Forecasting Principles and Practice, 3rd Ed. by Hyndman & Athanasopoulos.

Booknotes by Fernando San Segundo.

Last updated: 2021-04-15

Índice

Preliminaries and Book Information.

Software requirements

```
library(tidyverse)
```

```
if(!require(fpp3))install.packages("fpp3")
```

```
## Loading required package: fpp3

## -- Attaching packages -------------------------------------------- fpp3 0.4.0 --

## v lubridate   1.7.10     v feasts      0.2.1
## v tsibble     1.0.0      v fable       0.3.0
## v tsibbledata 0.3.0

## -- 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()
```

Useful links

Continue reading at

Text of the book

Chapter 1, Getting started

1.1 What can be forecast?

The predictability of an event or a quantity depends on several factors including:

1. how well we understand the factors that contribute to it;
2. how much data is available;
3. how similar the future is to the past;
4. whether the forecasts can affect the thing we are trying to forecast.

1.2 Forecasting, goals and planning

- **Forecasting** is about predicting the future as accurately as possible, given all of the information available, including historical data and knowledge of any future events that might impact the forecasts.
- **Goals** are what you would like to have happen. Goals should be linked to forecasts and plans, but this does not always occur. Too often, goals are set without any plan for how to achieve them, and no forecasts for whether they are realistic.
- **Planning** is a response to forecasts and goals. Planning involves determining the appropriate actions that are required to make your forecasts match your goals.

1.3 Determining what to forecast

1.4 Forecasting data and methods

Quantitative forecasting can be applied when two conditions are satisfied:

1. numerical information about the past is available;
2. it is reasonable to assume that some aspects of the past patterns will continue into the future.

Most quantitative prediction problems use either time series data (collected at regular intervals over time) or cross-sectional data (collected at a single point in time). *In this book we are concerned with forecasting future data, and we concentrate on the time series domain.*

Time series forecasting

In this book, we will only consider time series that are observed at **regular intervals of time** (e.g., hourly, daily, weekly, monthly, quarterly, annually). Irregularly spaced time series can also occur, but are beyond the scope of this book.

The simplest time series forecasting methods use only information on the variable to be forecast, and make no attempt to discover the factors that affect its behaviour. Therefore they will extrapolate trend and seasonal patterns.

Decomposition methods are helpful for studying the trend and seasonal patterns in a time series; these are discussed in Chapter 3. Popular time series models used for forecasting include exponential smoothing models and ARIMA models, discussed in Chapters 8 and 9 respectively.

Predictor variables and time series forecasting

A model with predictor variables might be of the form

$$\begin{aligned}\text{ED} = &f(\text{current temperature, strength of economy, population,}\\ &\text{time of day, day of week, error}).\end{aligned}$$

A suitable time series forecasting model is of the form

$$\text{ED}_{t+1} = f(\text{ED}_t, \text{ED}_{t-1}, \text{ED}_{t-2}, \text{ED}_{t-3}, \dots, \text{error})$$

There is also a third type of model which combines the features of the above two models. For example, it might be given by

$$\text{ED}_{t+1} = f(\text{ED}_t, \text{current temperature, time of day, day of week, error}).$$

These types of "mixed models" have been given various names in different disciplines. They are known as dynamic regression models, panel data models, longitudinal models, transfer function models.

There are several reasons a forecaster might select a time series model rather than an explanatory or mixed model. in particular, it is necessary to know or forecast the future values of the various predictors in order to be able to forecast the variable of interest, and this may be too difficult.

1.5 Some case studies

1.6 The basic steps in a forecasting task

- Step 1: Problem definition. Often the most difficult part of forecasting.
- Step 2: Gathering information. Data and expert knowñedge are required.
- Step 3: Preliminary (exploratory) analysis.
- Step 4: Choosing and fitting models. Each model is itself an artificial construct that is based on a set of assumptions (explicit and implicit) and usually involves one or more parameters which must be estimated using the known historical data. We will discuss regression models (Chapter 7), exponential smoothing methods (Chapter 8), Box-Jenkins ARIMA models (Chapter 9), Dynamic regression models (Chapter 10), Hierarchical forecasting (Chapter 11), and several advanced methods including neural networks and vector autoregression (Chapter 12).
- Step 5: Using and evaluating a forecasting model.

1.7 The statistical forecasting perspective

In most forecasting situations, the variation associated with the thing we are forecasting will shrink as the event approaches. In other words, the further ahead we forecast, the more uncertain we are.

When we obtain a forecast, we are estimating the middle of the range of possible values the random variable could take. Often, a forecast is accompanied by a prediction interval giving a range of values the random variable could take with relatively high probability.

We will use the subscript t for time. Thus, y_t will denote the observation at time t. The symbol $y_t|\mathcal{I}$ means "the random variable y_t given the information $\mathcal{I}$ that what we know. The set of values that this random variable could take, along with their relative probabilities, is known as the "forecasting distribution of $y_t|\mathcal{I}$. The *"forecast"* usually means mean the (estimated?) average value of the forecast distribution, denoted by $\hat{y}_t$. Also we will write, for example, $\hat{y}_{t|t-1}$ to mean the forecast of y_t taking account of all previous observations $(y_1, \ldots, y_{t-1})$.

Chapter 2 Time series graphics

2.1 tsibble objec

The index variablets

```
library(fpp3)
(y <- tsibble(
  Year = 2015:2019,
  Observation = c(123, 39, 78, 52, 110),
  index = Year
))
```

```
## # A tsibble: 5 x 2 [1Y]
##    Year Observation
##   <int>       <dbl>
## 1  2015         123
## 2  2016          39
## 3  2017          78
## 4  2018          52
## 5  2019         110
```

Sometimes we need to use a time class function on the index.

```
(z = tribble(
  ~Month, ~Observation,
"2019 Jan", 50,
"2019 Feb", 23,
"2019 Mar", 34,
"2019 Apr", 30,
"2019 May", 25
))
```

```
## # A tibble: 5 x 2
##   Month    Observation
##   <chr>          <dbl>
## 1 2019 Jan          50
## 2 2019 Feb          23
## 3 2019 Mar          34
## 4 2019 Apr          30
## 5 2019 May          25
```

```
Sys.getlocale()
```

```
## [1] "es_ES.UTF-8/es_ES.UTF-8/es_ES.UTF-8/C/es_ES.UTF-8/es_ES.UTF-8"
```

```
Sys.setlocale(category = "LC_ALL", locale = "en_US.UTF-8")
```

```
## [1] "en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/es_ES.UTF-8"
```

```
z %>%
  mutate(Month = yearmonth(Month)) %>%
  as_tsibble(index = Month)
```

```
## # A tsibble: 5 x 2 [1M]
##      Month Observation
##      <mth>       <dbl>
## 1 2019 Jan          50
## 2 2019 Feb          23
## 3 2019 Mar          34
## 4 2019 Apr          30
## 5 2019 May          25
```

Other time class functions can be used:

Frequency	Function
Annual	start:end
Quarterly	yearquarter()
Monthly	yearmonth()
Weekly	yearweek()
Daily	as_date(), ymd()
Sub-daily	as_datetime(), ymd_hms()

The key variables

```
olympic_running
```

```
## # A tsibble: 312 x 4 [4Y]
## # Key:       Length, Sex [14]
##     Year Length Sex    Time
##    <int>  <int> <chr> <dbl>
##  1  1896    100 men    12
##  2  1900    100 men    11
##  3  1904    100 men    11
##  4  1908    100 men    10.8
##  5  1912    100 men    10.8
##  6  1916    100 men    NA
##  7  1920    100 men    10.8
##  8  1924    100 men    10.6
##  9  1928    100 men    10.8
## 10  1932    100 men    10.3
## # ... with 302 more rows
```

```
olympic_running %>% distinct(Length)
```

```
## # A tibble: 7 x 1
##   Length
##    <int>
## 1    100
## 2    200
## 3    400
## 4    800
## 5   1500
## 6   5000
## 7  10000
```

Working with tsibble objects

```
PBS
```

```
## # A tsibble: 67,596 x 9 [1M]
## # Key:       Concession, Type, ATC1, ATC2 [336]
##       Month Concession  Type   ATC1  ATC1_desc   ATC2  ATC2_desc   Scripts  Cost
```

```
##        <mth> <chr>        <chr>  <chr> <chr>        <chr> <chr>          <dbl> <dbl>
##  1 1991 Jul Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    18228 67877
##  2 1991 Aug Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    15327 57011
##  3 1991 Sep Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    14775 55020
##  4 1991 Oct Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    15380 57222
##  5 1991 Nov Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    14371 52120
##  6 1991 Dec Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    15028 54299
##  7 1992 Jan Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    11040 39753
##  8 1992 Feb Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    15165 54405
##  9 1992 Mar Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    16898 61108
## 10 1992 Apr Concession~ Co-pa~ A     Alimentary~ A01   STOMATOLOG~    18141 65356
## # ... with 67,586 more rows
```

```
PBS %>%
  filter(ATC2 == "A10")
```

```
## # A tsibble: 816 x 9 [1M]
## # Key:       Concession, Type, ATC1, ATC2 [4]
##        Month Concession   Type   ATC1  ATC1_desc   ATC2  ATC2_desc  Scripts   Cost
##        <mth> <chr>        <chr>  <chr> <chr>       <chr> <chr>        <dbl>  <dbl>
##  1 1991 Jul Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   89733 2.09e6
##  2 1991 Aug Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   77101 1.80e6
##  3 1991 Sep Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   76255 1.78e6
##  4 1991 Oct Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   78681 1.85e6
##  5 1991 Nov Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   70554 1.69e6
##  6 1991 Dec Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   75814 1.84e6
##  7 1992 Jan Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   64186 1.56e6
##  8 1992 Feb Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   75899 1.73e6
##  9 1992 Mar Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   89445 2.05e6
## 10 1992 Apr Concession~ Co-pa~ A     Alimentary~ A10   ANTIDIABE~   97315 2.23e6
## # ... with 806 more rows
```

```
PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost)
```

```
## # A tsibble: 816 x 4 [1M]
## # Key:       Concession, Type [4]
##        Month Concession   Type           Cost
##        <mth> <chr>        <chr>         <dbl>
##  1 1991 Jul Concessional Co-payments 2092878
##  2 1991 Aug Concessional Co-payments 1795733
##  3 1991 Sep Concessional Co-payments 1777231
##  4 1991 Oct Concessional Co-payments 1848507
##  5 1991 Nov Concessional Co-payments 1686458
##  6 1991 Dec Concessional Co-payments 1843079
##  7 1992 Jan Concessional Co-payments 1564702
##  8 1992 Feb Concessional Co-payments 1732508
##  9 1992 Mar Concessional Co-payments 2046102
## 10 1992 Apr Concessional Co-payments 2225977
## # ... with 806 more rows
```

```r
PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost) %>%
  summarise(TotalC = sum(Cost))
```

```
## # A tsibble: 204 x 2 [1M]
##       Month  TotalC
##       <mth>   <dbl>
##  1 1991 Jul 3526591
##  2 1991 Aug 3180891
##  3 1991 Sep 3252221
##  4 1991 Oct 3611003
##  5 1991 Nov 3565869
##  6 1991 Dec 4306371
##  7 1992 Jan 5088335
##  8 1992 Feb 2814520
##  9 1992 Mar 2985811
## 10 1992 Apr 3204780
## # ... with 194 more rows
```

```r
PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost) %>%
  summarise(TotalC = sum(Cost)) %>%
  mutate(Cost = TotalC/1e6)
```

```
## # A tsibble: 204 x 3 [1M]
##       Month  TotalC  Cost
##       <mth>   <dbl> <dbl>
##  1 1991 Jul 3526591  3.53
##  2 1991 Aug 3180891  3.18
##  3 1991 Sep 3252221  3.25
##  4 1991 Oct 3611003  3.61
##  5 1991 Nov 3565869  3.57
##  6 1991 Dec 4306371  4.31
##  7 1992 Jan 5088335  5.09
##  8 1992 Feb 2814520  2.81
##  9 1992 Mar 2985811  2.99
## 10 1992 Apr 3204780  3.20
## # ... with 194 more rows
```

```r
a10 = PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost) %>%
  summarise(TotalC = sum(Cost)) %>%
  mutate(Cost = TotalC / 1e6)
```

Read a csv file and convert to a tsibble

```r
if(!file.exists("../data/prison_population.csv")){
  prison <- readr::read_csv("https://OTexts.com/fpp3/extrafiles/prison_population.csv")
  write_csv(prison, file = "../data/prison_population.csv")
} else {
  prison = readr::read_csv("../data/prison_population.csv")
}
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Date = col_date(format = ""),
##   State = col_character(),
##   Gender = col_character(),
##   Legal = col_character(),
##   Indigenous = col_character(),
##   Count = col_double()
## )
```

```r
prison
```

```
## # A tibble: 3,072 x 6
##    Date       State Gender Legal     Indigenous Count
##    <date>     <chr> <chr>  <chr>     <chr>      <dbl>
##  1 2005-03-01 ACT   Female Remanded  ATSI           0
##  2 2005-03-01 ACT   Female Remanded  Non-ATSI       2
##  3 2005-03-01 ACT   Female Sentenced ATSI           0
##  4 2005-03-01 ACT   Female Sentenced Non-ATSI       5
##  5 2005-03-01 ACT   Male   Remanded  ATSI           7
##  6 2005-03-01 ACT   Male   Remanded  Non-ATSI      58
##  7 2005-03-01 ACT   Male   Sentenced ATSI           5
##  8 2005-03-01 ACT   Male   Sentenced Non-ATSI     101
##  9 2005-03-01 NSW   Female Remanded  ATSI          51
## 10 2005-03-01 NSW   Female Remanded  Non-ATSI     131
## # ... with 3,062 more rows
```

```r
prison <- prison %>%
  mutate(Quarter = yearquarter(Date)) %>%
  select(-Date) %>%
  as_tsibble(key = c(State, Gender, Legal, Indigenous),
             index = Quarter)

prison
```

```
## # A tsibble: 3,072 x 6 [1Q]
## # Key:       State, Gender, Legal, Indigenous [64]
##    State Gender Legal    Indigenous Count Quarter
##    <chr> <chr>  <chr>    <chr>      <dbl>   <qtr>
##  1 ACT   Female Remanded ATSI           0 2005 Q1
##  2 ACT   Female Remanded ATSI           1 2005 Q2
##  3 ACT   Female Remanded ATSI           0 2005 Q3
##  4 ACT   Female Remanded ATSI           0 2005 Q4
```

```
## 5 ACT   Female Remanded ATSI           1 2006 Q1
## 6 ACT   Female Remanded ATSI           1 2006 Q2
## 7 ACT   Female Remanded ATSI           1 2006 Q3
## 8 ACT   Female Remanded ATSI           0 2006 Q4
## 9 ACT   Female Remanded ATSI           0 2007 Q1
## 10 ACT  Female Remanded ATSI           1 2007 Q2
## # ... with 3,062 more rows
```

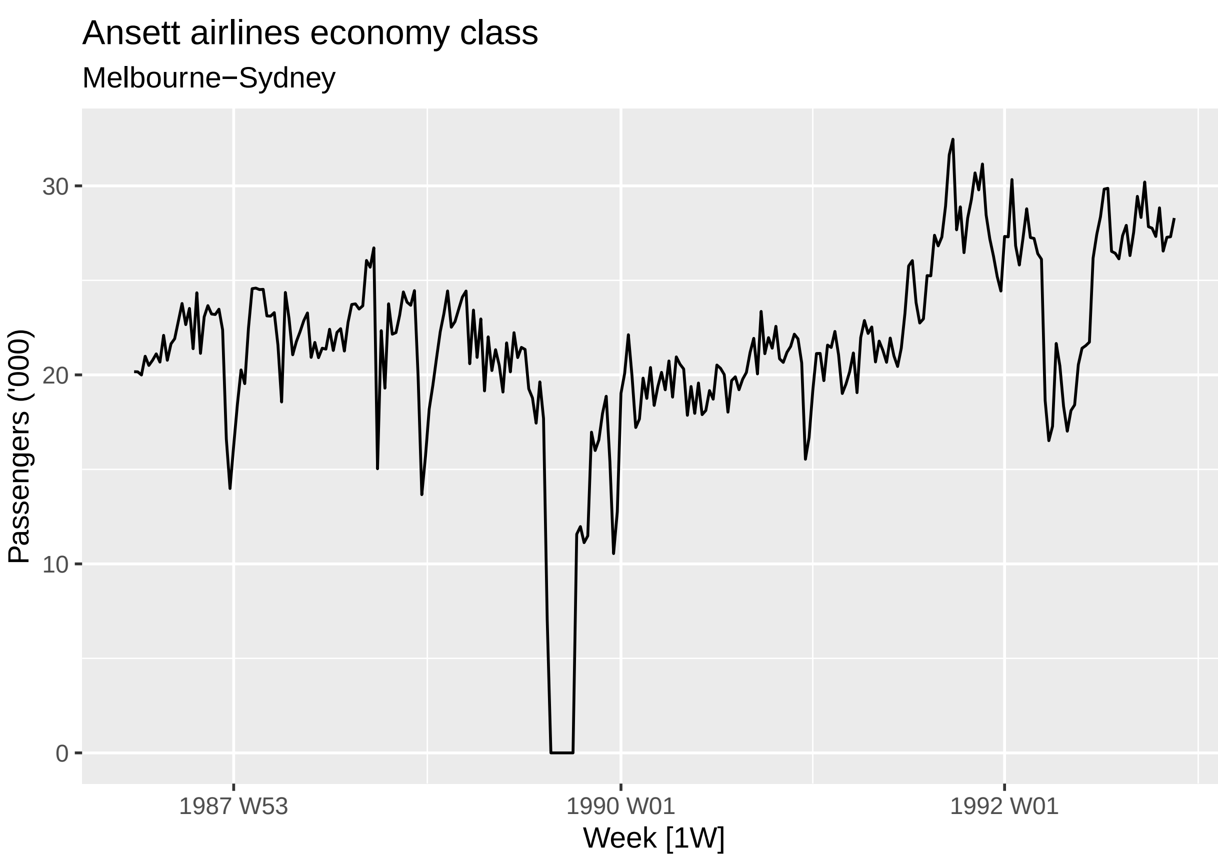
The seasonal period

The seasonal period is the number of observations before the seasonal pattern repeats. In most cases, this will be automatically detected using the time index variable.

Year > Quarters > Months > Weeks > Days > Hours > Minutes > Seconds

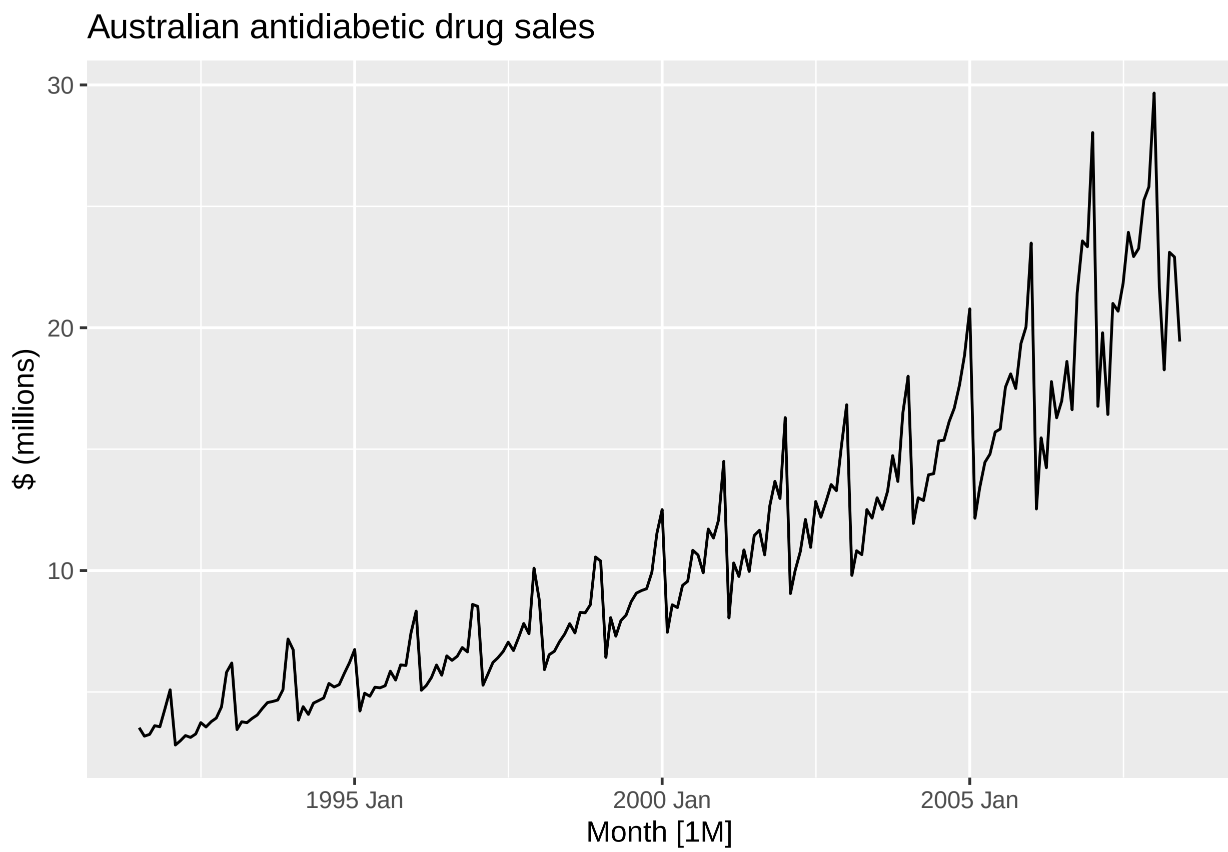
2.2 Time plots

```
melsyd_economy <- ansett %>%
  filter(Airports == "MEL-SYD", Class == "Economy") %>%
    mutate(Passengers = Passengers/1000)

autoplot(melsyd_economy, Passengers) +
    labs(title = "Ansett airlines economy class",
       subtitle = "Melbourne-Sydney",
       y = "Passengers ('000)")
```



The autoplot() command automatically produces an appropriate plot of whatever you pass to it in the first argument. In this case, it recognises melsyd_economy as a time series and produces a time plot.

```r
autoplot(a10, Cost) +
  labs(y = "$ (millions)",
       title = "Australian antidiabetic drug sales")
```



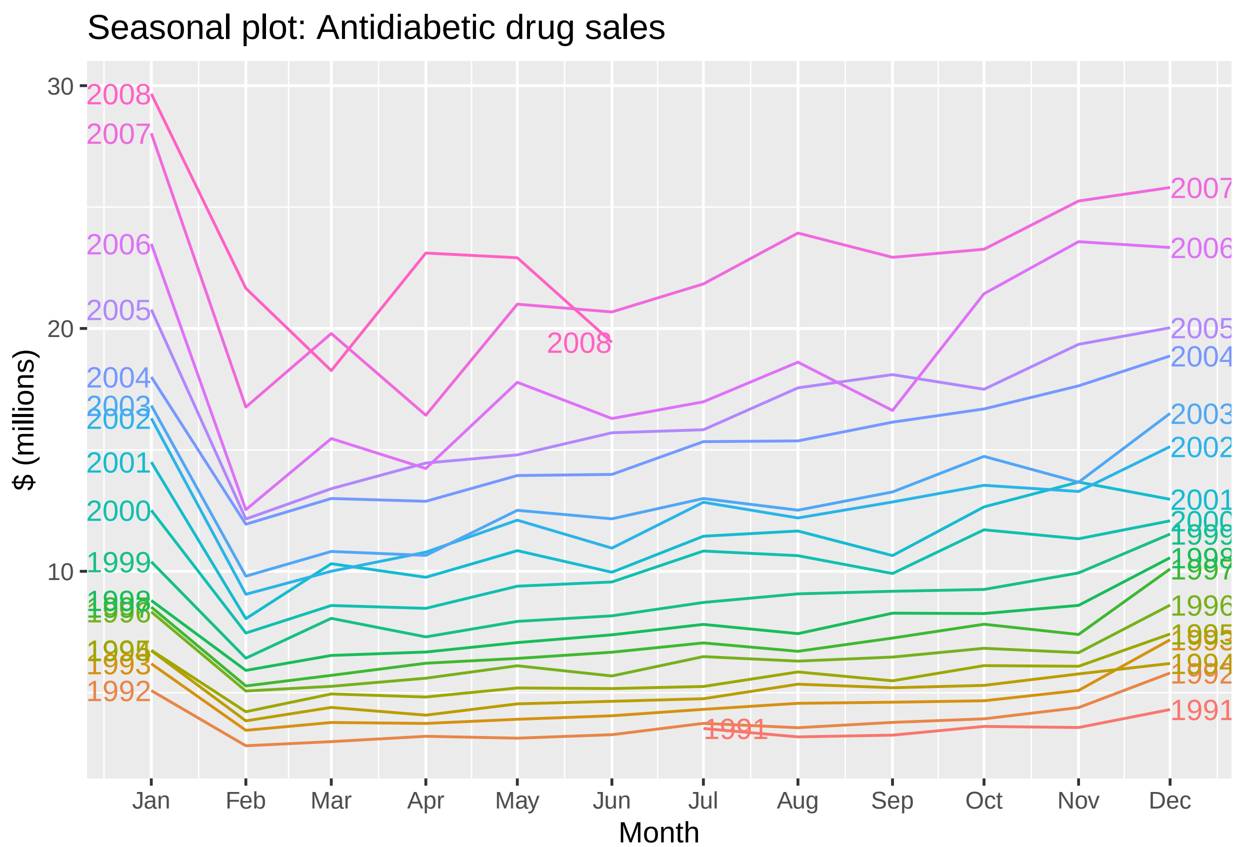
2.3 Time series patterns

- Trend. A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes we will refer to a trend as "changing direction."
- Seasonal. A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known period.
- Cyclic. A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions, and are often related to the "business cycle." The duration of these fluctuations is usually at least 2 years. In general, the average length of cycles is longer than the length of a seasonal pattern, and the magnitudes of cycles tend to be more variable than the magnitudes of seasonal patterns.

