Chapter 3 Time series decomposition

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.

In Section 2.3 we discussed three types of time series patterns: trend, seasonality and cycles. 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.

When decomposing a time series, it is sometimes helpful to first transform or adjust the series in order to make the decomposition (and later analysis) as simple as possible. So we will begin by discussing transformations and adjustments.

3.1 Transformations and adjustments

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.

This can be seen in the `global_economy` dataset, where a common transformation of GDP is GDP per-capita.

```python
global_economy = pd.read_csv("../data/global_economy.csv")
df = global_economy.query('unique_id == "Australia"')[
    ["unique_id", "ds", "GDP", "Population"]
].assign(y=lambda x: x["GDP"] / x["Population"])

plot_series(df, xlabel="Year [1Y]", ylabel="$US", title="GDP per capita")
```

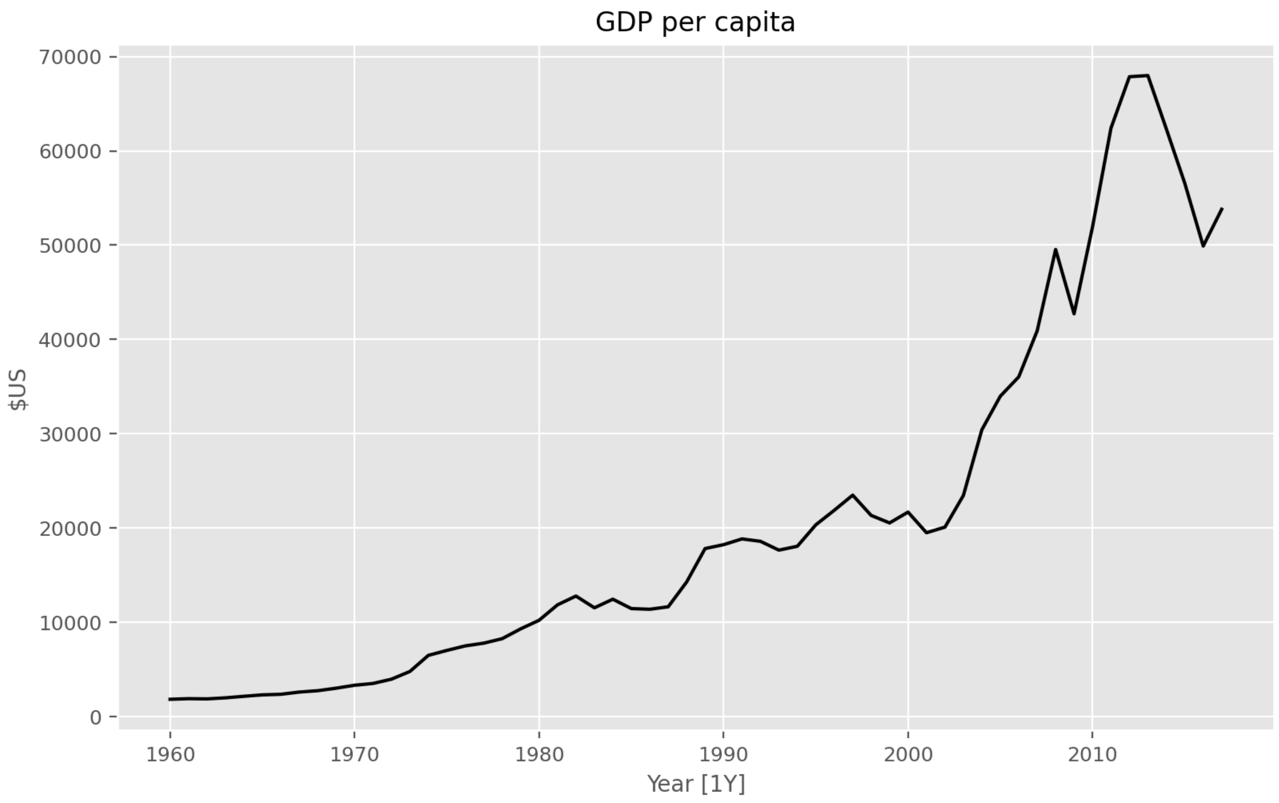


Figure 3.1: Australian GDP per capita.

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).

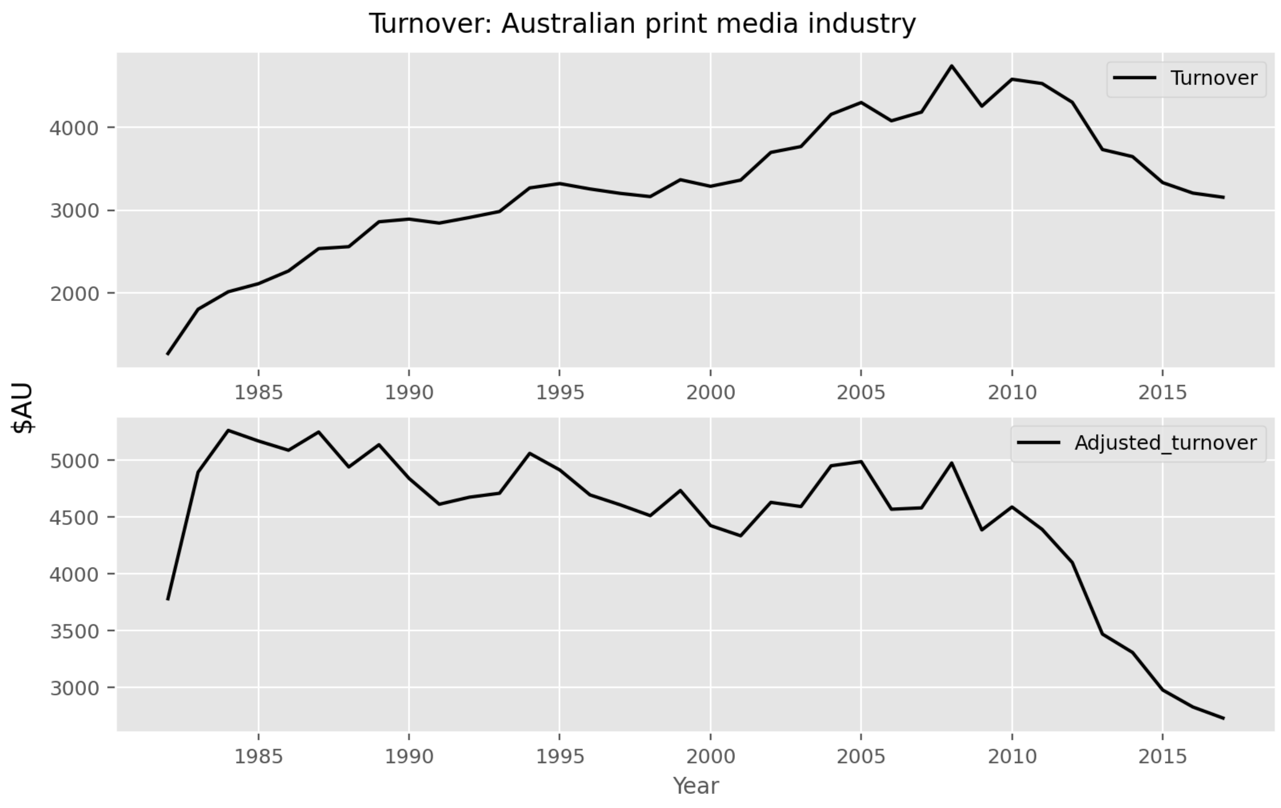
This allows us to compare the growth or decline of industries relative to a common price value. For example, looking at aggregate annual "newspaper and book" retail turnover from `aus_retail`, and adjusting the data for inflation using CPI from `global_economy` allows us to understand the changes over time.

```python
aus_retail = pd.read_csv("../data/aus_retail.csv", parse_dates=["Month"])
print_retail = (
    aus_retail.query('Industry == "Newspaper and book retailing"')
    .groupby(["Month"], as_index=False)["Turnover"]
    .sum()
)
print_retail["ds"] = print_retail["Month"].dt.year
print_retail = print_retail.groupby("ds")["Turnover"].sum().reset_index()

aus_economy = global_economy.query('unique_id == "Australia"')

df = pd.merge(aus_economy, print_retail, on="ds", how="left")
df["Adjusted_turnover"] = (df["Turnover"] / df["CPI"]) * 100
df.dropna(inplace=True)

fig, axes = plt.subplots(nrows=2, ncols=1)
sns.lineplot(data=df, x='ds', y='Turnover', ax=axes[0], color='black',
    label="Turnover")
sns.lineplot(data=df, x='ds', y='Adjusted_turnover', ax=axes[1],
    color='black', label="Adjusted_turnover")
axes[0].set_xlabel('')
axes[1].set_xlabel("Year")
axes[0].set_ylabel("")
axes[1].set_ylabel("")
fig.suptitle("Turnover: Australian print media industry")
fig.supylabel("$AU")
plt.show()
```



By adjusting for inflation using the CPI, we can see that Australia's newspaper and book retailing industry has been in decline much longer than the original data suggests. The adjusted turnover is in 2010 Australian dollars, as CPI is 100 in 2010 in this data set.

Mathematical transformations

If the data shows variation that increases or decreases with the level of the series, then a transformation can be useful. For example, a logarithmic transformation is often useful. If we denote the original observations as y_{1},\dots,y_{T} and the transformed observations as w_{1}, \dots, w_{T}, then w_t = \log(y_t). Logarithms are useful because they are interpretable:

changes in a log value are relative (or percentage) changes on the original scale. So if log base 10 is used, then an increase of 1 on the log scale corresponds to a multiplication of 10 on the original scale. If any value of the original series is zero or negative, then logarithms are not possible.

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 and Cox 1964), which depend on the parameter \lambda and are defined as follows: w_t = \begin{cases} \log(y_t) & \text{if $\lambda=0$}; \\ (\text{sign}(y_t)|y_t|^\lambda-1)/\lambda & \text{otherwise}. \end{cases} \tag{3.1} This is actually a modified Box-Cox transformation, discussed in Bickel and Doksum (1981), which allows for negative values of y_t provided \lambda > 0.

