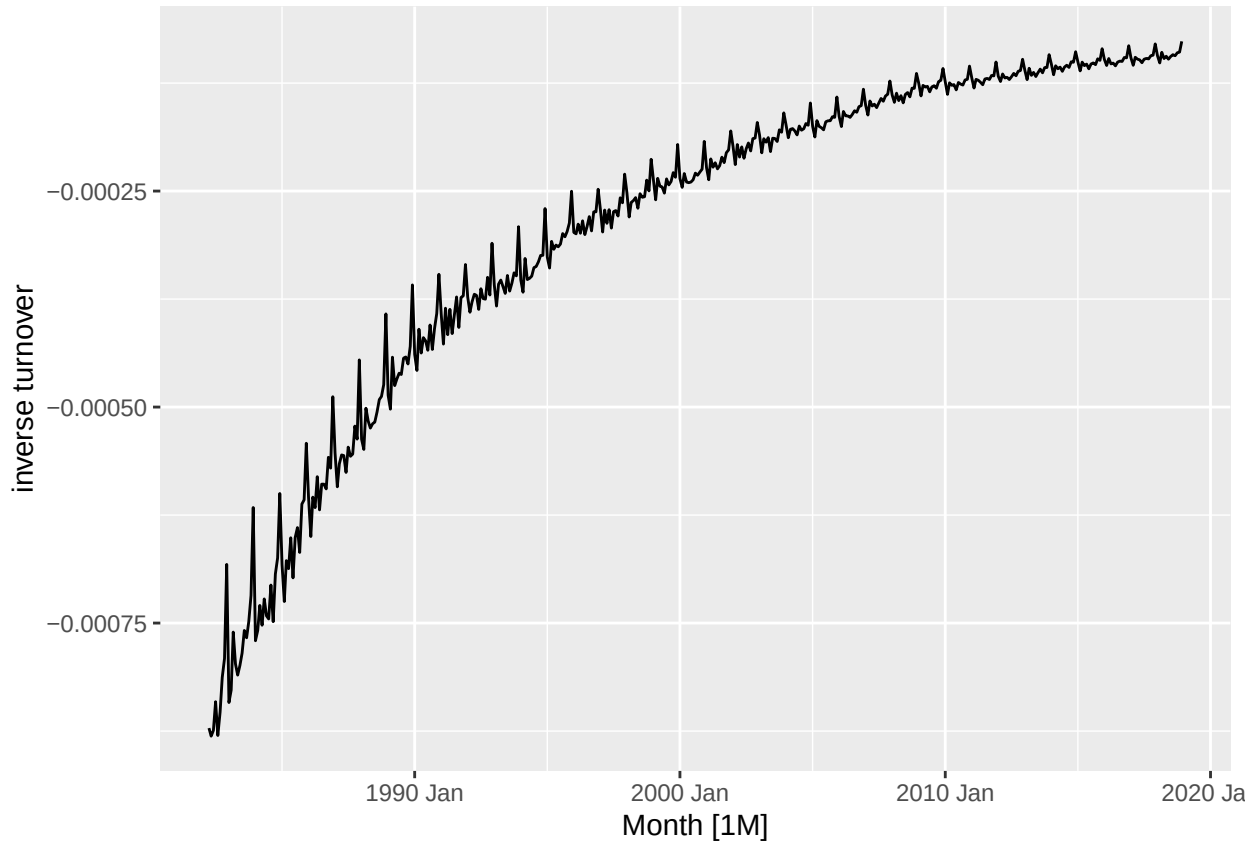
Now, when we take the log, it makes much more even and fluctuations becomes down.

```
food |> autoplot(log(Turnover)) +
  labs(y = 'Log turnover')
```



Now, when we take the inverse transformations.

```
food |> autoplot(-1/Turnover) +
  labs(y = 'inverse turnover')
```



Here, variations at the top end are smaller than the bottom end.

Box-Cox Transformations

Sometimes other transformations are also used (although they are not so interpretable). For example, square roots and cube roots can be used. These are called *power transformations* because they can be written in the form $w_t = y_t^p$.

A useful family of transformations, that includes both logarithms and power transformations, is the family of Box-Cox transformations (Box & Cox, 1964), which depend on the parameter λ

Each of these transformations is close to a member of the family of Box-cox transformations:

$w_t = \log y_t$ when $\lambda = 0$ and

$w_t = (sign(y_t)|y_t|^\lambda - 1)/\lambda$ when $\lambda \neq 0$

Here, λ describes how strong the transformation is.

This is actually a modified Box-Cox transformation, discussed in Bickel & Doksum (1981), which allows for negative values of y_t provided $\lambda > 0$.

- Actually the Bickel-Doksum transformation (not a box-cox transformation and it came after the box-cox transformation) (allowing for $y_t < 0$)
- $\lambda = 1$: No Substantive transformation, we are just shifting here.
- $\lambda = 1/2$: Square root transformation plus linear transformation
- $\lambda = 0$: Natural Logarithm

- $\lambda = -1$: (inverse plus 1)

Box-Cox transformation removes the hetrogenity in the data.

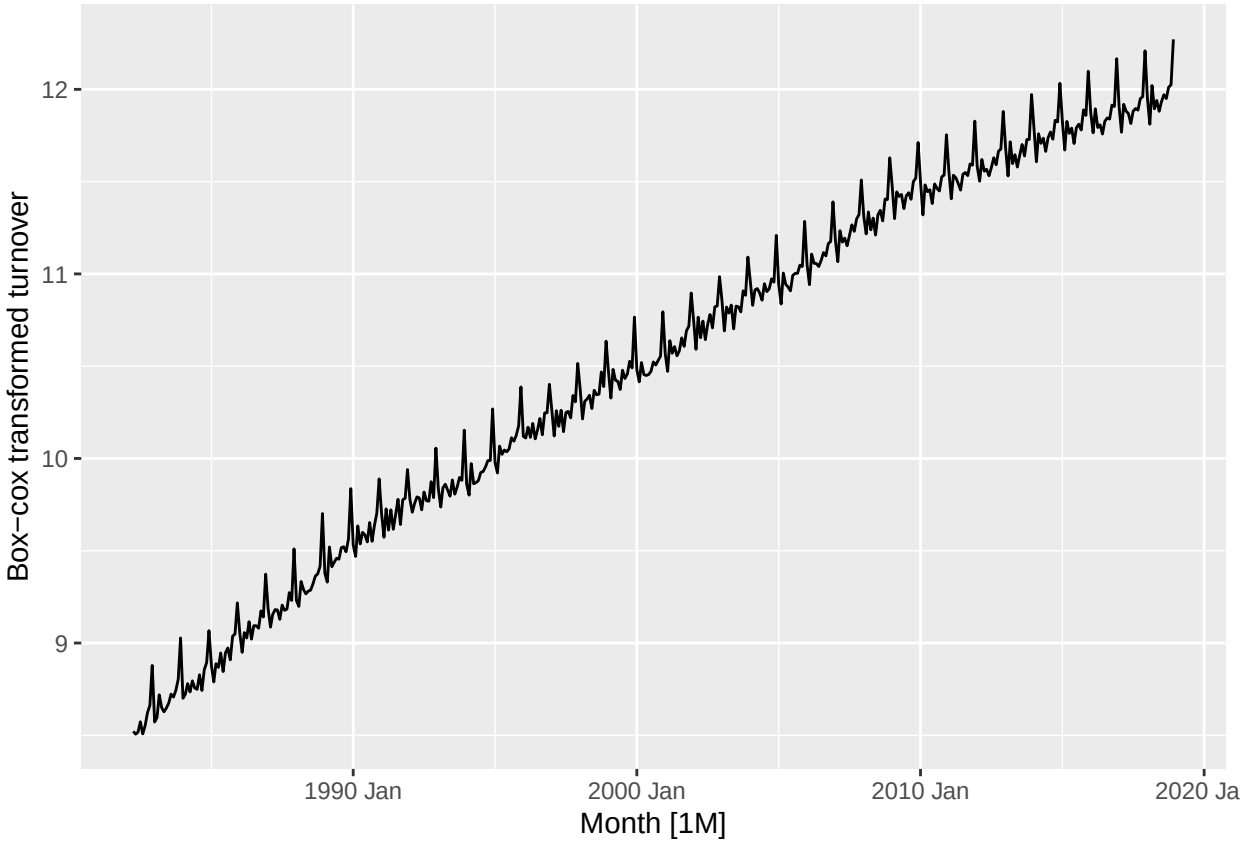
There is a function in the package which enables you to automatically choose λ which is meant to be optimal in some sense which is called a Guerrero (a maxican statistician) transformation.

```
food |>
  features(Turnover,features = guerrero)
```

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

- This attempts to balance the seasonal fluctuations and random variation across the series.
- Always check the results.
- A low value of lambda gives extremely large prediction intervals that we may not be want.

```
food |> autoplot(box_cox(Turnover,0.0524))+
  labs(y="Box-cox transformed turnover")
```



- Often no transform is needed. If you need then try simple ones like log and don't take optimal lambda because it does not much change the result

- Simple transformations are easier to explain and work well enough.
- Transformations can have very large effect on PI(Prediction Interval).
- If some data are zero or negative then use $lambda > 0$
- LOG 1P transformation i.e. log1p() can also be useful for data with zeros.Here we take log(log(data)+1)
- Choosing logs is a simple way to force forecasts to be positive
- Transformations must be reversed to obtain forecasts on the original scale. (Handle automatically by 'fable').

Adjusting the historical data can often lead to a simpler time series. Here, we deal with four kinds of adjustments: calendar adjustments, population adjustments, inflation adjustments and mathematical transformations. The purpose of these adjustments and transformations is to simplify the patterns in the historical data by removing known sources of variation, or by making the pattern more consistent across the whole data set. Simpler patterns are usually easier to model and lead to more accurate forecasts.

Calendar adjustments

Some of the variation seen in seasonal data may be due to simple calendar effects. In such cases, it is usually much easier to remove the variation before doing any further analysis.

For example, if you are studying the total monthly sales in a retail store, there will be variation between the months simply because of the different numbers of trading days in each month, in addition to the seasonal variation across the year. It is easy to remove this variation by computing average sales per trading day in each month, rather than total sales in the month. Then we effectively remove the calendar variation.

Population adjustments

Any data that are affected by population changes can be adjusted to give per-capita data. That is, consider the data per person (or per thousand people, or per million people) rather than the total. For example, if you are studying the number of hospital beds in a particular region over time, the results are much easier to interpret if you remove the effects of population changes by considering the number of beds per thousand people. Then you can see whether there have been real increases in the number of beds, or whether the increases are due entirely to population increases. It is possible for the total number of beds to increase, but the number of beds per thousand people to decrease. This occurs when the population is increasing faster than the number of hospital beds. For most data that are affected by population changes, it is best to use per-capita data rather than the totals.

Inflation adjustments

Data which are affected by the value of money are best adjusted before modelling. For example, the average cost of a new house will have increased over the last few decades due to inflation. A $200,000 house this year is not the same as a $200,000 house twenty years ago. For this reason, financial time series are usually adjusted so that all values are stated in dollar values from a particular year. For example, the house price data may be stated in year 2000 dollars.

To make these adjustments, a price index is used. If z_t denotes the price index and y_t denotes the original house price in year t, then $x_t = (y_t/z_t) * z_2000$ gives the adjusted house price at year 2000 dollar values. Price indexes are often constructed by government agencies. For consumer goods, a common price index is the Consumer Price Index (or CPI).