2.4 Seasonal plots

Is similar to a time plot except that the data are plotted against the individual "seasons" in which the data were observed.

```
a10 %>%
  gg_season(Cost, labels = "both") +
  labs(y = "$ (millions)",
       title = "Seasonal plot: Antidiabetic drug sales") +
  expand_limits(x = ymd(c("1972-12-28", "1973-12-04")))
```

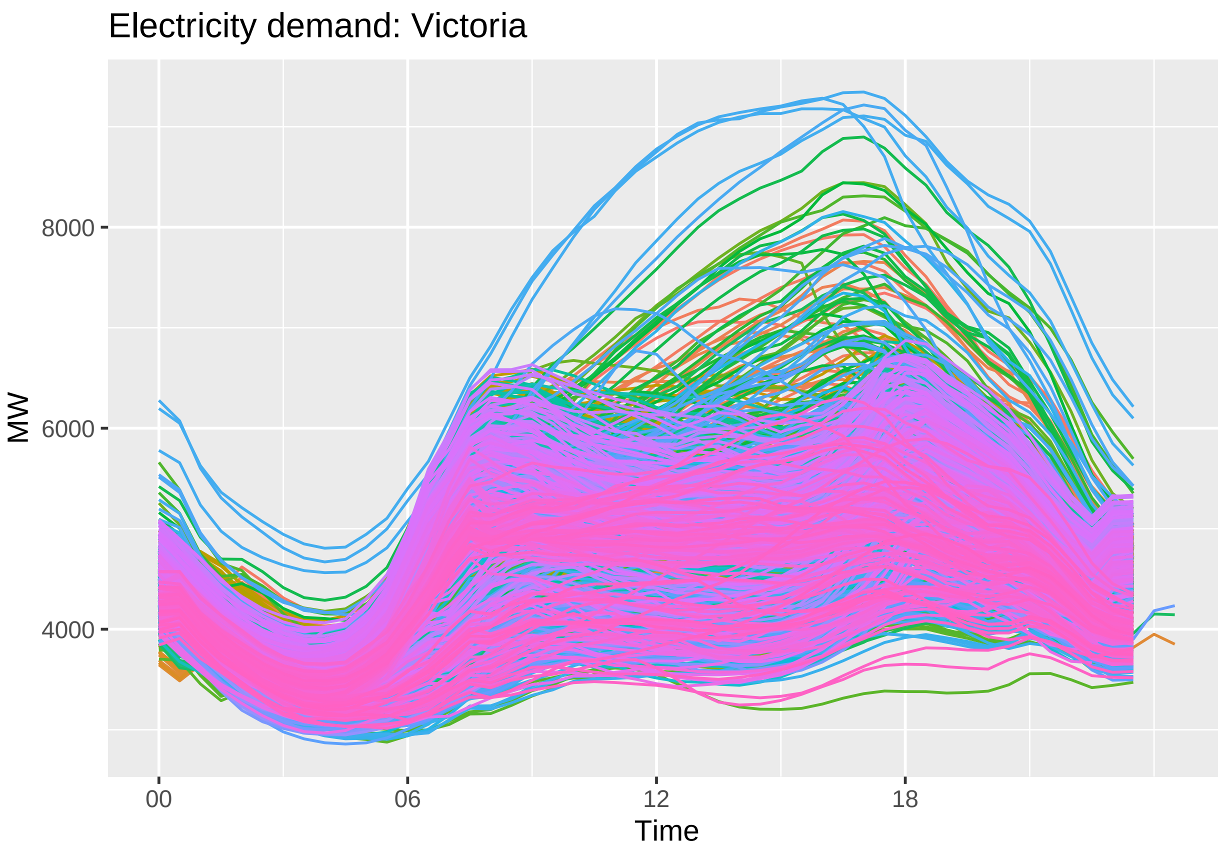


Multiple seasonal periods

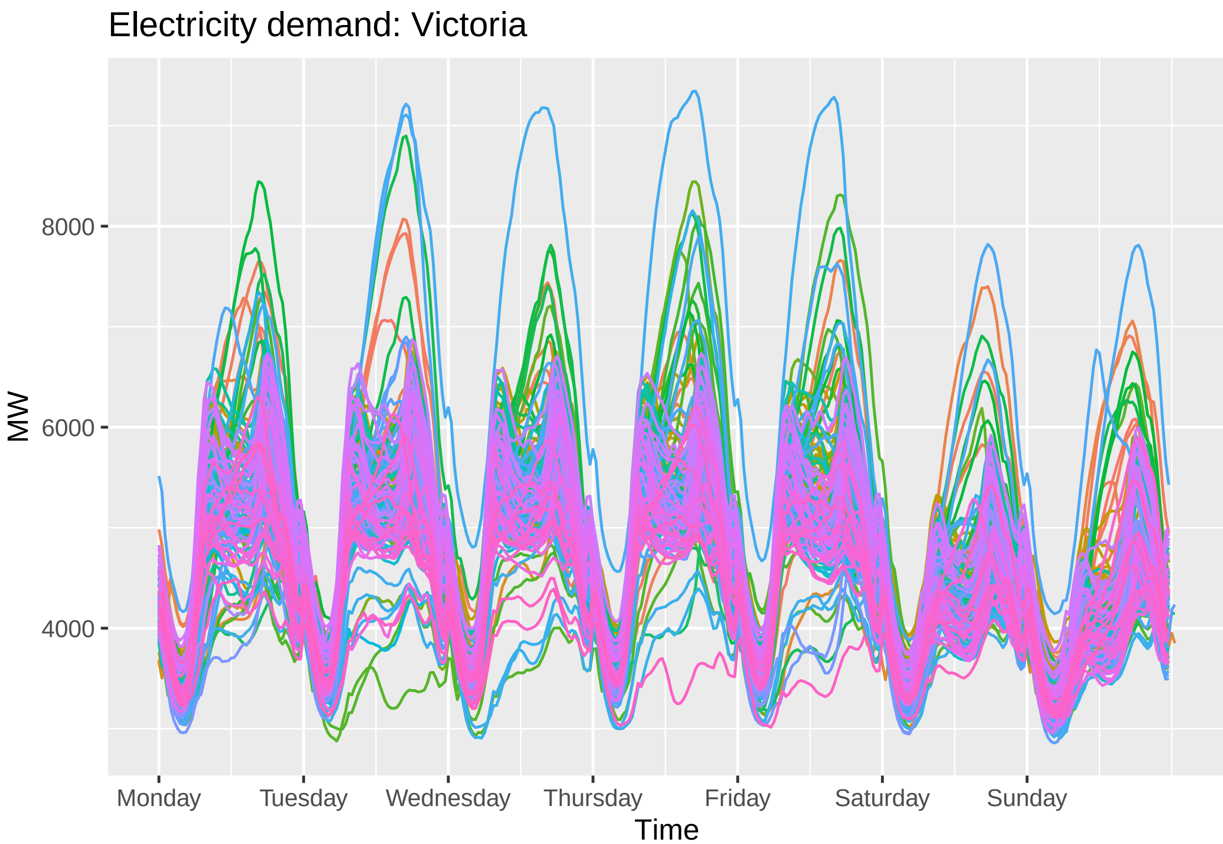
```
vic_elec
```

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

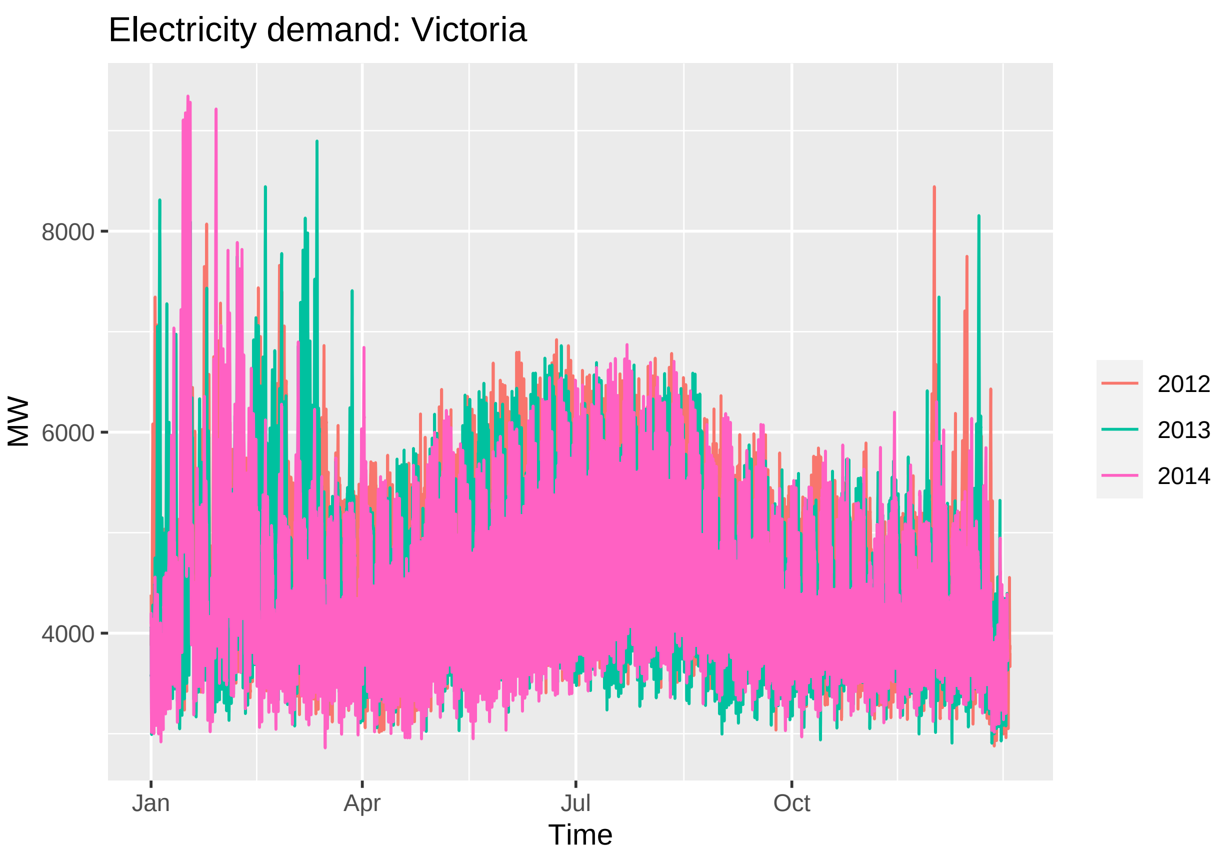
```r
vic_elec %>% gg_season(Demand, period = "day") +
  theme(legend.position = "none") +
  labs(y="MW", title="Electricity demand: Victoria")
```




```r
vic_elec %>% gg_season(Demand, period = "week") +
  theme(legend.position = "none") +
  labs(y="MW", title="Electricity demand: Victoria")
```

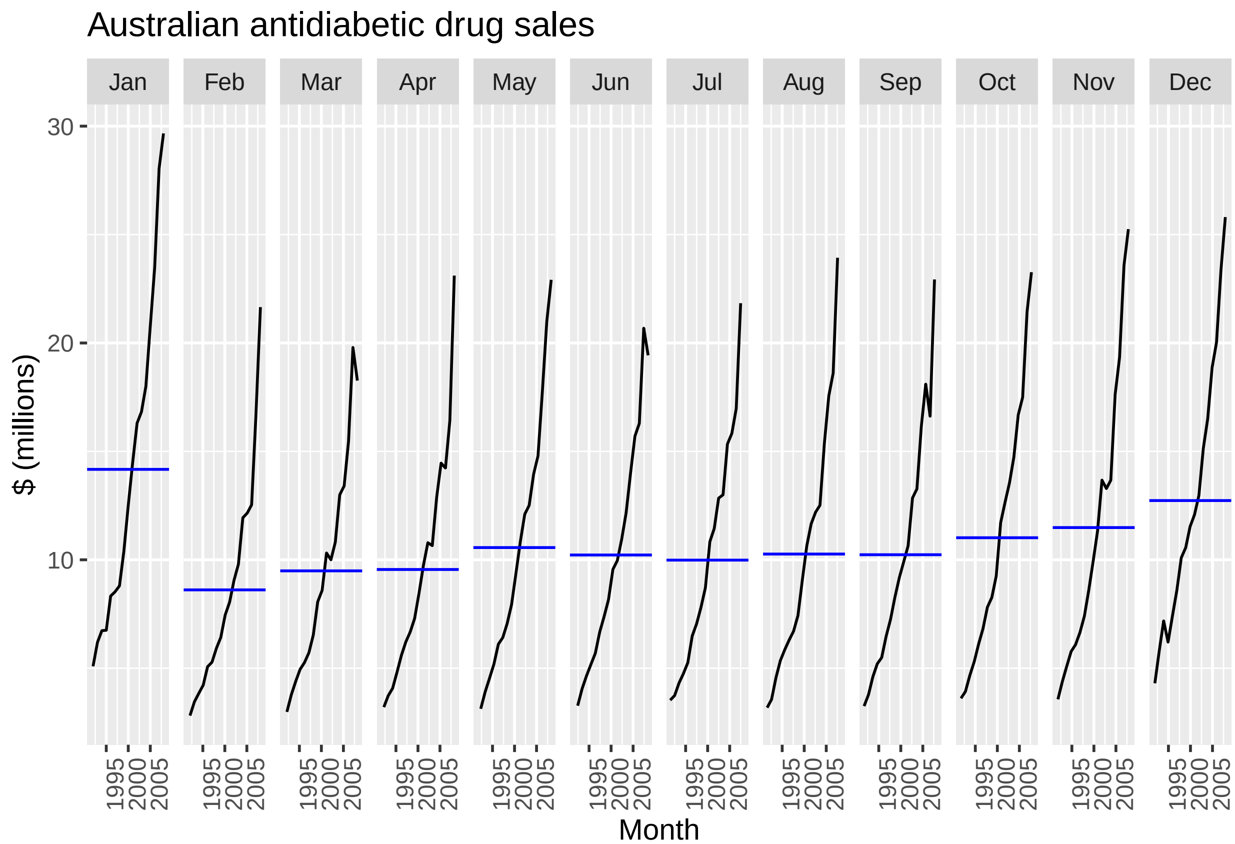



```
vic_elec %>% gg_season(Demand, period = "year") +
  labs(y="MW", title="Electricity demand: Victoria")
```



2.5 Seasonal subseries plots

```
a10 %>%
  gg_subseries(Cost) +
  labs(
    y = "$ (millions)",
    title = "Australian antidiabetic drug sales"
  )
```

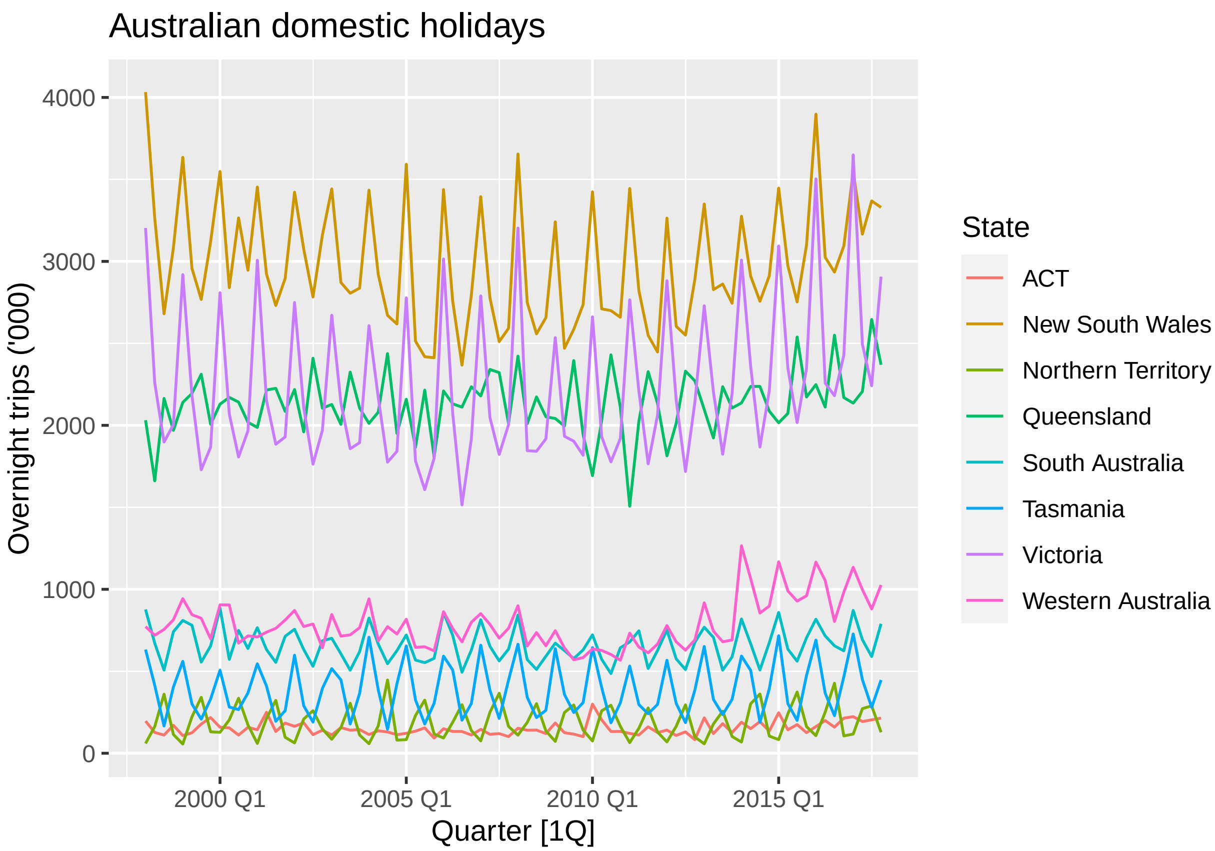


Example: Australian holiday tourism

```
(holidays <- tourism %>%
  filter(Purpose == "Holiday") %>%
  group_by(State) %>%
  summarise(Trips = sum(Trips)))
```

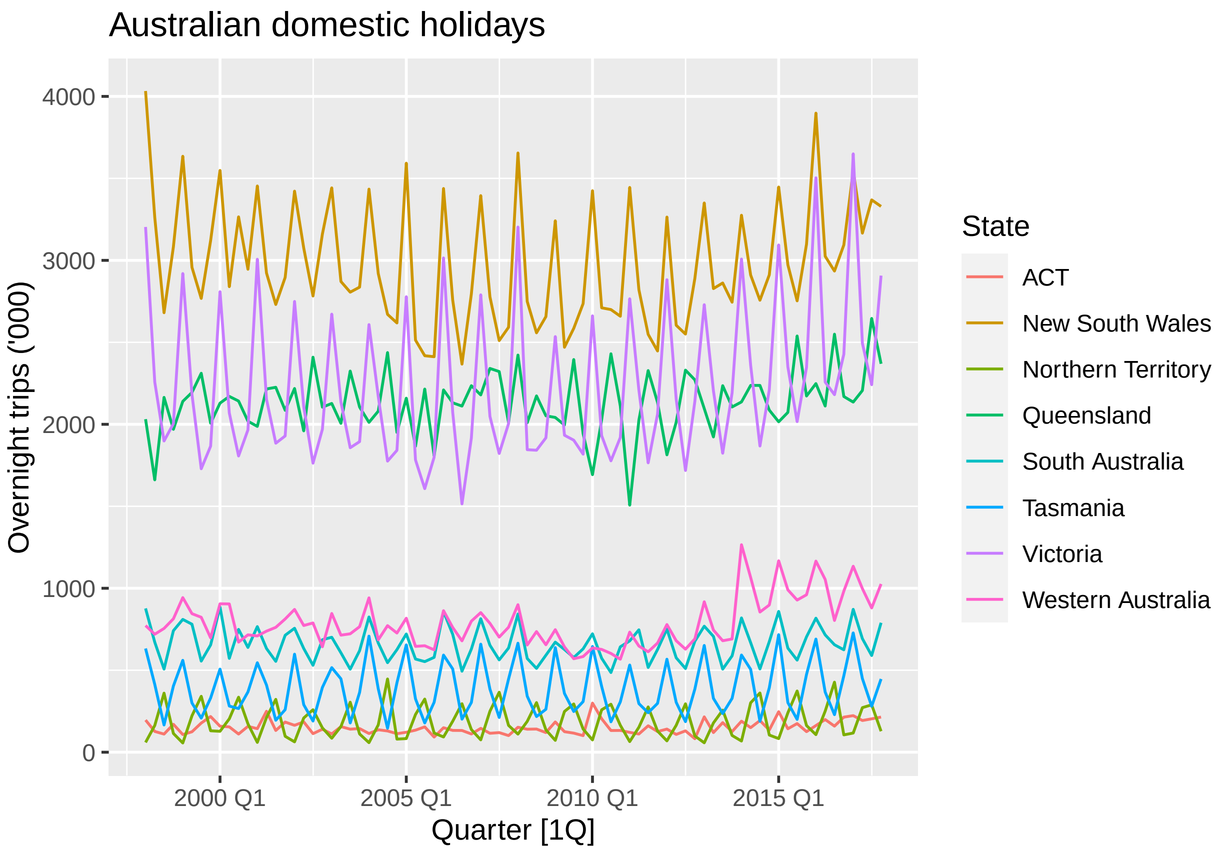
```
## # A tsibble: 640 x 3 [1Q]
## # Key:       State [8]
##    State Quarter Trips
##    <chr>   <qtr> <dbl>
##  1 ACT   1998 Q1  196.
##  2 ACT   1998 Q2  127.
##  3 ACT   1998 Q3  111.
##  4 ACT   1998 Q4  170.
##  5 ACT   1999 Q1  108.
##  6 ACT   1999 Q2  125.
##  7 ACT   1999 Q3  178.
##  8 ACT   1999 Q4  218.
##  9 ACT   2000 Q1  158.
## 10 ACT   2000 Q2  155.
## # ... with 630 more rows
```

```r
autoplot(holidays, Trips) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```

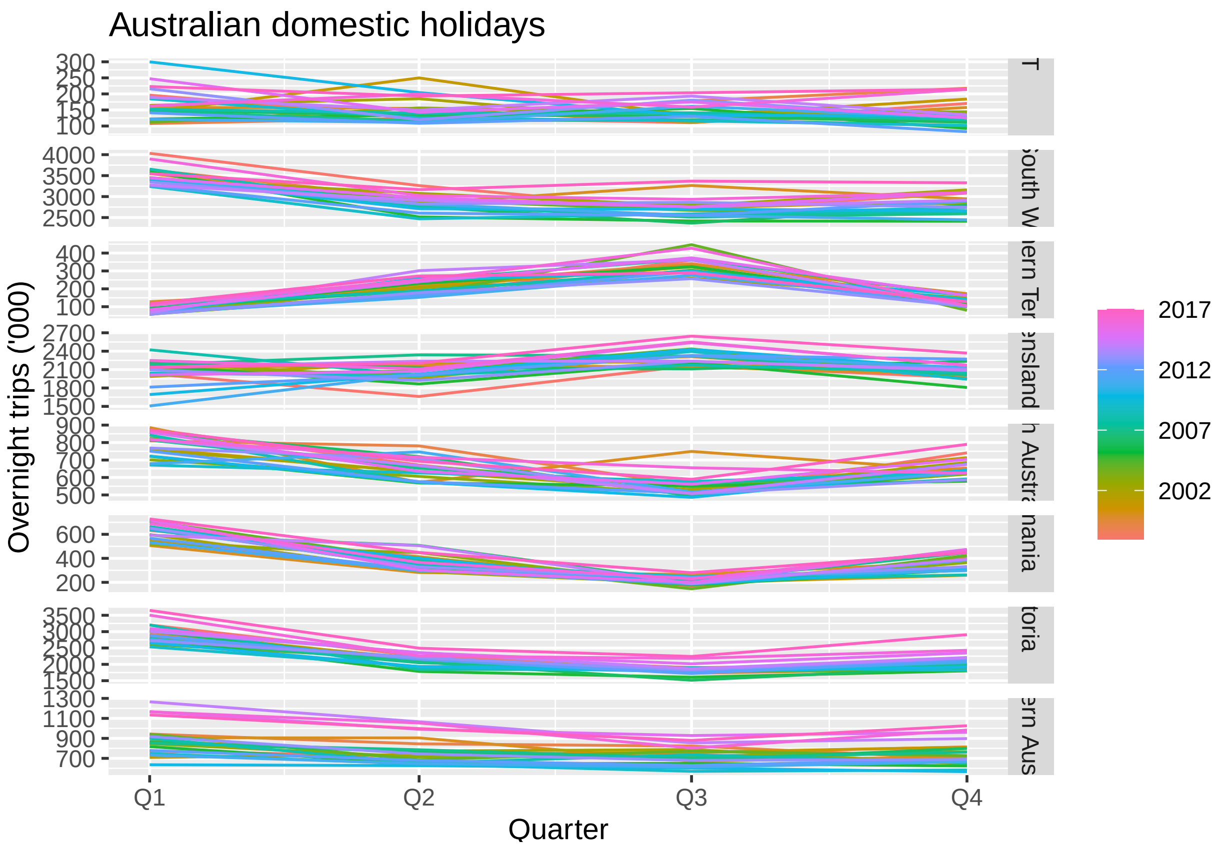



```r
autoplot(holidays) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```

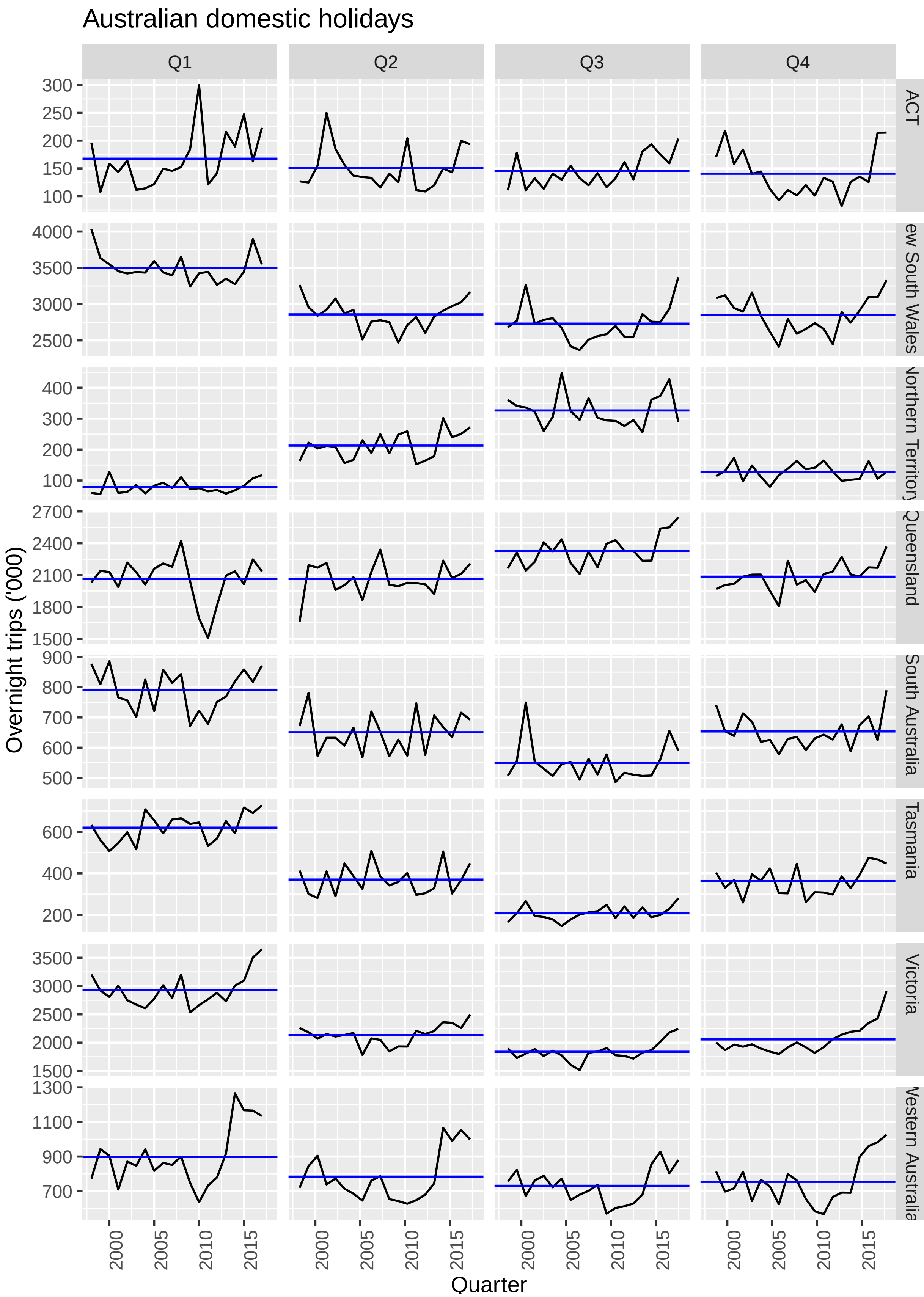
```
## Plot variable not specified, automatically selected `.vars = Trips`
```




```
gg_season(holidays, Trips) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```

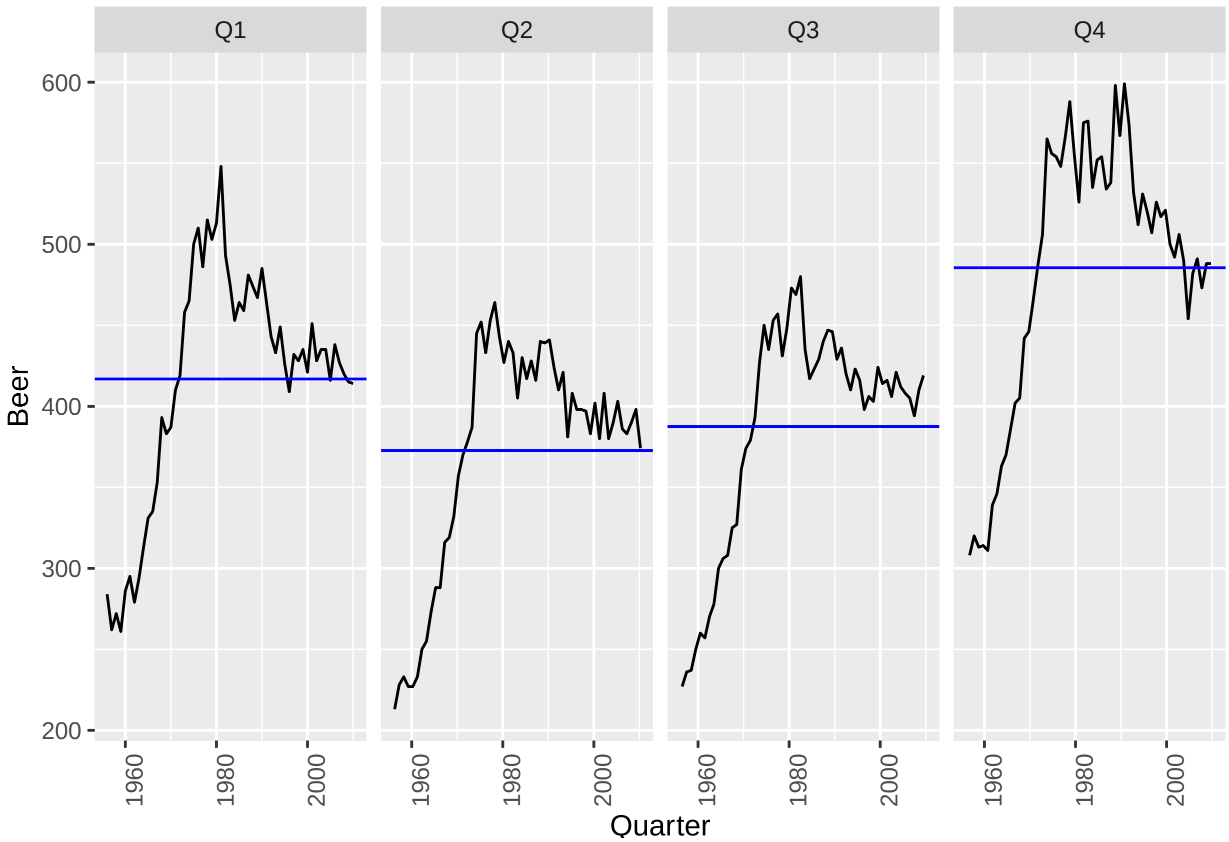