The logarithm in a Box-Cox transformation is always a natural logarithm (i.e., to base e). So if \lambda=0, natural logarithms are used, but if \lambda\ne0, a power transformation is used, followed by some simple scaling.

If \lambda=1, then w_t = y_t-1, so the transformed data is shifted downwards but there is no change in the shape of the time series. For all other values of \lambda, the time series will change shape.

A good value of \lambda is one which makes the size of the seasonal variation about the same across the whole series, as that makes the forecasting model simpler.

To determine a \lambda for the Box-Cox transformation, use the `boxcox_lambda` function from `CoreForecast`. This function requieres the `season_length` of the series and uses the method by Guerrero (1993) to choose a value for \lambda.

You can apply the Box-Cox transformation directly to the data using the `boxcox` function from `CoreForecast`. This function accepts any \lambda you provide, including the optimal value determined by `boxcox_lambda`.

```python
aus_production = pd.read_csv("../data/aus_production.csv",
    parse_dates=["ds"])
df = aus_production[["ds", "Gas"]].copy()
df.insert(0, "unique_id", "Gas")
df.rename(columns={"Gas": "y"}, inplace=True)

optim_lambda = boxcox_lambda(df["y"].to_numpy(), method="guerrero",
    season_length=4)
y_transformed = boxcox(df["y"].to_numpy(), optim_lambda)

df["y_transformed"] = y_transformed
rounded_lambda = round(optim_lambda, 2)
plot_series(df, target_col="y_transformed",
    xlabel="Quarter [1Q]", ylabel="",
    title=f"Transformed gas production with $\\lambda$ = {rounded_lambda}")
```

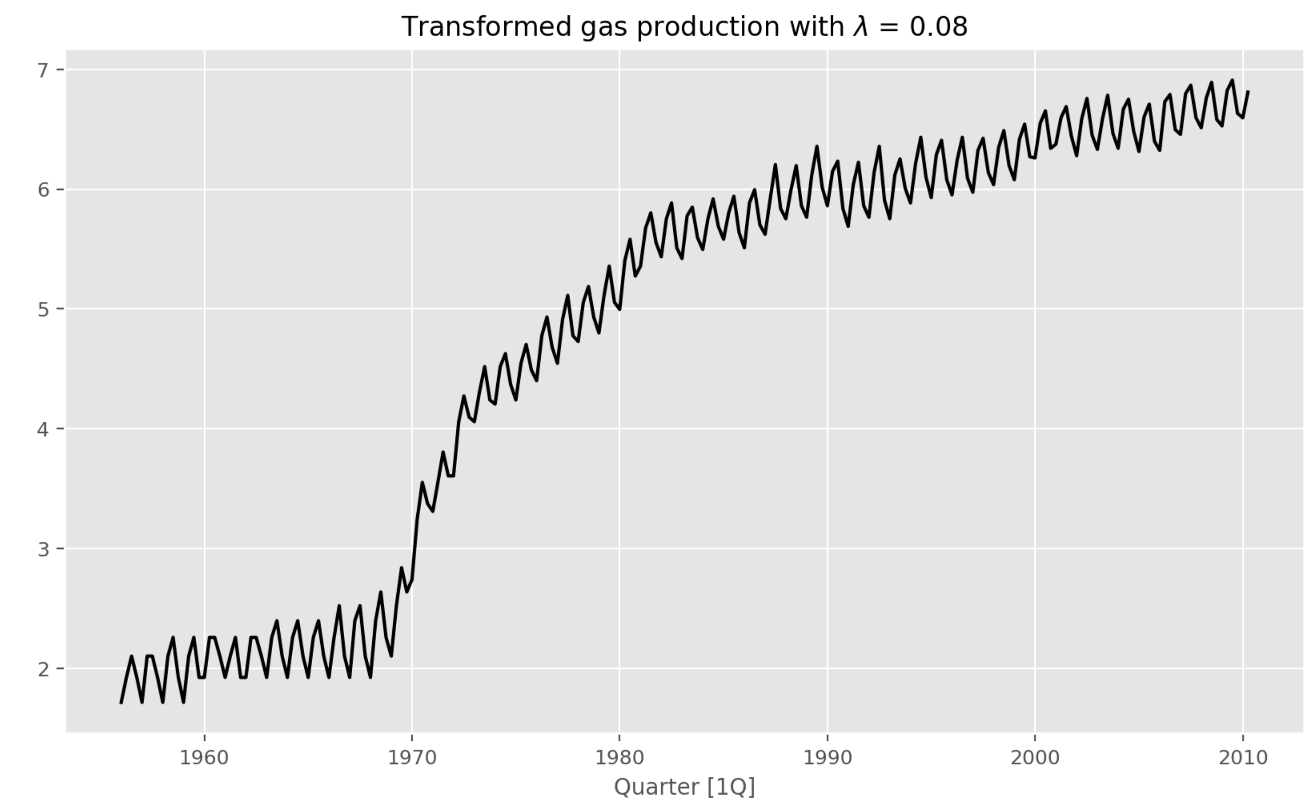


Figure 3.2: Transformed Australian gas production with \lambda chosen using the `boxcox_lambda` function from `CoreForecast`.

3.2 Time series components

If we assume an additive decomposition, then we can write y_{t} = S_{t} + T_{t} + R_t, where y_{t} is the data, S_{t} is the seasonal component, T_{t} is the trend-cycle component, and R_t is the remainder component, all at period t. Alternatively, a multiplicative decomposition would be written as y_{t} = S_{t} \times T_{t} \times R_t.

The additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative decomposition is more appropriate. Multiplicative decompositions are common with economic time series.

An alternative to using a multiplicative decomposition is to first transform the data until the variation in the series appears to be stable over time, then use an additive decomposition. When a log transformation has been used, this is equivalent to using a multiplicative decomposition on the original data because y_{t} = S_{t} \times T_{t} \times R_t \quad\text{is equivalent to}\quad \log y_{t} = \log S_{t} + \log T_{t} + \log R_t.

Example: Employment in the US retail sector

We will look at several methods for obtaining the components S_{t}, T_{t} and R_{t} later in this chapter, but first it is helpful to see an example. We will decompose the number of persons employed in retail as shown in Figure 3.3. The data shows the total monthly number of persons in thousands employed in the retail sector across the US since 1990.

```
us_employment = pd.read_csv("../data/us_employment.csv", parse_dates=["ds"])
us_retail_employment = us_employment.query(
    '(unique_id == "Retail Trade") & (ds >= "1990-01-01")'
)

plot_series(us_retail_employment,
            xlabel="Month [1M]",
            ylabel="Persons (thousands)",
            title="Total employment in US retail")
```

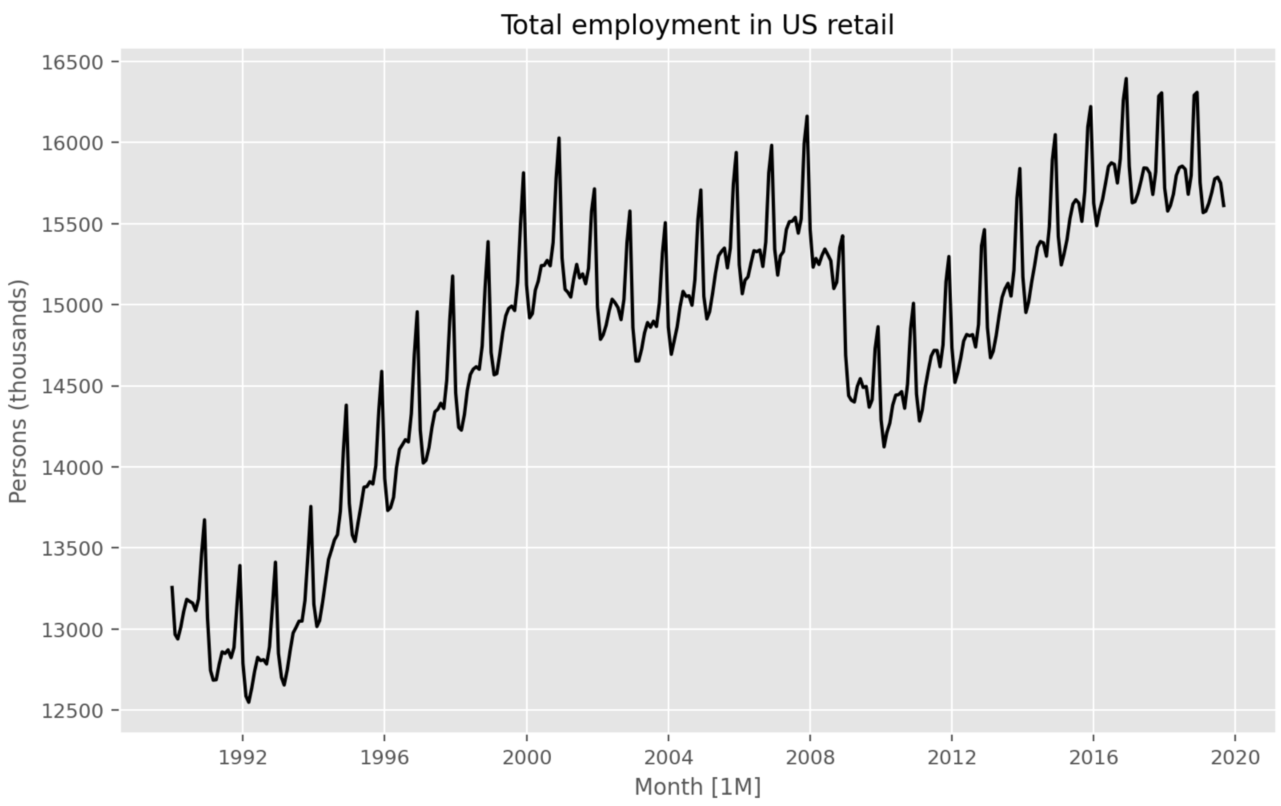


Figure 3.3: Total number of persons employed in US retail.