3.2 Time series components

Time Series Decomposition

Here, we split the time series data into the components trend-cycle component, seasonal component and the remainder component.

$$y_t = f(S_t, T_t, R_t)$$

where

$y_t =$ data at period t

$T_t =$ trend-cycle component at period t

$S_t = $ Seasonal component at period t

$R_t = $ Remainder component at period t

Additive decomposition: $y_t = S_t + T_t + R_t$

Multiplicative decomposition: $y_t = S_t \times T_t \times R_t$

We are mostly going to talk about additive decomposition here.

- Additive model appropriate if magnitude of the seasonal fluctuations does not vary with level. By using Box-Cox transformations, we can ensure that seasonal fluctuations are even and don't change with the level of the series.
- If seasonal patterns are proportional to the level of series then multiplicative model appropriate.
- Multiplicative decomposition more prevalent with economic series. So, Multiplicative decompositions are common with economic time series.
- Alternative for the multiplicative decomposition: Use a Box-Cox transformation, and then use a additive decomposition
- Logs turn multiplicative relationship into an additive relationship:

$$y_t = S_t \times T_t \times R_t$$
$$\rightarrow \log y_t = \log S_t + \log T_t + \log R_t$$

Example: US Retail Employment

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

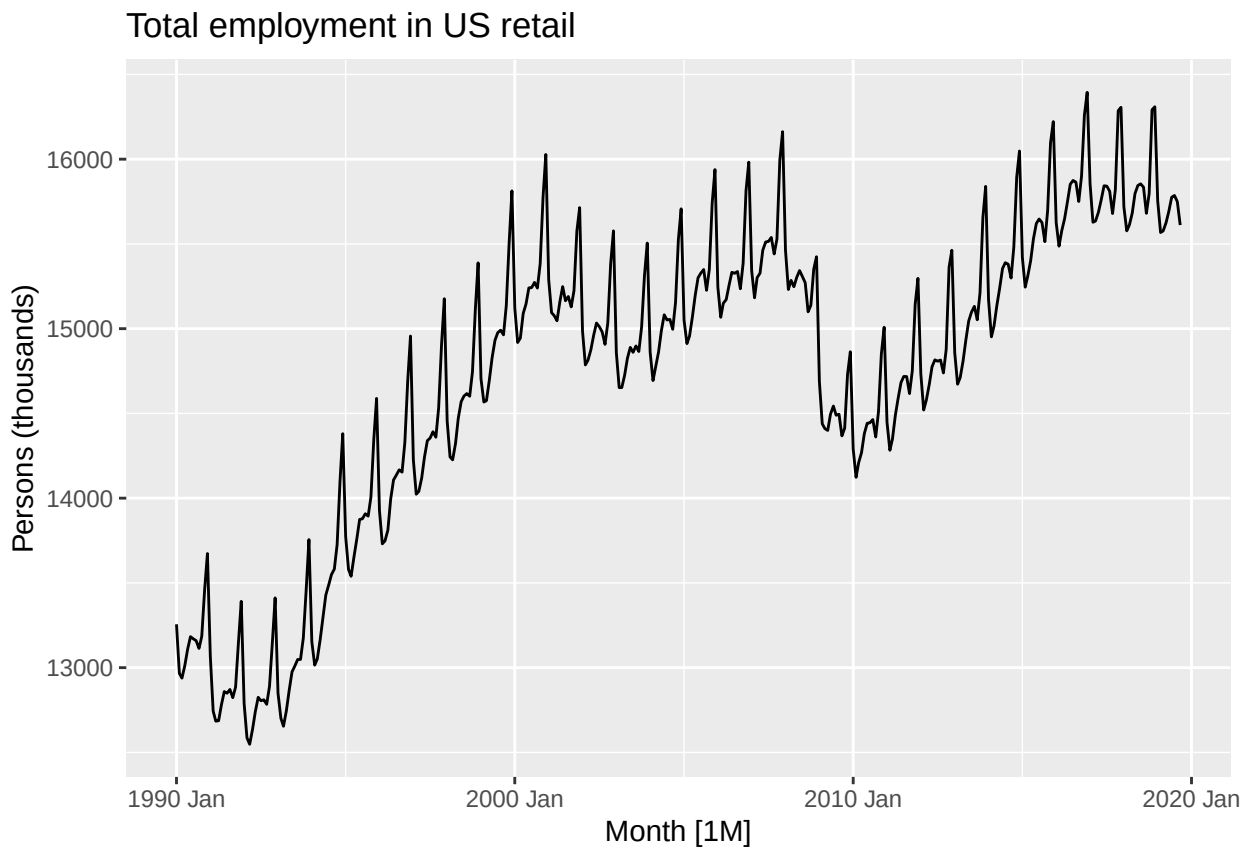
```
## # A tsibble: 357 x 3 [1M]
##       Month Title        Employed
##       <mth> <chr>           <dbl>
##  1 1990 Jan Retail Trade   13256.
##  2 1990 Feb Retail Trade   12966.
```

```
##  3 1990 Mar Retail Trade   12938.
##  4 1990 Apr Retail Trade   13012.
##  5 1990 May Retail Trade   13108.
##  6 1990 Jun Retail Trade   13183.
##  7 1990 Jul Retail Trade   13170.
##  8 1990 Aug Retail Trade   13160.
##  9 1990 Sep Retail Trade   13113.
## 10 1990 Oct Retail Trade   13185.
## # i 347 more rows
```

Here, we are interested in the column "Employed"and we are trying to decompose that into the components.

Before any decomposition, we have to look at the data.

```
us_retail_employment |>
  autoplot(Employed) +
  labs(y="Persons (thousands)",title = "Total employment in US retail")
```



Here we can see the trend and cycle component (see 1990 to 2005 and then repeats)

Seasonal pattern means something happening every year so here you can see every year you get a heart beat symbol like pattern.

We are going try to estimate it and whatever is left is a remainder

So, we take a dataset and pipe it into the model function and say we want to use the STL model which is the decomposition method we are going to use mostly. STL stands for seasonal trend and lowest decomposition.

```
us_retail_employment |>
  model(stl=STL(Employed))
```

```
## # A mable: 1 x 1
##       stl
##   <model>
## 1   <STL>
```

Here in this case we have one time series and one model so output is 1*1.

Now, we store the above stl object in dcmp and use the decomposition() function that gives the components.

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

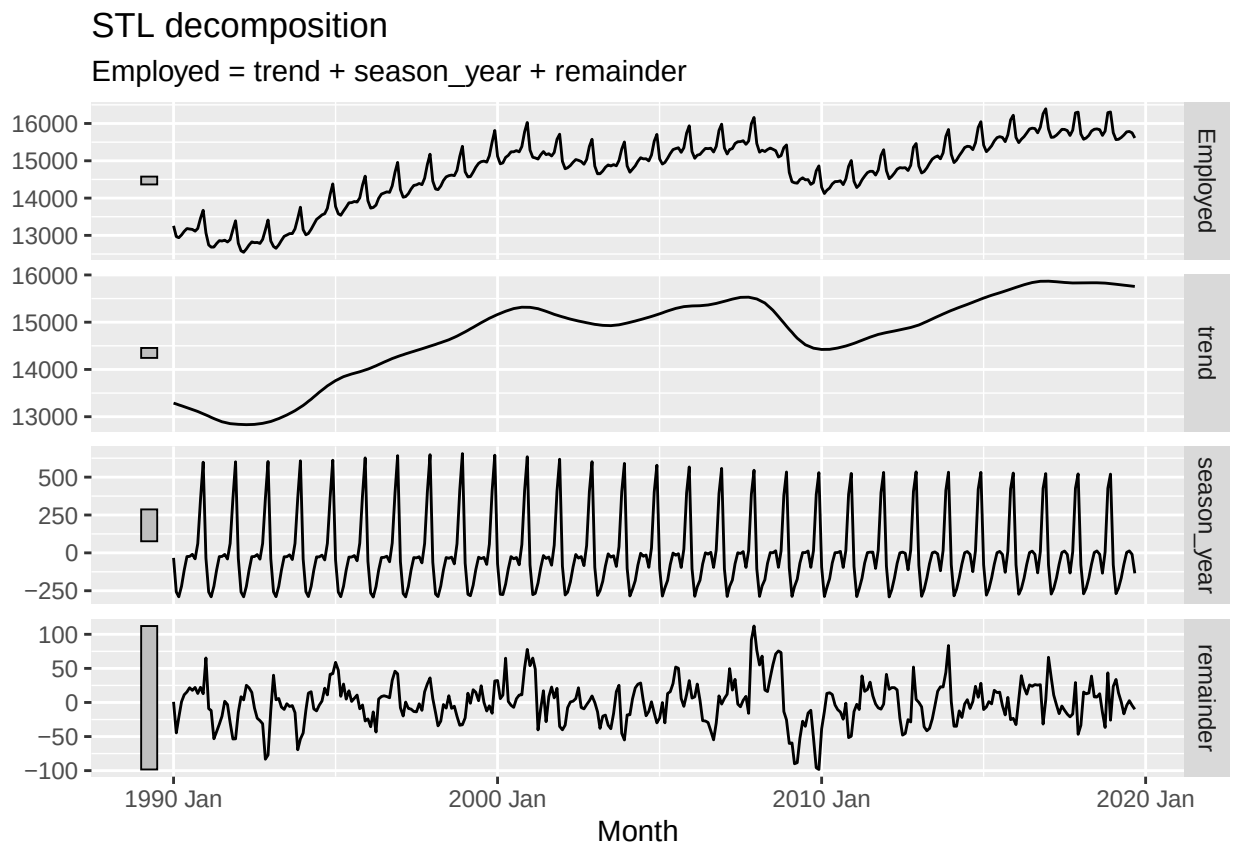
```
## # A dable: 357 x 7 [1M]
## # Key:     .model [1]
## # :        Employed = trend + season_year + remainder
##    .model    Month Employed  trend season_year remainder season_adjust
##    <chr>     <mth>    <dbl>  <dbl>       <dbl>     <dbl>         <dbl>
##  1 stl    1990 Jan   13256. 13288.      -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.
## # i 347 more rows
```

Here the output is dable object which looks like a sibble but it is decomposition table. It has a index column which is "Month" here and it is also having key which is a model column, here we have only one model.

Here, we can see the values $y_t = T_t + S_t + R_t$ where Y_t is the values in Employed column, T_t is trend values, S_t is season_year values and R_t is remainder values.