```
holidays %>%
  gg_subseries(Trips) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```


Australian domestic holidays
Q1
Q2
Q3
Q4
ACT
New South Wales
Northern Territory
Queensland
South Australia
Tasmania
Victoria
Western Australia
Overnight trips ('000)
Quarter
2000
2005
2010
2015

```
aus_production %>%
  select(Beer) %>%
  gg_subseries()
```

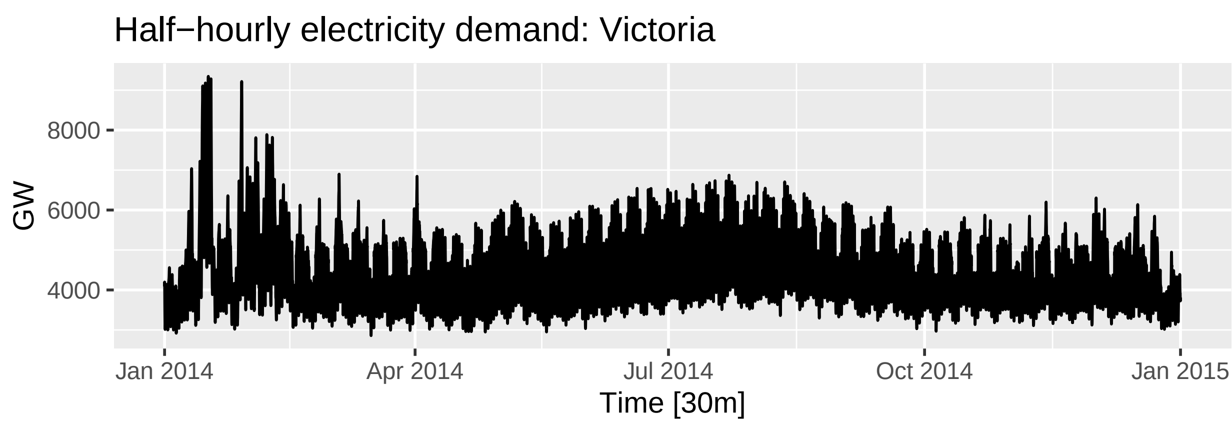
```
## Plot variable not specified, automatically selected `y = Beer`
```

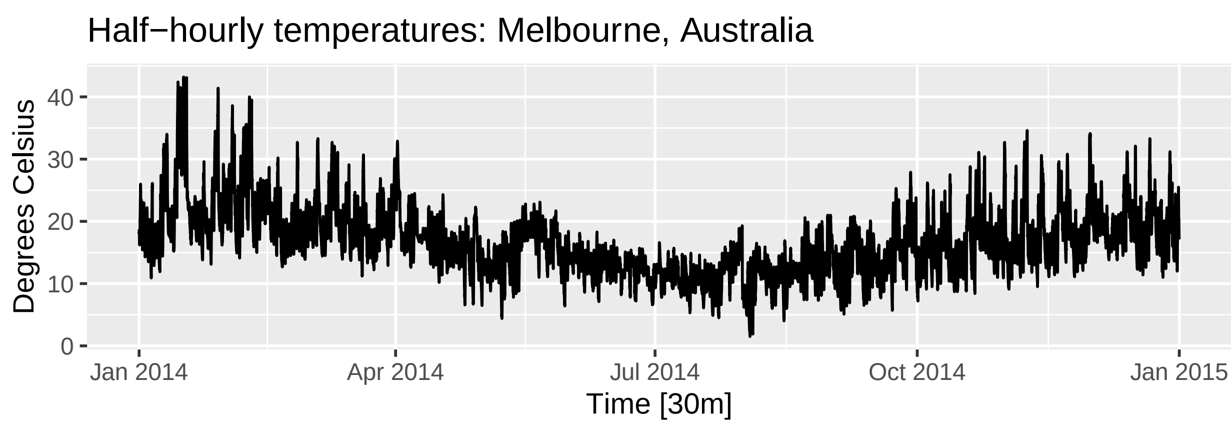


2.6 Scatterplots

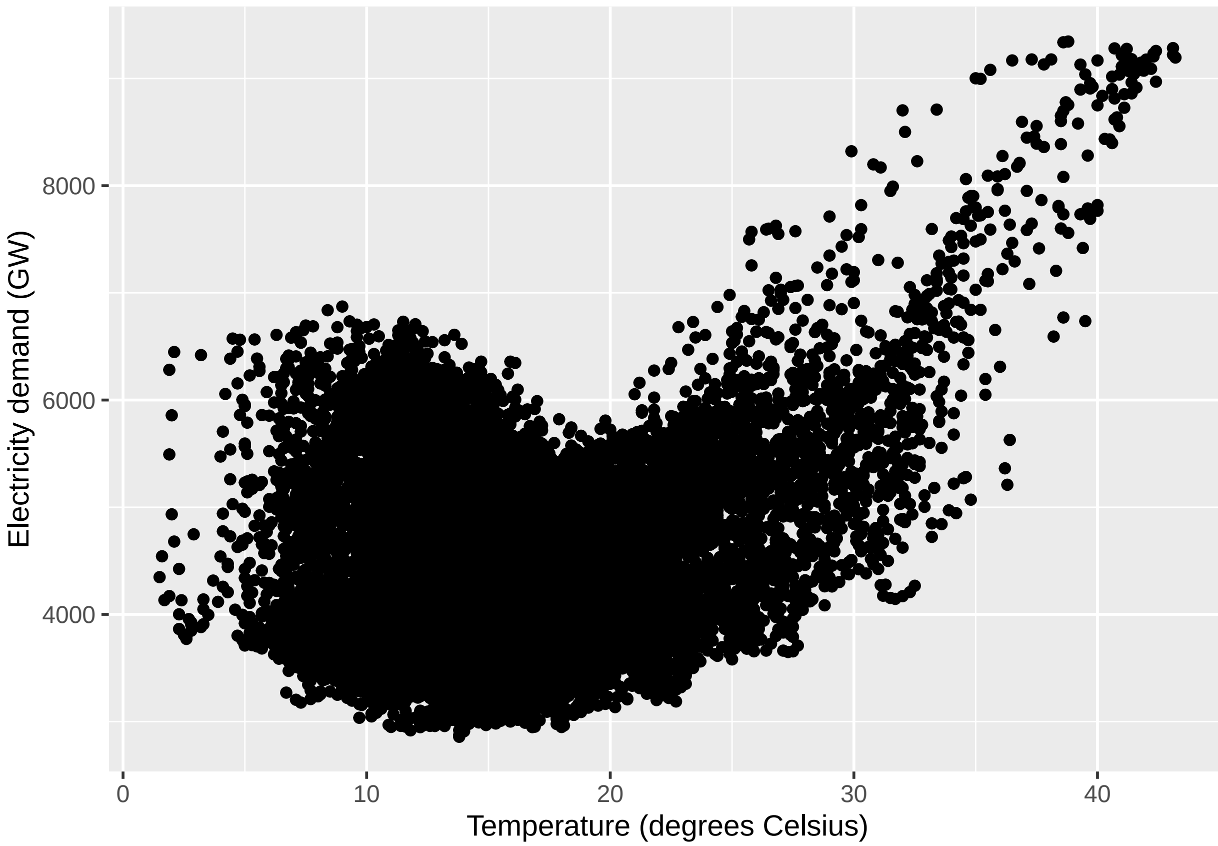
These are useful to explore relationships *between* time series.

```
vic_elec %>%
  filter(year(Time) == 2014) %>%
  autoplot(Demand) +
  labs(y = "GW",
       title = "Half-hourly electricity demand: Victoria") -> p1

p2 = vic_elec %>%
  filter(year(Time) == 2014) %>%
  autoplot(Temperature) +
  labs(
    y = "Degrees Celsius",
    title = "Half-hourly temperatures: Melbourne, Australia"
  )

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





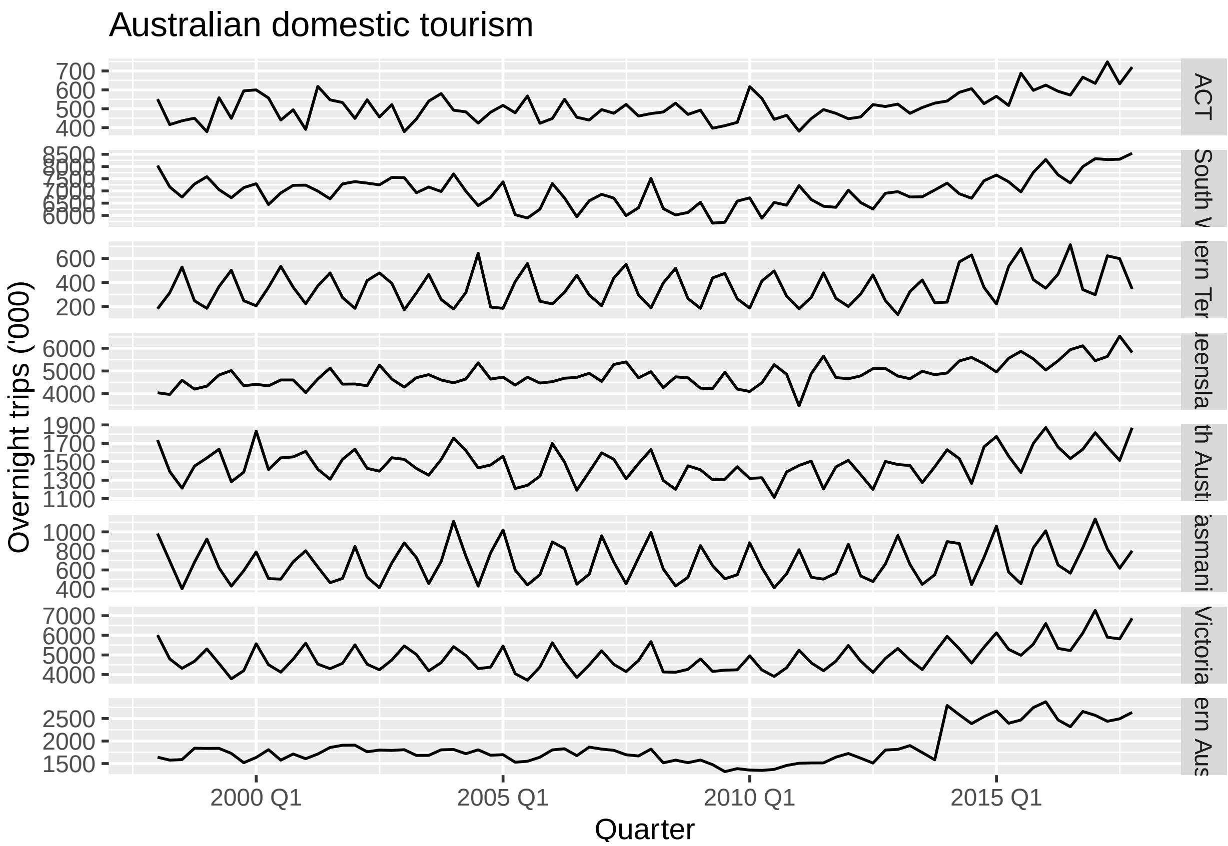

```
vic_elec %>%
  filter(year(Time) == 2014) %>%
  ggplot(aes(x = Temperature, y = Demand)) +
  geom_point() +
  labs(x = "Temperature (degrees Celsius)",
       y = "Electricity demand (GW)")
```



Correlation

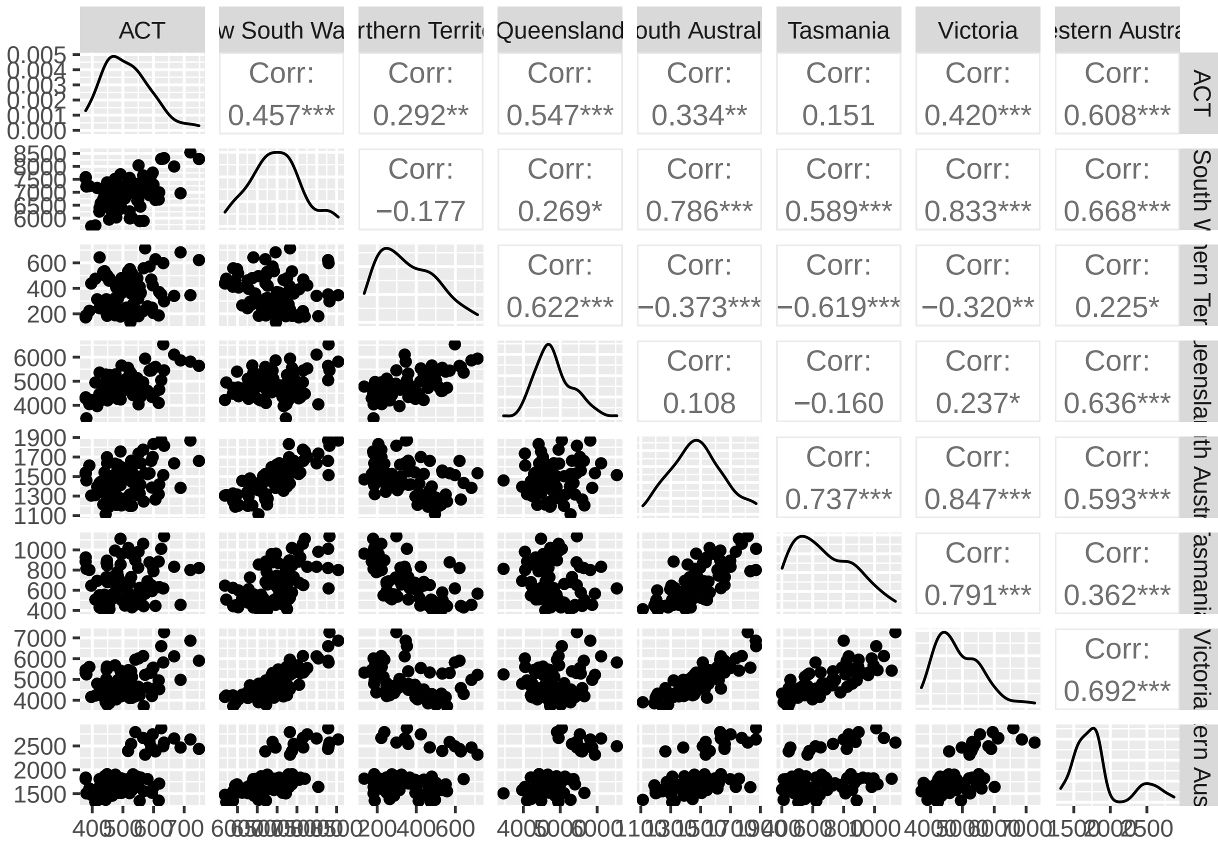
Scatterplot matrices

```
visitors <- tourism %>%
  group_by(State) %>%
  summarise(Trips = sum(Trips))
visitors %>%
  ggplot(aes(x = Quarter, y = Trips)) +
  geom_line() +
  facet_grid(vars(State), scales = "free_y") +
  labs(title = "Australian domestic tourism",
       y= "Overnight trips ('000)")
```



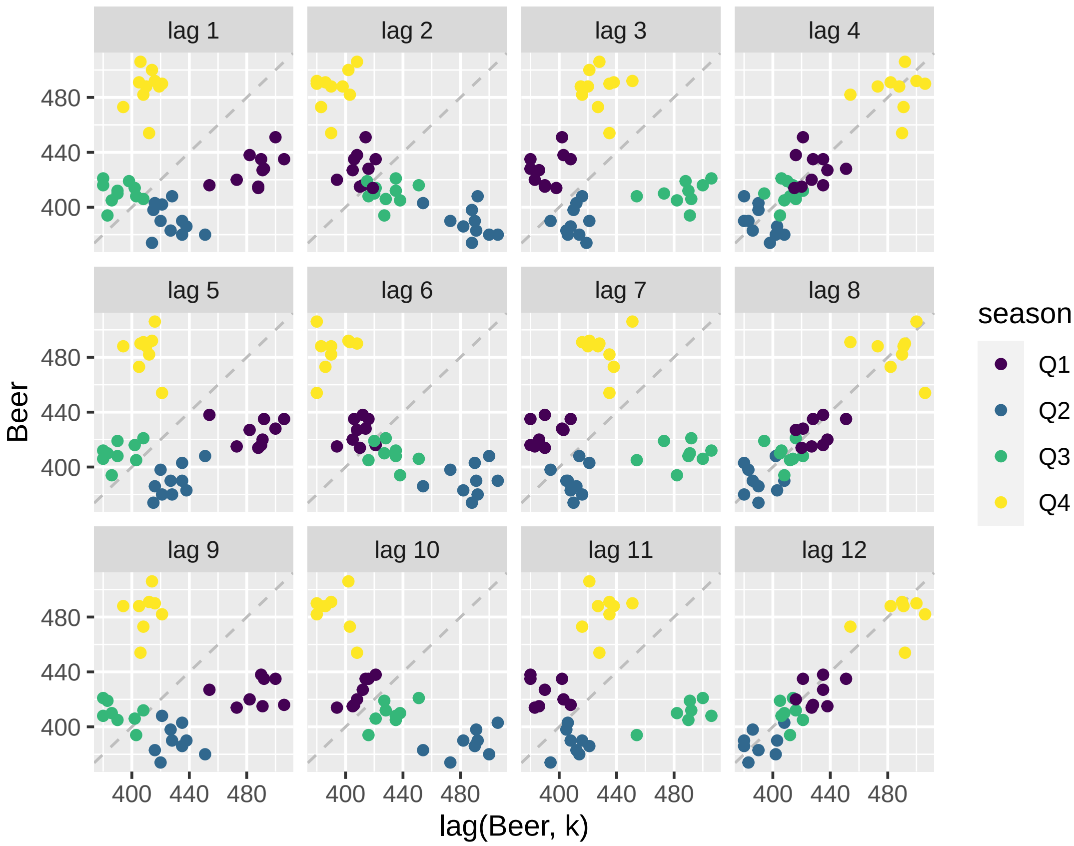

```
visitors %>%
  pivot_wider(values_from=Trips, names_from=State) %>%
  GGally::ggpairs(columns = 2:9, progress =FALSE)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```



2.7 Lag plots

```r
recent_production <- aus_production %>%
  filter(year(Quarter) >= 2000)
recent_production %>%
  gg_lag(Beer, geom = "point", lags = 1:12) +
  labs(x = "lag(Beer, k)")
```



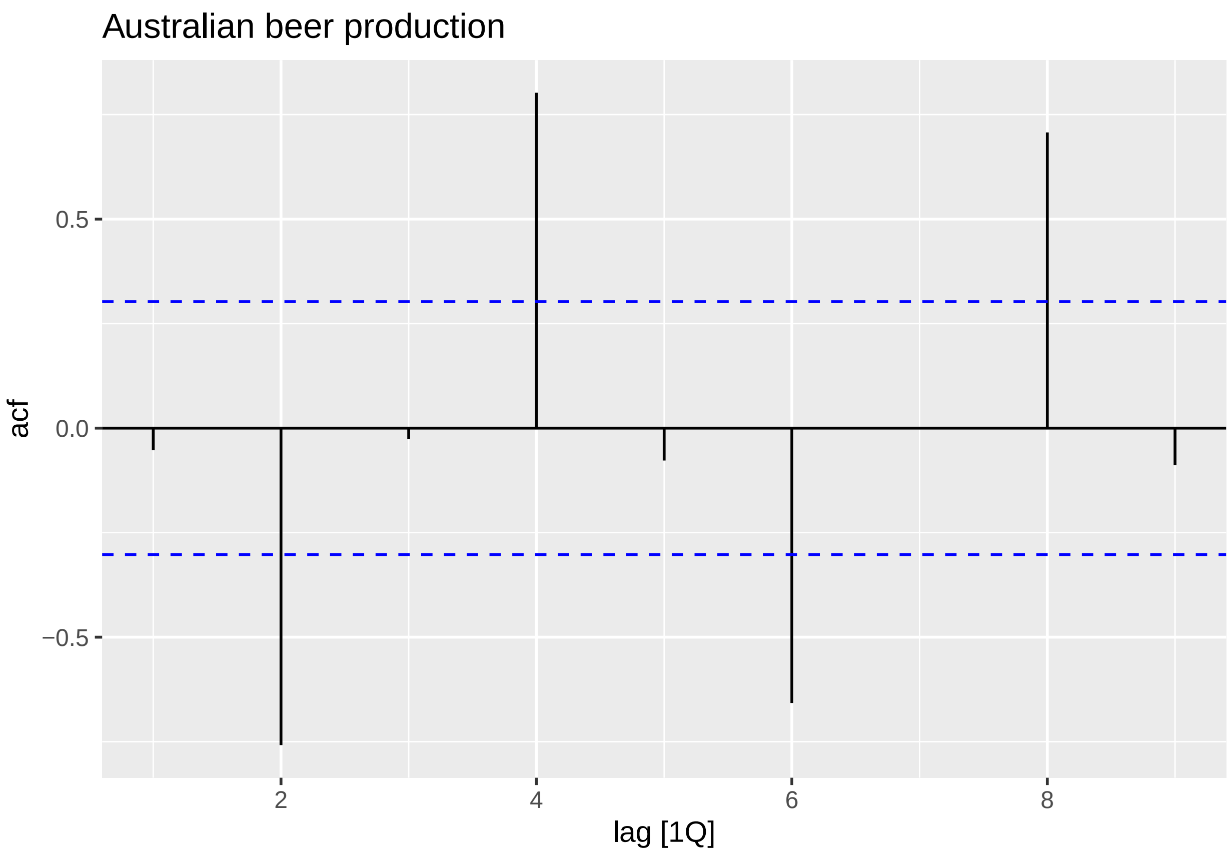
2.8 Autocorrelation

```
recent_production %>%
  ACF(Beer, lag_max = 9) %>%
  arrange(desc(acf))
```

```
## Warning: Current temporal ordering may yield unexpected results.
## i Suggest to sort by ‘‘, ‘lag‘ first.

## Warning: Current temporal ordering may yield unexpected results.
## i Suggest to sort by ‘‘, ‘lag‘ first.

## # A tsibble: 9 x 2 [1Q]
##     lag      acf
##   <lag>    <dbl>
## 1    4Q  0.802
## 2    8Q  0.707
## 3    7Q  0.00119
## 4    3Q -0.0262
## 5    1Q -0.0530
## 6    5Q -0.0775
## 7    9Q -0.0888
## 8    6Q -0.657
## 9    2Q -0.758
```

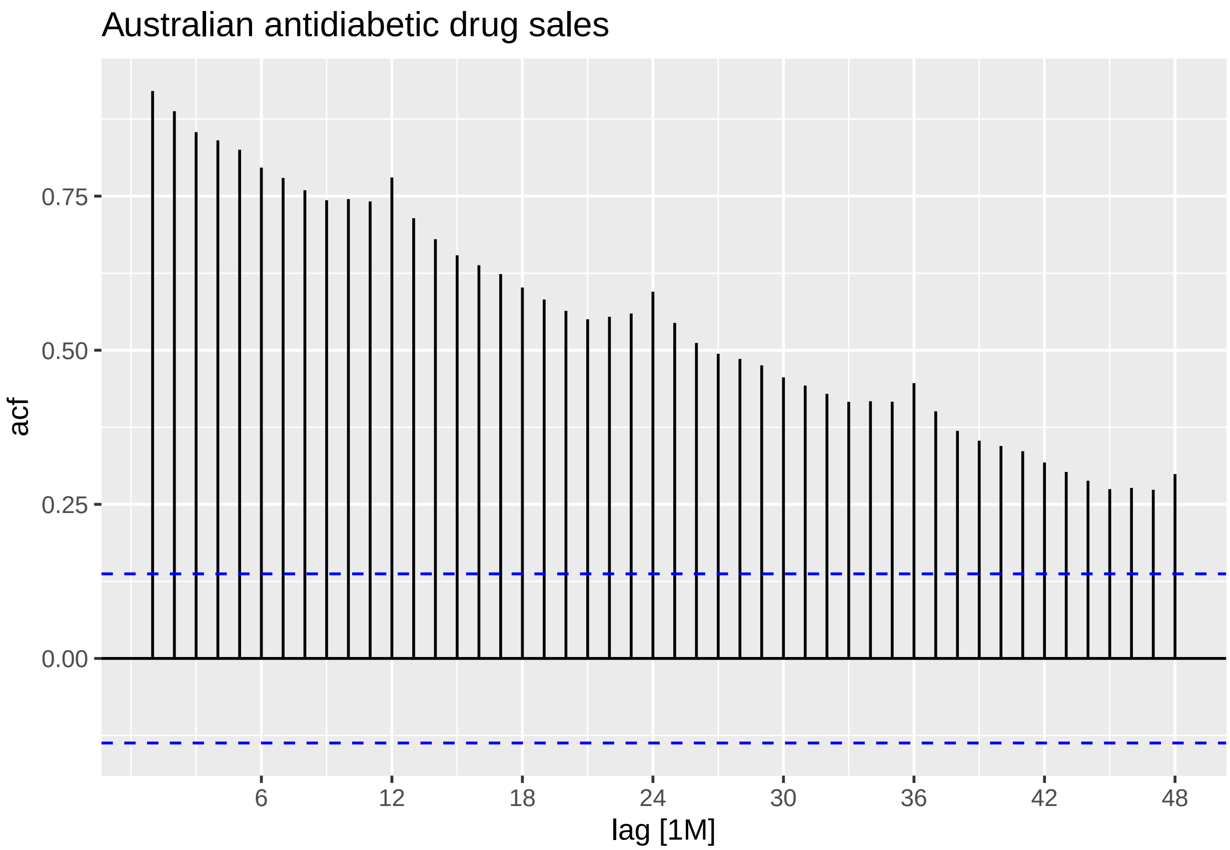
Correlogram

```
recent_production %>%
  ACF(Beer, lag_max = 9) %>%
  autoplot() + labs(title="Australian beer production")
```