To illustrate the ideas, we will use the STL decomposition method, which is discussed in Section 3.6.

```
stl = STL(us_retail_employment["y"], period=12)
res = stl.fit()

dcmp = pd.DataFrame({
    "ds": us_retail_employment["ds"],
    "data": us_retail_employment["y"],
    "trend": res.trend,
    "seasonal": res.seasonal,
    "remainder": res.resid
}).reset_index(drop=True)

dcmp.head()
```

	ds	data	trend	seasonal	remainder
0	1990-01-01	13255.8	13296.249	-3.700	-36.749
1	1990-02-01	12966.3	13276.085	-288.398	-21.387
2	1990-03-01	12938.2	13255.663	-306.658	-10.805
3	1990-04-01	13012.3	13234.986	-235.775	13.089
4	1990-05-01	13108.3	13214.071	-115.399	9.628

The output above shows the components of an STL decomposition. The original data is shown (as `data`), followed by the estimated components.

The `trend` column (containing the trend-cycle T_t) follows the overall movement of the series, ignoring any seasonality and random fluctuations, as shown in Figure 3.4.

```
fig, ax = plt.subplots()
sns.lineplot(data=dcmp, x="ds", y="data", label="Data", color="gray")
sns.lineplot(data=dcmp, x="ds", y="trend", label="Trend", color="#D55E00")
ax.set_title("Total employment in US retail")
ax.set_xlabel("Month [1M]")
ax.set_ylabel("Persons (thousands)")
plt.show()
```

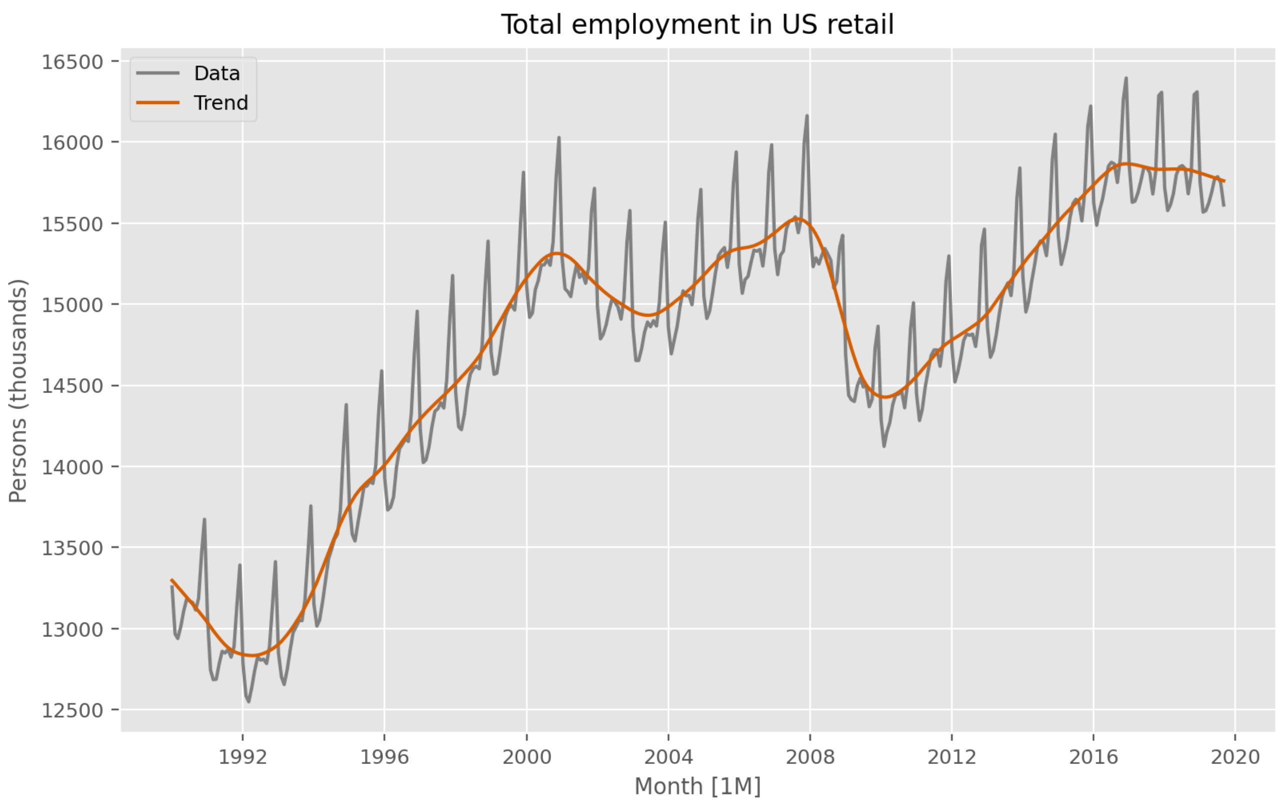


Figure 3.4: Total number of persons employed in US retail: the trend-cycle component (orange) and the raw data (grey).

We can plot all of the components of the decomposition, as shown in Figure 3.5.

```
fig, axes = plt.subplots(nrows=4, ncols=1, sharex=True, figsize=(8, 6))
sns.lineplot(data=dcmp, x=dcmp.index, y="data", ax=axes[0])
sns.lineplot(data=dcmp, x=dcmp.index, y="trend", ax=axes[1])
sns.lineplot(data=dcmp, x=dcmp.index, y="seasonal", ax=axes[2])
sns.lineplot(data=dcmp, x=dcmp.index, y="remainder", ax=axes[3])
axes[0].set_ylabel("Employed")
axes[1].set_ylabel("trend")
axes[2].set_ylabel("seasonal")
axes[3].set_ylabel("remainder")
fig.suptitle("STL decomposition")
fig.subplots_adjust(top=0.90)
fig.text(0.5, 0.95, "Employed = trend + seasonal + remainder", ha='center')
plt.xlabel("")
plt.show()
plt.show()
```

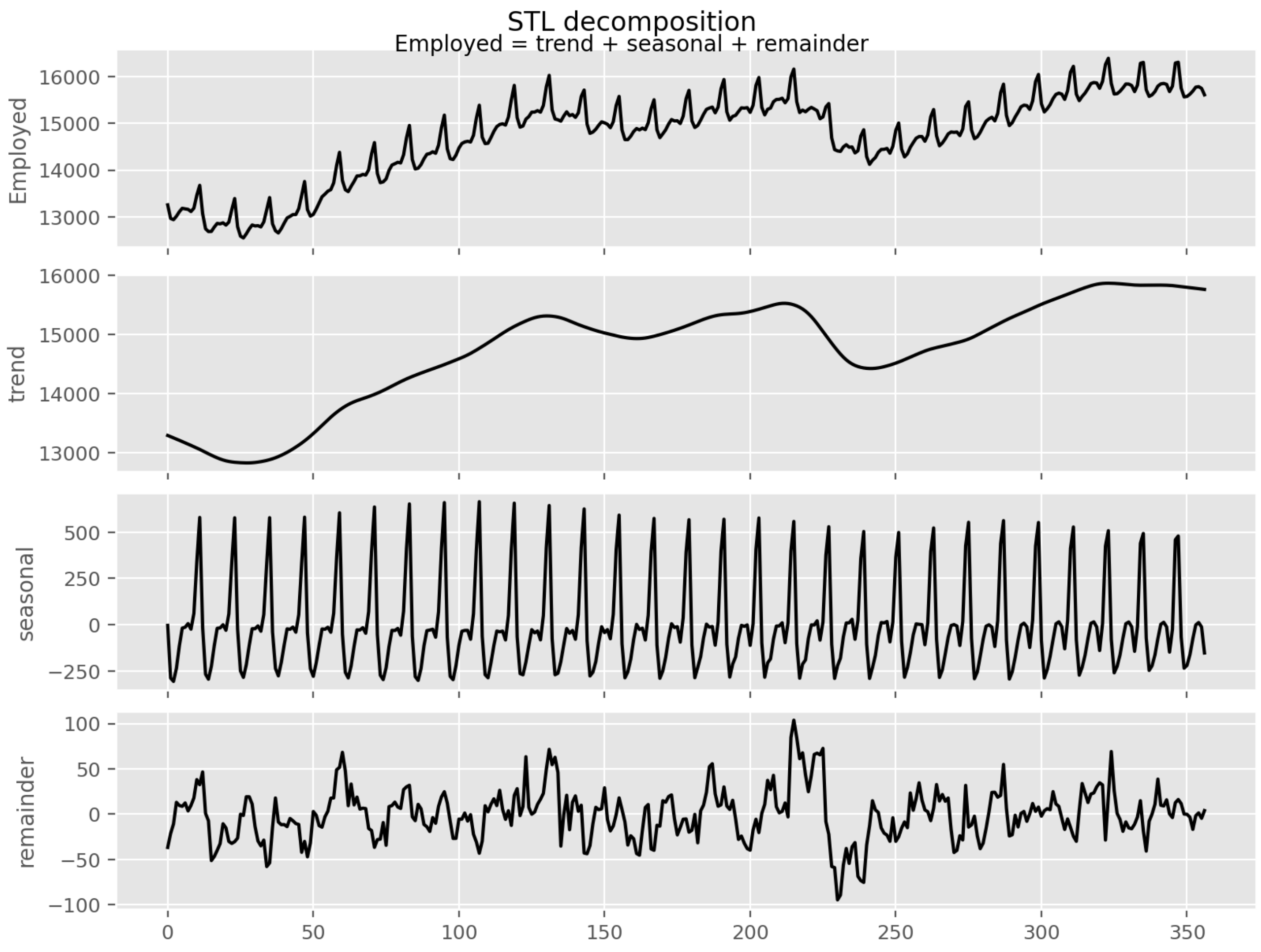


Figure 3.5: The total number of persons employed in US retail (top) and its three additive components.