Now we look at the plots

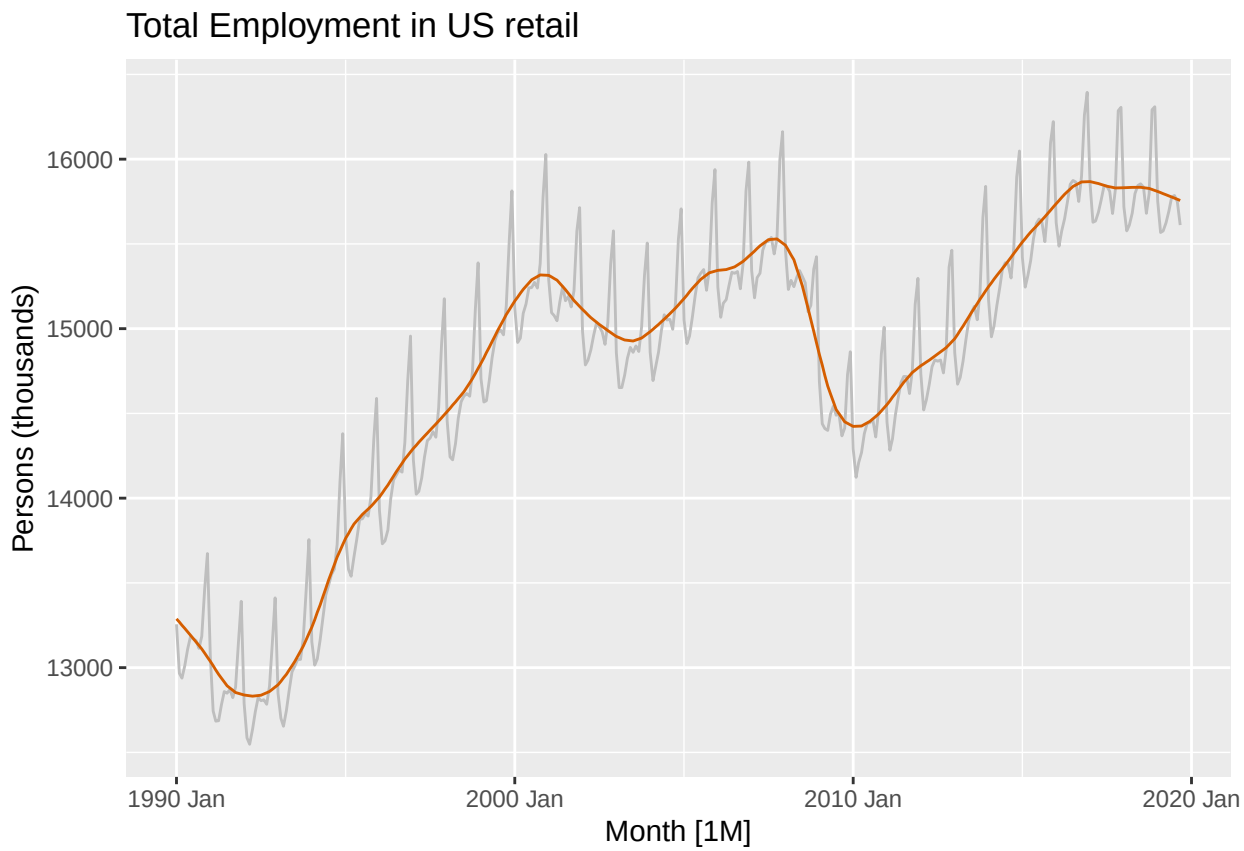
```
components(dcmp) |> autoplot()
```



In the above plot, the bars in the left shows the how the scales for each three plots different from each other (Each are of 200 length scale) but Remainder section has less magnitude than the original data or trend because bar of remainder is bigger than original data and trend.

Let's do some more plots.

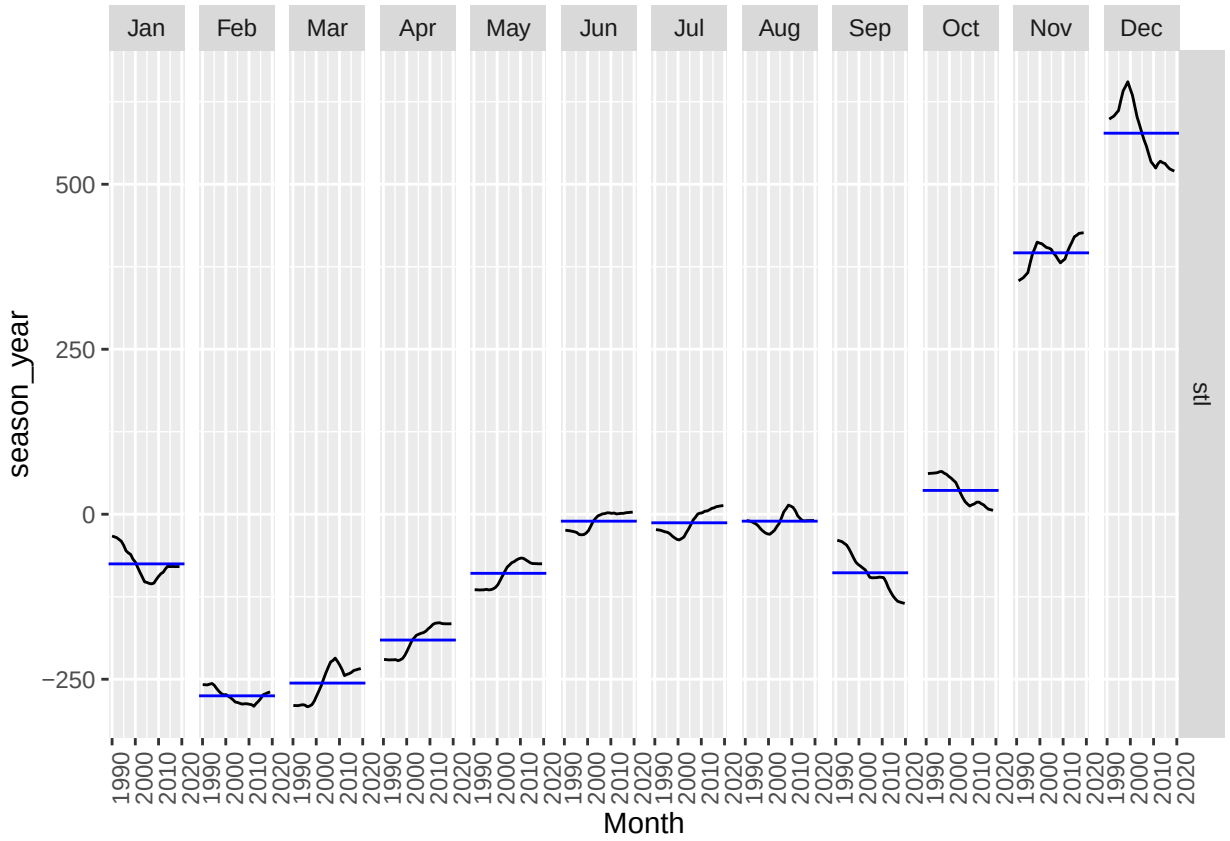
```
us_retail_employment |>
  autoplot(Employed, color="gray")+
  autolayer(components(dcmp),trend,color="#D55E00")+
  labs(y="Persons (thousands)", title = "Total Employment in US retail")
```



Here gray color plot shows the time series plot for the original data. For the orange color plot, we take the data from the components and pulling the trend variable. Here, trend curve provides a smooth varying pattern that goes through the dataset.

Now, we make the sub-series plots.

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



This is sub-series plot of the seasonal component. Here pattern does not change for some months but for December,September it changes.

Seasonal Adjustment:

- Useful by-product of decomposition: an easy way to calculate seasonally adjusted data.
- Additive decomposition: seasonally adjusted data given by

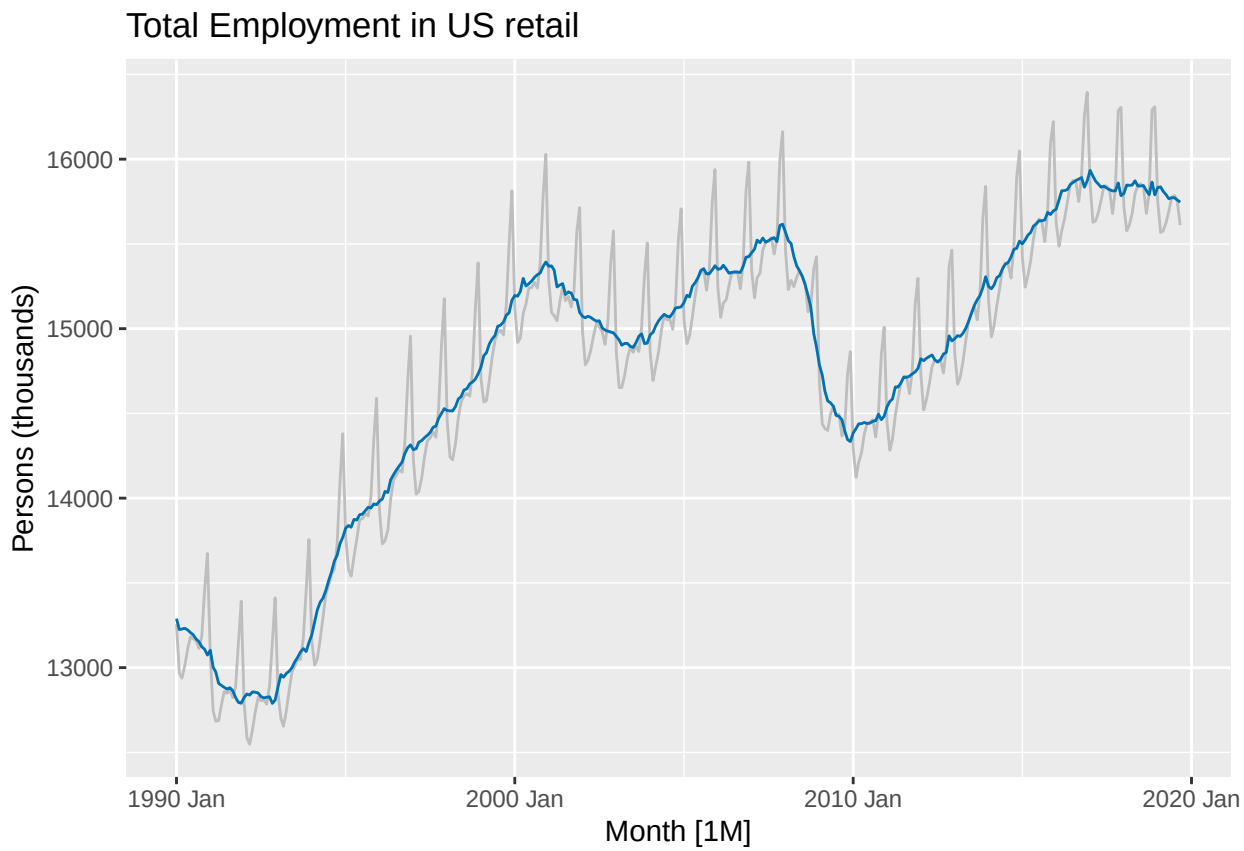
  $$y_t - S_t = T_t + R_t$$

- Multiplicative decomposition: seasonally adjusted data given by

  $$y_t/S_t = T_t \times R_t$$

This is actually the most common thing that people want to use a decomposition for they will fill a decomposition so that they can seasonally adjust the data and every National Statistics office in the world does this sort of thing. They will often use the seasonal adjustment on key economic variables such as the amount of unemployment in the country. They do these things because it helps to see the fluctuations in the data that are not affected by the seasonal pattern so they remove these regular patterns and see what is going on.

```
us_retail_employment |>
  autoplot(Employed, color="gray")+
  autolayer(components(dcmp),season_adjust,color="#0072B2")+
  labs(y="Persons (thousands)", title = "Total Employment in US retail")
```



Here notice that it's much more wiggly than the trend because remainder is still in there and it is both trend and remainder (only trend graph was smooth). We have taken the data and removed the seasonal component and we are left with trend and the remainder and this is done by most people.