Trend and seasonality in ACF plots

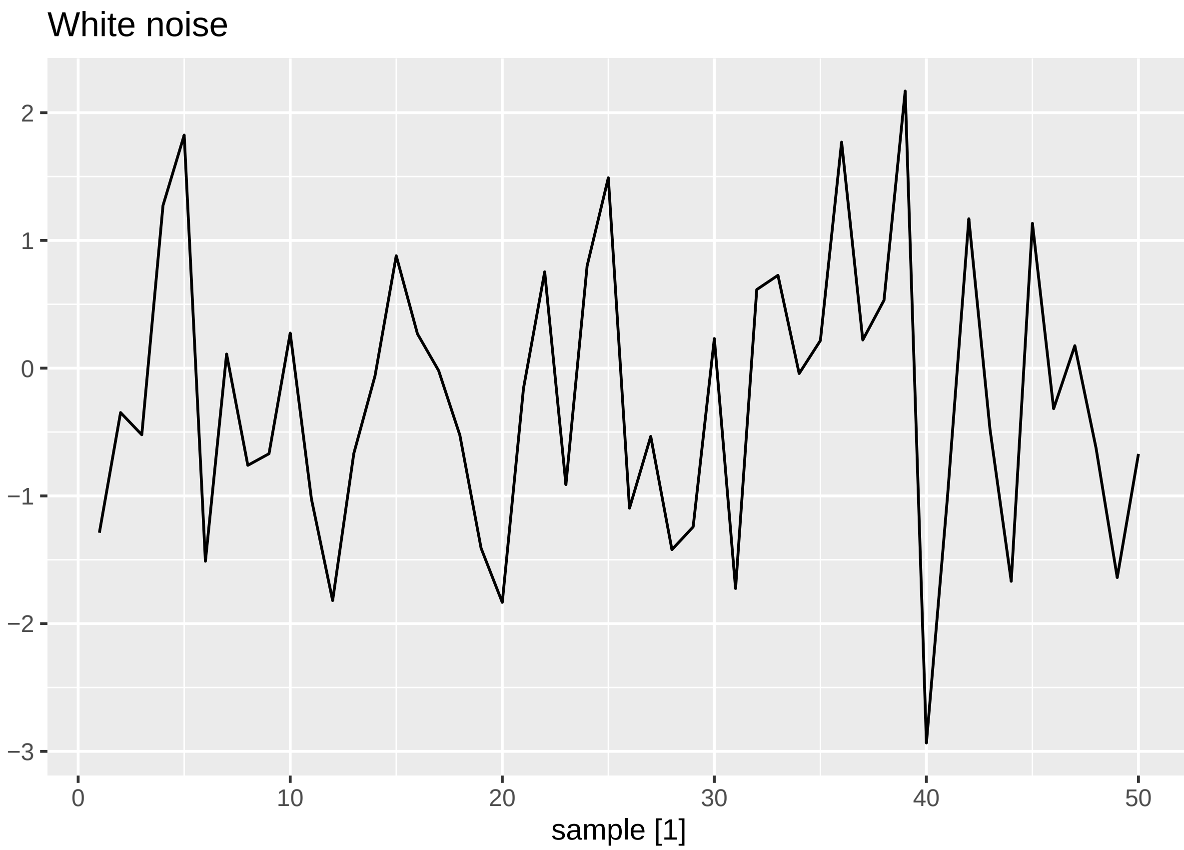
```
a10 %>%
  ACF(Cost, lag_max = 48) %>%
  autoplot() +
  labs(title="Australian antidiabetic drug sales")
```



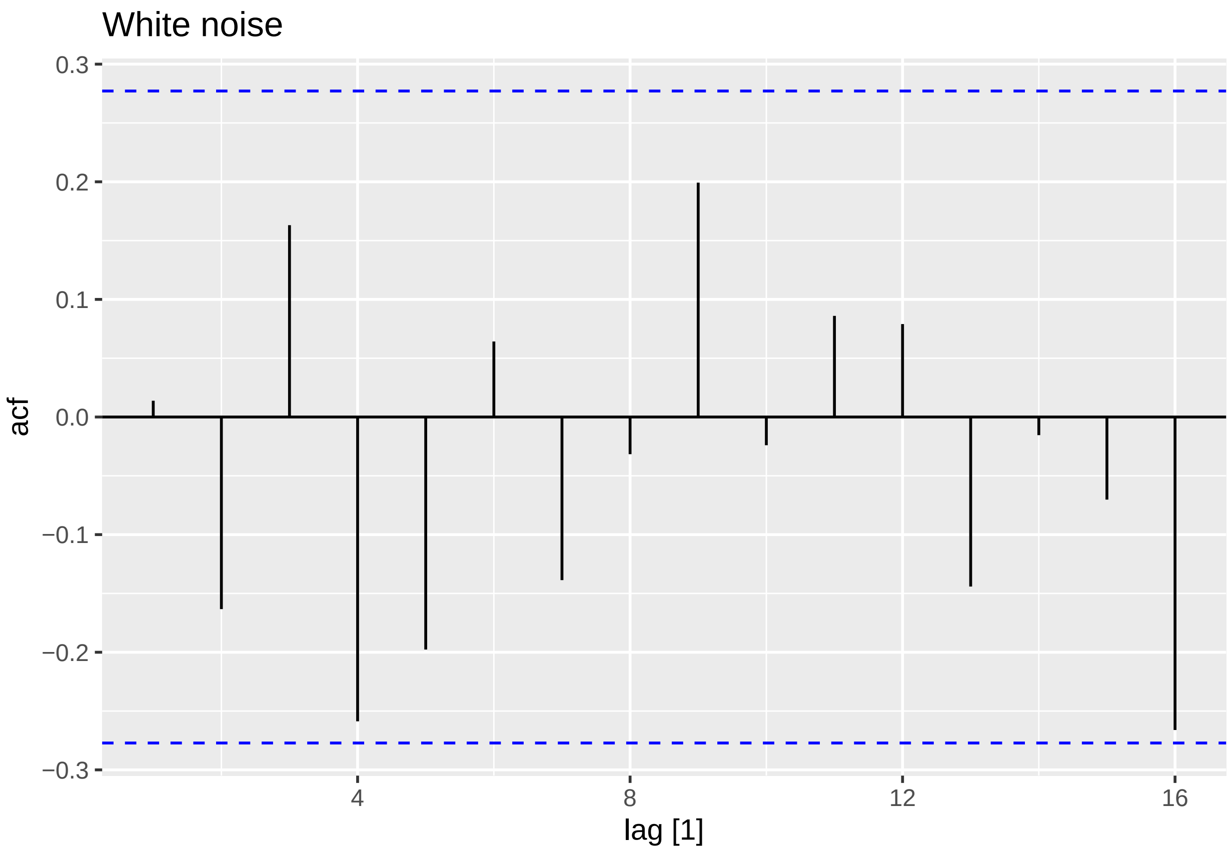
2.9 White noise

Time series that show no autocorrelation are called white noise

```
set.seed(30)
y <- tsibble(sample = 1:50, wn = rnorm(50), index = sample)
y %>% autoplot(wn) + labs(title = "White noise", y = "")
```




```
y %>%
  ACF(wn) %>%
  autoplot() + labs(title = "White noise")
```



Chapter 3 Time series decomposition

When we decompose a time series into components, we usually combine the trend and cycle into a single trend-cycle component. 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).

3.1 Transformations and adjustments

Calendar adjustments

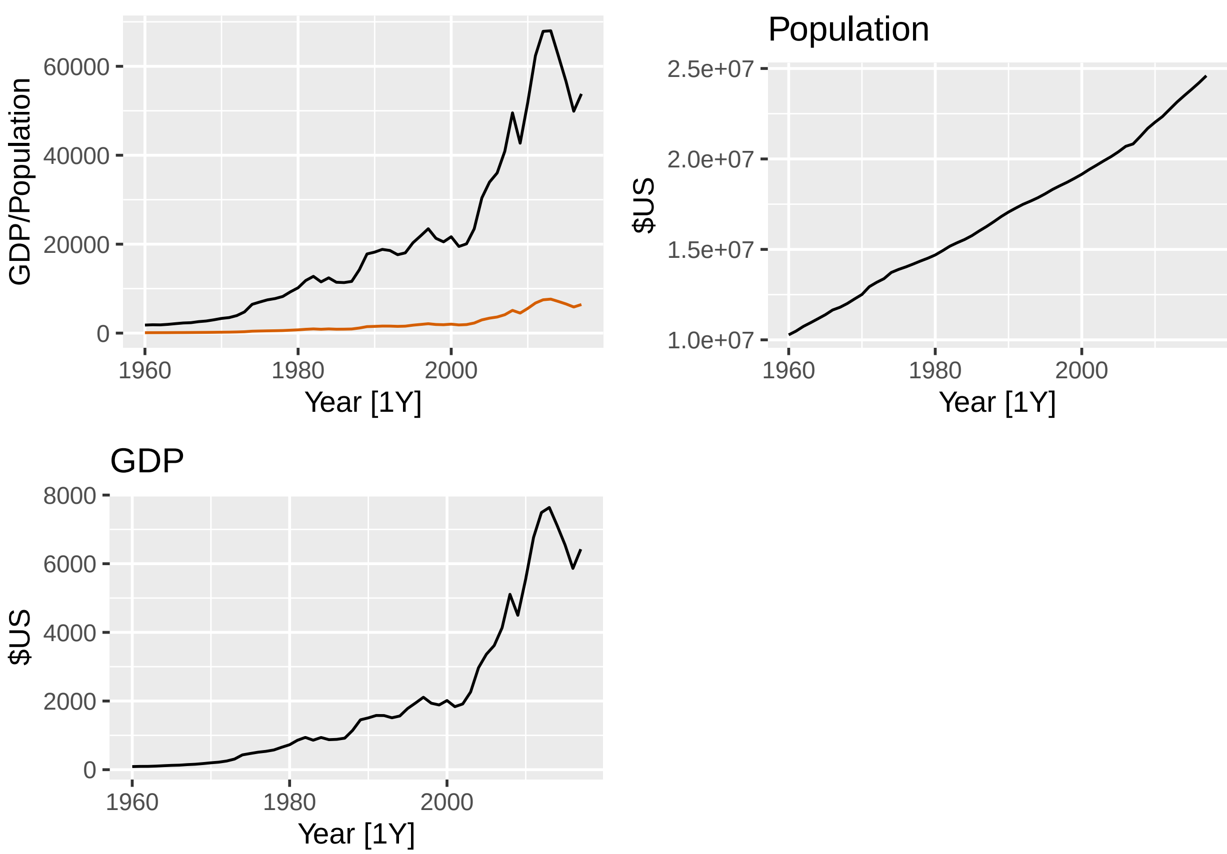
Population adjustments

```
meanPop = mean(global_economy$Population, na.rm = TRUE)

p1 = global_economy %>%
  filter(Country == "Australia") %>%
  autoplot(GDP/Population) +
  geom_line(aes(y= GDP / meanPop), color = "#D55E00")
  labs(title= "GDP per capita", y = "$US")
```

```
## $y
## [1] "$US"
```

```
##
## $title
## [1] "GDP per capita"
##
## attr(,"class")
## [1] "labels"
```

```
p2 = global_economy %>%
  filter(Country == "Australia") %>%
  autoplot(GDP / meanPop) +
  labs(title= "GDP", y = "$US")

p3 = global_economy %>%
  filter(Country == "Australia") %>%
  autoplot(Population) +
  labs(title= "Population", y = "$US")

gridExtra::grid.arrange(p1, p3, p2, nrow = 2)
```

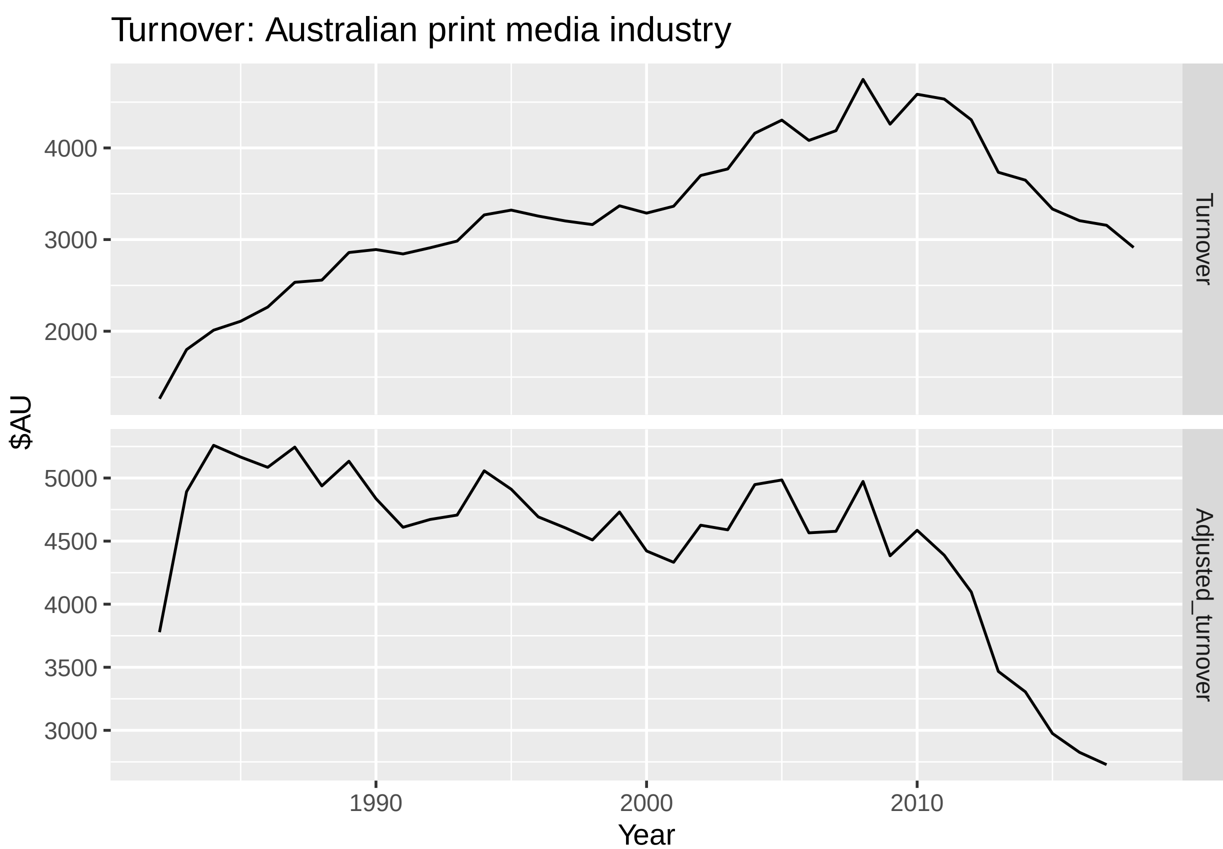


Inflation adjustments

```
print_retail <- aus_retail %>%
  filter(Industry == "Newspaper and book retailing") %>%
  group_by(Industry) %>%
```

```r
  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")
```

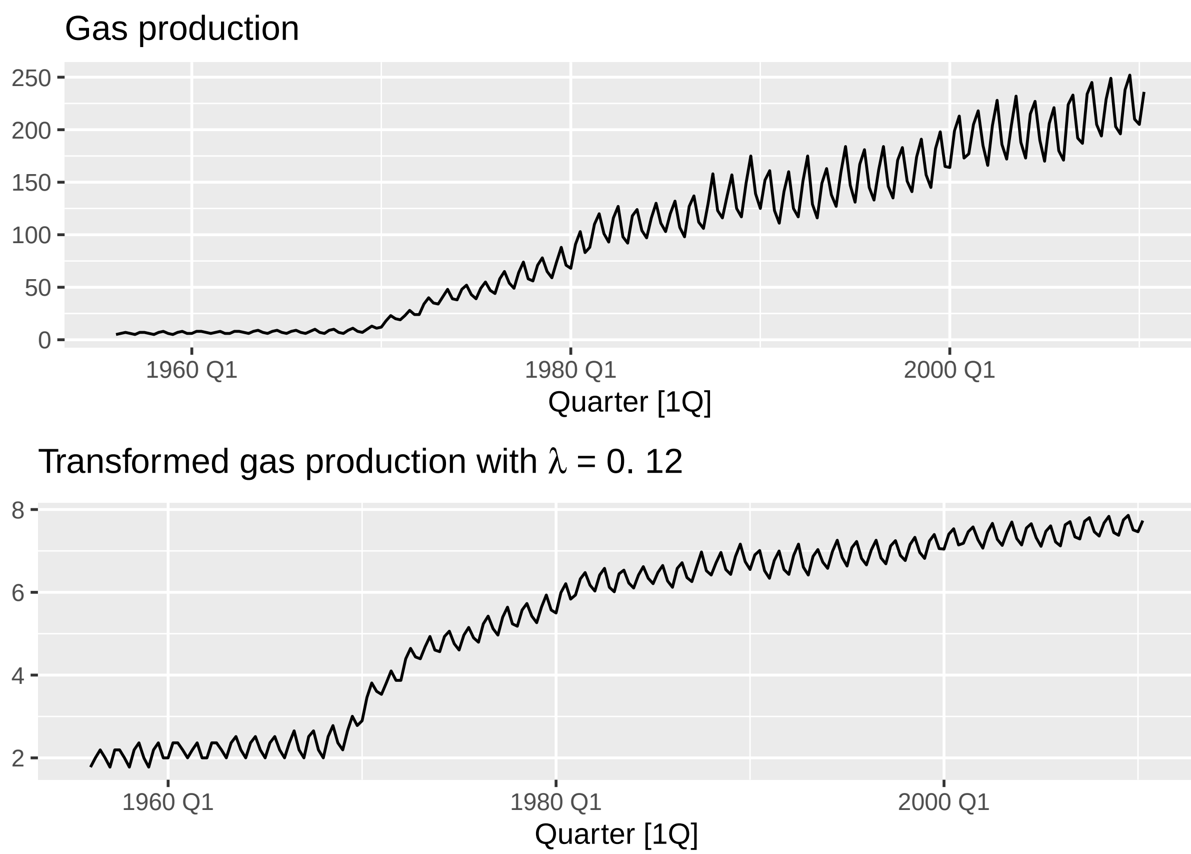
```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



Mathematical transformations

```r
p1 = aus_production %>%
  autoplot(Gas) +
  labs(y = "", title = "Gas production")
```

```
lambda <- aus_production %>%
  features(Gas, features = guerrero) %>%
  pull(lambda_guerrero)
p2 = aus_production %>%
  autoplot(box_cox(Gas, lambda)) +
  labs(y = "",
       title = latex2exp::TeX(paste0(
         "Transformed gas production with $\\lambda$ = ",
         round(lambda,2))))
gridExtra::grid.arrange(p1, p2, nrow = 2)
```



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 and R_t is the remainder.

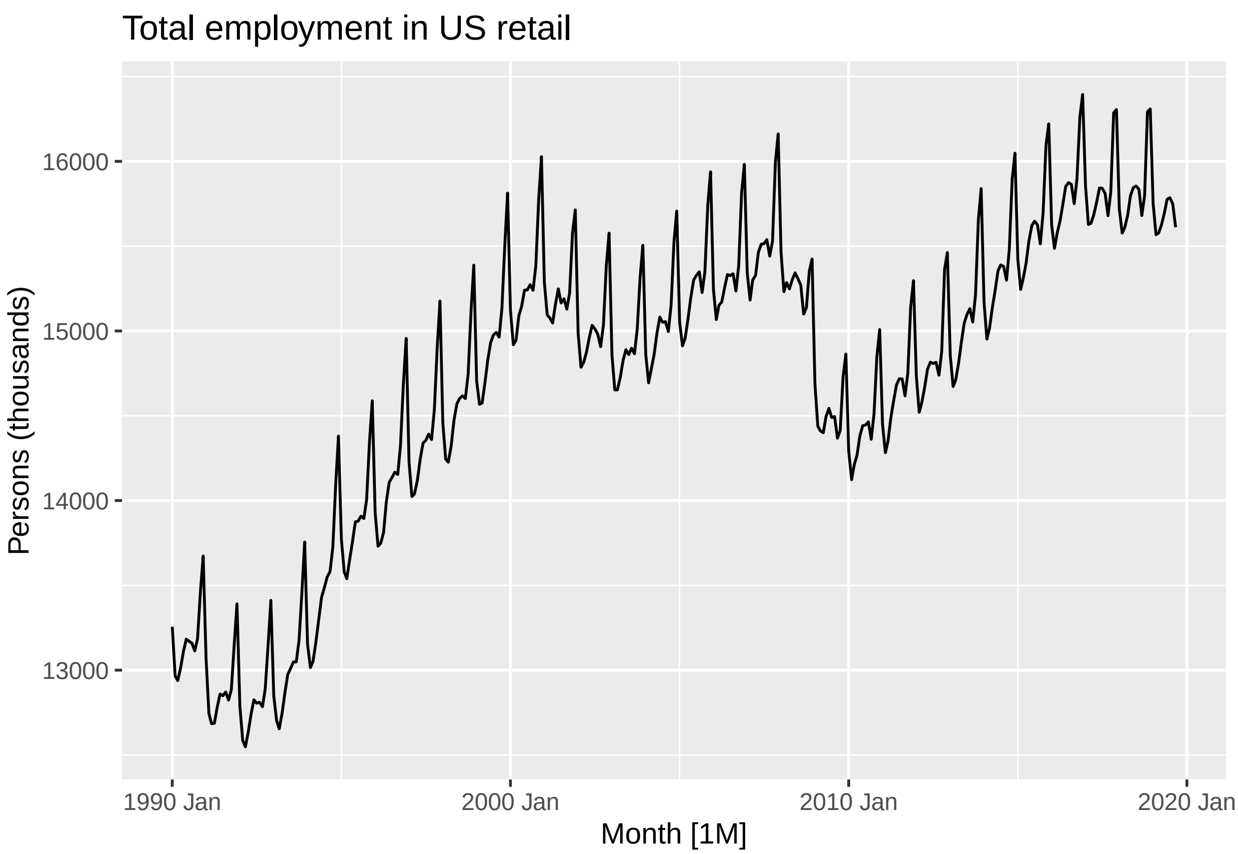
Alternatively, a multiplicative decomposition would be written as

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

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. But with a log-like transformation, multiplicative becomes additive.

Employment in the US retail sector

```
us_retail_employment <- us_employment %>%
  filter(year(Month) >= 1990, Title == "Retail Trade") %>%
  select(-Series_ID)
autoplot(us_retail_employment, Employed) +
  labs(y = "Persons (thousands)",
       title = "Total employment in US retail")
```



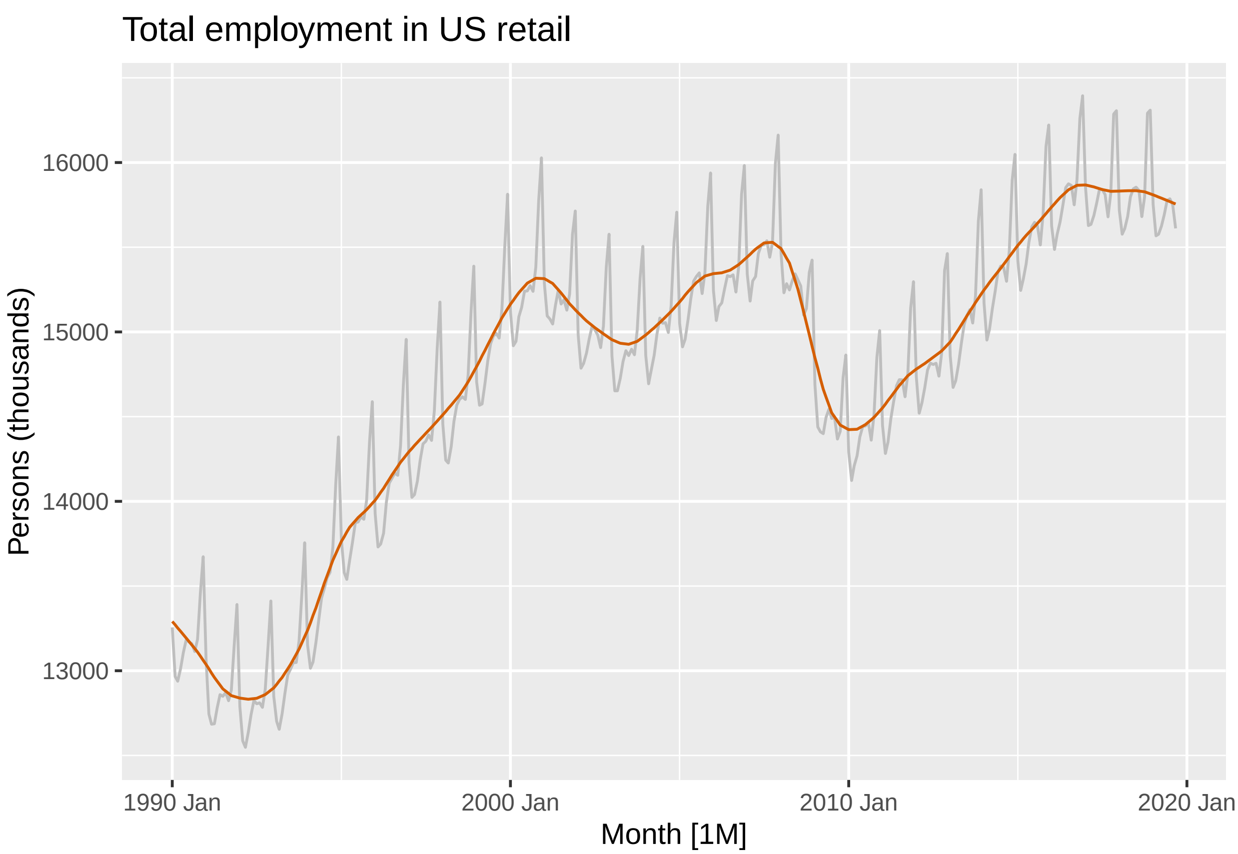

```
dcmp <- us_retail_employment %>%
  model(stl = STL(Employed))
components(dcmp)
```

```
## # 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. 13291.       -38.1    3.08          13294.
##  2 stl    1990 Feb   12966. 13272.      -261.   -44.2           13227.
##  3 stl    1990 Mar   12938. 13252.      -291.   -23.0           13229.
##  4 stl    1990 Apr   13012. 13233.      -221.     0.0892        13233.
##  5 stl    1990 May   13108. 13213.      -115.     9.98          13223.
##  6 stl    1990 Jun   13183. 13193.       -25.6   15.7           13208.
##  7 stl    1990 Jul   13170. 13173.       -24.4   22.0           13194.
```

```
## 8 stl    1990 Aug   13160. 13152.        -11.8   19.5            13171.
## 9 stl    1990 Sep   13113. 13131.        -43.4   25.7            13157.
## 10 stl   1990 Oct   13185. 13110.         62.5   12.3            13123.
## # ... with 347 more rows
```

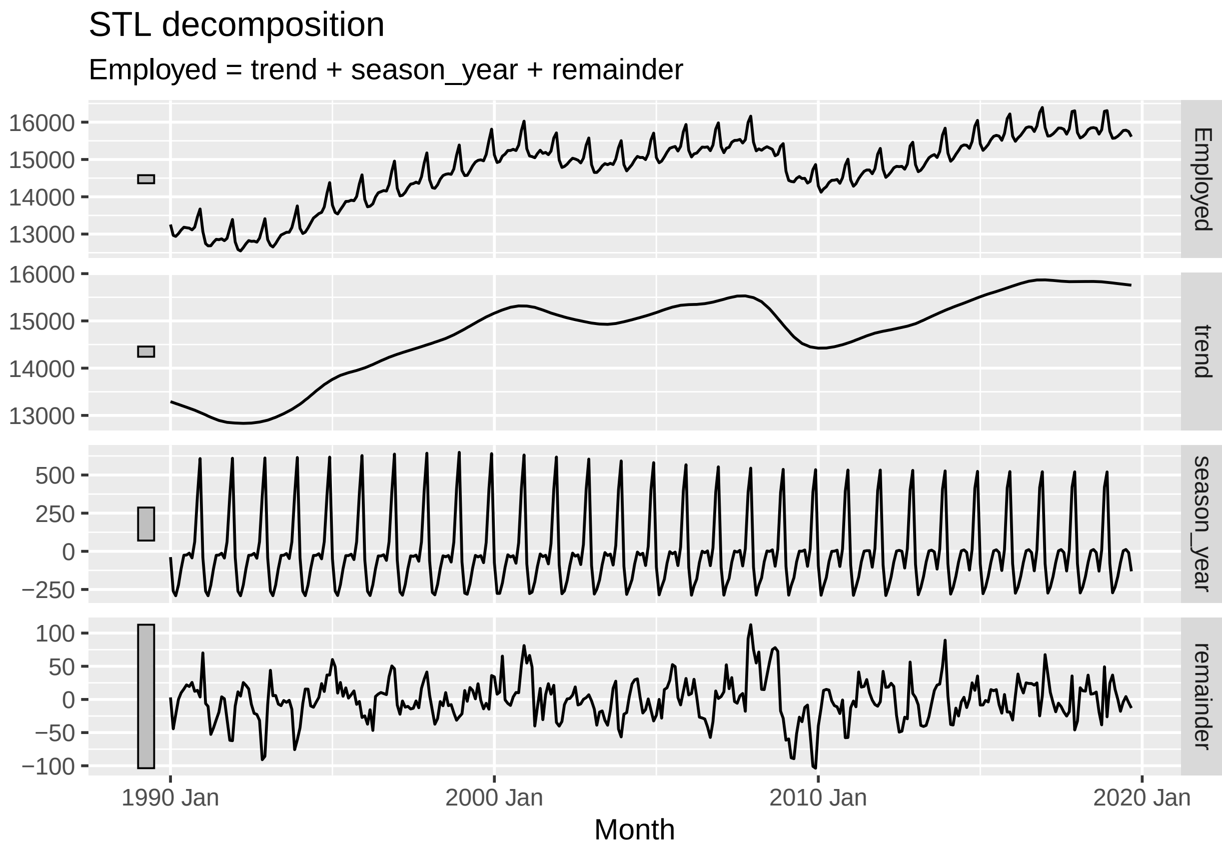
This output forms a "dable" or decomposition table. The header to the table shows that the Employed series has been decomposed additively.