The three components are shown separately in the bottom three panels. These components can be added together to reconstruct the data shown in the top panel. Notice that the seasonal component changes over time, so that any two consecutive years have similar patterns, but years far apart may have different seasonal patterns. The remainder component shown in the bottom panel is what is left over when the seasonal and trend-cycle components have been subtracted from the data.

Seasonally adjusted data

If the seasonal component is removed from the original data, the resulting values are the “seasonally adjusted” data. For an additive decomposition, the seasonally adjusted data are given by y_{t}-S_{t}, and for multiplicative data, the seasonally adjusted values are obtained using y_{t}/S_{t}.

Figure 3.6 shows the seasonally adjusted number of persons employed.

```
dcmp["season_adjust"] = dcmp["data"] - dcmp["seasonal"]

fig, ax = plt.subplots()
sns.lineplot(data=dcmp, x="ds", y="data", label="Data", color="gray")
sns.lineplot(
    data=dcmp,
    x="ds",
    y="season_adjust",
    label="Seasonally adjusted data",
    color="#0072B2",
)
ax.set_title("Total employment in US retail")
ax.set_xlabel("Month [1M]")
ax.set_ylabel("Persons (thousands)")
plt.show()
```

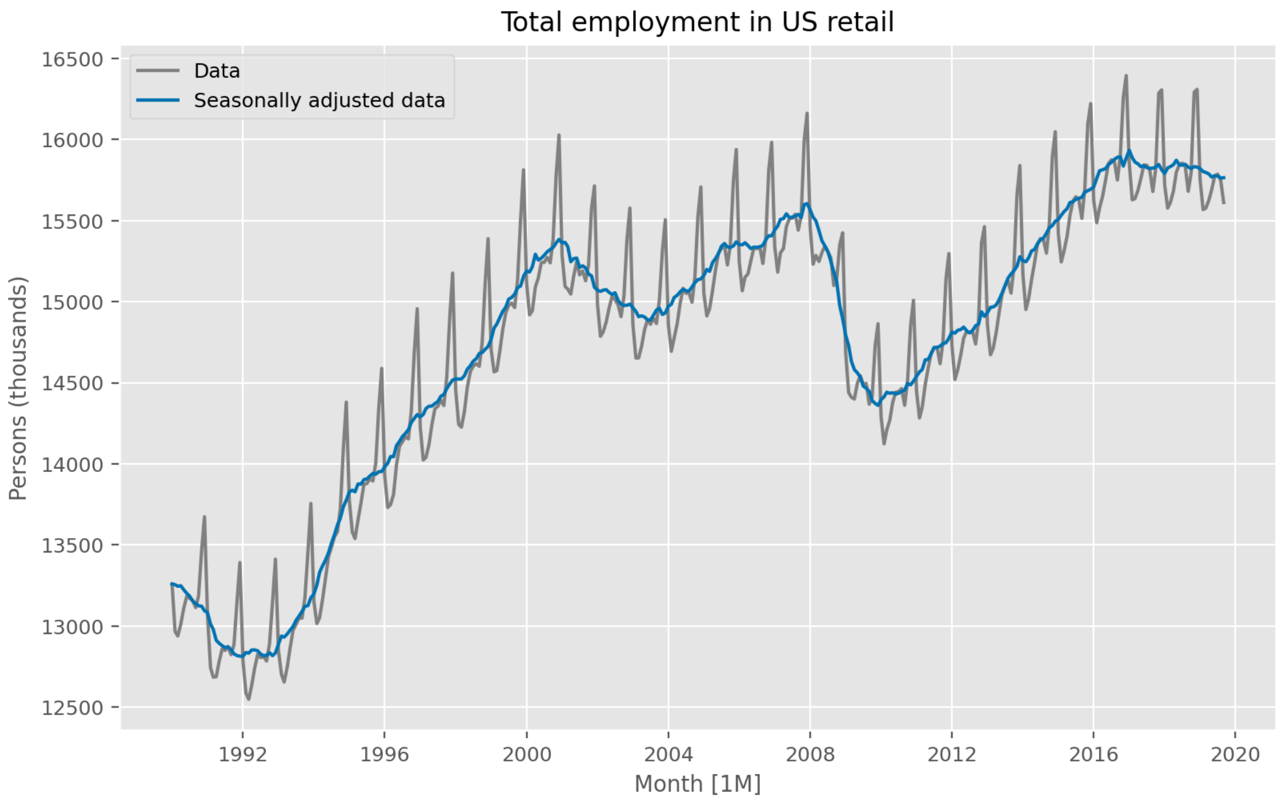


Figure 3.6: Seasonally adjusted retail employment data (blue) and the original data (grey).

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 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. The first step in a classical decomposition is to use a moving average method to estimate the trend-cycle, so we begin by discussing moving averages.

Moving average smoothing

A moving average of order m can be written as $\hat{T}_{t} = \frac{1}{m} \sum_{j=-k}^k y_{t+j}, \tag{3.2}$ where m=2k+1. That is, the estimate of the trend-cycle at time t is obtained by averaging values of the time series within k periods of t. Observations that are nearby in time are also likely to be close in value. 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.

For example, consider Figure 3.7 which shows exports of goods and services for Australia as a percentage of GDP from 1960 to 2017. The data are also shown in the table below.

```
global_economy = pd.read_csv("../data/global_economy.csv")
aus_exports = global_economy.query('unique_id == "Australia"')[
    ["unique_id", "ds", "Exports"]
]

plot_series(aus_exports, target_col="Exports",
            xlabel="Year [1Y]", ylabel="% of GDP",
            title="Total Australian exports")
```

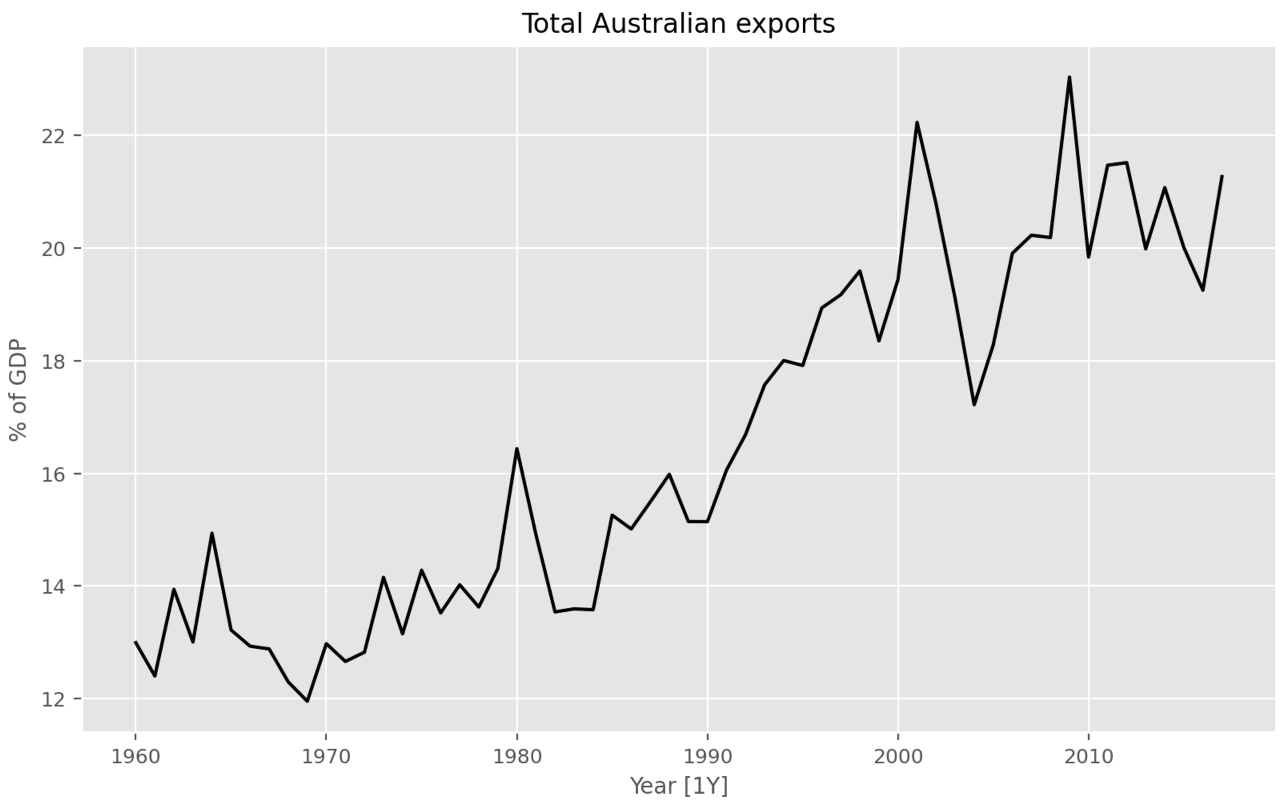


Figure 3.7: Australian exports of goods and services: 1960–2017.

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
1965	13.22	13.40
1966	12.93	13.25
1967	12.88	12.66
…	…	…
2010	19.84	21.21
2011	21.47	21.17
2012	21.52	20.78
2013	19.99	20.81
2014	21.08	20.37

Year	Exports	5-MA
2015	20.01	20.32
2016	19.25	
2017	21.27	

In the last column of this table, a moving average of order 5 is shown, providing an estimate of the trend-cycle. The first value in this column is the average of the first five observations, 1960–1964; the second value in the 5-MA column is the average of the values for 1961–1965; and so on. Each value in the 5-MA column is the average of the observations in the five year window centred on the corresponding year. In the notation of Equation 3.2, column 5-MA contains the values of \hat{T}_{t} with k=2 and m=2k+1=5. There are no values for either the first two years or the last two years, because we do not have two observations on either side. Later we will use more sophisticated methods of trend-cycle estimation which do allow estimates near the endpoints.