- We use estimates of S based on the past values to seasonally adjust a current value.
- Seasonally adjusted series reflect remainders as well as trend. Therefore they are not "smooth" and "downturns" or "upturns" can be misleading.Because of noises, it is having wiggles, so be careful. So it is better to use trend-cycle component rather than seasonally adjusted data.
- It is better to use the trend-cycle component to look for the turning points.

If the variation due to seasonality is not of primary interest, the seasonally adjusted series can be useful. For example, monthly unemployment data are usually seasonally adjusted in order to highlight variation due to the underlying state of the economy rather than the seasonal variation. An increase in unemployment due to school leavers seeking work is seasonal variation, while an increase in unemployment due to an economic recession is non-seasonal. Most economic analysts who study unemployment data are more interested in the non-seasonal variation. Consequently, employment data (and many other economic series) are usually seasonally adjusted.

Seasonally adjusted series contain the remainder component as well as the trend-cycle. Therefore, they are not "smooth", and "downturns" or "upturns" can be misleading. If the purpose is to look for turning points in a series, and interpret any changes in direction, then it is better to use the trend-cycle component rather than the seasonally adjusted data.

3.3 Moving averages

The traditional way of performing time series decomposition is called a *Classical Decomposition*. This method originated in 1920s but even today from some of the bases of decomposition methods actually used in practice. Hence it is important to understand how classical decomposition works.

The first step in the classical decomposition would be to actually get an estimate of the trend component and to get an estimate of the trend component, we use moving averages.

The simplest estimate of the trend-cycle uses *moving averages*.

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

where $k = \frac{m-1}{2}$

Here $\hat{T}_t$ is the estimate of the trend and is obtained by averaging values of the time series within k periods.

Example

Suppose we have to perform a moving average of order $m = 7$ then $k = \frac{7-1}{2} = 3$. Hence our estimate of the trend is $\hat{T}_t = \frac{1}{7}\Sigma_{j=-3}^{j=3} y_{t+j} = \frac{1}{7}(y_{t-3} + y_{t-2} + y_{t-1} + y_t + y_{t+1} + y_{t+2} + y_{t+3})$

Hence, our estimate of Trend at time "t" will be the average between three observations prior to time "t" and three observations post time "t".

Example

```
global_economy |> filter(Country == "Australia") |>
  autoplot(Exports) +
  labs(y= "% of GDP", title = "Total Australian Exports")
```

Now, suppose we have to implement moving average of order "5". So, we have to take the average of export values at 1960,1961,1962,1963,1964 and put the 5-MA value at 1962.

So, we get:

Year | Exports | 5-MA

1960 12.99
1961 12.40
1962 13.94 13.46
1963 13.01 13.50
1964 14.94 13.61

Now, we can plot it also.

As order of moving average increase, the trend estimate becomes smoother. As we are increasing the order of moving order, we are missing the trend estimates at both ends of our sample. So, for forecasting, more recent data is a big disadvantage.

Recap

So, a moving average is an *average of nearby points*

- observations nearby in time are also likely to be *close in value*
- average eliminates some *randomness* in the data, leaving a smooth trend-cycle component.

3-MA: $\hat{T}_t = (y_{t-1} + y_t + y_{t+1})/3$

5-MA: $\hat{T}_t = (y_{t-2} + y_{t-1} + y_t + y_{t+1} + y_{t+2})/5$

- Each average computed by dropping *oldest* observations and including *next* observation.
- averaging *moves* through time series until trend-cycle computed at each observation possible

Why is there no estimate at ends ?

- For a 3-MA, there cannot be estimates at time 1 or time T because the observations at time 0 and T+1 are not available.
- Generally, there cannot be estimates at times near the endpoints.

The order of the MA

- Larger order means smoother, flatter curve
- Larger order means more points lost at ends
- *order = length of season* or cycle removes seasonal pattern completely. So, if we have got daily data and we use the order=7 then it removes the seasonal pattern completely. So our trend estimate will be average across 7 days. Hence no seasonality is left over.
- But so for even orders ?

4 MA:

$\frac{1}{4}(y_{t-2} + y_{t-1} + y_t + y_{t+1})$ or

$\frac{1}{4}(y_{t-1} + y_t + y_{t+1} + y_{t+2})$

So, it's actually not centered.

Solution: Take a further 2-MA to "centre" result

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

Moving average of the same length as the season removes the seasonal pattern.

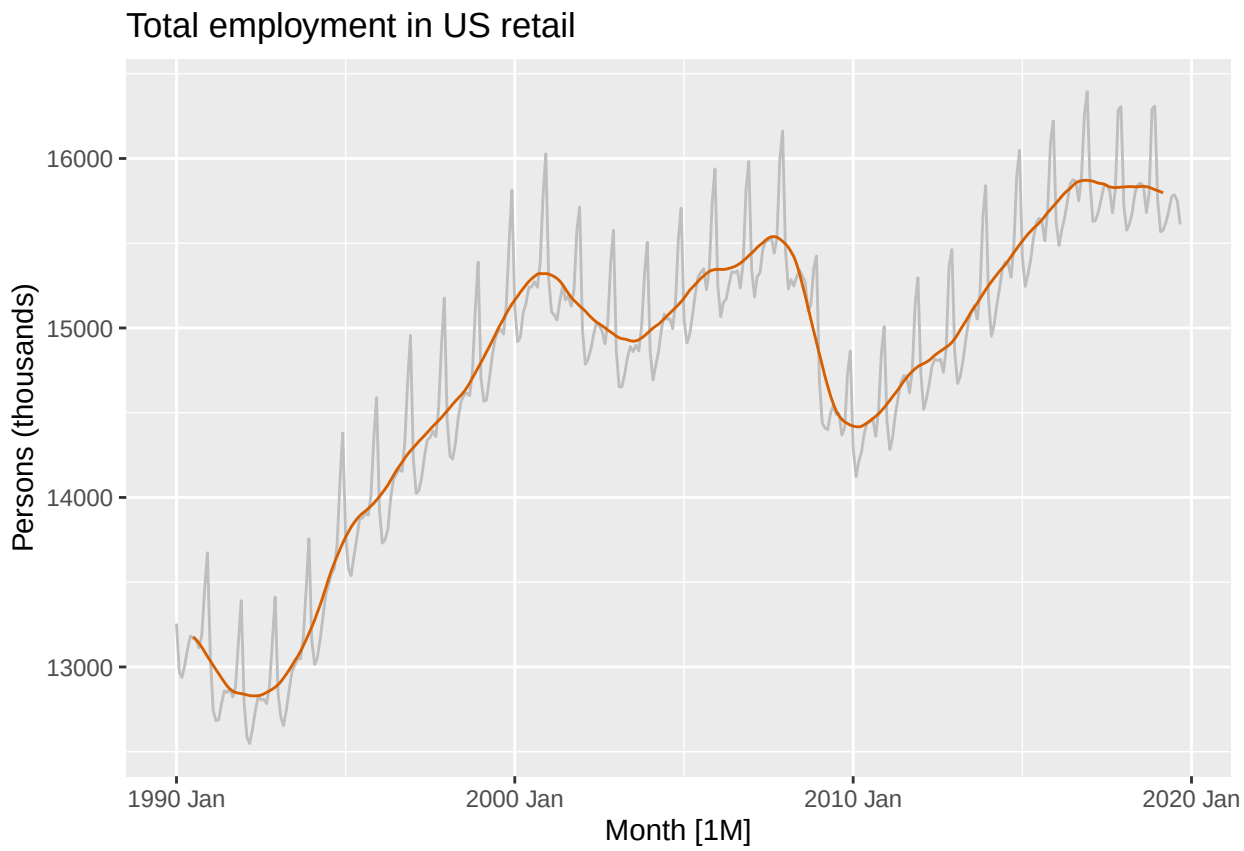
- For quarterly data: use a 2×4 MA
- For monthly data: use a 2×12 MA
  $\hat{T}_t = \frac{1}{24}y_{t-6} + \frac{1}{12}y_{t-5} + ... + \frac{1}{12}y_{t+5} + \frac{1}{24}y_{t+6}$

```
us_retail_employment_ma <- us_retail_employment |>
  mutate(
    `12-MA` = slider:: slide_dbl(Employed,mean,
            .before = 5, .after = 6,.complete = TRUE),
    `2*12-MA` = slider:: slide_dbl(`12-MA`,mean,
            .before = 1, .after = 0,.complete = TRUE)
  )
```

```
us_retail_employment_ma |>
  autoplot(Employed,color='gray') +
  autolayer(us_retail_employment_ma,vars(`2*12-MA`),
            color = '#D55E00')+
  labs(y="Persons (thousands)", title = "Total employment in US retail" )
```

Implementation

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```



The classical method of time series decomposition originated in the 1920s and was widely used until the 1950s. It still forms the basis of many time series decomposition methods, so it is important to understand how it works. Therefore, the average eliminates some of the randomness in the data, leaving a smooth trend-cycle component. We call this an m-MA, meaning a moving average of order m.

Notice that the trend-cycle (in orange) is smoother than the original data and captures the main movement of the time series without all of the minor fluctuations. The order of the moving average determines the smoothness of the trend-cycle estimate. In general, a larger order means a smoother curve.

It is possible to apply a moving average to a moving average. One reason for doing this is to make an even-order moving average symmetric.

When a 2-MA follows a moving average of an even order (such as 4), it is called a "centred moving average of order 4". This is because the results are now symmetric.

Other combinations of moving averages are also possible. For example, a 3×3-MA is often used, and consists of a moving average of order 3 followed by another moving average of order 3. In general, an even order MA should be followed by an even order MA to make it symmetric. Similarly, an odd order MA should be followed by an odd order MA.

3.4 Classical decomposition

Additive decomposition: $y_t = T_t + S_t + R_t = \hat{T}_t + \hat{S}_t + \hat{R}_t$

Multiplicative decomposition: $y_t = T_t \times S_t \times R_t = \hat{T}_t \times \hat{S}_t \times \hat{R}_t$

- Estimate $\hat{T}$ using $(2 \times m) - MA$ if m is even. Otherwise, estimate $\hat{T}$ using $m - MA$

Compute de-trended series

- Additive decomposition: $y_t - \hat{T}_t$
- Multiplicative decomposition: $y_t / \hat{T}_t$

De-trending Remove smoothed series $\hat{T}_t$ from y_t to leave S_t and R_t.

- Additive model: $y_t - \hat{T}_t = (\hat{T}_t + \hat{S}_t + \hat{R}_t) - \hat{T}_t = \hat{S}_t + \hat{R}_t$
- Multiplicative model: $y_t / \hat{T}_t = (\hat{T}_t \times \hat{S}_t \times \hat{R}_t)/\hat{T}_t = \hat{S}_t \times \hat{R}_t$

Estimating the Seasonal Component

- Seasonal index for each season is estimated as an *average* of the detrended series for that season of successive years.
- E.g. Take averages across all Januaries to get $S^{(1)}$ if your data is monthly. (Similarly if you have quarterly data, get average for each quarters)
- If necessary, adjust the seasonal indices so that:

–> For additive: $S^{(1)} + S^{(2)} + ... + S^{(12)} = 0$

–> For multiplicative: $S^{(1)} + S^{(2)} + ... + S^{(12)} = m$

- The seasonal component $\hat{S}$ simply consists of replications of the seasonal indices.