```
components(dcmp) %>%
  as_tsibble() %>%
  autoplot(Employed, color="gray") +
  geom_line(aes(y=trend), color = "#D55E00") +
  labs(
    y = "Persons (thousands)",
    title = "Total employment in US retail"
  )
```



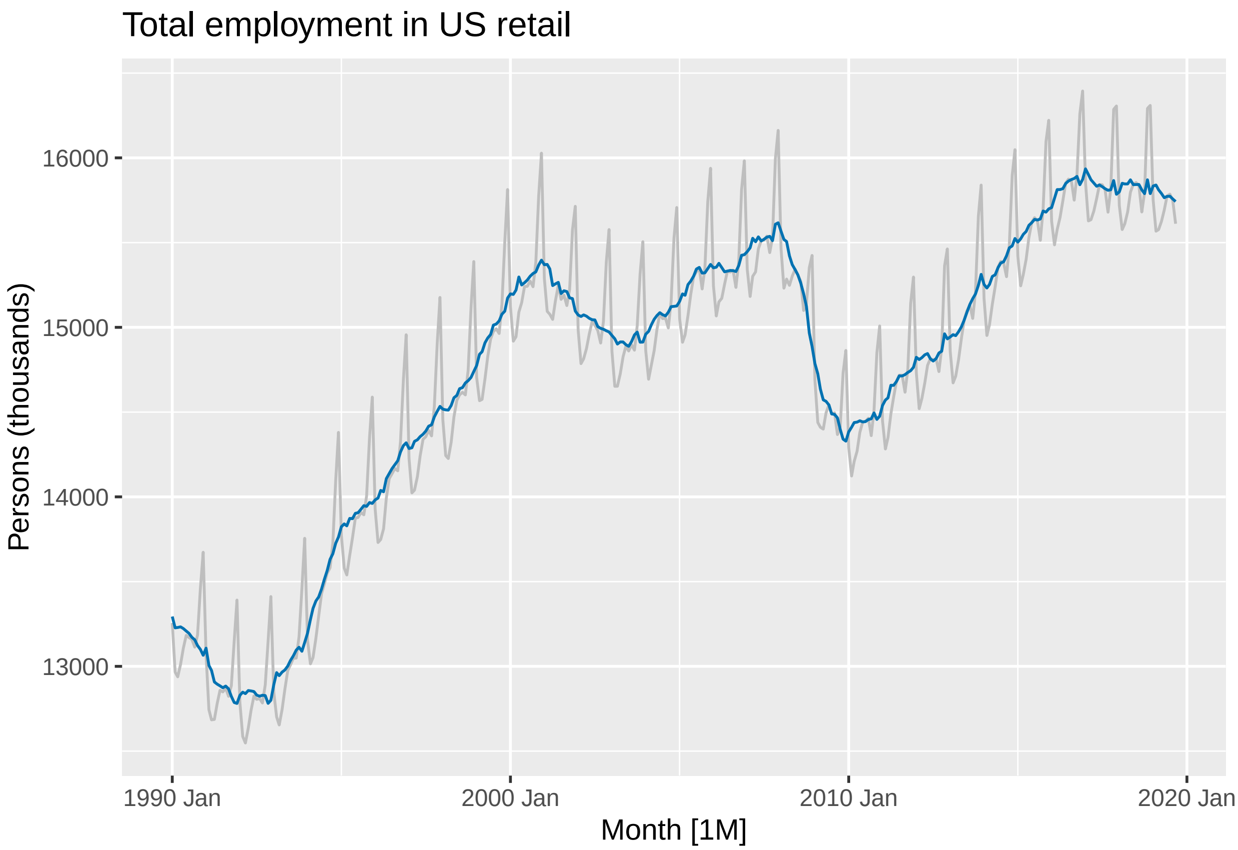
We can plot all of the components in a single figure using autoplot()

```
components(dcmp) %>% autoplot()
```



Seasonally adjusted data

```
components(dcmp) %>%
  as_tsibble() %>%
  autoplot(Employed, color = "gray") +
  geom_line(aes(y=season_adjust), color = "#0072B2") +
  labs(y = "Persons (thousands)",
       title = "Total employment in US retail")
```



3.3 Moving averages

Moving average smoothing

A moving average of order m can be written as

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}, \tag{3.2}$$

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

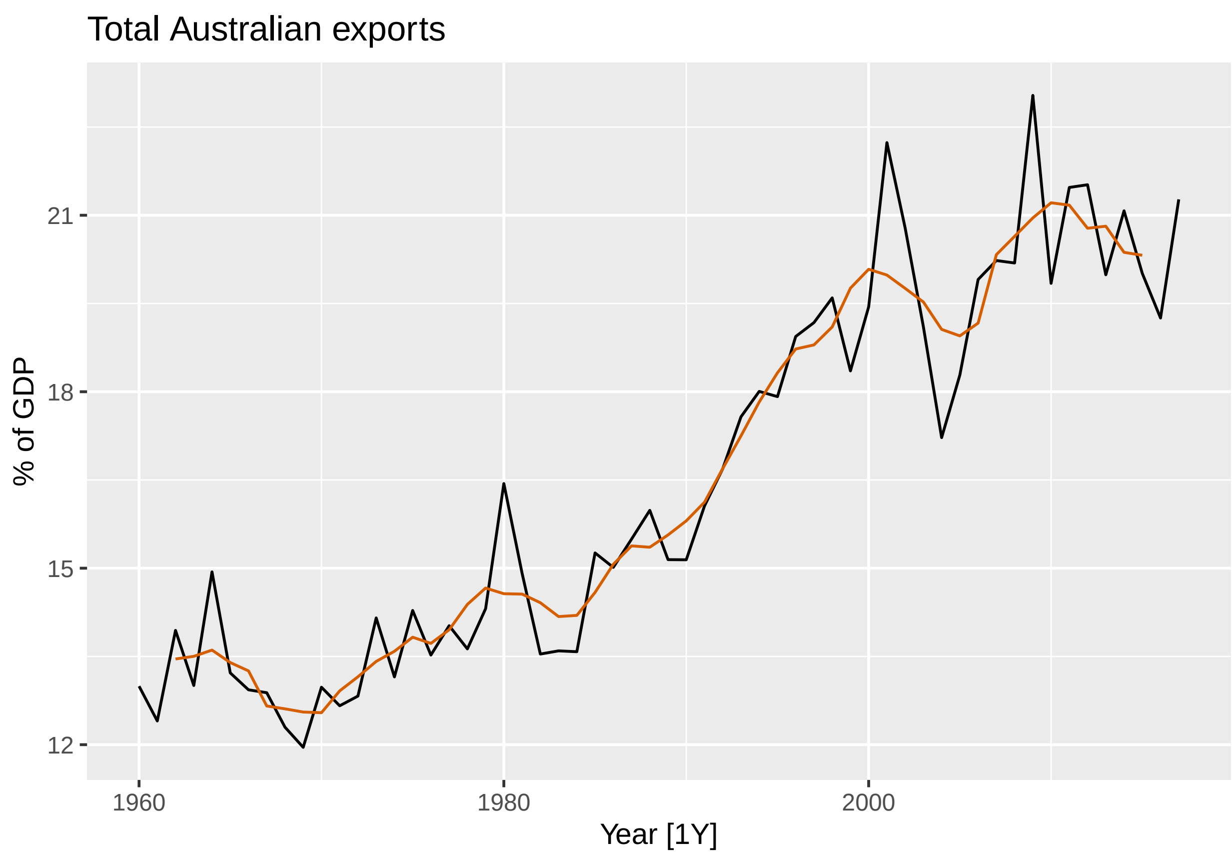

```
aus_exports <- global_economy %>%
  filter(Country == "Australia") %>%
  mutate(
    `5-MA` = slider::slide_dbl(Exports, mean,
                .before = 2, .after = 2, .complete = TRUE)
  )

aus_exports %>% select(Year, Exports, `5-MA`) %>% head(20) %>% kable()
```

Year	Exports	5-MA
1960	12.99445	NA
1961	12.40310	NA
1962	13.94301	13.45694
1963	13.00589	13.50208
1964	14.93825	13.60794
1965	13.22018	13.39608
1966	12.93238	13.25444
1967	12.88373	12.65776
1968	12.29767	12.60913
1969	11.95486	12.55491
1970	12.97704	12.54332
1971	12.66127	12.91479
1972	12.82576	13.15422
1973	14.15502	13.41511
1974	13.15200	13.58748

Year	Exports	5-MA
1975	14.28150	13.82691
1976	13.52313	13.72154
1977	14.02290	13.95354
1978	13.62817	14.38513
1979	14.31198	14.66292

```
aus_exports %>%
  autoplot(Exports) +
  geom_line(aes(y = `5-MA`), color = "#D55E00") +
  guides(colour = guide_legend(title = "series"), size = guide_legend("title")) +
  labs(y = "% of GDP",
       title = "Total Australian exports")
```

```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```



Moving averages of moving averages

```
beer <- aus_production %>%
  filter(year(Quarter) >= 1992) %>%
  select(Quarter, Beer)
beer_ma <- beer %>%
```

```
  mutate(
    `4-MA` = slider::slide_dbl(Beer, mean,
                .before = 1, .after = 2, .complete = TRUE),
    `2x4-MA` = slider::slide_dbl(`4-MA`, mean,
                .before = 1, .after = 0, .complete = TRUE)
  )
```

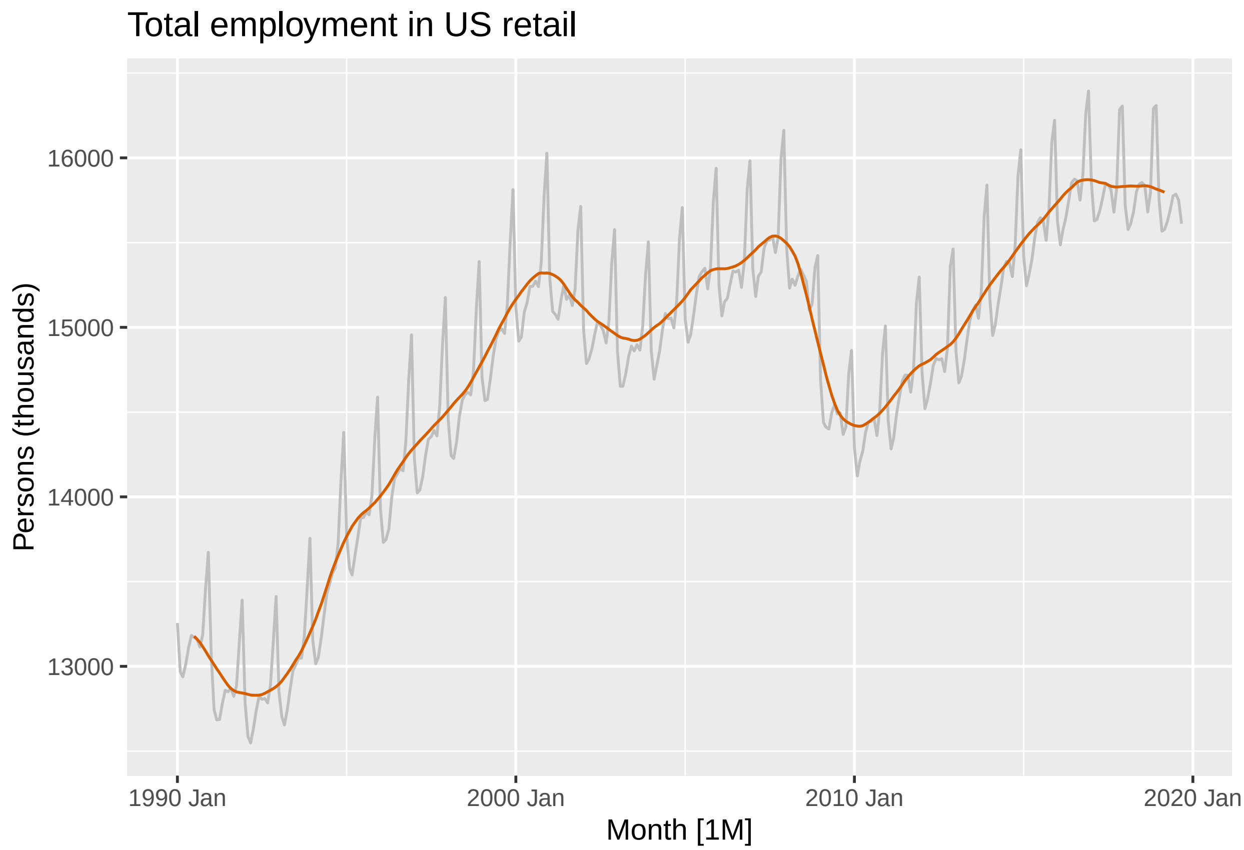
Estimating the trend-cycle with seasonal data

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 2Öm-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 2Ö12-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.

Example: Employment in the US retail sector

```
us_retail_employment_ma <- us_retail_employment %>%
  mutate(
    `12-MA` = slider::slide_dbl(Employed, mean,
                .before = 5, .after = 6, .complete = TRUE),
    `2x12-MA` = slider::slide_dbl(`12-MA`, mean,
                .before = 1, .after = 0, .complete = TRUE)
  )
us_retail_employment_ma %>%
  autoplot(Employed, color = "gray") +
  geom_line(aes(y = `2x12-MA`), color = "#D55E00") +
  labs(y = "Persons (thousands)",
       title = "Total employment in US retail")
```

```
## Warning: Removed 12 row(s) containing missing values (geom_path).
```



Weighted moving averages

Combinations of moving averages result in weighted moving averages. In general, a weighted m-MA can be written as

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

where $k = (m-1)/2$ It is important that the weights all sum to one and that they are symmetric so that $a_j = a_{-j}$.

3.4 Classical decomposition

Let us consider that a time series has seasonal period m. In classical decomposition, we assume that the seasonal component is constant from year to year.

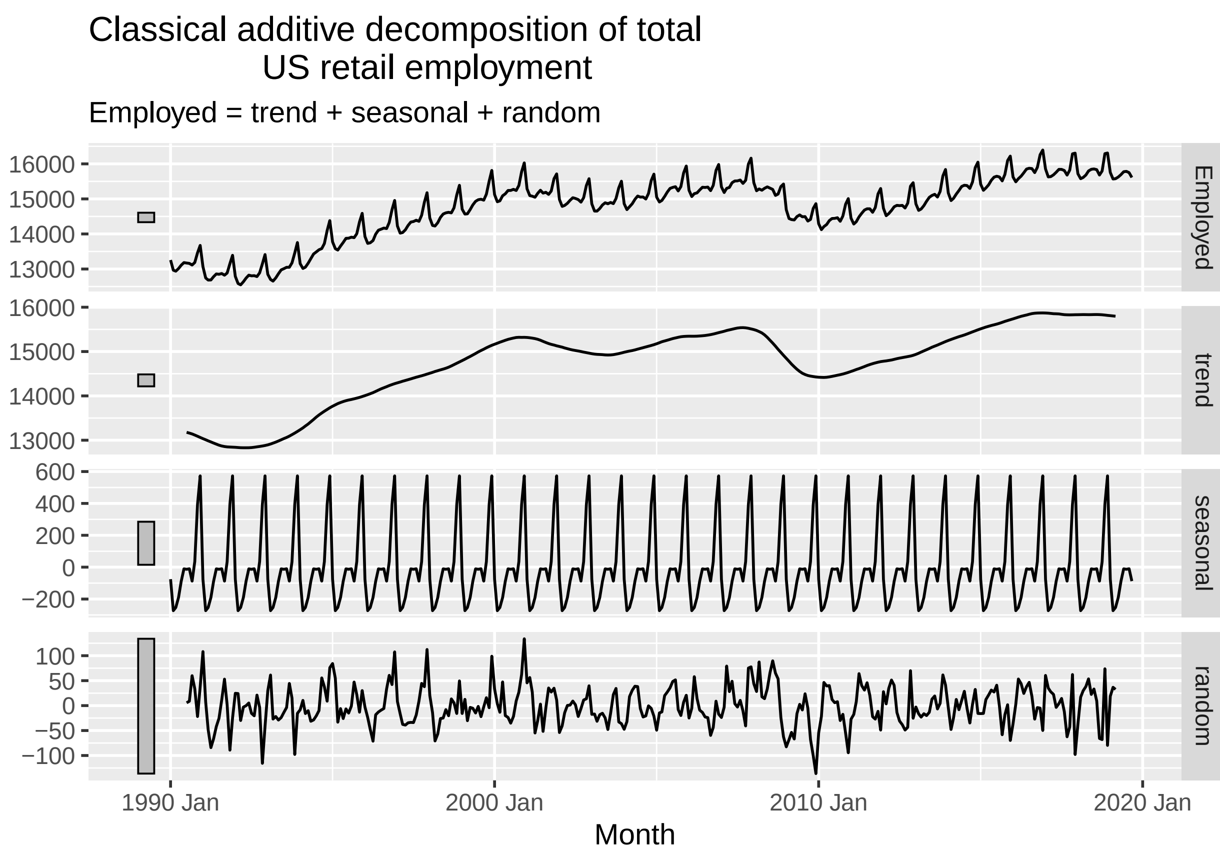
Additive decomposition

- Step 1: If m is odd, compute the trend-cycle component $\hat{T}_t$ with a m-MA. If m is even, compute the trend-cycle component $\hat{T}_t$ with a $2 \times m$-MA.
- Step 2: Compute 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. 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 give $\hat{S}_t$

- Step 4: The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$

```
us_retail_employment %>%
  model(
    classical_decomposition(Employed, type = "additive")
  ) %>%
  components() %>%
  autoplot() +
  labs(title = "Classical additive decomposition of total
                  US retail employment")
```

```
## Warning: Removed 6 row(s) containing missing values (geom_path).
```



Multiplicative decomposition

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

- Step 1: If m is odd, compute the trend-cycle component $\hat{T}_t$ with a m-MA. If m is even, compute the trend-cycle component $\hat{T}_t$ with a $2 \times m$-MA.
- Step 2: Compute 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. These seasonal component values are then adjusted to ensure that they add to m. The *seasonal component* is obtained by stringing together these monthly values, and then replicating the sequence for each year of data. This give $\hat{S}_t$

- 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

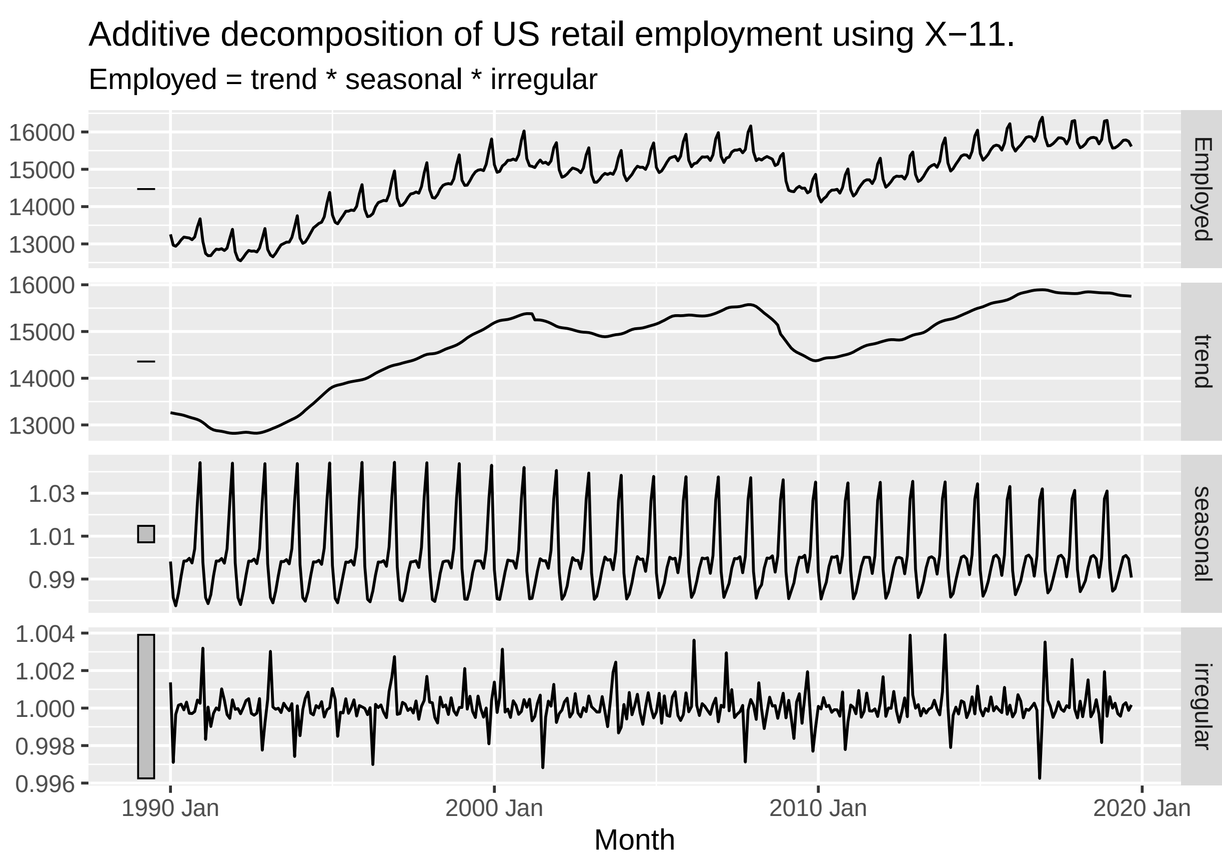
While classical decomposition is still widely used, it is not recommended, because of these problems (among others):

- The estimate of the trend-cycle is unavailable for the first few and last few observations.
- 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.
- Occasionally, the values of the time series in a small number of periods may be particularly unusual. The classical method is not robust to these kinds of unusual values.

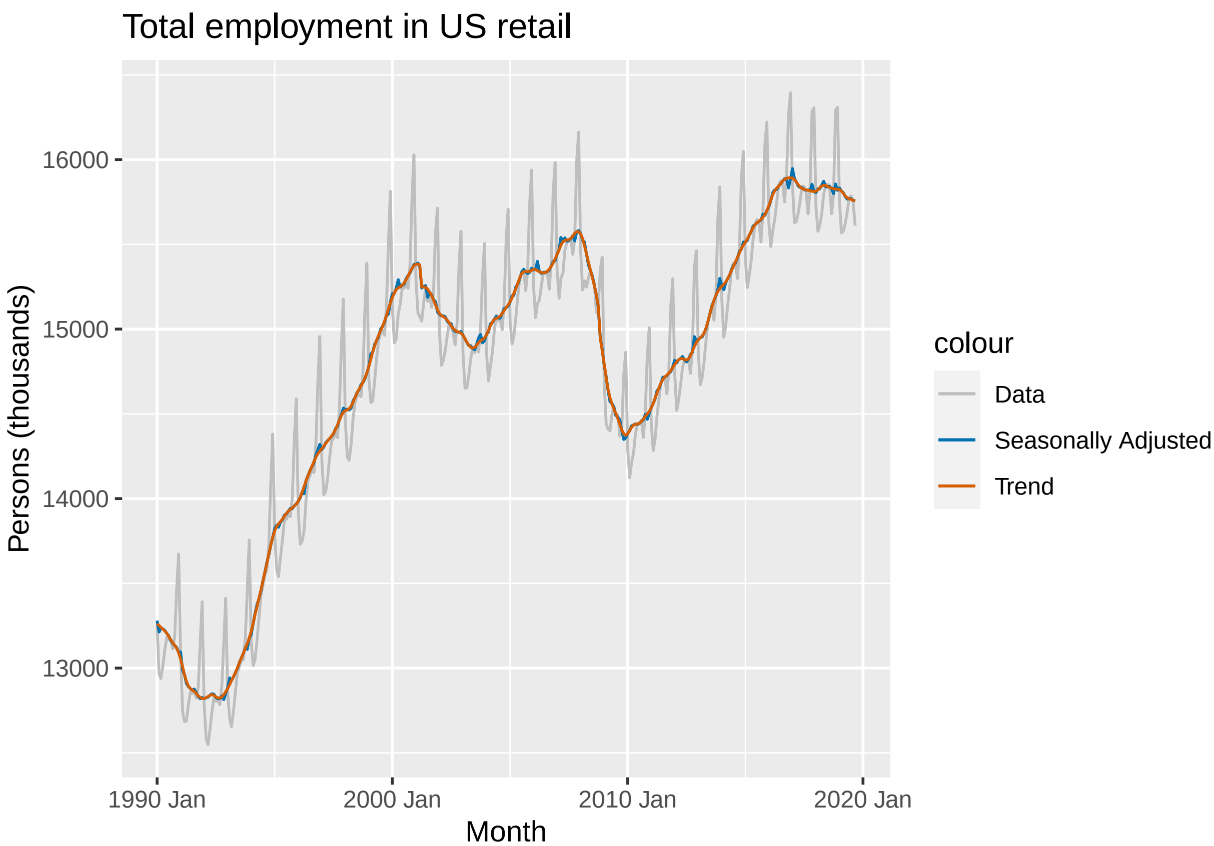
3.5 Methods used by official statistics agencies