This is easily computed using the `rolling` function from `pandas`. In this case, we use a window of size 5. Note that we need to set `center=True` so that the moving average is centered around each observation.

```
aus_exports["5-MA"] = \
    aus_exports["Exports"].rolling(window=5, center=True).mean()
```

To see what the trend-cycle estimate looks like, we plot it along with the original data in Figure 3.8.

```
fig, ax = plt.subplots()
sns.lineplot(data=aus_exports, x="ds", y="Exports", label="Exports",
    color="black")
sns.lineplot(data=aus_exports, x="ds", y="5-MA", label="5-MA",
    color="#D55E00")
ax.set_title("Total Australian exports")
ax.set_xlabel("Year")
ax.set_ylabel("% of GDP")
plt.show()
```

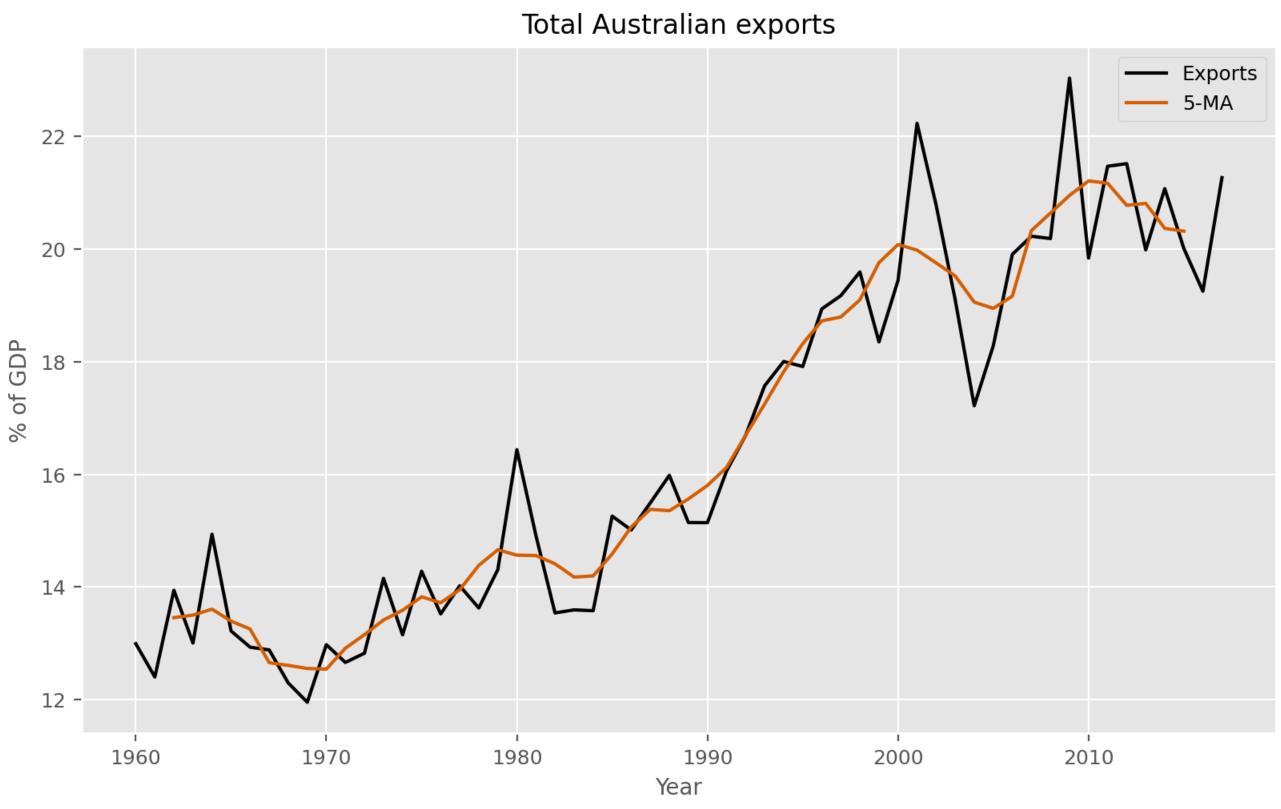


Figure 3.8: Australian exports (black) along with the 5-MA estimate of the trend-cycle (orange).

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. Figure 3.9 shows the effect of changing the order of the moving average for the Australian exports data.

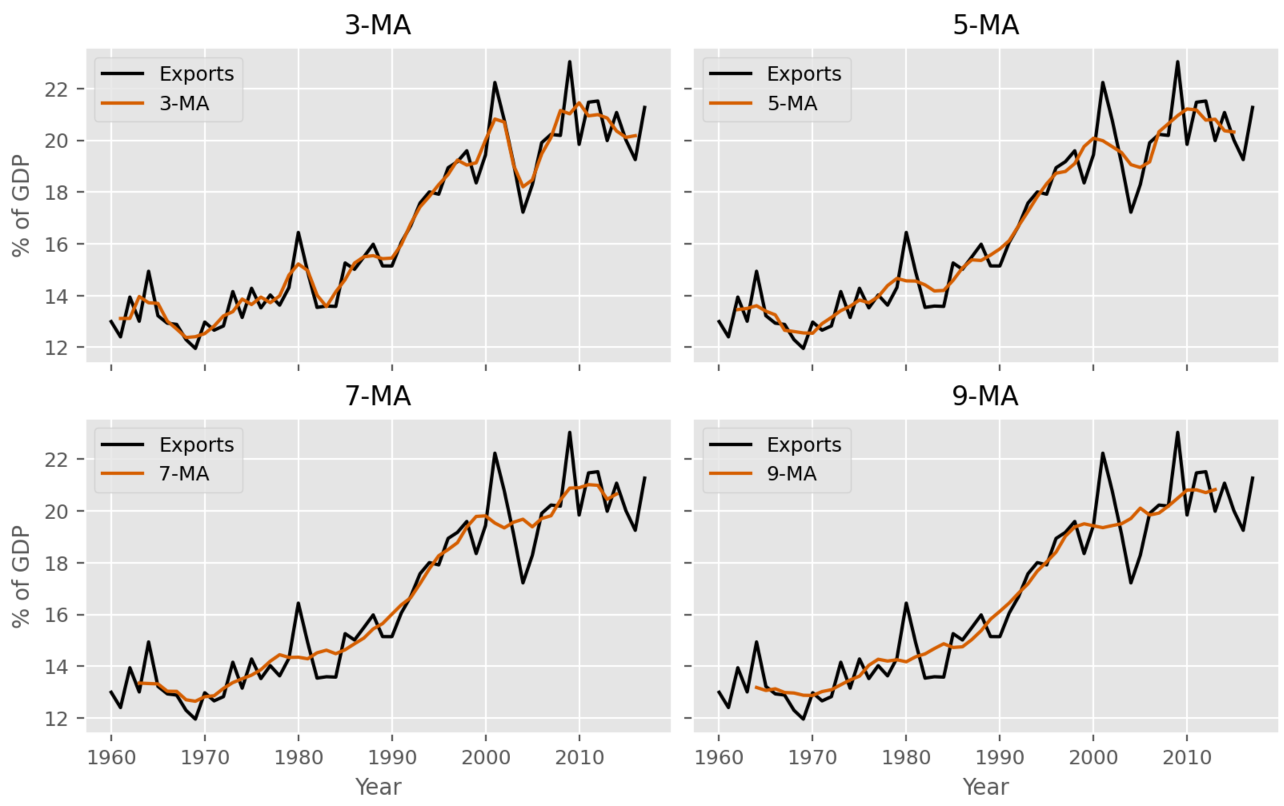


Figure 3.9: Different moving averages applied to the Australian exports data.

Simple moving averages such as these are usually of an odd order (e.g., 3, 5, 7, etc.). This is so they are symmetric: in a moving average of order m=2k+1, the middle observation, and k observations on either side, are averaged. But if m was even, it would no longer be symmetric.

Moving averages of moving averages

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.

For example, we might take a moving average of order 4, and then apply another moving average of order 2 to the results. In the following table, this has been done for the first few years of the Australian quarterly beer production data.

```
aus_production = pd.read_csv("../data/aus_production.csv",
    parse_dates=["ds"])
beer = aus_production.query('ds >= "1992-01-01"')[["ds", "Beer"]].copy()
beer.insert(0, "unique_id", "beer")
beer.rename(columns={"Beer": "y"}, inplace=True)
beer["4-MA"] = beer["y"].rolling(window=4, center=True).mean()
beer["2x4-MA"] = beer["4-MA"].rolling(window=2, center=True).mean()
```

Table 3.1: A moving average of order 4 applied to the quarterly beer data, followed by a moving average of order 2.

	unique_id	ds	y	4-MA	2x4-MA
144	beer	1992-01-01	443	NaN	NaN
145	beer	1992-04-01	410	NaN	NaN
146	beer	1992-07-01	420	451.25	NaN
147	beer	1992-10-01	532	448.75	450.000
148	beer	1993-01-01	433	451.50	450.125
...	...	...	...	...	...
213	beer	2009-04-01	398	430.00	428.875
214	beer	2009-07-01	419	430.00	430.000
215	beer	2009-10-01	488	429.75	429.875
216	beer	2010-01-01	414	423.75	426.750
217	beer	2010-04-01	374	NaN	NaN

74 rows × 5 columns

The notation “2\times4-MA” in the last column means a 4-MA followed by a 2-MA. The values in the last column are obtained by taking a moving average of order 2 of the values in the previous column. For example, the first two values in the 4-MA column are 451.25=(443+410+420+532)/4 and 448.75=(410+420+532+433)/4 The first value in the 2x4-MA column is the average of these two: 450=(452.25+448.75)/2.

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. To see that this is the case, we can write the 2\times4-MA as follows: \begin{align*} \hat{T}_{t} &= \frac{1}{2}\Big[\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})\Big] \\ &= \frac{1}{8}y_{t-2}+\frac14y_{t-1} + \frac14y_{t}+\frac14y_{t+1}+\frac18y_{t+2}. \end{align*} It is now a weighted average of observations that is symmetric.