Remainder Component:

Additive decomposition: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$

Multiplicative decomposition: $\hat{R}_t = y_t / (\hat{T}_t \times \hat{S}_t)$

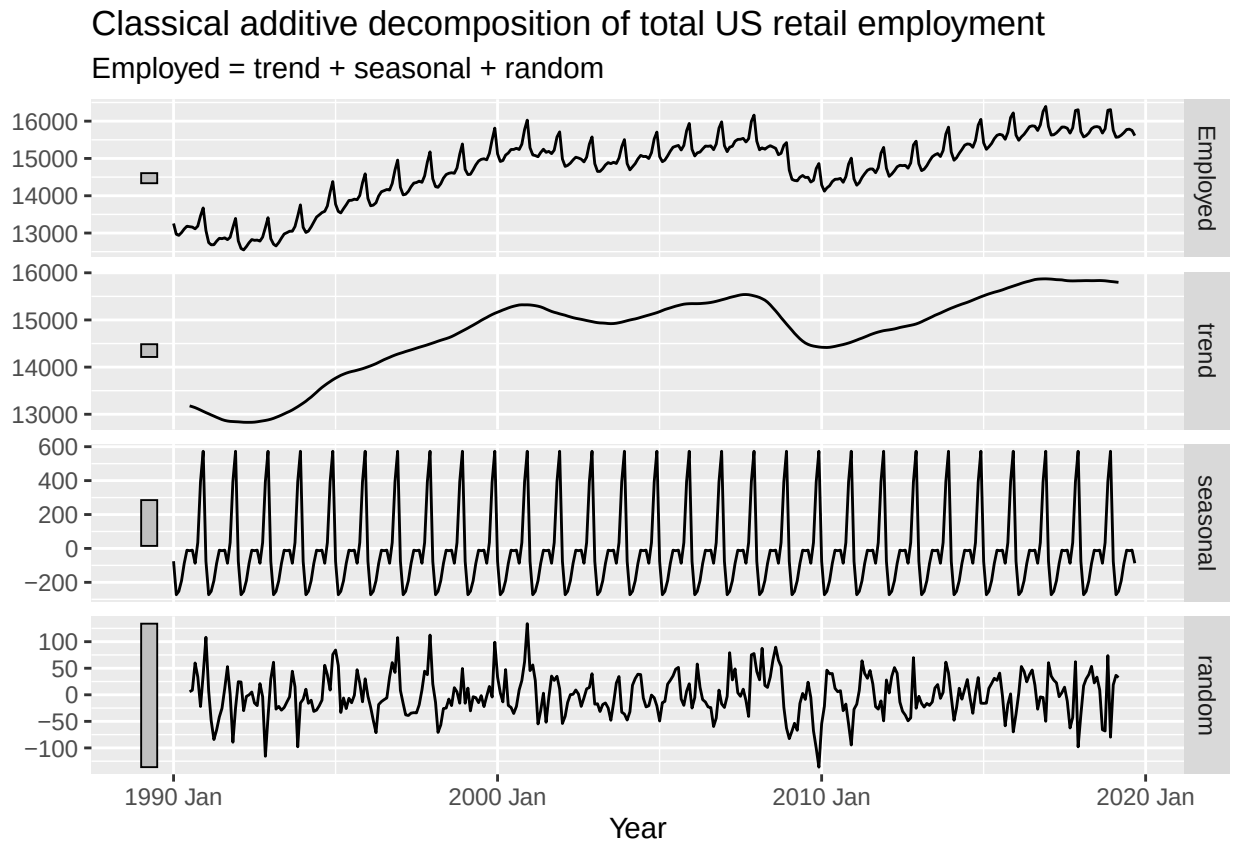
Classical decomposition:

- Choose additive or multiplicative depending on which gives the most stable components.
- For multiplicative model, this method of estimation is known as *ratio-to-moving-average method*.

Implementation:

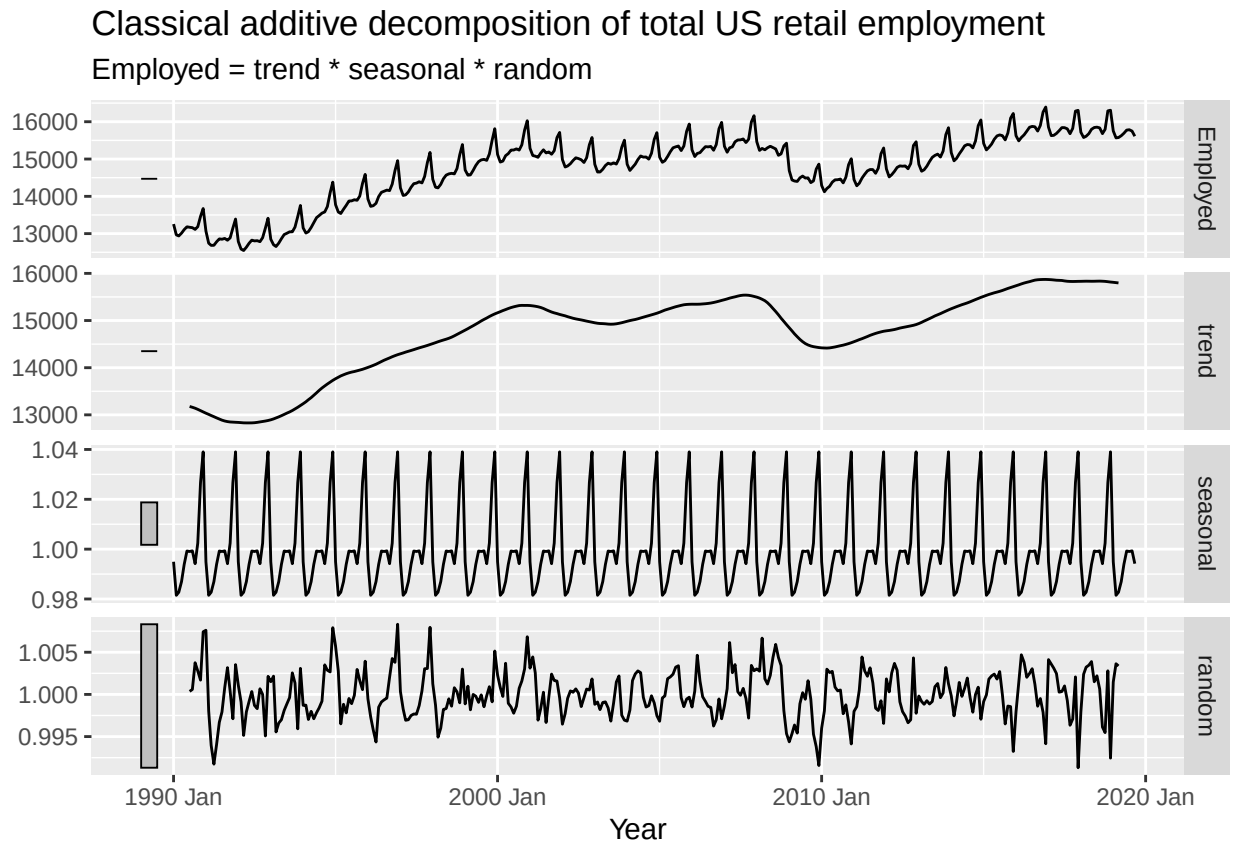
```
us_retail_employment |>
  model(classical_decomposition(Employed, type="additive")) |>
  components() |>
  autoplot() + xlab("Year") +
  ggtitle("Classical additive decomposition of total US retail employment")
```

```
## Warning: Removed 6 rows containing missing values (`geom_line()`).
```




```
us_retail_employment |>
  model(classical_decomposition(Employed, type="multiplicative")) |>
  components() |>
  autoplot() + xlab("Year") +
  ggtitle("Classical additive decomposition of total US retail employment")
```

```
## Warning: Removed 6 rows containing missing values (`geom_line()`).
```



Comments on Classical Decomposition:

- Estimate of trend is *unavailable* for first few and last few observations
- *Seasonal component repeats* from year to year. May not be realistic (example: electricity demand)
- *Not robust to outliers* because it relies on outliers and averages are very sensitive to outliers. A huge outlier will move that trend quite rapidly and significantly.
- Newer methods designed to overcome these problems.

The classical decomposition method originated in the 1920s. It is a relatively simple procedure, and forms the starting point for most other methods of time series decomposition. There are two forms of classical decomposition: an additive decomposition and a multiplicative decomposition. These are described below for a time series with seasonal period m (e.g.,$m = 4$ for quarterly data, $m = 12$ for monthly data, $m = 7$ for daily data with a weekly pattern).

In classical decomposition, we assume that the seasonal component is constant from year to year. For multiplicative seasonality, the m values that form the seasonal component are sometimes called the "seasonal indices".

Additive decomposition

Step 1

If m is an even number, compute the trend-cycle component $\hat{T}_t$ using a $2 \times m - MA$.

If m is an odd number, compute the trend-cycle component $\hat{T}_t$ using an $m - MA$

Step 2:

Calculate the detrended series: $y_t - \hat{T}_t$.

Step 3:

To estimate the seasonal component for each season, simply average the detrended values for that season. For example, with monthly data, the seasonal component for March is the average of all the detrended March values in the data. These seasonal component values are then adjusted to ensure that they add to zero. The seasonal component is obtained by stringing together these monthly values, and then replicating the sequence for each year of data. This gives $\hat{S}_t$

Step 4:

The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components: $\overset{R}{t} = y_t - \hat{T}_t - \hat{S}_t$

(For multiplicative decomposition, change, - to /).

Comments on classical decomposition

While classical decomposition is still widely used, it is not recommended, as there are now several much better methods. Some of the problems with classical decomposition are summarised below.

The estimate of the trend-cycle is unavailable for the first few and last few observations. For example, if m=12 , there is no trend-cycle estimate for the first six or the last six observations. Consequently, there is also no estimate of the remainder component for the same time periods.

The trend-cycle estimate tends to over-smooth rapid rises and falls in the data. Classical decomposition methods assume that the seasonal component repeats from year to year. For many series, this is a reasonable assumption, but for some longer series it is not. For example, electricity demand patterns have changed over time as air conditioning has become more widespread. In many locations, the seasonal usage pattern from several decades ago had its maximum demand in winter (due to heating), while the current seasonal pattern has its maximum demand in summer (due to air conditioning). Classical decomposition methods are unable to capture these seasonal changes over time. Occasionally, the values of the time series in a small number of periods may be particularly unusual. For example, the monthly air passenger traffic may be affected by an industrial dispute, making the traffic during the dispute different from usual. The classical method is not robust to these kinds of unusual values.

3.5 Methods used by official statistics agencies

Time series decomposition is commonly used in official statistics bureaus. Here we will talk about the methods that they use in these sort of organisations.

History of Time series decomposition

- Time series decomposition began back in 1920s using the classical decomposition methods.