X-11 method

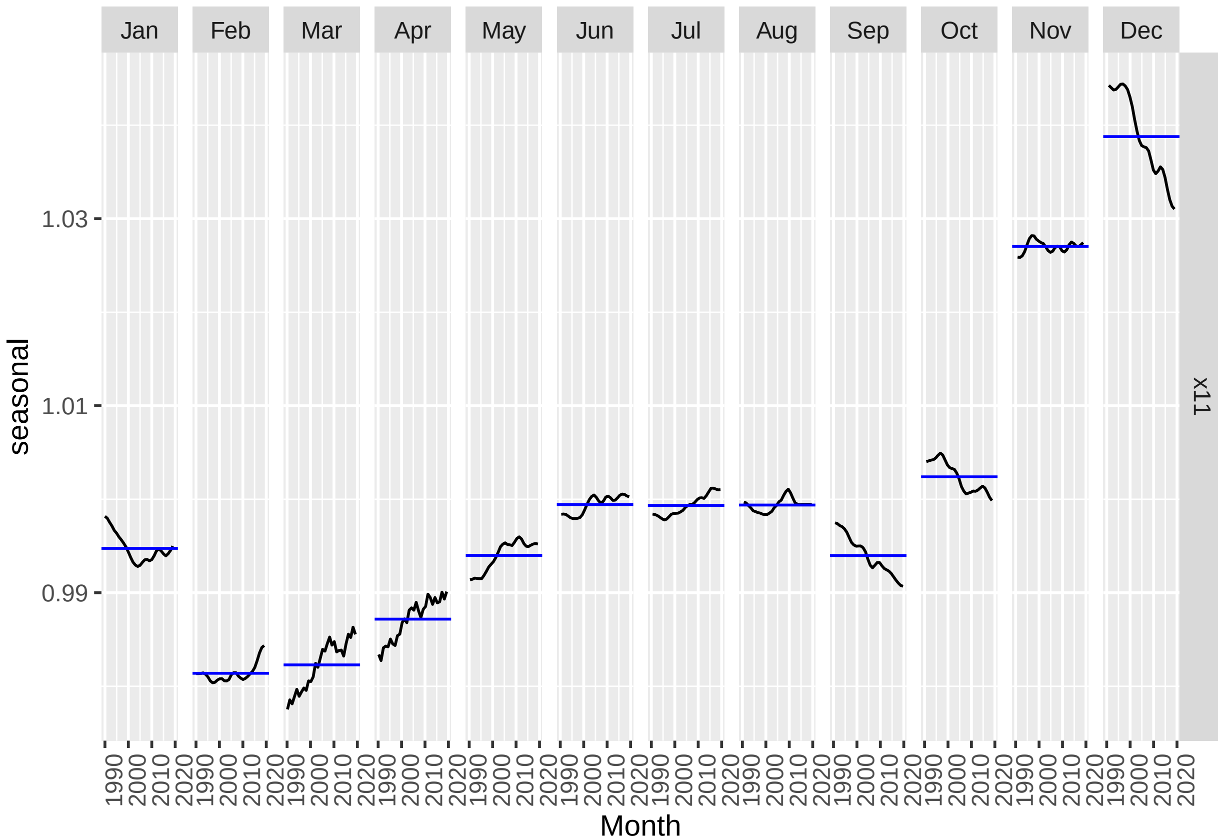
```
x11_dcmp <- us_retail_employment %>%
  model(x11 = X_13ARIMA_SEATS(Employed ~ x11())) %>%
  components()
autoplot(x11_dcmp) +
  labs(title =
    "Additive decomposition of US retail employment using X-11.")
```



```
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")
  )
```



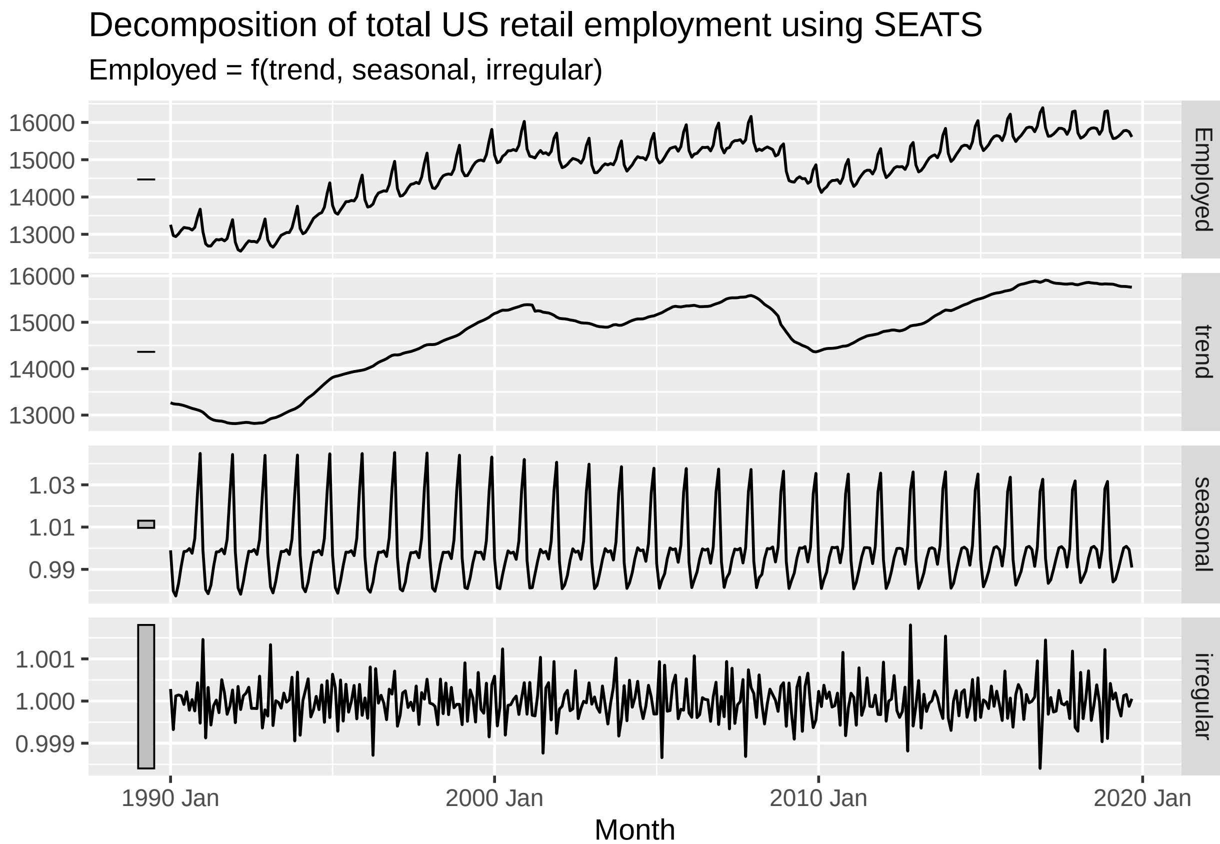

```
x11_dcmp %>%
  gg_subseries(seasonal)
```



SEATS method

This procedure was developed at the Bank of Spain, and is now widely used by government agencies around the world.

```
seats_dcmp <- us_retail_employment %>%
  model(seats = X_13ARIMA_SEATS(Employed ~ seats())) %>%
  components()
autoplot(seats_dcmp) +
  labs(title =
    "Decomposition of total US retail employment using SEATS")
```



3.6 STL decomposition

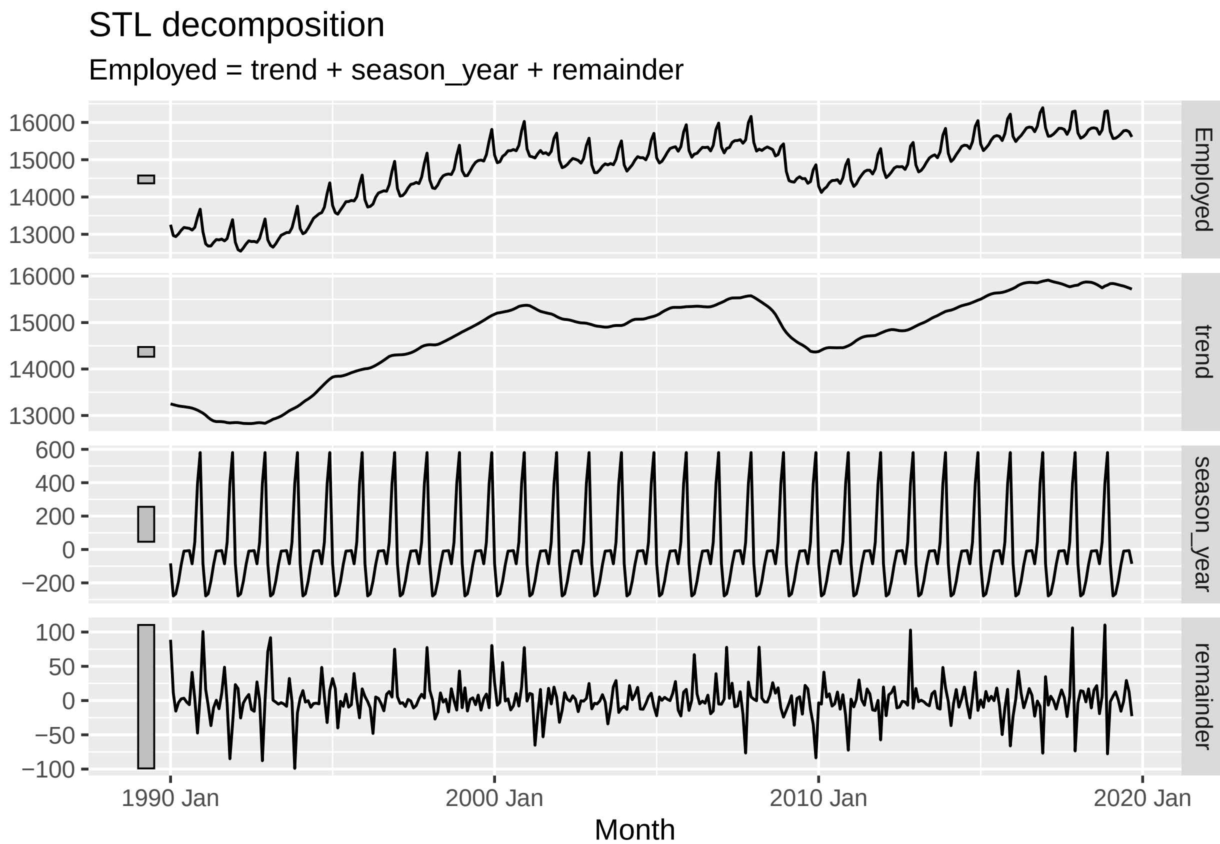
STL is a versatile and robust method for decomposing time series. STL is an acronym for "Seasonal and Trend decomposition using Loess.

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.

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

```
us_retail_employment %>%
  model(
    STL(Employed ~ trend(window = 7) +
                   season(window = "periodic"),
    robust = TRUE)) %>%
  components() %>%
  autoplot()
```



The two main parameters to be chosen when using STL are the trend-cycle window trend(window = ?) and the seasonal window season(window = ?). 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 season(window='periodic') (i.e., identical across years).

Chapter 4 Time series features

The feasts package includes functions for computing FEatures And Statistics from Time Series (hence the name).

4.1 Some simple statistics

```
tourism %>%
  features(Trips, list(mean = mean)) %>%
  arrange(mean)
```

```
## # A tibble: 304 x 4
##    Region          State              Purpose   mean
##    <chr>           <chr>              <chr>    <dbl>
##  1 Kangaroo Island South Australia    Other    0.340
##  2 MacDonnell      Northern Territory Other    0.449
```

```
##  3 Wilderness West Tasmania           Other    0.478
##  4 Barkly          Northern Territory Other    0.632
##  5 Clare Valley    South Australia    Other    0.898
##  6 Barossa         South Australia    Other    1.02
##  7 Kakadu Arnhem   Northern Territory Other    1.04
##  8 Lasseter        Northern Territory Other    1.14
##  9 Wimmera         Victoria           Other    1.15
## 10 MacDonnell      Northern Territory Visiting 1.18
## # ... with 294 more rows
```

```
tourism %>% features(Trips, quantile)
```

```
## # A tibble: 304 x 8
##    Region         State              Purpose    `0%` `25%`   `50%`  `75%` `100%`
##    <chr>          <chr>              <chr>     <dbl>  <dbl>   <dbl>  <dbl>  <dbl>
##  1 Adelaide       South Australia    Busine~  68.7   134.   153.    177.   242.
##  2 Adelaide       South Australia    Holiday 108.    135.   154.    172.   224.
##  3 Adelaide       South Australia    Other    25.9    43.9   53.8    62.5  107.
##  4 Adelaide       South Australia    Visiti~ 137.    179.   206.    229.   270.
##  5 Adelaide Hills South Australia    Busine~   0       0      1.26    3.92  28.6
##  6 Adelaide Hills South Australia    Holiday   0       5.77   8.52   14.1   35.8
##  7 Adelaide Hills South Australia    Other     0       0      0.908   2.09   8.95
##  8 Adelaide Hills South Australia    Visiti~   0.778   8.91  12.2    16.8   81.1
##  9 Alice Springs  Northern Territo~ Busine~   1.01    9.13  13.3    18.5   34.1
## 10 Alice Springs  Northern Territo~ Holiday   2.81   16.9   31.5    44.8   76.5
## # ... with 294 more rows
```

4.2 ACF features

The feat_acf() function computes a selection of the autocorrelations discussed here. It will return six or seven features:

1. the first autocorrelation coefficient from the original data;
2. the sum of squares of the first ten autocorrelation coefficients from the original data;
3. the first autocorrelation coefficient from the differenced data;
4. the sum of squares of the first ten autocorrelation coefficients from the differenced data;
5. the first autocorrelation coefficient from the twice differenced data;
6. the sum of squares of the first ten autocorrelation coefficients from the twice differenced data;
7. For seasonal data, the autocorrelation coefficient at the first seasonal lag is also returned.

```
tourism %>% features(Trips, feat_acf)
```

```
## # A tibble: 304 x 10
##    Region    State      Purpose     acf1 acf10 diff1_acf1 diff1_acf10 diff2_acf1
##    <chr>     <chr>      <chr>      <dbl> <dbl>      <dbl>       <dbl>      <dbl>
##  1 Adelaide  South Aus~ Busine~  0.0333  0.131     -0.520       0.463     -0.676
##  2 Adelaide  South Aus~ Holiday  0.0456  0.372     -0.343       0.614     -0.487
##  3 Adelaide  South Aus~ Other    0.517   1.15      -0.409       0.383     -0.675
##  4 Adelaide  South Aus~ Visiti~  0.0684  0.294     -0.394       0.452     -0.518
##  5 Adelaide~ South Aus~ Busine~  0.0709  0.134     -0.580       0.415     -0.750
##  6 Adelaide~ South Aus~ Holiday  0.131   0.313     -0.536       0.500     -0.716
```

```
##  7 Adelaide~ South Aus~ Other    0.261    0.330     -0.253        0.317     -0.457
##  8 Adelaide~ South Aus~ Visiti~  0.139    0.117     -0.472        0.239     -0.626
##  9 Alice Sp~ Northern ~ Busine~  0.217    0.367     -0.500        0.381     -0.658
## 10 Alice Sp~ Northern ~ Holiday -0.00660 2.11      -0.153        2.11      -0.274
## # ... with 294 more rows, and 2 more variables: diff2_acf10 <dbl>,
## #   season_acf1 <dbl>
```

4.3 STL Features

A time series decomposition can be used to measure the strength of trend and seasonality in a time series. Recall that the decomposition is written as

$$y_t = T_t + S_t + R_t,$$

For strongly trended data, the seasonally adjusted data should have much more variation than the remainder component. To measure this we consider

$$F_T = \text{máx}\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(T_t + R_t)}\right).$$

This will give a measure of the strength of the trend between 0 (no trend) and 1 (strong trend).

The strength of seasonality is defined similarly, but with respect to the detrended data rather than the seasonally adjusted data:

$$F_S = \text{máx}\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(S_t + R_t)}\right).$$

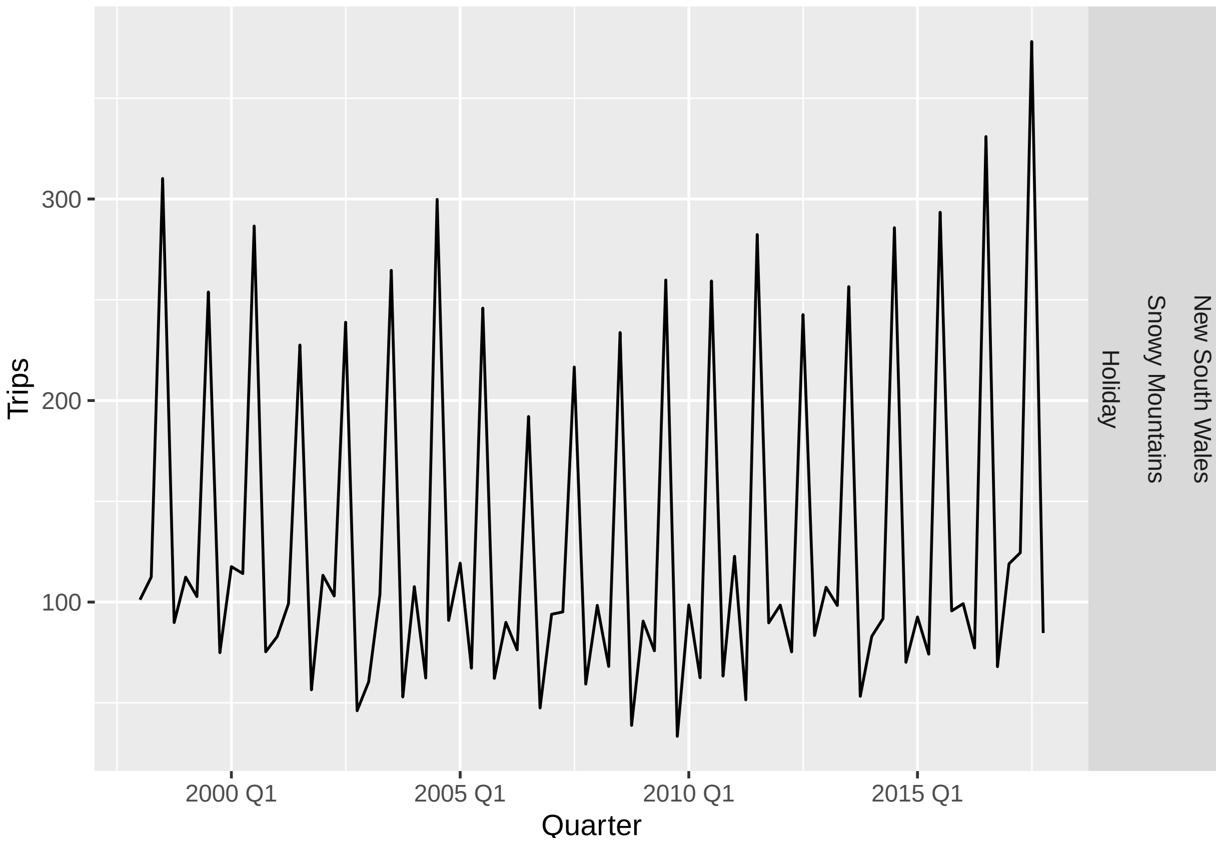
A series with seasonal strength F_S close to 0 exhibits almost no seasonality, while a series with strong seasonality will have F_S close to 1.

```
tourism %>%
  features(Trips, feat_stl)
```

```
## # A tibble: 304 x 12
##    Region    State     Purpose trend_strength seasonal_strengt~ seasonal_peak_y~
##    <chr>     <chr>     <chr>            <dbl>             <dbl>            <dbl>
##  1 Adelaide  South Au~ Busine~          0.451             0.380                3
##  2 Adelaide  South Au~ Holiday          0.541             0.601                1
##  3 Adelaide  South Au~ Other            0.743             0.189                2
##  4 Adelaide  South Au~ Visiti~          0.433             0.446                1
##  5 Adelaide~ South Au~ Busine~          0.453             0.140                3
##  6 Adelaide~ South Au~ Holiday          0.512             0.244                2
##  7 Adelaide~ South Au~ Other            0.584             0.374                2
##  8 Adelaide~ South Au~ Visiti~          0.481             0.228                0
##  9 Alice Sp~ Northern~ Busine~          0.526             0.224                0
## 10 Alice Sp~ Northern~ Holiday          0.377             0.827                3
## # ... with 294 more rows, and 6 more variables: seasonal_trough_year <dbl>,
## #   spikiness <dbl>, linearity <dbl>, curvature <dbl>, stl_e_acf1 <dbl>,
## #   stl_e_acf10 <dbl>
```

```
tourism %>%
  features(Trips, feat_stl) %>%
  ggplot(aes(x = trend_strength, y = seasonal_strength_year,
             col = Purpose)) +
  geom_point() +
  facet_wrap(vars(State))
```


```
tourism %>%
  features(Trips, feat_stl) %>%
  filter(
    seasonal_strength_year == max(seasonal_strength_year)
  ) %>%
  left_join(tourism, by = c("State", "Region", "Purpose")) %>%
  ggplot(aes(x = Quarter, y = Trips)) +
  geom_line() +
  facet_grid(vars(State, Region, Purpose))
```



This shows holiday trips to the most popular ski region of Australia.

The feat_stl() function returns several more features other than those discussed above.

- seasonal_peak_year indicates the timing of the peaks — which month or quarter contains the largest seasonal component. This tells us something about the nature of the seasonality.
- seasonal_trough_year indicates the timing of the troughs — which month or quarter contains the smallest seasonal component.
- spikiness measures the prevalence of spikes in the remainder component of the STL decomposition. It is the variance of its leave-one-out variances.
- linearity measures the linearity of the trend component of the STL decomposition. It is based on the coefficient of a linear regression applied to the trend component.
- curvature measures the curvature of the trend component of the STL decomposition. It is based on the coefficient from an orthogonal quadratic regression applied to the trend component.
- stl_e_acf1 is the first autocorrelation coefficient of the remainder series.
- stl_e_acf10 is the sum of squares of the first ten autocorrelation coefficients of the remainder series.

4.4 Other features

4.5 Exploring Australian tourism data

```
tourism_features <- tourism %>%
  features(Trips, feature_set(pkgs = "feasts"))
```

```
## Warning: ‘n_flat_spots()‘ was deprecated in feasts 0.1.5.
## Please use ‘longest_flat_spot()‘ instead.
```

```r
tourism_features
```

```
## # A tibble: 304 x 51
##    Region    State     Purpose trend_strength seasonal_strengt~ seasonal_peak_y~
##    <chr>     <chr>     <chr>            <dbl>             <dbl>            <dbl>
##  1 Adelaide  South Au~ Busine~          0.451             0.380                3
##  2 Adelaide  South Au~ Holiday          0.541             0.601                1
##  3 Adelaide  South Au~ Other            0.743             0.189                2
##  4 Adelaide  South Au~ Visiti~          0.433             0.446                1
##  5 Adelaide~ South Au~ Busine~          0.453             0.140                3
##  6 Adelaide~ South Au~ Holiday          0.512             0.244                2
##  7 Adelaide~ South Au~ Other            0.584             0.374                2
##  8 Adelaide~ South Au~ Visiti~          0.481             0.228                0
##  9 Alice Sp~ Northern~ Busine~          0.526             0.224                0
## 10 Alice Sp~ Northern~ Holiday          0.377             0.827                3
## # ... with 294 more rows, and 45 more variables: seasonal_trough_year <dbl>,
## #   spikiness <dbl>, linearity <dbl>, curvature <dbl>, stl_e_acf1 <dbl>,
## #   stl_e_acf10 <dbl>, acf1 <dbl>, acf10 <dbl>, diff1_acf1 <dbl>,
## #   diff1_acf10 <dbl>, diff2_acf1 <dbl>, diff2_acf10 <dbl>, season_acf1 <dbl>,
## #   pacf5 <dbl>, diff1_pacf5 <dbl>, diff2_pacf5 <dbl>, season_pacf <dbl>,
## #   zero_run_mean <dbl>, nonzero_squared_cv <dbl>, zero_start_prop <dbl>,
## #   zero_end_prop <dbl>, lambda_guerrero <dbl>, kpss_stat <dbl>,
## #   kpss_pvalue <dbl>, pp_stat <dbl>, pp_pvalue <dbl>, ndiffs <int>,
## #   nsdiffs <int>, bp_stat <dbl>, bp_pvalue <dbl>, lb_stat <dbl>,
## #   lb_pvalue <dbl>, var_tiled_var <dbl>, var_tiled_mean <dbl>,
## #   shift_level_max <dbl>, shift_level_index <dbl>, shift_var_max <dbl>,
## #   shift_var_index <dbl>, shift_kl_max <dbl>, shift_kl_index <dbl>,
## #   spectral_entropy <dbl>, n_crossing_points <int>, longest_flat_spot <int>,
## #   coef_hurst <dbl>, stat_arch_lm <dbl>
```

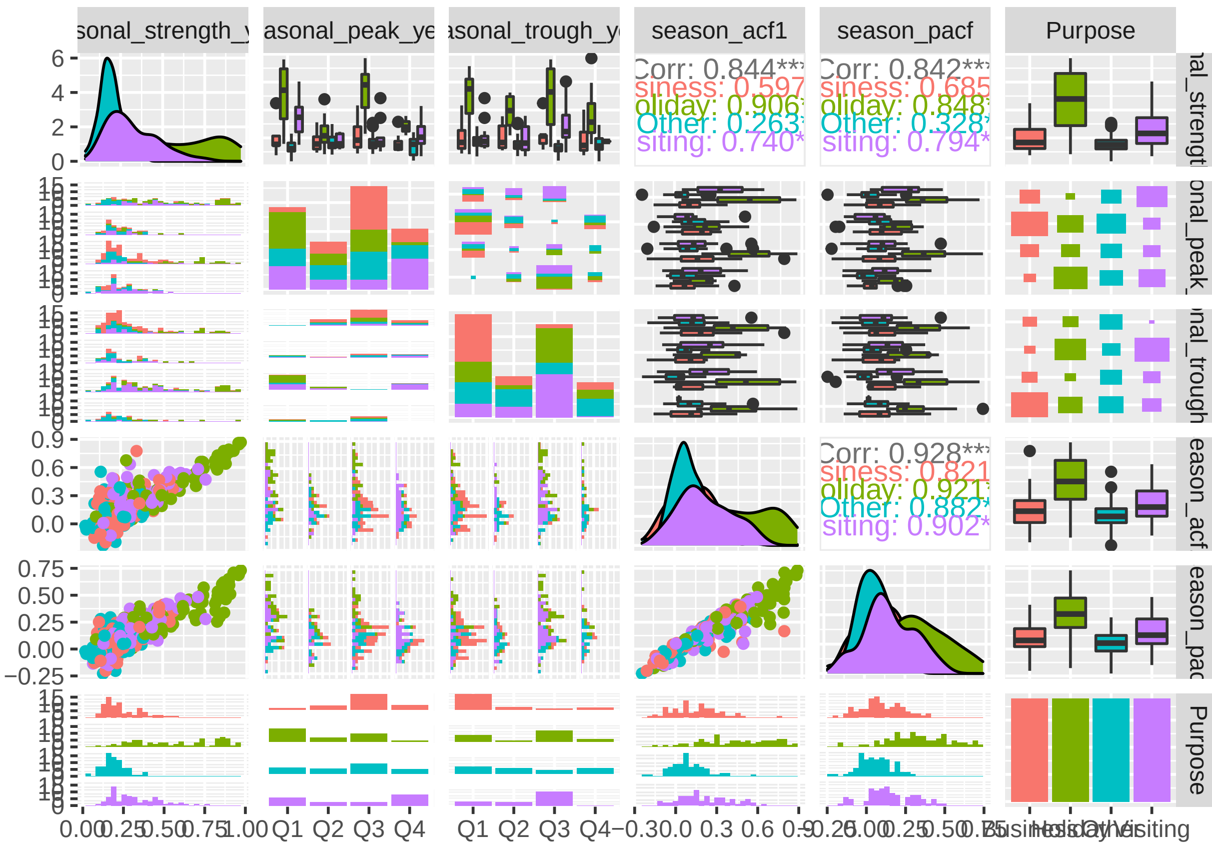
```r
library(glue)
```

```
##
## Attaching package: 'glue'

## The following object is masked from 'package:dplyr':
##
##     collapse
```