Other combinations of moving averages are also possible. For example, a 3\times3-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.

Estimating the trend-cycle with seasonal data

The most common use of centred moving averages is for estimating the trend-cycle from seasonal data. Consider the 2\times4-MA: \hat{T}_{t} = \frac{1}{8}y_{t-2} + \frac14y_{t-1} + \frac14y_{t} + \frac14y_{t+1} + \frac18y_{t+2}. When applied to quarterly data, each quarter of the year is given equal weight as the first and last terms apply to the same quarter in consecutive years. Consequently, the seasonal variation will be averaged out and the resulting values of \hat{T}_t will have little or no seasonal variation remaining. A similar effect would be obtained using a 2\times 8-MA or a 2\times 12-MA to quarterly data.

In general, a 2\times m-MA is equivalent to a weighted moving average of order m+1 where all observations take the weight 1/m, except for the first and last terms which take weights 1/(2m). So, if the seasonal period is even and of order m, we use a 2\times m-MA to estimate the trend-cycle. If the seasonal period is odd and of order m, we use a m-MA to estimate the trend-cycle. For example, a 2\times 12-MA can be used to estimate the trend-cycle of monthly data with annual seasonality and a 7-MA can be used to estimate the trend-cycle of daily data with a weekly seasonality.

Other choices for the order of the MA will usually result in trend-cycle estimates being contaminated by the seasonality in the data.

Example: Employment in the US retail sector

```
us_retail_employment_ma = us_retail_employment.copy()
us_retail_employment_ma["12-MA"] = (
    us_retail_employment_ma["y"].rolling(window=12, center=True).mean()
)
us_retail_employment_ma["2x12-MA"] = (
    us_retail_employment_ma["12-MA"].rolling(window=2, center=True).mean()
)

fig, ax = plt.subplots()
sns.lineplot(data=us_retail_employment_ma, x="ds", y="y", label="Data",
    color="grey")
sns.lineplot(
    data=us_retail_employment_ma, x="ds", y="2x12-MA", label="2x12-MA",
    color="#D55E00"
)
ax.set_title("Total employment in US retail")
ax.set_xlabel("Month [1M]")
ax.set_ylabel("Persons (thousands)")
plt.show()
```

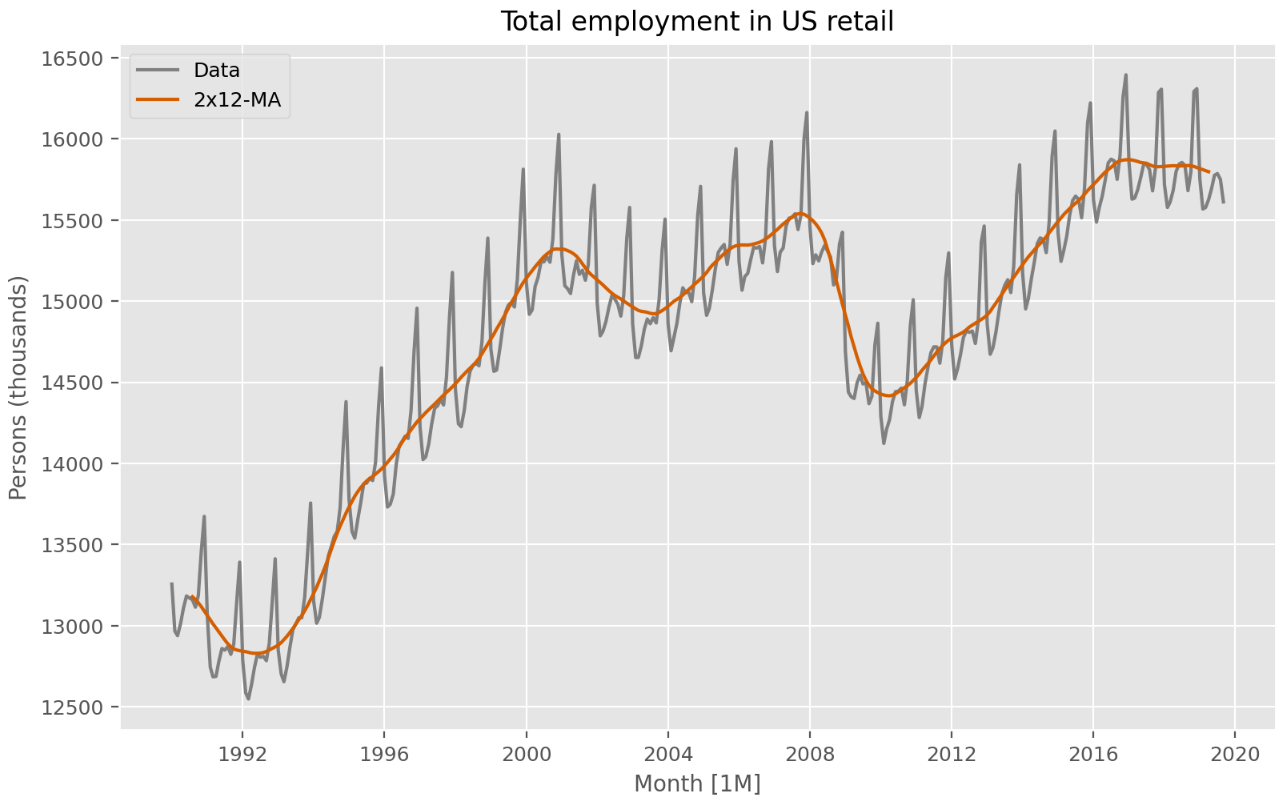


Figure 3.10: A 2x12-MA applied to the US retail employment series.

Figure 3.10 shows a 2\times12-MA applied to the total number of persons employed in the US retail sector. Notice that the smooth line shows no seasonality; it is almost the same as the trend-cycle shown in Figure 3.4, which was estimated using a much more sophisticated method than a moving average. Any other choice for the order of the moving average (except for 24, 36, etc.) would have resulted in a smooth line that showed some seasonal fluctuations.

Weighted moving averages

Combinations of moving averages result in weighted moving averages. For example, the 2\times4-MA discussed above is equivalent to a weighted 5-MA with weights given by \left[\frac{1}{8},\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{8}\right]. In general, a weighted m-MA can be written as \hat{T}_t = \sum_{j=-k}^k a_j y_{t+j}, where k=(m-1)/2, and the weights are given by \left[a_{-k},\dots,a_k\right]. It is important that the weights all sum to one and that they are symmetric so that a_j = a_{-j}. The simple m-MA is a special case where all of the weights are equal to 1/m.

A major advantage of weighted moving averages is that they yield a smoother estimate of the trend-cycle. Instead of observations entering and leaving the calculation at full weight, their weights slowly increase and then slowly decrease, resulting in a smoother curve.

3.4 Classical decomposition

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

Figure 3.11 shows a classical decomposition of the total retail employment series across the US.

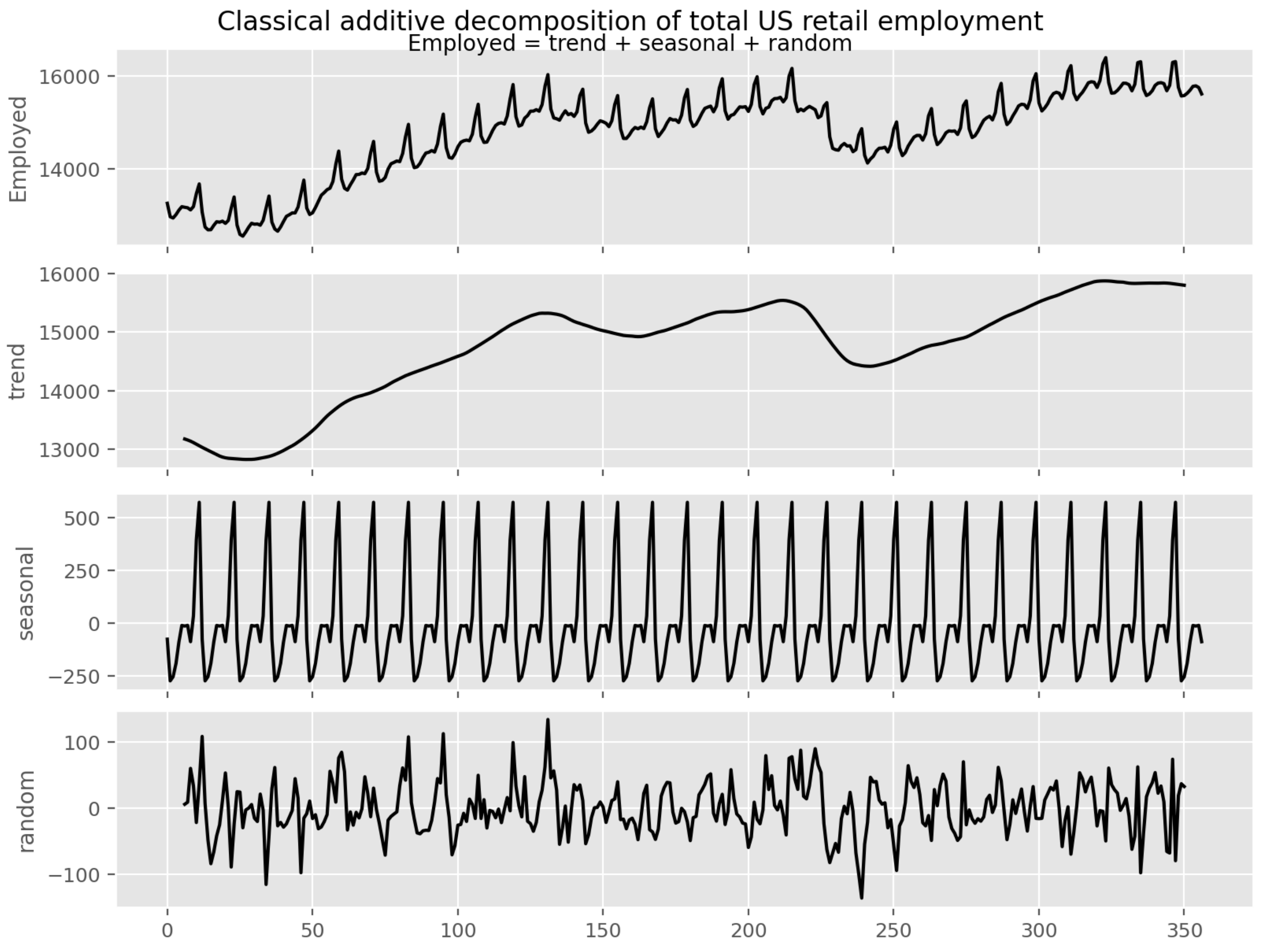


Figure 3.11: A classical additive decomposition of US retail employment.