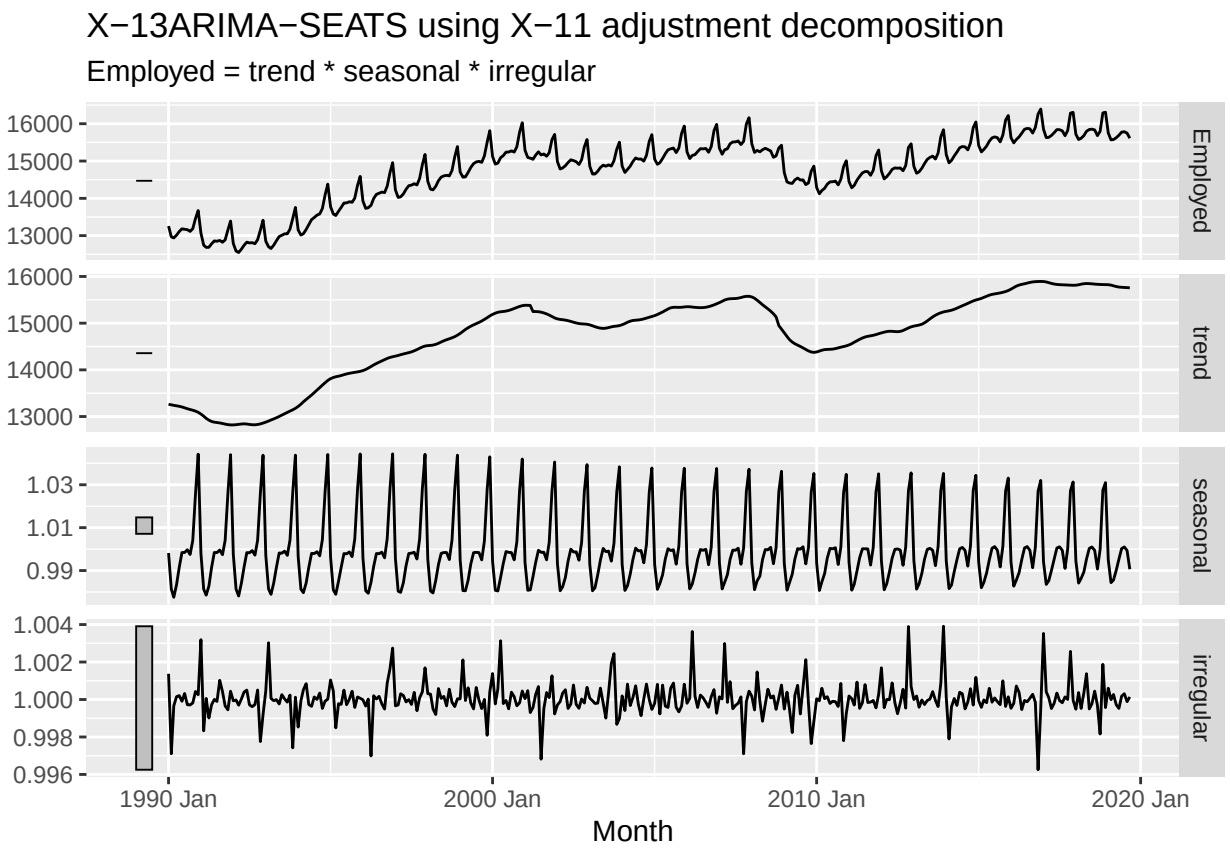
- Then in 1957s, US census bureau developed a new most sophasticated and robust approach and they called it as census-2 and that became the basis for the X-11 method and its variants (including X-12 ARIMA and X-13 ARIMA)
- STL method introduced in 1983 at Bell labs
- Another collection of methods are developed in Europe known as TRAMO and SEATS in 1990s.

National Statistics Offices

- Australian Bureau Statistics (ABS) uses X-12 ARIMA
- US Census Bureau uses X-13 ARIMA-SEATS
- Statistics Canada uses X-12 ARIMA
- ONS(UK) uses X-12 ARIMA
- EuroStat use X-13 ARIMA-SEATS.

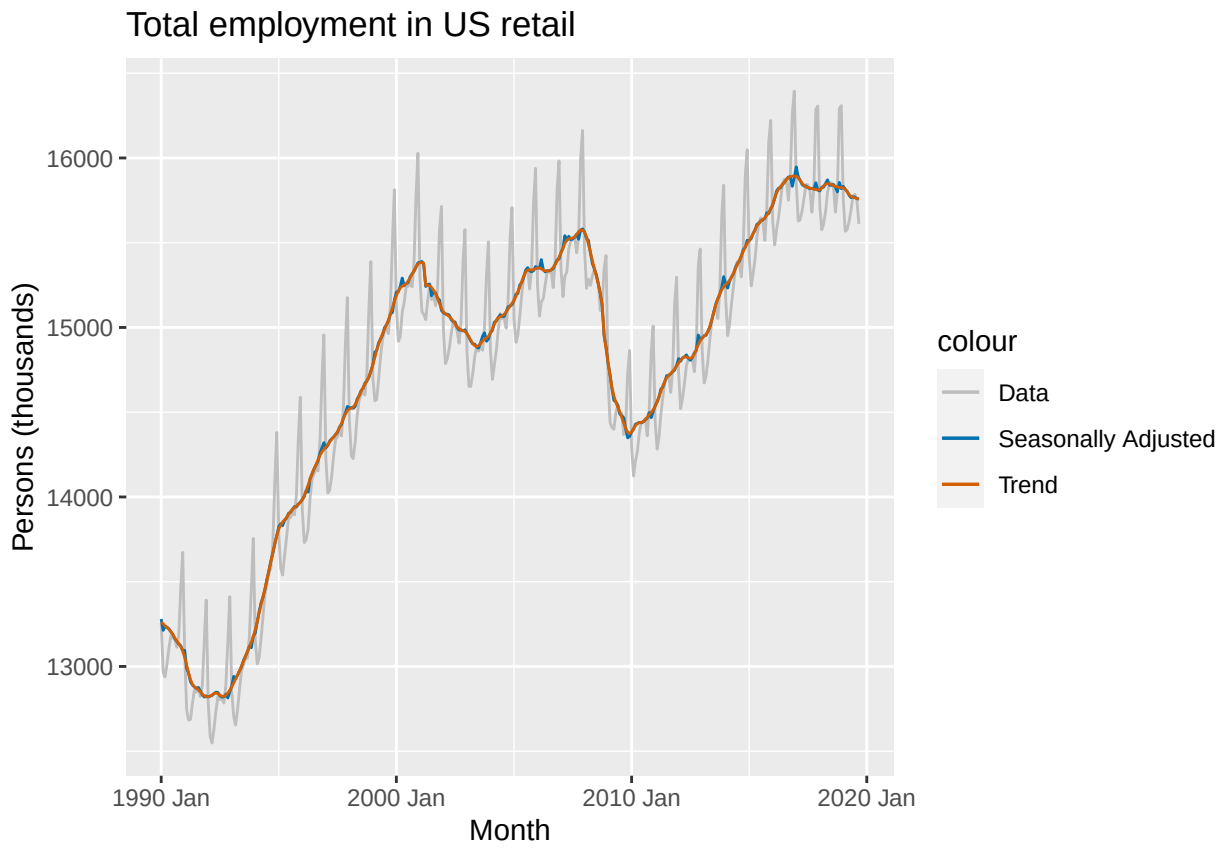
X-11 decomposition

```
x11_dcmp <- us_retail_employment |>
  model(x11 = X_13ARIMA_SEATS(Employed ~ x11())) |>
  components()
autoplot(x11_dcmp)
```



As we can see it is a multiplicative decomposition whereas STL is always an additive decomposition.

```
x11_dcmp |>
  ggplot(aes(x = Month)) +
  geom_line(aes(y = Employed, colour = "Data")) +
  geom_line(aes(y = season_adjust,
                colour = "Seasonally Adjusted")) +
  geom_line(aes(y = trend, colour = "Trend")) +
  labs(y = "Persons (thousands)",
       title = "Total employment in US retail") +
  scale_colour_manual(
    values = c("gray", "#0072B2", "#D55E00"),
    breaks = c("Data", "Seasonally Adjusted", "Trend")
  )
```



"SEATS" stands for "Seasonal Extraction in ARIMA Time Series".

Advantages of X-11 decomposition

- It is quite robust to outliers
- Completely automated choices for trend and seasonal changes
- Very widely tested on economic data over a long period of time

Disadvantages of X-11 decomposition

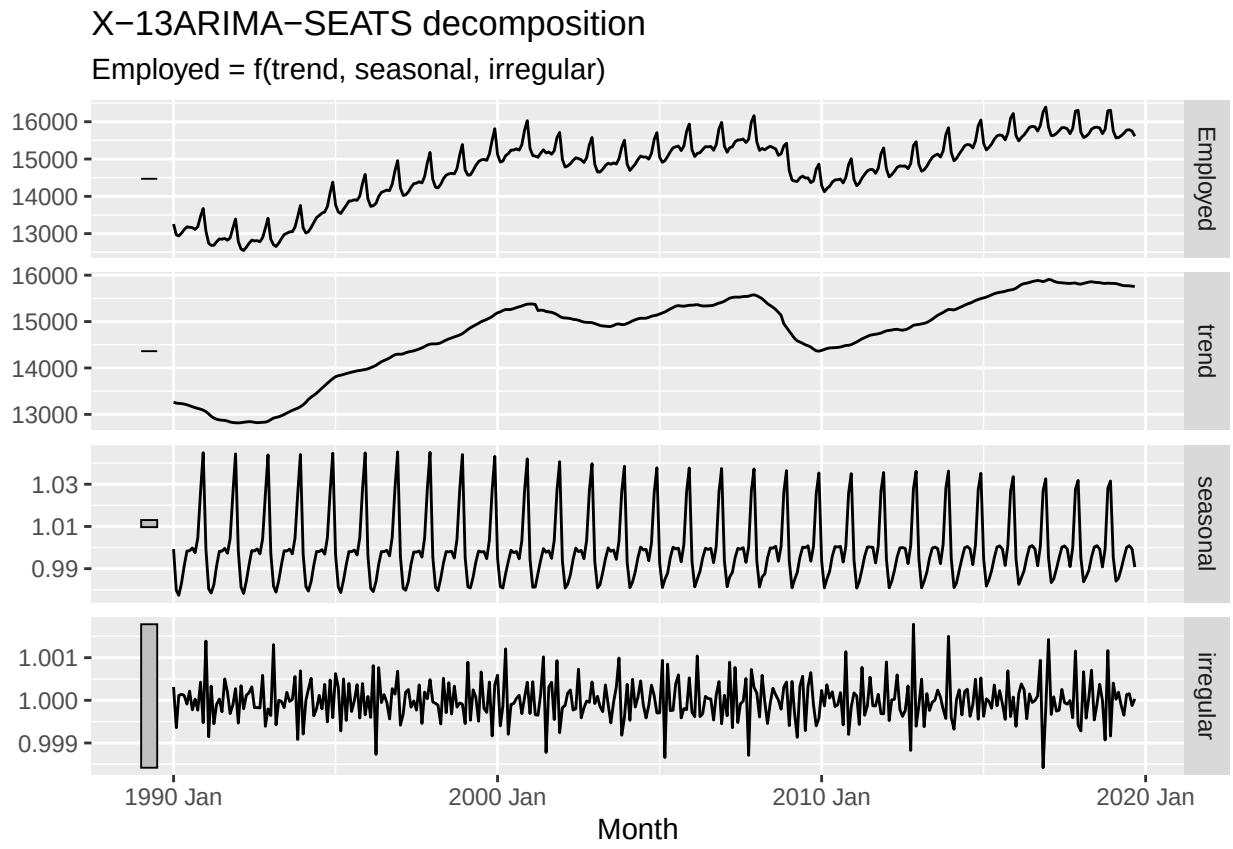
- No prediction/confidence intervals
- Ad hoc method with no underlying model
- Only developed for quarterly and monthly data

Extensions: X-12 ARIMA and X-13 ARIMA

- The X-11, X-12-ARIMA and X-13-ARIMA methods are based on Census-2 decomposition
- They allow adjustments for trading days and other explanatory variables
- Known outliers can be omitted
- Level shifts and ramp effects can be modelled
- Missing values estimated and replaced
- Holiday factors (e.g. Easter, Labour day) can be estimated

X-13 ARIMA-SEATS

```
seats_dcmp <- us_retail_employment |>
  model(seats= X_13ARIMA_SEATS(Employed ~ seats())) |>
  components()
autoplot(seats_dcmp)
```



It is also multiplicative decomposition and it is a model based approach which means we can get the confidence interval for the trend or seasonality.