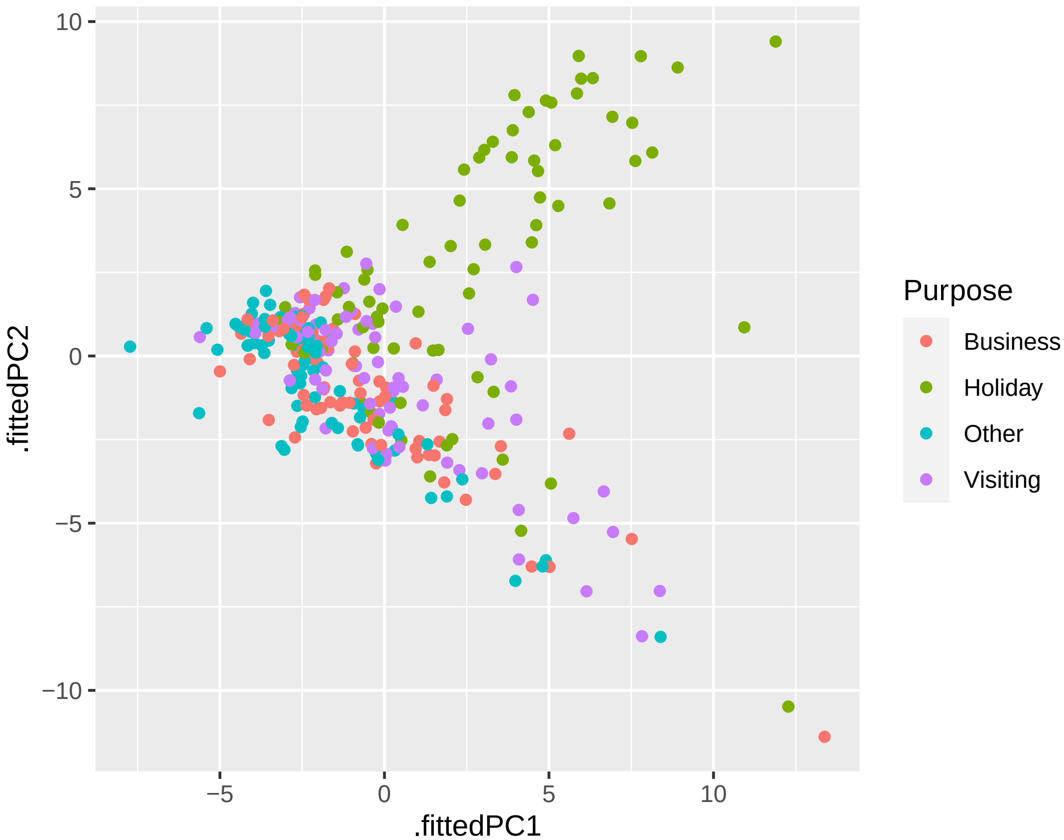
```r
tourism_features %>%
  select_at(vars(contains("season"), Purpose)) %>%
  mutate(
    seasonal_peak_year = seasonal_peak_year +
      4*(seasonal_peak_year==0),
    seasonal_trough_year = seasonal_trough_year +
      4*(seasonal_trough_year==0),
    seasonal_peak_year = glue("Q{seasonal_peak_year}"),
    seasonal_trough_year = glue("Q{seasonal_trough_year}"),
  ) %>%
  GGally::ggpairs(mapping = aes(colour = Purpose), progress=FALSE)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



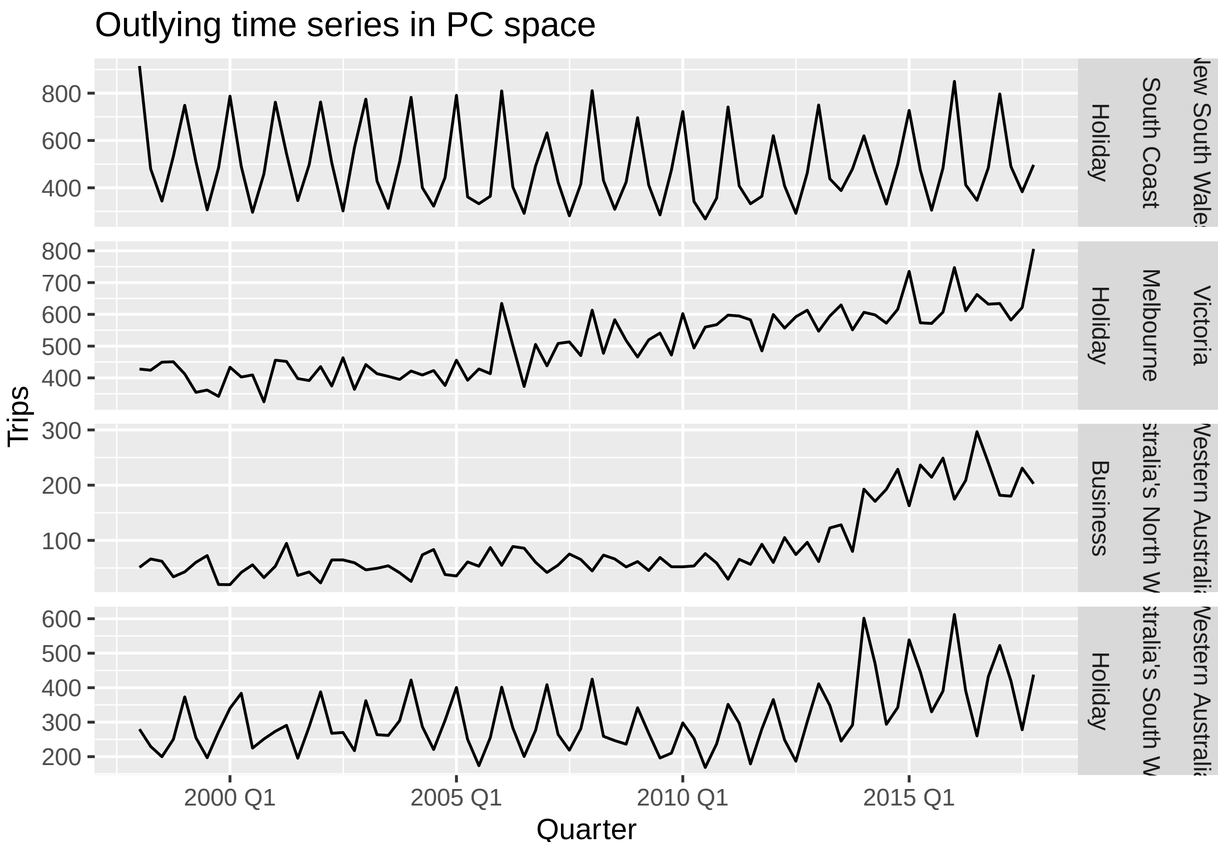

```r
library(broom)
pcs <- tourism_features %>%
  select(-State, -Region, -Purpose) %>%
  prcomp(scale = TRUE) %>%
  augment(tourism_features)
pcs %>%
  ggplot(aes(x = .fittedPC1, y = .fittedPC2, col = Purpose)) +
  geom_point() +
  theme(aspect.ratio = 1)
```




```
outliers <- pcs %>%
  filter(.fittedPC1 > 10) %>%
  select(Region, State, Purpose, .fittedPC1, .fittedPC2)
outliers
```

```
## # A tibble: 4 x 5
##   Region                 State             Purpose  .fittedPC1 .fittedPC2
##   <chr>                  <chr>             <chr>         <dbl>      <dbl>
## 1 Australia's North West Western Australia Business       13.4    -11.4
## 2 Australia's South West Western Australia Holiday        10.9      0.857
## 3 Melbourne              Victoria          Holiday        12.3    -10.5
## 4 South Coast            New South Wales   Holiday        11.9      9.40
```

```
outliers %>%
  left_join(tourism, by = c("State", "Region", "Purpose")) %>%
  mutate(
    Series = glue("{State}", "{Region}", "{Purpose}",
                  .sep = "\n\n")
  ) %>%
  ggplot(aes(x = Quarter, y = Trips)) +
  geom_line() +
  facet_grid(Series ~ ., scales = "free") +
  labs(title = "Outlying time series in PC space")
```



Chapter 5 The forecaster's toolbox

5.1 A tidy forecasting workflow

5.2 Some simple forecasting methods

```
bricks <- aus_production %>%
  filter_index("1970 Q1" ~ "2004 Q4")
```

```
# Re-index based on trading days
google_stock <- gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) >= 2015) %>%
  mutate(day = row_number()) %>%
  update_tsibble(index = day, regular = TRUE)
# Filter the year of interest
google_2015 <- google_stock %>% filter(year(Date) == 2015)
# Fit the models
google_fit <- google_2015 %>%
  model(
    Mean = MEAN(Close),
    `Naïve` = NAIVE(Close),
    Drift = NAIVE(Close ~ drift())
  )
```

```r
# Produce forecasts for the trading days in January 2016
google_jan_2016 <- google_stock %>%
  filter(yearmonth(Date) == yearmonth("2016 Jan"))
google_fc <- google_fit %>%
  forecast(new_data = google_jan_2016)
# Plot the forecasts
google_fc %>%
  autoplot(google_2015, level = NULL) +
  autolayer(google_jan_2016, Close, color = "black") +
  labs(y = "$US",
       title = "Google daily closing stock prices",
       subtitle = "(Jan 2015 - Jan 2016)") +
  guides(colour = guide_legend(title = "Forecast"))
```

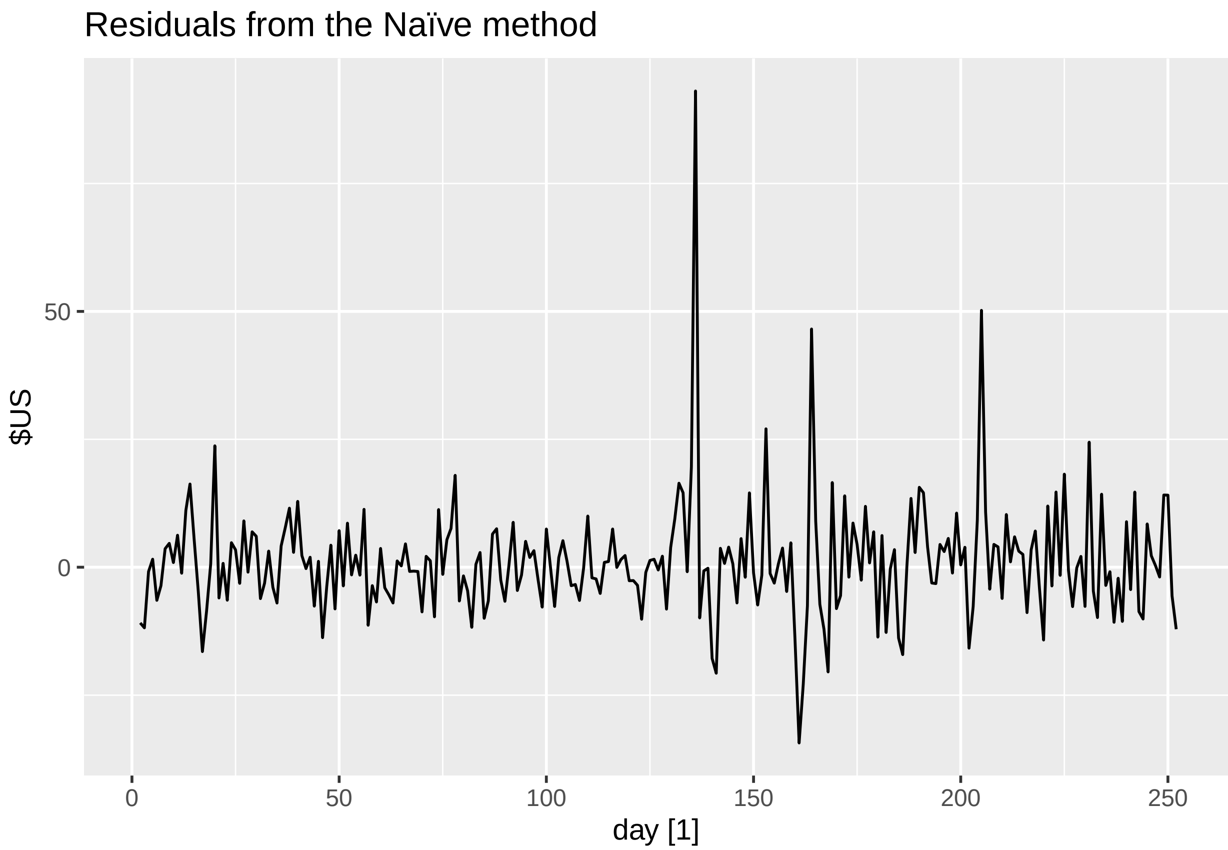
5.4 Residual diagnostics

Example: Forecasting Google daily closing stock prices

```r
autoplot(google_2015, Close) +
  labs(y = "$US",
       title = "Google daily closing stock prices in 2015")
```


```
aug <- google_2015 %>%
  model(NAIVE(Close)) %>%
  augment()
autoplot(aug, .innov) +
  labs(y = "$US",
       title = "Residuals from the Naïve method")
```

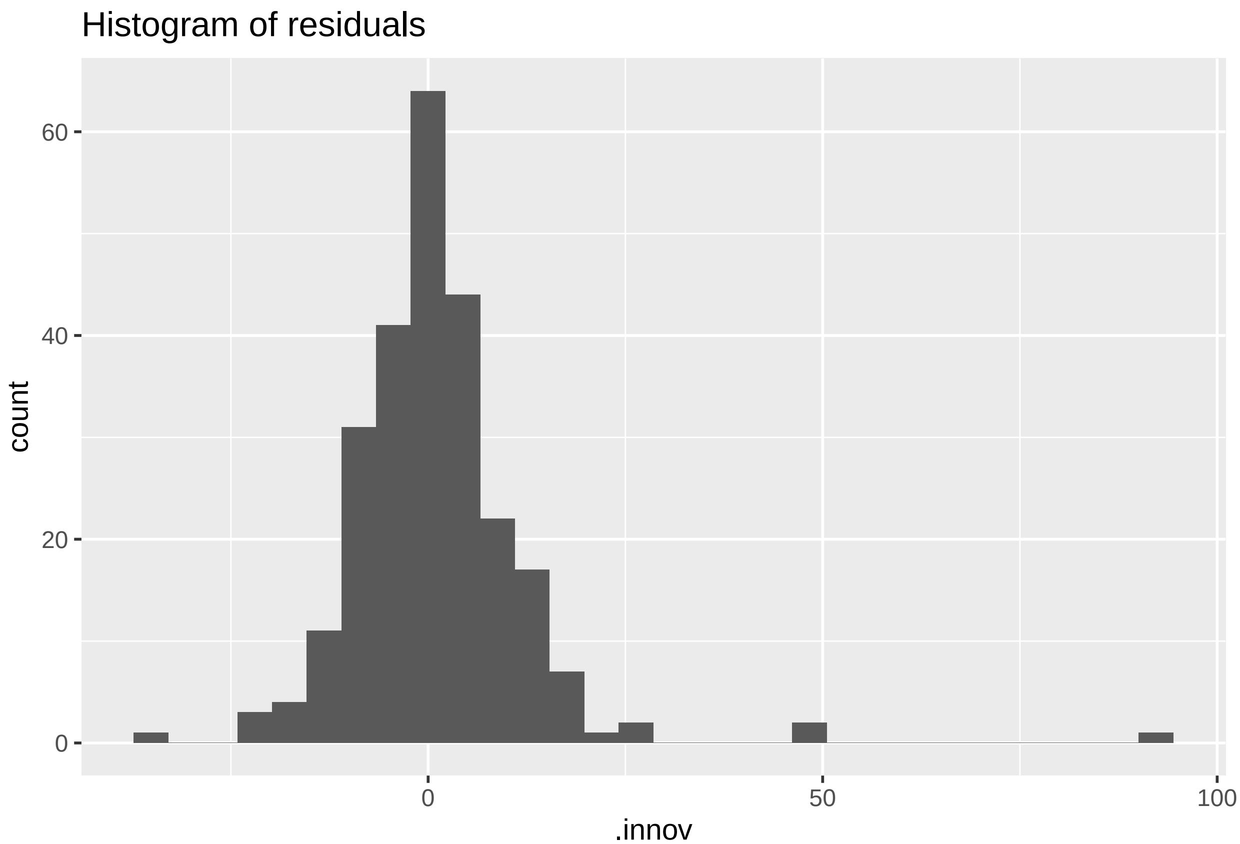
```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



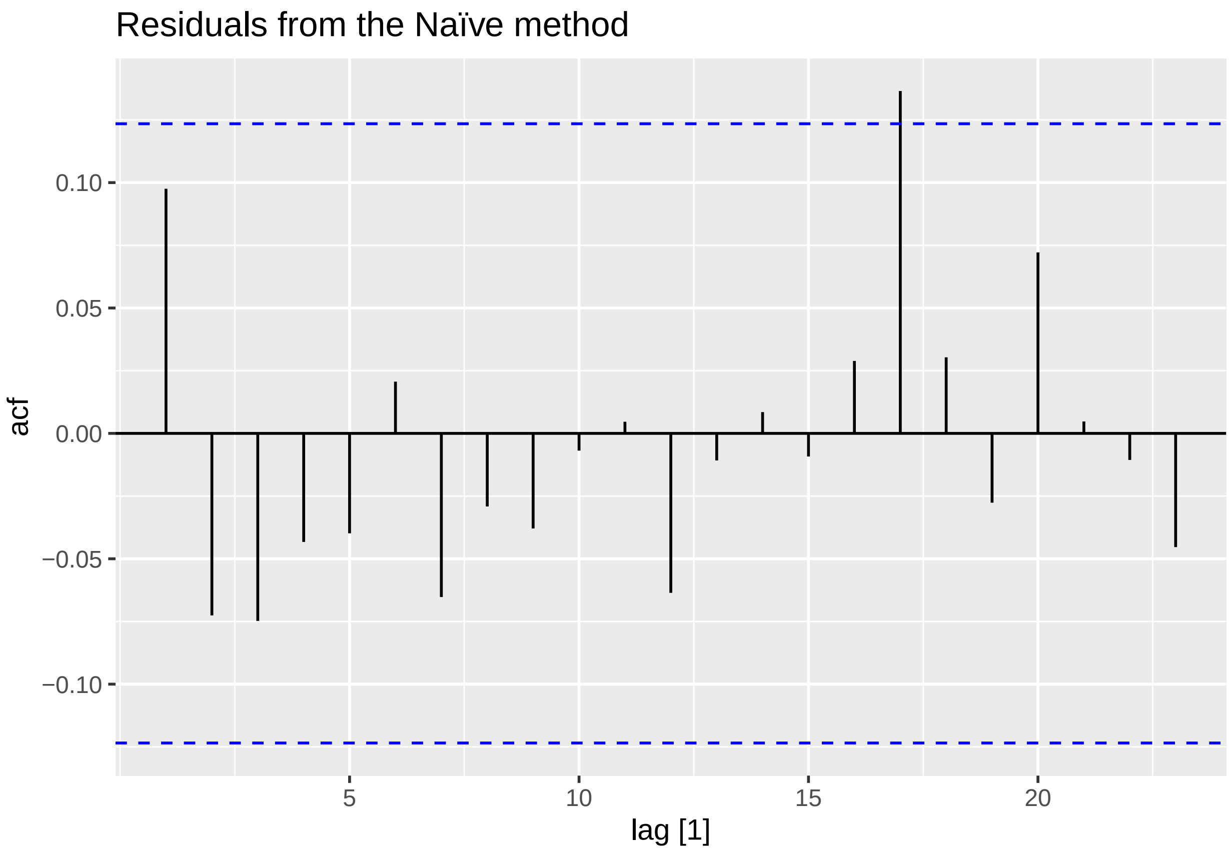

```
aug %>%
  ggplot(aes(x = .innov)) +
  geom_histogram() +
  labs(title = "Histogram of residuals")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

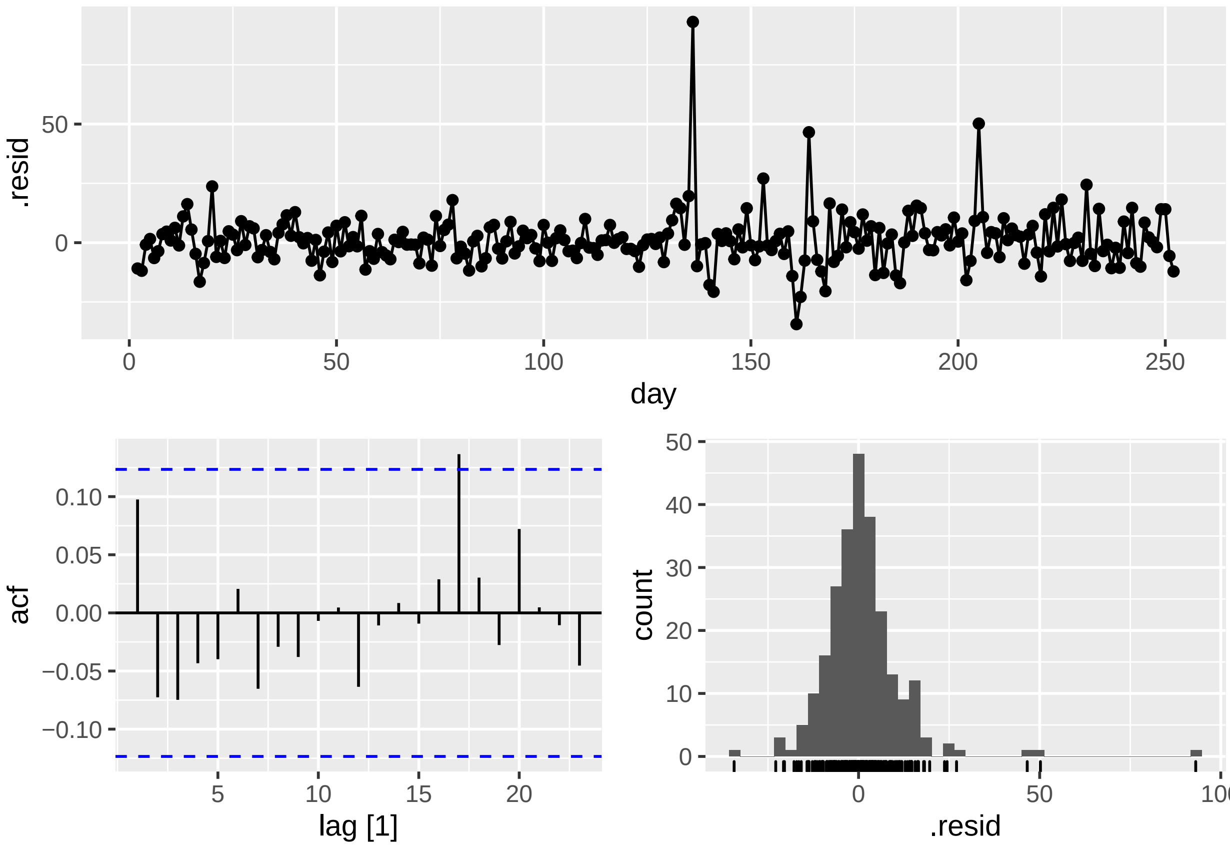



```
aug %>%
  ACF(.innov) %>%
  autoplot() +
  labs(title = "Residuals from the Naïve method")
```




```
google_2015 %>%
  model(NAIVE(Close)) %>%
  gg_tsresiduals()
```

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

## Warning: Removed 1 rows containing missing values (geom_point).

## Warning: Removed 1 rows containing non-finite values (stat_bin).
```



Portmanteau tests for autocorrelation

```
aug %>% features(.innov, box_pierce, lag = 10, dof = 0)
```

```
## # A tibble: 1 x 4
##   Symbol .model       bp_stat bp_pvalue
##   <chr>  <chr>          <dbl>     <dbl>
## 1 GOOG   NAIVE(Close)    7.74     0.654
```

```
aug %>% features(.innov, ljung_box, lag = 10, dof = 0)
```

```
## # A tibble: 1 x 4
##   Symbol .model       lb_stat lb_pvalue
##   <chr>  <chr>          <dbl>     <dbl>
## 1 GOOG   NAIVE(Close)    7.91     0.637
```

```
fit <- google_2015 %>% model(RW(Close ~ drift()))
tidy(fit)
```

```
## # A tibble: 1 x 7
##   Symbol .model               term  estimate std.error statistic p.value
##   <chr>  <chr>                <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 GOOG   RW(Close ~ drift()) b         0.944     0.705      1.34   0.182
```

```
augment(fit) %>% features(.innov, ljung_box, lag=10, dof=1)
```

```
## # A tibble: 1 x 4
##   Symbol .model              lb_stat lb_pvalue
##   <chr>  <chr>                 <dbl>     <dbl>
## 1 GOOG   RW(Close ~ drift())    7.91     0.543
```

5.5 Distributional forecasts and prediction intervals

Forecast distributions

Prediction intervals

$$\hat{y}_{T+h|T} \pm c\hat{\sigma}_h$$

One-step prediction intervals

$$\hat{\sigma} = \sqrt{\frac{1}{T-K}\sum_{t=1}^{T} e_t^2}, \tag{5.1}$$

Multi-step prediction intervals

Benchmark methods

```
google_2015 %>%
  model(NAIVE(Close)) %>%
  forecast(h = 10) %>%
  hilo()
```

```
## # A tsibble: 10 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model   day        Close .mean                  `80%`
##    <chr>  <chr>  <dbl>       <dist> <dbl>                 <hilo>
##  1 GOOG   NAIVE~   253  N(759, 125)  759. [744.5400, 773.2200]80
##  2 GOOG   NAIVE~   254  N(759, 250)  759. [738.6001, 779.1599]80
##  3 GOOG   NAIVE~   255  N(759, 376)  759. [734.0423, 783.7177]80
##  4 GOOG   NAIVE~   256  N(759, 501)  759. [730.1999, 787.5601]80
##  5 GOOG   NAIVE~   257  N(759, 626)  759. [726.8147, 790.9453]80
##  6 GOOG   NAIVE~   258  N(759, 751)  759. [723.7543, 794.0058]80
##  7 GOOG   NAIVE~   259  N(759, 876)  759. [720.9399, 796.8202]80
##  8 GOOG   NAIVE~   260 N(759, 1002)  759. [718.3203, 799.4397]80
##  9 GOOG   NAIVE~   261 N(759, 1127)  759. [715.8599, 801.9001]80
## 10 GOOG   NAIVE~   262 N(759, 1252)  759. [713.5329, 804.2272]80
## # ... with 1 more variable: `95%` <hilo>
```

```r
google_2015 %>%
  model(NAIVE(Close)) %>%
  forecast(h = 10) %>%
  autoplot(google_2015) +
  labs(title="Google daily closing stock price", y="$US" )
```

Prediction intervals from bootstrapped residuals

```r
fit <- google_2015 %>%
  model(NAIVE(Close))
sim <- fit %>% generate(h = 30, times = 5, bootstrap = TRUE)
sim
```

```
## # A tsibble: 150 x 5 [1]
## # Key:       Symbol, .model, .rep [5]
##    Symbol .model         day .rep   .sim
##    <chr>  <chr>        <dbl> <chr> <dbl>
##  1 GOOG   NAIVE(Close)   253 1      767.
##  2 GOOG   NAIVE(Close)   254 1      764.
##  3 GOOG   NAIVE(Close)   255 1      764.
##  4 GOOG   NAIVE(Close)   256 1      751.
##  5 GOOG   NAIVE(Close)   257 1      732.
##  6 GOOG   NAIVE(Close)   258 1      729.
##  7 GOOG   NAIVE(Close)   259 1      722.
```

```
##  8 GOOG   NAIVE(Close)   260 1      725.
##  9 GOOG   NAIVE(Close)   261 1      712.
## 10 GOOG   NAIVE(Close)   262 1      716.
## # ... with 140 more rows
```