Multiplicative decomposition

A classical multiplicative decomposition is similar, except that the subtractions are replaced by divisions.

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

Step 4
: The remainder component is calculated by dividing out the estimated seasonal and trend-cycle components: $\hat{R}_{t} = y_t /(\hat{T}_t \hat{S}_t)$.

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

Official statistics agencies (such as the US Census Bureau and the Australian Bureau of Statistics) are responsible for a large number of official economic and social time series. These agencies have developed their own decomposition procedures which are used for seasonal adjustment. Most of them use variants of the X-11 method, or the SEATS method, or a combination of the two. These methods are designed specifically to work with quarterly and monthly data, which are the most common series handled by official statistics agencies. They will not handle seasonality of other kinds, such as daily data, or hourly data, or weekly data.

X-11 method

The X-11 method originated in the US Census Bureau and was further developed by Statistics Canada. It is based on classical decomposition, but includes many extra steps and features in order to overcome the drawbacks of classical decomposition that were discussed in the previous section. In particular, trend-cycle estimates are available for all observations including the end points, and the seasonal component is allowed to vary slowly over time. X-11 also handles trading day variation, holiday effects and the effects of known predictors. There are methods for both additive and multiplicative decomposition. The process is entirely automatic and tends to be highly robust to outliers and level shifts in the time series. The details of the X-11 method are described in Dagum and Bianconcini (2016).

SEATS method

“SEATS” stands for “Seasonal Extraction in ARIMA Time Series” (ARIMA models are discussed in Chapter 9). This procedure was developed at the Bank of Spain, and is now widely used by government agencies around the world. The details are beyond the scope of this book. However, a complete discussion of the method is available in Dagum and Bianconcini (2016).

3.6 STL decomposition

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 Cleveland et al. (1990), and later extended to handle multiple seasonal patterns by Bandara, Hyndman, and Bergmeir (2022).

STL has several advantages over classical decomposition, and the SEATS and X-11 methods:

- Unlike SEATS and X-11, STL will handle any type of seasonality, not only monthly and quarterly data.
- The seasonal component is allowed to change over time, and the rate of change can be controlled by the user.
- The smoothness of the trend-cycle can also be controlled by the user.
- It can be robust to outliers (i.e., the user can specify a robust decomposition), so that occasional unusual observations will not affect the estimates of the trend-cycle and seasonal components. They will, however, affect the remainder component.

On the other hand, STL has some disadvantages. In particular, it does not handle trading day or calendar variation automatically, and it only provides facilities for additive decompositions.

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<\lambda<1. A value of \lambda=0 gives a multiplicative decomposition while \lambda=1 gives an additive decomposition.

The best way to begin learning how to use STL is to see some examples and experiment with the settings. Figure 3.5 showed an example of an STL decomposition applied to the total US retail employment series. Figure 3.12 shows an alternative STL decomposition where the trend-cycle is more flexible, the seasonal pattern is fixed, and the robust option has been used.

```python
stl = STL(us_retail_employment["y"], period=12, seasonal=13, trend=21, robust=True)
res_stl = stl.fit()

dcmp = pd.DataFrame({
    "ds": us_retail_employment["ds"],
    "data": us_retail_employment["y"],
    "trend": res_stl.trend,
    "seasonal": res_stl.seasonal,
    "remainder": res_stl.resid
}).reset_index(drop=True)


fig, axes = plt.subplots(nrows=4, ncols=1, sharex=True, figsize=(8, 6))
sns.lineplot(data=dcmp, x=dcmp.index, y="data", ax=axes[0])
sns.lineplot(data=dcmp, x=dcmp.index, y="trend", ax=axes[1])
sns.lineplot(data=dcmp, x=dcmp.index, y="seasonal", ax=axes[2])
sns.lineplot(data=dcmp, x=dcmp.index, y="remainder", ax=axes[3])
axes[0].set_ylabel("Employed")
axes[1].set_ylabel("trend")
axes[2].set_ylabel("season_year")
axes[3].set_ylabel("remainder")
fig.suptitle("STL decomposition")
fig.subplots_adjust(top=0.90)
fig.text(0.5, 0.95, "Employed = trend + seasonal + random", ha="center")
plt.xlabel("")
plt.show()
```

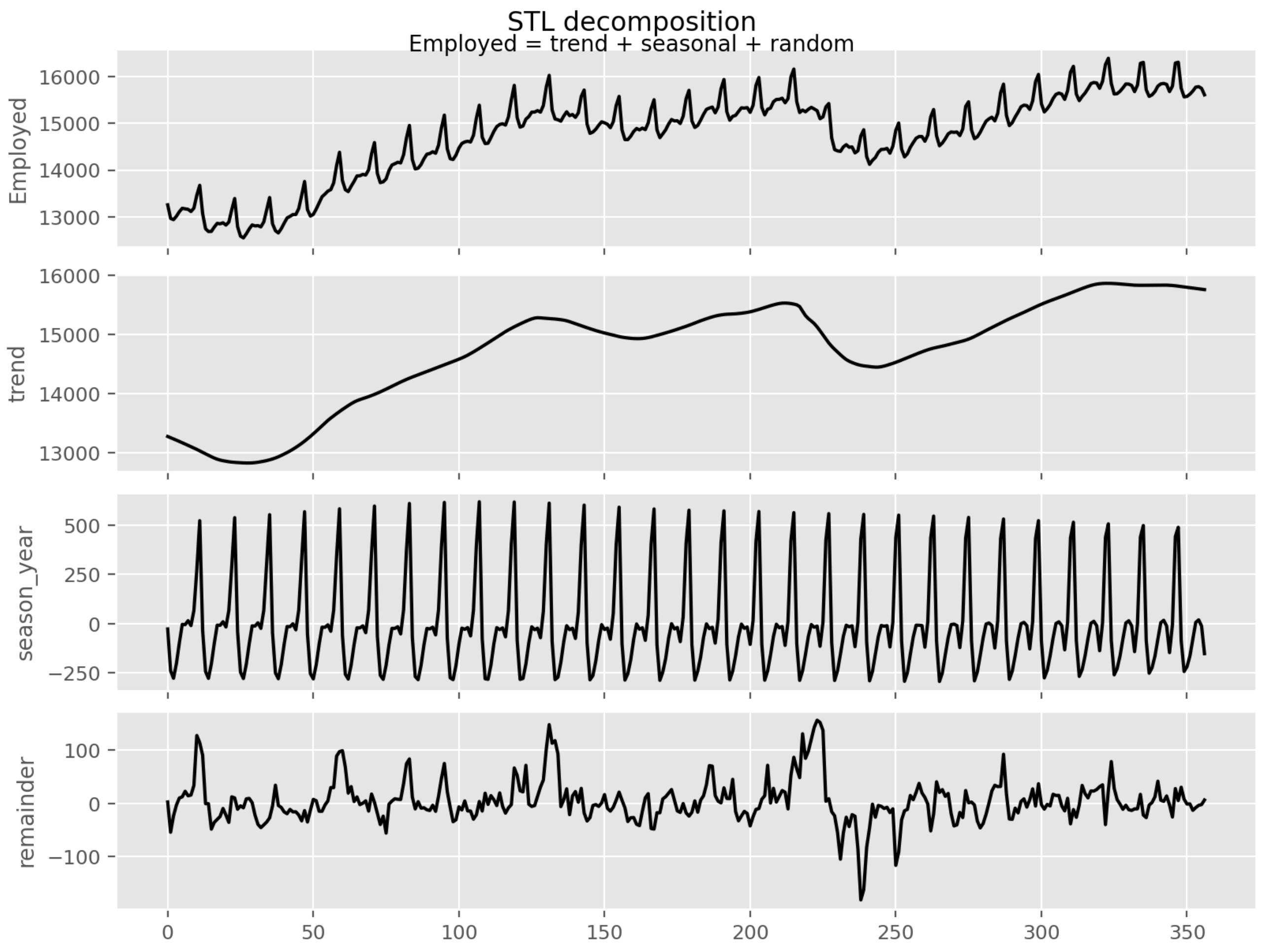


Figure 3.12: Total US retail employment (top) and its three additive components obtained from a robust STL decomposition with flexible trend-cycle and fixed seasonality.

The two main parameters to be chosen when using STL are the `season` and the `trend`. The `season` is the length of the seasonal smoother, while the `trend` is the length of the trend smoother. These parameters 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.

By default, the `STL()` function in `statsmodels` provides a convenient automated STL decomposition. 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 3.5. Selecting a shorter trend window as in Figure 3.12 improves this.

3.7 Exercises

1. Consider the GDP information in `global_economy`. Plot the GDP per capita for each country over time. Which country has the highest GDP per capita? How has this changed over time?

2. For each of the following series, make a graph of the data. If transforming seems appropriate, do so and describe the effect.

    - United States GDP from `global_economy`.
    - Slaughter of Victorian “Bulls, bullocks and steers” in `aus_livestock`.
    - Victorian Electricity Demand from `vic_elec`.
    - Gas production from `aus_production`.