Advantages of X13-ARIMA-SEATS

- Model-based
- Smooth trend estimate
- Allows estimates for the end points
- Allows changing seasonality
- Developed for economic data

Disadvantages

- Only developed for quarterly and monthly data

3.6 STL decomposition

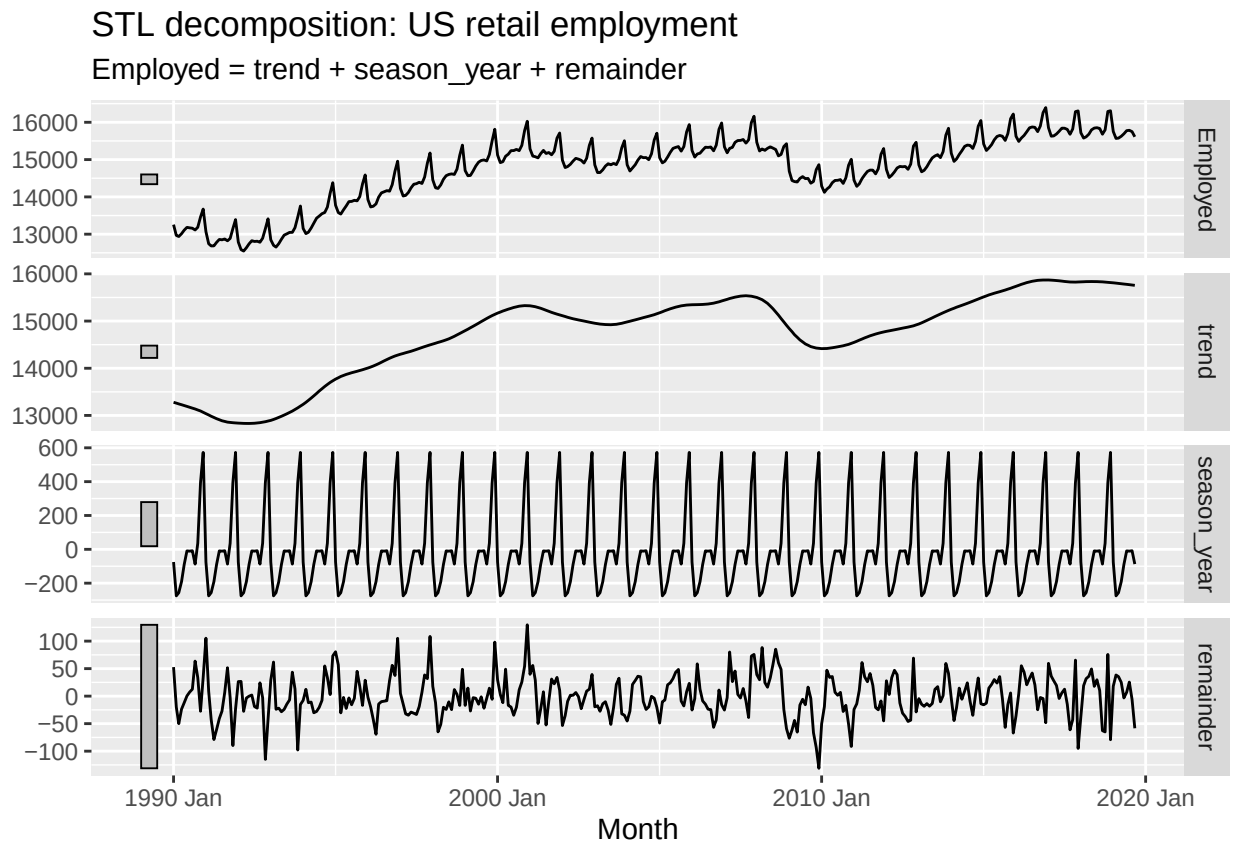
- STL stands for *Seasonal and Trend decomposition using Loess*. Loess is a form of smoothing based on locally linear regressions.
- It is a very versatile and robust tool.

- Unlike X-12 ARIMA, STL will handle any type of seasonality not just monthly or quarterly data.
- Seasonal components are allowed to change over time and rate of change controlled by user i.e. how quickly it can be changed over time.
- Smoothness of trend-cycle can also be controlled by user
- Robust to outliers.
- There are something which X12 ARIMA and SEATS can do and STL can't do including trading day and calender adjustments or handling exogenous variables.
- STL has only additive method and not multiplicative method.
- Take logs to get multiplicative decomposition and then fitting additive decompositions.
- Use Box-Cox transformations to get other decompositions which is between additive and multiplicative decompositions.

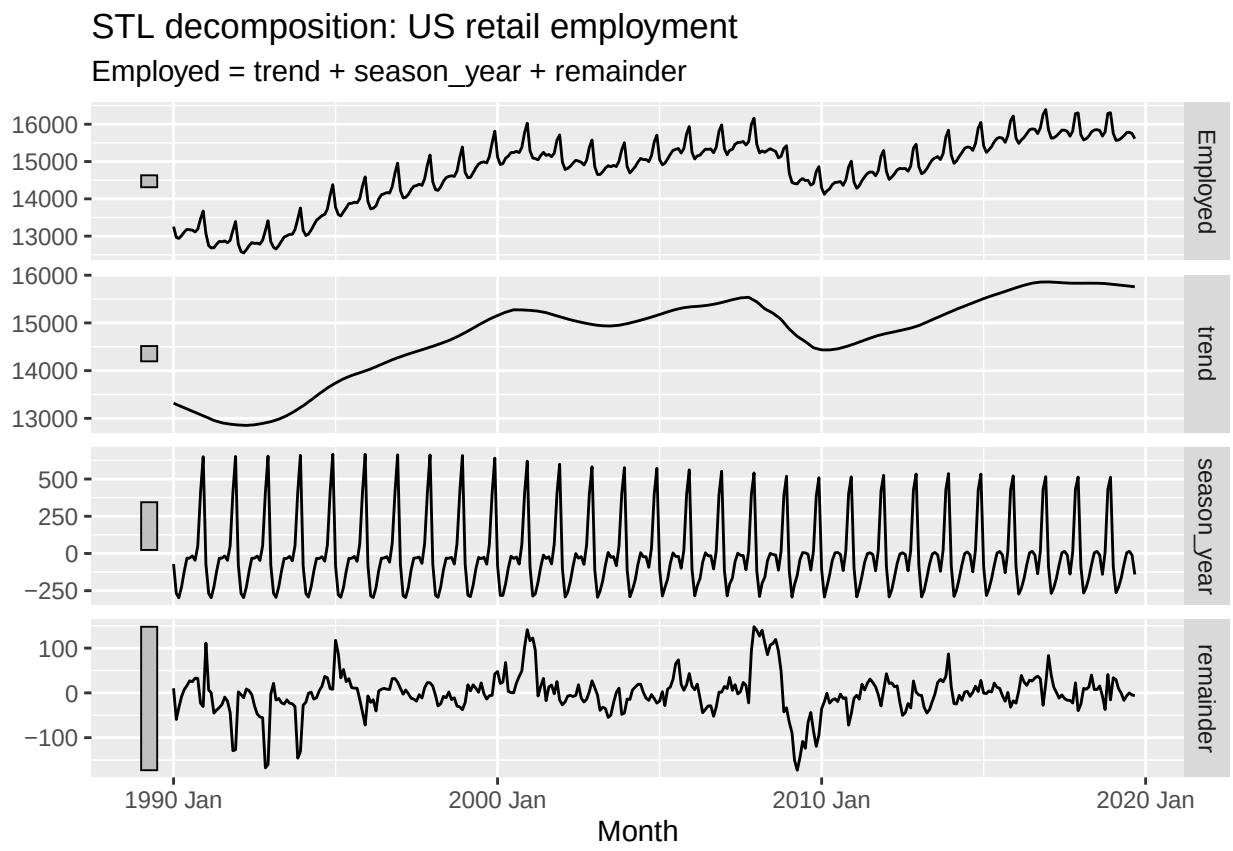
Example

When we don't want to change seasonal patterns over time, we plot it as:

```
us_retail_employment |>
  model(STL(Employed ~ season(window="periodic"))) |>
  components() |>
  autoplot() + labs(title = "STL decomposition: US retail employment")
```

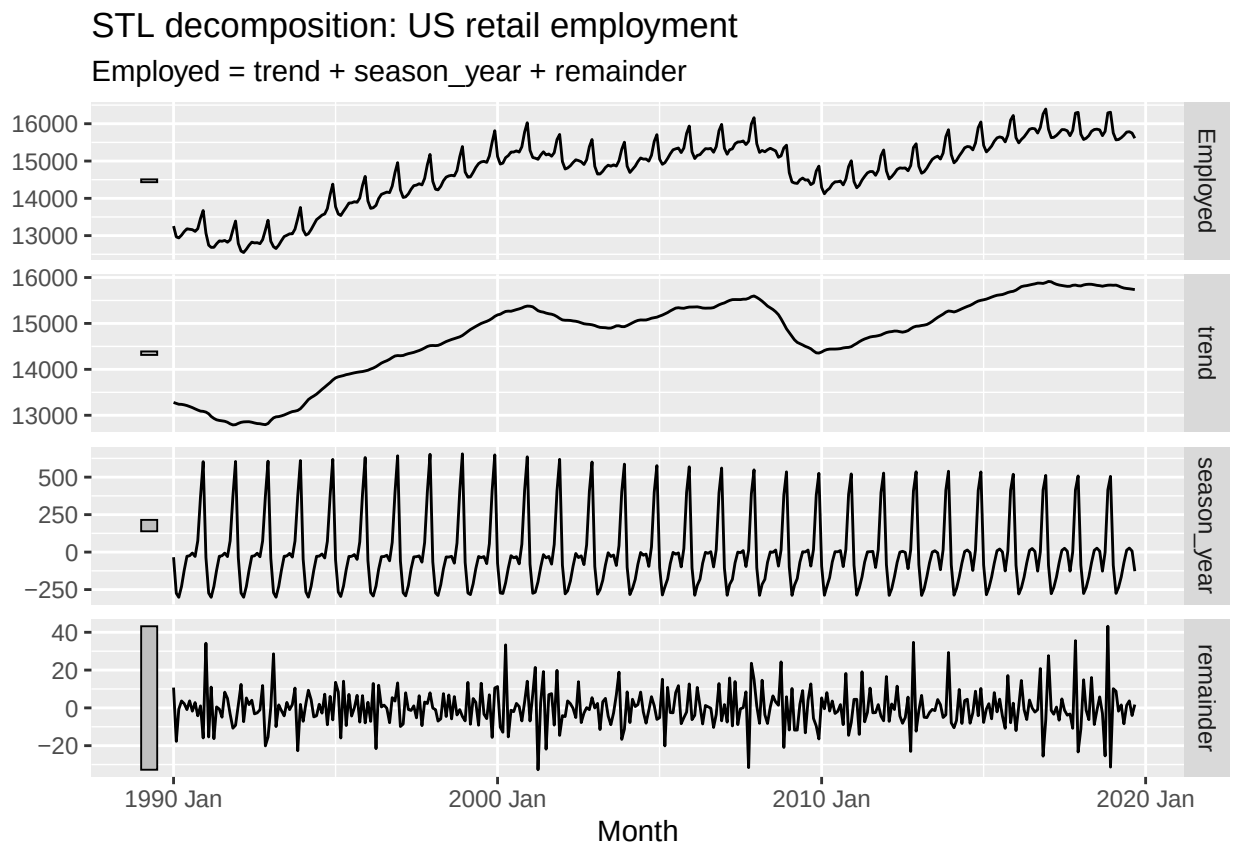


```
us_retail_employment |>
  model(STL(Employed ~ season(window=9),robust=TRUE)) |>
  components() |>
  autoplot() + labs(title = "STL decomposition: US retail employment")
```



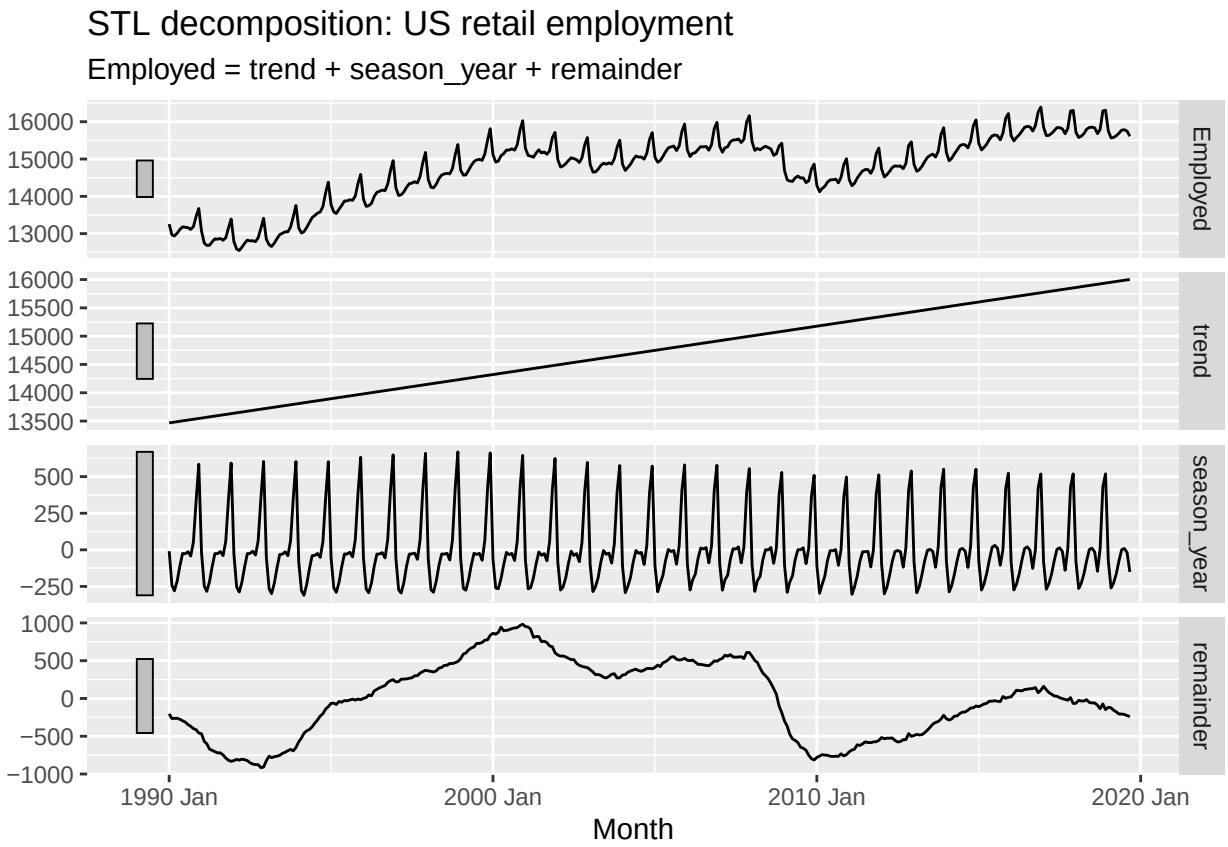
Here, window = 9 means, it averages the seasonal patterns to form the moving average of seasonal components, it's going to use a window of 9 years but due to whightening, we get the small variations.

```
us_retail_employment |>
  model(STL(Employed ~ season(window=9)+ trend(window=5))) |>
  components() |>
  autoplot() + labs(title = "STL decomposition: US retail employment")
```



Here, we make the trend more wiggly from the smooth trend which we got previously.

```
us_retail_employment |>
  model(STL(Employed ~ season(window=9)+ trend(window=6665))) |>
  components() |>
  autoplot() + labs(title = "STL decomposition: US retail employment")
```



Now trend becomes a straight line and so remainder component changed now.

"Robust" attribute is not always desirable, sometimes we need it and sometimes not. Its default value is FALSE.

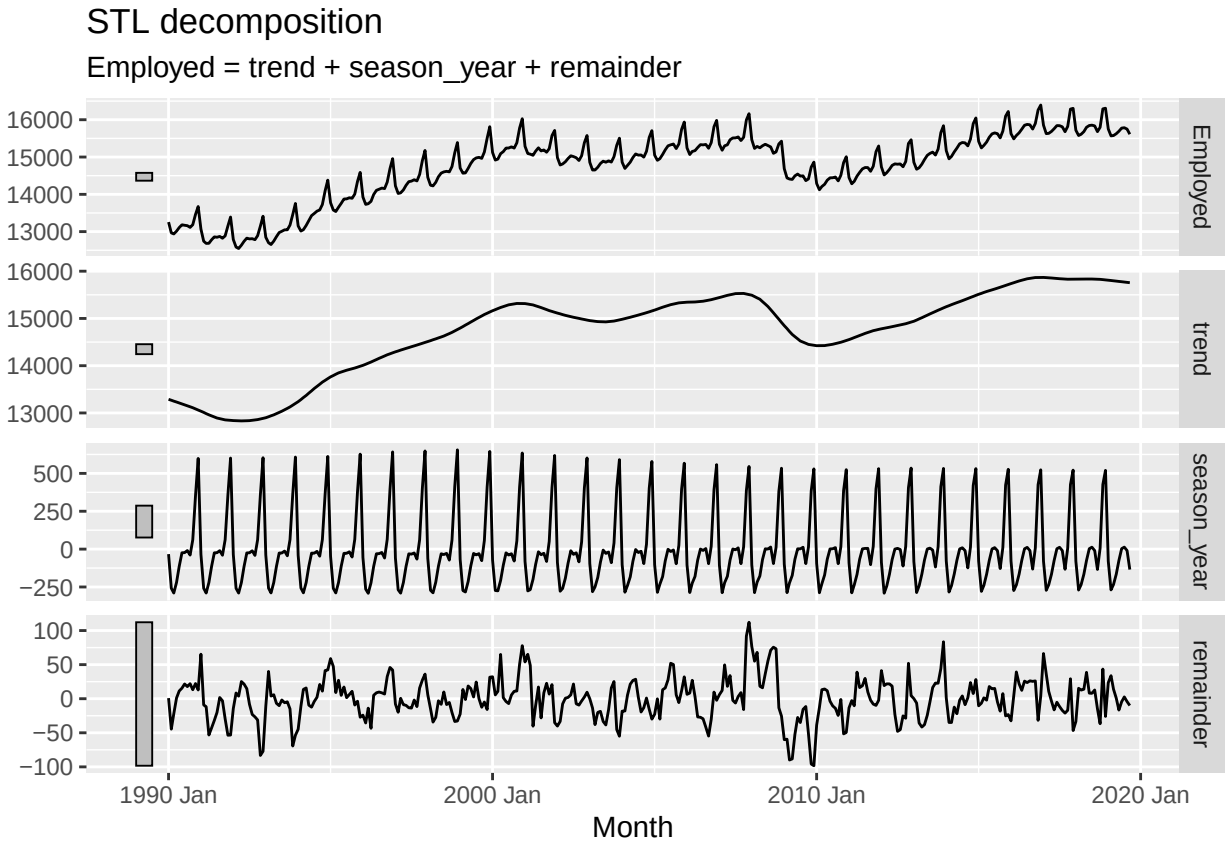
Summary

trend(window = ?) controls the wiggliness of the trend component. Because it is using low value so it uses the weighted least square fit. when the value of window is low, it uses the fewer observations to fit at the local linear line for the trend component.

season(window = ?) controls variation on seasonal component. Here window value tells that how many years we are using to estimate the seasonal component.

trend(window = 'periodic') is equivalent to infinite window. Here seasonal patterns does not change over time.

```
us_retail_employment |>
  model(STL(Employed)) |>
  components() |>
  autoplot()
```



Default above plot is good. STL() chooses season(window=13) by default.

It can include transformations.

- Default Trend value is given by the expression

window = nextodd(ceiling((1.5*period)/(1-(1.5/seasonal_window))))

How it works:

- Algorithm that updates trend and seasonal components iteratively.
- Starts with $\hat{T}_t = 0$ i.e. it starts with no trend and estimate the seasonality something like in classical decomposition
- After estimating the seasonal patterns, it removes that and tries to estimate the trend using a loess curve i.e. locally linear curve and then it removes that and then tries to do seasonal term again and iterates between these two successively and getting more accurate and better refined estimates of both Trend and seasonal components. So, it uses a mixture of loess and moving averages to successively refine the trend and seasonal estimates.
- The trend window controls loess bandwidth applied to deseasonalised value.
- The season window controls loess bandwidth applied to detrended subseries.
- Robustness weights based on remainder

STL is a versatile and robust method for decomposing time series. STL is an acronym for "Seasonal and Trend decomposition using Loess", while loess is a method for estimating nonlinear relationships. The STL method was developed by R. B. Cleveland et al. (1990).

A multiplicative decomposition can be obtained by first taking logs of the data, then back-transforming the components. Decompositions that are between additive and multiplicative can be obtained using a Box-Cox transformation of the data with
$0< <1$. A value of $=0$ gives a multiplicative decomposition while $=1$ gives an additive decomposition.

The control how rapidly the trend-cycle and seasonal components can change. Smaller values allow for more rapid changes. Both trend and seasonal windows should be odd numbers; trend window is the number of consecutive observations to be used when estimating the trend-cycle; season window is the number of consecutive years to be used in estimating each value in the seasonal component. Setting the seasonal window to be infinite is equivalent to forcing the seasonal component to be periodic.

. The default setting for monthly data is trend(window=21). This usually gives a good balance between overfitting the seasonality and allowing it to slowly change over time. But, as with any automated procedure, the default settings will need adjusting for some time series. In this case the default trend window setting produces a trend-cycle component that is too rigid. As a result, signal from the 2008 global financial crisis has leaked into the remainder component, as can be seen in the bottom panel of Figure 3.7. Selecting a shorter trend window as in Figure 3.18 improves this.