3. Why is a Box-Cox transformation unhelpful for the `canadian_gas` data?

4. What Box-Cox transformation would you select for your retail data (from Exercise 7 in Section 2.10)?

5. For the following series, find an appropriate Box-Cox transformation in order to stabilise the variance. Tobacco from `aus_production`, Economy class passengers between Melbourne and Sydney from `ansett`, and Pedestrian counts at Southern Cross Station from `pedestrian`.

6. Show that a 3\times5 MA is equivalent to a 7-term weighted moving average with weights of 0.067, 0.133, 0.200, 0.200,

0.200, 0.133, and 0.067.

7. Consider the last five years of the Gas data from `aus_production`.

```
gas = aus_production.query('unique_id == "Gas"').tail(5*4)
```

   a. Plot the time series. Can you identify seasonal fluctuations and/or a trend-cycle?
   b. Use `seasonal_decompose` with `model='multiplicative'` to calculate the trend-cycle and seasonal indices.
   c. Do the results support the graphical interpretation from part a?
   d. Compute and plot the seasonally adjusted data.
   e. Change one observation to be an outlier (e.g., add 300 to one observation), and recompute the seasonally adjusted data. What is the effect of the outlier?
   f. Does it make any difference if the outlier is near the end rather than in the middle of the time series?

8. Recall your retail time series data (from Exercise 7 in Section 2.10). Decompose the series using the MSTL method. Does it reveal any outliers, or unusual features that you had not noticed previously?

9. Figures 3.13 and 3.14 show the result of decomposing the number of persons in the civilian labour force in Australia each month from February 1978 to August 1995.

   a. Write about 3–5 sentences describing the results of the decomposition. Pay particular attention to the scales of the graphs in making your interpretation.
   b. Is the recession of 1991/1992 visible in the estimated components?

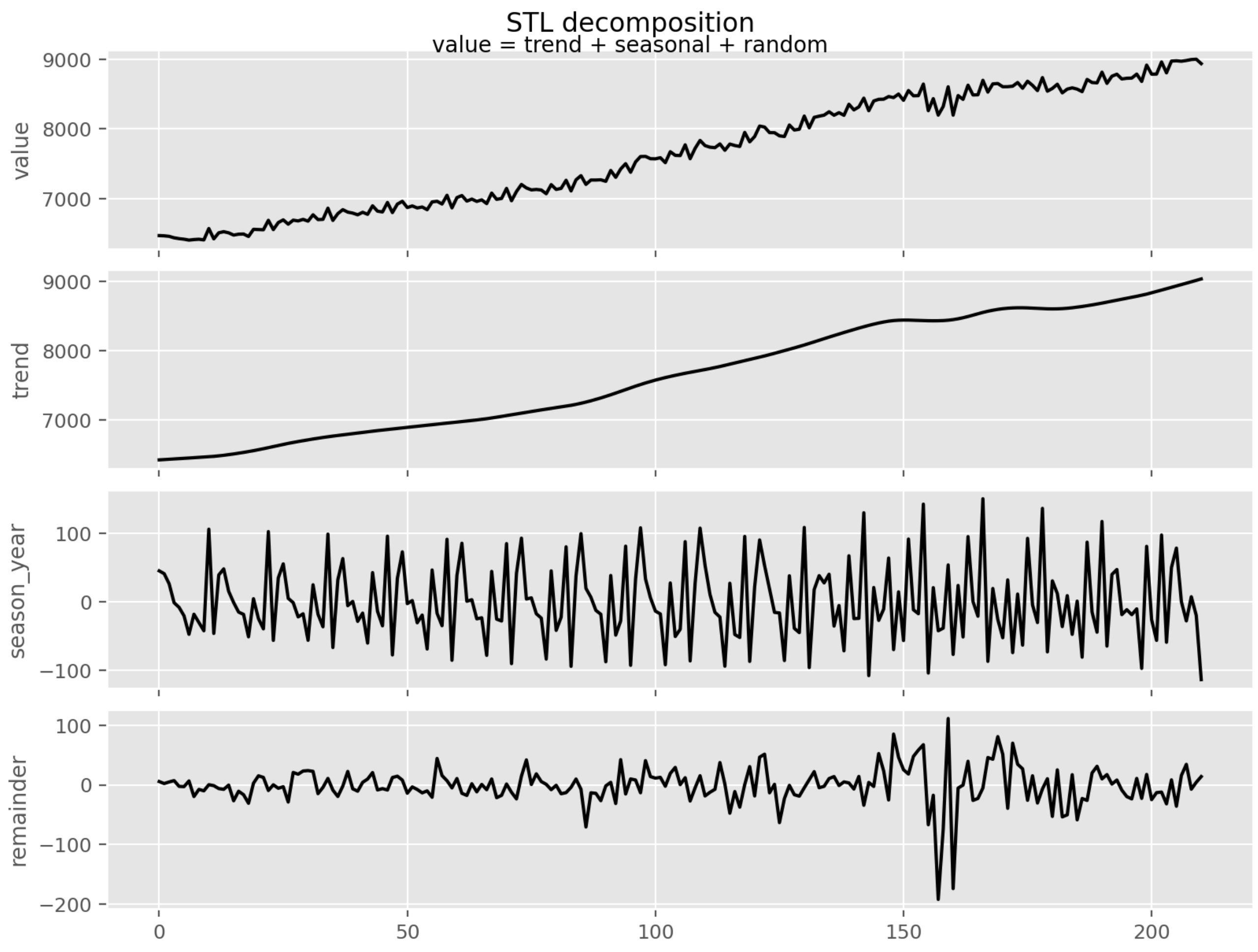


Figure 3.13: Decomposition of the number of persons in the civilian labour force in Australia each month from February 1978 to August 1995.

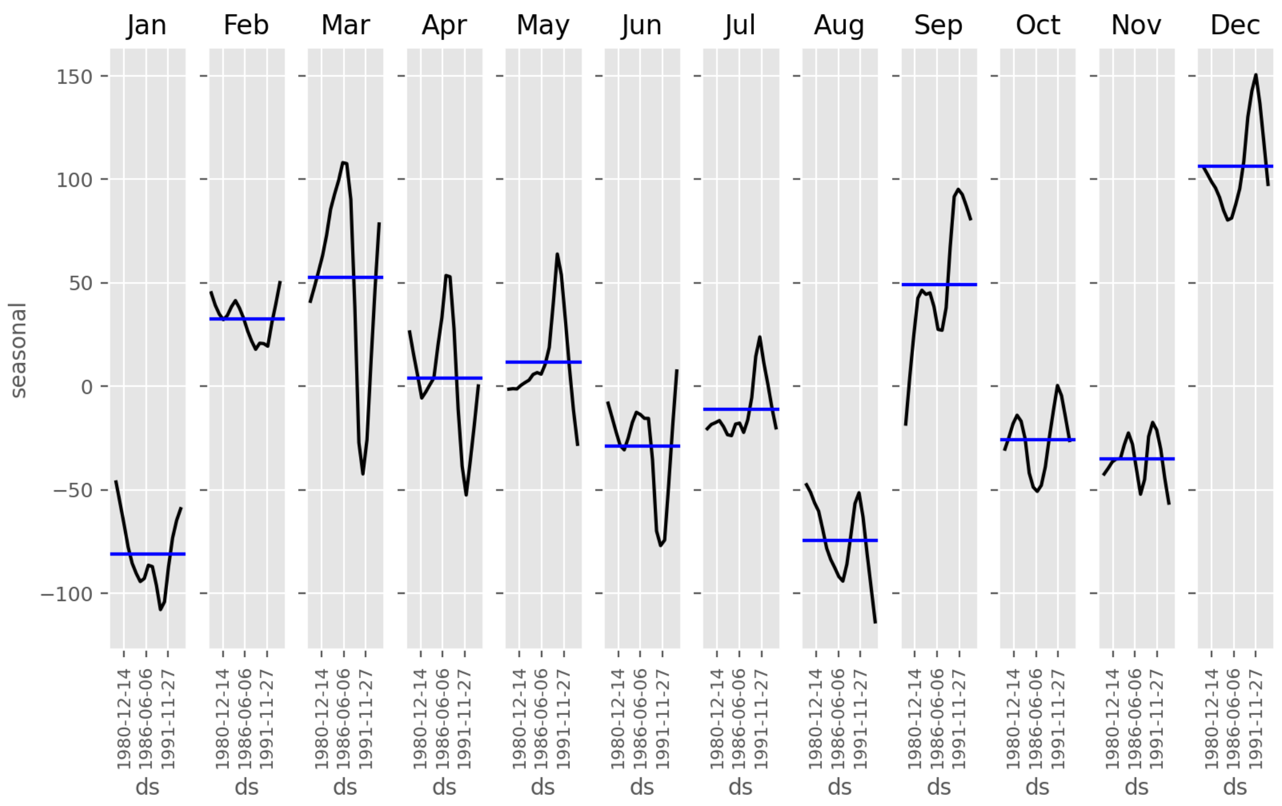


Figure 3.14: Seasonal component from the decomposition shown in the previous figure.

10. This exercise uses the `canadian_gas` data (monthly Canadian gas production in billions of cubic metres, January 1960 – February 2005).
    a. Plot the data using to look at the effect of the changing seasonality over time.[1]
    b. Do an STL decomposition of the data using `STL()`.
    c. How does the seasonal shape change over time? [Hint: Try plotting the seasonal component.]
    d. Can you produce a plausible seasonally adjusted series?

3.8 Further reading

- Cleveland et al. (1990) introduced STL, and still provides the best description of the algorithm. Bandara, Hyndman, and Bergmeir (2022) introduced MSTL, which is an extension of STL that allows for multiple seasonal patterns.

3.9 Used modules and classes

CoreForecast

- `boxcox` function - For applying Box-Cox transformations
- `boxcox_lambda` function - For finding optimal Box-Cox transformation parameters

UtilsForecast

- `plot_series` utility - For creating time series visualizations

. The evolving seasonal pattern is possibly due to changes in the regulation of gas prices — thanks to Lewis Kirvan for pointing this out. ↩︎

← Chapter 2 Time series graphics

Chapter 4 Time series features →