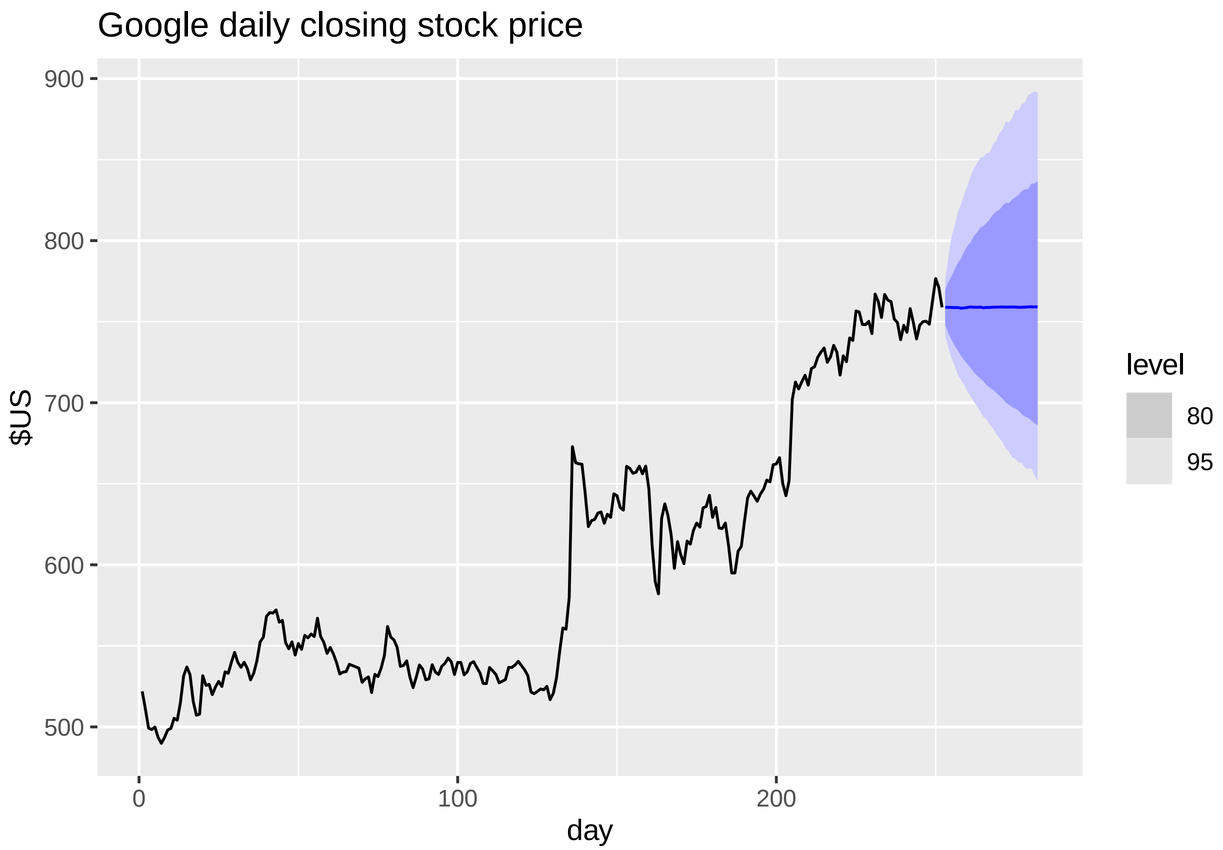
```
google_2015 %>%
  ggplot(aes(x = day)) +
  geom_line(aes(y = Close)) +
  geom_line(aes(y = .sim, colour = as.factor(.rep)),
    data = sim) +
  labs(title="Google daily closing stock price", y="$US" ) +
  guides(col = FALSE)
```


```
fc <- fit %>% forecast(h = 30, bootstrap = TRUE)
fc
```

```
## # A fable: 30 x 5 [1]
## # Key:     Symbol, .model [1]
##    Symbol .model         day        Close .mean
##    <chr>  <chr>        <dbl>       <dist> <dbl>
##  1 GOOG   NAIVE(Close)   253 sample[5000]  759.
##  2 GOOG   NAIVE(Close)   254 sample[5000]  759.
##  3 GOOG   NAIVE(Close)   255 sample[5000]  759.
##  4 GOOG   NAIVE(Close)   256 sample[5000]  759.
##  5 GOOG   NAIVE(Close)   257 sample[5000]  759.
##  6 GOOG   NAIVE(Close)   258 sample[5000]  758.
```

```
##   7 GOOG   NAIVE(Close)   259 sample[5000]  758.
##   8 GOOG   NAIVE(Close)   260 sample[5000]  759.
##   9 GOOG   NAIVE(Close)   261 sample[5000]  759.
##  10 GOOG   NAIVE(Close)   262 sample[5000]  759.
## # ... with 20 more rows
```

```
autoplot(fc, google_2015) +
  labs(title="Google daily closing stock price", y="$US" )
```




```
google_2015 %>%
  model(NAIVE(Close)) %>%
  forecast(h = 10, bootstrap = TRUE, times = 1000) %>%
  hilo()
```

```
## # A tsibble: 10 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model   day        Close .mean                  `80%`
##    <chr>  <chr>  <dbl>       <dist> <dbl>                 <hilo>
##  1 GOOG   NAIVE~   253 sample[1000]  759. [747.8160, 769.9869]80
##  2 GOOG   NAIVE~   254 sample[1000]  759. [742.7850, 774.5886]80
##  3 GOOG   NAIVE~   255 sample[1000]  759. [737.9107, 778.1048]80
##  4 GOOG   NAIVE~   256 sample[1000]  759. [734.2978, 784.0971]80
##  5 GOOG   NAIVE~   257 sample[1000]  759. [732.0624, 786.6164]80
##  6 GOOG   NAIVE~   258 sample[1000]  760. [730.0388, 791.1085]80
##  7 GOOG   NAIVE~   259 sample[1000]  760. [727.1831, 792.9552]80
##  8 GOOG   NAIVE~   260 sample[1000]  760. [726.0862, 797.5936]80
```

```
## 9 GOOG   NAIVE~   261 sample[1000]  760. [723.2396, 800.5478]80
## 10 GOOG  NAIVE~   262 sample[1000]  761. [720.5341, 803.6017]80
## # ... with 1 more variable: '95%' <hilo>
```

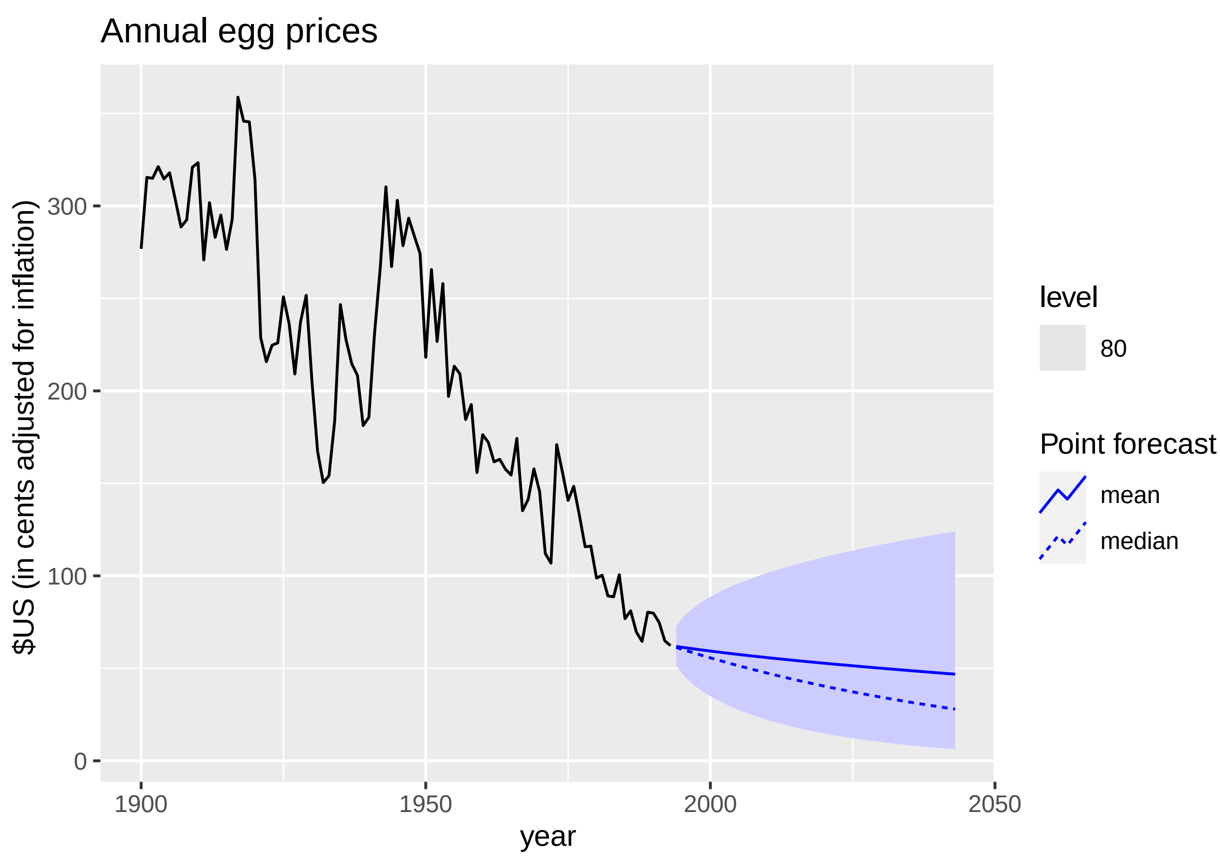
5.6 Forecasting using transformations

Prediction intervals with transformations

Bias adjustments

```
prices %>%
  filter(!is.na(eggs)) %>%
  model(RW(log(eggs) ~ drift())) %>%
  forecast(h = 50) %>%
  autoplot(prices %>% filter(!is.na(eggs)),
    level = 80, point_forecast = lst(mean, median)
  ) +
  labs(title = "Annual egg prices",
       y = "$US (in cents adjusted for inflation) ")
```

```
## Warning: Ignoring unknown aesthetics: linetype
```



5.7 Forecasting with decomposition

To forecast a decomposed time series,

$$y_t = \hat{S}_t + \hat{A}_t,$$

we forecast the seasonal component $\hat{S}_t$ and the seasonally adjusted component $\hat{A}_t$

It is usually assumed that the seasonal component is unchanging, or changing extremely slowly, so it is forecast by simply taking the last year of the estimated component. In other words, a Seasonal naïve method is used for the seasonal component. To forecast the seasonally adjusted component, any non-seasonal forecasting method may be used.

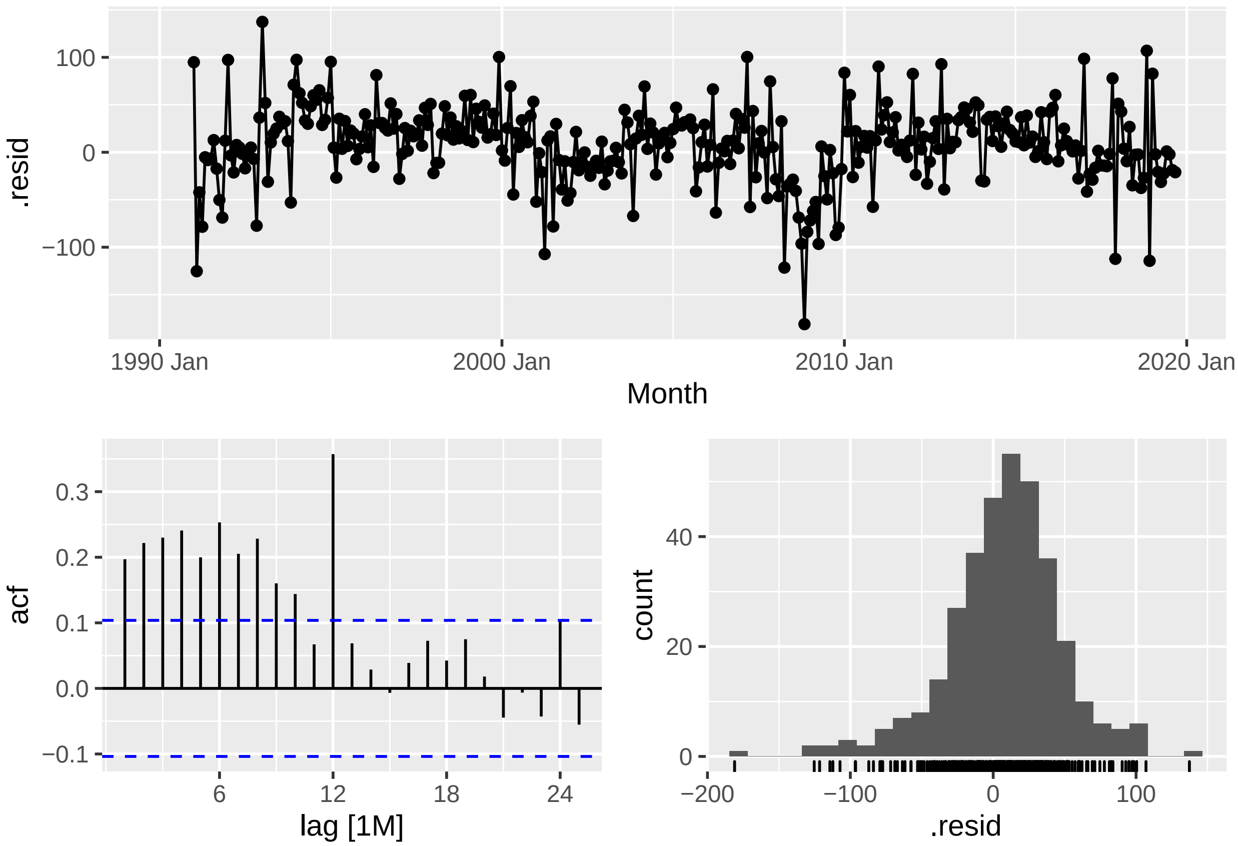
Example: Employment in the US retail sector

```
us_retail_employment <- us_employment %>%
  filter(year(Month) >= 1990, Title == "Retail Trade")
dcmp <- us_retail_employment %>%
  model(STL(Employed ~ trend(window = 7), robust = TRUE)) %>%
  components() %>%
  select(-.model)
dcmp %>%
  model(NAIVE(season_adjust)) %>%
  forecast() %>%
  autoplot(dcmp) +
  labs(y = "Number of people",
       title = "US retail employment")
```

This is made easy with the decomposition_model() function,

```
fit_dcmp <- us_retail_employment %>%
  model(stlf = decomposition_model(
    STL(Employed ~ trend(window = 7), robust = TRUE),
    NAIVE(season_adjust)
  ))
fit_dcmp %>%
  forecast() %>%
  autoplot(us_retail_employment)+
  labs(y = "Number of people",
       title = "Monthly US retail employment")
```


```
fit_dcmp %>% gg_tsresiduals()

## Warning: Removed 12 row(s) containing missing values (geom_path).

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

## Warning: Removed 12 rows containing non-finite values (stat_bin).
```



The ACF of the residuals shown in Figure 5.20, display significant autocorrelations. These are due to the Naïve method not capturing the changing trend in the seasonally adjusted series.

5.8 Evaluating point forecast accuracy

Training and test sets

It is standard practice to separate the available data into two portions, training and test data, where the training data is used to estimate any parameters of a forecasting method and the test data is used to evaluate its accuracy. The test set should ideally be at least as large as the maximum forecast horizon required.

- A model which fits the training data well will not necessarily forecast well.
- A perfect fit can always be obtained by using a model with enough parameters.
- Overfitting a model to data is just as bad as failing to identify a systematic pattern in the data.

Functions to subset a time series

```
(aus_production %>% filter(year(Quarter) >= 1995))
```

```
## # A tsibble: 62 x 7 [1Q]
##    Quarter  Beer Tobacco Bricks Cement Electricity   Gas
##      <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>
##  1 1995 Q1   426    4714    430   1626       41768   131
##  2 1995 Q2   408    3939    457   1703       43686   167
```

```
## 3 1995 Q3   416    6137    417   1733        46022   181
## 4 1995 Q4   520    4739    370   1545        42800   145
## 5 1996 Q1   409    4275    310   1526        43661   133
## 6 1996 Q2   398    5239    358   1593        44707   162
## 7 1996 Q3   398    6293    379   1706        46326   184
## 8 1996 Q4   507    5575    369   1699        43346   146
## 9 1997 Q1   432    4802    330   1511        43938   135
## 10 1997 Q2  398    5523    390   1785        45828   171
## # ... with 52 more rows
```

```
(aus_production %>%
  slice(n()-19:0))
```

```
## # A tsibble: 20 x 7 [1Q]
##    Quarter  Beer Tobacco Bricks Cement Electricity   Gas
##      <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>
##  1 2005 Q3   408      NA     NA   2340       56043   221
##  2 2005 Q4   482      NA     NA   2265       54992   180
##  3 2006 Q1   438      NA     NA   2027       57112   171
##  4 2006 Q2   386      NA     NA   2278       57157   224
##  5 2006 Q3   405      NA     NA   2427       58400   233
##  6 2006 Q4   491      NA     NA   2451       56249   192
##  7 2007 Q1   427      NA     NA   2140       56244   187
##  8 2007 Q2   383      NA     NA   2362       55036   234
##  9 2007 Q3   394      NA     NA   2536       59806   245
## 10 2007 Q4   473      NA     NA   2562       56411   205
## 11 2008 Q1   420      NA     NA   2183       59118   194
## 12 2008 Q2   390      NA     NA   2558       56660   229
## 13 2008 Q3   410      NA     NA   2612       64067   249
## 14 2008 Q4   488      NA     NA   2373       59045   203
## 15 2009 Q1   415      NA     NA   1963       58368   196
## 16 2009 Q2   398      NA     NA   2160       57471   238
## 17 2009 Q3   419      NA     NA   2325       58394   252
## 18 2009 Q4   488      NA     NA   2273       57336   210
## 19 2010 Q1   414      NA     NA   1904       58309   205
## 20 2010 Q2   374      NA     NA   2401       58041   236
```

```
aus_retail %>%
  group_by(State, Industry) %>%
  slice(1:12)
```

```
## # A tsibble: 1,824 x 5 [1M]
## # Key:       State, Industry [152]
## # Groups:    State, Industry [152]
##    State                Industry                   `Series ID`    Month Turnover
##    <chr>                <chr>                      <chr>          <mth>    <dbl>
##  1 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Apr      4.4
##  2 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 May      3.4
##  3 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Jun      3.6
##  4 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Jul      4
##  5 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Aug      3.6
##  6 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Sep      4.2
##  7 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Oct      4.8
```

```
##  8 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Nov      5.4
##  9 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1982 Dec      6.9
## 10 Australian Capital~ Cafes, restaurants and cat~ A3349849A   1983 Jan      3.8
## # ... with 1,814 more rows
```

This will subset the first year (the index is quarter) of data from each time series in the data.

Forecast errors

$e_{T+h} = y_{T+h} - \hat{y}_{T+h|T},$

Scale-dependent errors

Accuracy measures that are based only on the e_i are therefore scale-dependent and cannot be used to make comparisons between series that involve different units.

$$\begin{aligned}\text{Mean absolute error: MAE} &= \text{mean}(|e_t|),\\ \text{Root mean squared error: RMSE} &= \sqrt{\text{mean}(e_t^2)}.\end{aligned}$$

Percentage errors

The percentage error is given by

$$p_t = 100e_t/y_t$$

and it can be used to define

$$\text{Mean absolute percentage error: MAPE} = \text{mean}(|p_t|).$$

Scaled errors

For a non-seasonal time series, a useful way to define a scaled error uses Naïve forecasts:

$$q_j = \frac{e_j}{\dfrac{1}{T-1}\displaystyle\sum_{t=2}^{T}|y_t - y_{t-1}|}.$$

Because of the quotient it is independent of the scale of the data.

For seasonal time series, a scaled error can be defined using Seasonal naïve forecasts:

$$q_j = \frac{e_j}{\dfrac{1}{T-m}\displaystyle\sum_{t=m+1}^{T}|y_t - y_{t-m}|}.$$

And we define:

$$\text{MASE} = \text{mean}(|q_j|).$$

Similarly, the root mean squared scaled error is given by

$$\text{RMSSE} = \sqrt{\text{mean}(q_j^2)},$$

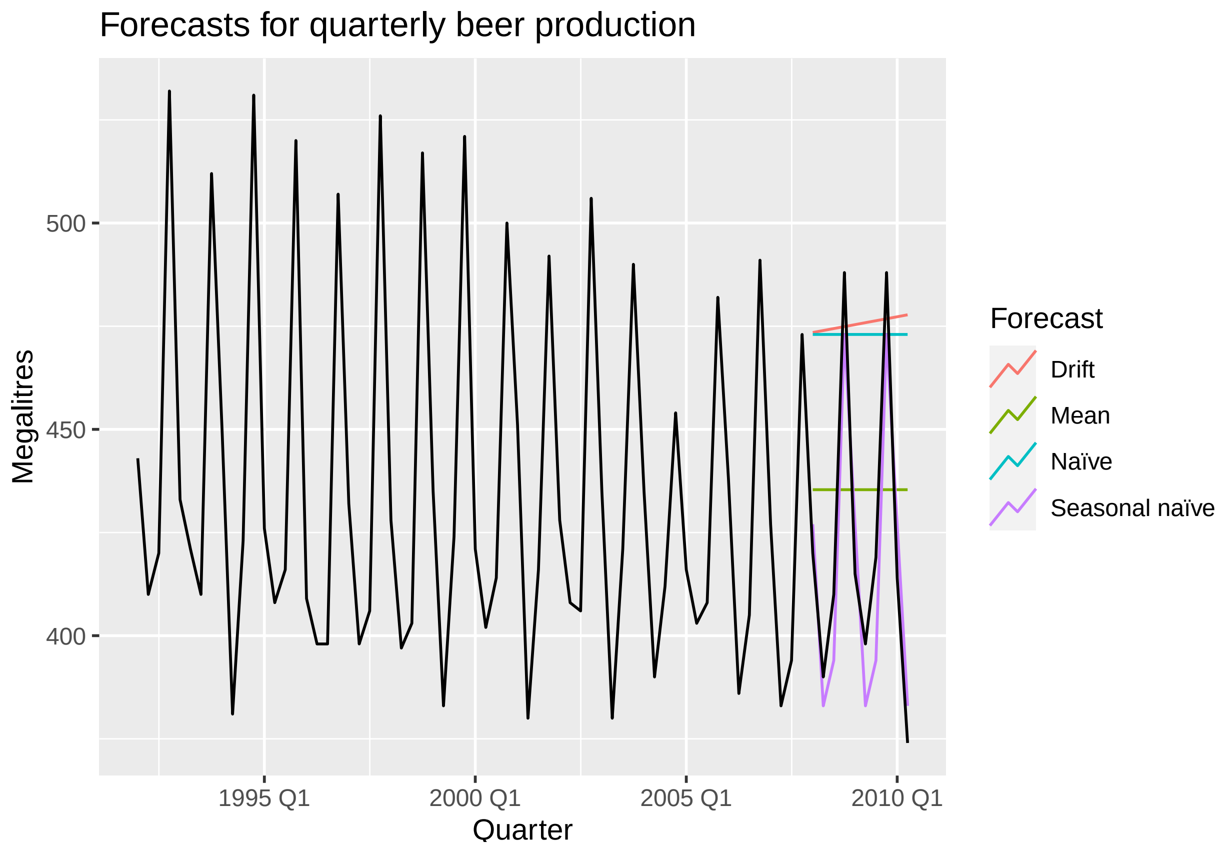
where

$$q_j^2 = \frac{e_j^2}{\frac{1}{T-m}\sum_{t=m+1}^{T}(y_t - y_{t-m})^2},$$

with $m = 1$ for non-seasonal data.

Examples

```
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
beer_train <- recent_production %>%
  filter(year(Quarter) <= 2007)

beer_fit <- beer_train %>%
  model(
    Mean = MEAN(Beer),
    `Naïve` = NAIVE(Beer),
    `Seasonal naïve` = SNAIVE(Beer),
    Drift = RW(Beer ~ drift())
  )

beer_fc <- beer_fit %>%
  forecast(h = 10)

beer_fc %>%
  autoplot(
    aus_production %>% filter(year(Quarter) >= 1992),
    level = NULL
  ) +
  labs(
    y = "Megalitres",
    title = "Forecasts for quarterly beer production"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```




```
accuracy(beer_fc, recent_production)
```

```
## # A tibble: 4 x 10
##   .model         .type    ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
##   <chr>          <chr> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 Drift          Test  -54.0  64.9  58.9 -13.6  14.6  4.12  3.87  -0.0741
## 2 Mean           Test  -13.8  38.4  34.8  -3.97  8.28 2.44  2.29  -0.0691
## 3 Naïve          Test  -51.4  62.7  57.4 -13.0  14.2  4.01  3.74  -0.0691
## 4 Seasonal naïve Test    5.2  14.3  13.4   1.15  3.17 0.937 0.853  0.132
```

A non-seasonal example:

```
google_fit <- google_2015 %>%
  model(
    Mean = MEAN(Close),
    `Naïve` = NAIVE(Close),
    Drift = RW(Close ~ drift())
  )

google_fc <- google_fit %>%
  forecast(google_jan_2016)
google_fc %>%
  autoplot(bind_rows(google_2015, google_jan_2016),
    level = NULL) +
  labs(y = "$US",
       title = "Google closing stock prices from Jan 2015") +
  guides(colour = guide_legend(title = "Forecast"))
```


```
accuracy(google_fc, google_stock)
```

```
## # A tibble: 3 x 11
##   .model Symbol .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>  <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Drift  GOOG   Test  -49.8  53.1  49.8 -6.99  6.99  6.99  4.74 0.604
## 2 Mean   GOOG   Test  117.  118.  117.  16.2  16.2  16.4  10.5  0.496
## 3 Naïve  GOOG   Test  -40.4  43.4  40.4 -5.67  5.67  5.67  3.88 0.496
```

5.9 Evaluating distributional forecast accuracy

Quantile scores

```
google_fc %>%
  filter(.model == "Naïve") %>%
  autoplot(bind_rows(google_2015, google_jan_2016), level=80)+
  labs(y = "$US",
       title = "Google closing stock prices")
```


```
google_fc %>%
  filter(.model == "Naïve", Date == "2016-01-04") %>%
  accuracy(google_stock, list(qs=quantile_score), probs=0.10)
```

```
## # A tibble: 1 x 4
##   .model Symbol .type    qs
##   <chr>  <chr>  <chr> <dbl>
## 1 Naïve  GOOG   Test   4.86
```

The Winkler score is designed to evaluate prediction intervals:

$$W_{\alpha,t} = \begin{cases} (u_{\alpha,t} - \ell_{\alpha,t}) + \frac{2}{\alpha}(\ell_{\alpha,t} - y_t) & \text{if } y_t < \ell_{\alpha,t} \\ (u_{\alpha,t} - \ell_{\alpha,t}) & \text{if } \ell_{\alpha,t} \le y_t \le u_{\alpha,t} \\ (u_{\alpha,t} - \ell_{\alpha,t}) + \frac{2}{\alpha}(y_t - u_{\alpha,t}) & \text{if } y_t > u_{\alpha,t}. \end{cases}$$

For observations that fall within the interval, the Winkler score is simply the length of the interval. So low scores are associated with narrow intervals. However, if the observation falls outside the interval, the penalty applies, with the penalty proportional to how far the observation is outside the interval.

```
google_fc %>%
  filter(.model == "Naïve", Date == "2016-01-04") %>%
  accuracy(google_stock,
    list(winkler = winkler_score), level = 80)
```

```
## # A tibble: 1 x 4
```

```
##   .model Symbol .type winkler
##   <chr>  <chr>  <chr>   <dbl>
## 1 Naïve  GOOG   Test     55.7
```

Continuous Ranked Probability Score

Often we are interested in the whole forecast distribution, rather than particular quantiles or prediction intervals. In that case, we can average the quantile scores over all values of p to obtain the Continuous Ranked Probability Score or CRPS.

```
google_fc %>%
  accuracy(google_stock, list(crps = CRPS))
```

```
## # A tibble: 3 x 4
##   .model Symbol .type  crps
##   <chr>  <chr>  <chr> <dbl>
## 1 Drift  GOOG   Test   33.5
## 2 Mean   GOOG   Test   76.7
## 3 Naïve  GOOG   Test   26.5
```

Scale-free comparisons using skill scores

With skill scores, we compute a forecast accuracy measure relative to some benchmark method.

$$\frac{\text{CRPS}_{\text{Naïve}} - \text{CRPS}_{\text{Drift}}}{\text{CRPS}_{\text{Naïve}}}.$$

This gives the proportion that the Drift method improves over the Naïve method based on CRPS.

```
google_fc %>%
  accuracy(google_stock, list(skill = skill_score(CRPS)))
```

```
## # A tibble: 3 x 4
##   .model Symbol .type  skill
##   <chr>  <chr>  <chr>  <dbl>
## 1 Drift  GOOG   Test  -0.266
## 2 Mean   GOOG   Test  -1.90
## 3 Naïve  GOOG   Test   0
```

The skill_score() function, will always compute the CRPS for the appropriate benchmark forecasts, even if these are not included in the fable object.

5.10 Time series cross-validation

```
# Time series cross-validation accuracy
google_2015_tr <- google_2015 %>%
  stretch_tsibble(.init = 3, .step = 1) %>%
  relocate(Date, Symbol, .id)
google_2015_tr
```

```
## # A tsibble: 31,875 x 10 [1]
## # Key:       Symbol, .id [250]
##    Date       Symbol   .id  Open  High   Low Close Adj_Close  Volume   day
##    <date>     <chr>  <int> <dbl> <dbl> <dbl> <dbl>     <dbl>   <dbl> <int>
##  1 2015-01-02 GOOG       1  526.  528.  521.  522.      522. 1447600     1
##  2 2015-01-05 GOOG       1  520.  521.  510.  511.      511. 2059800     2
##  3 2015-01-06 GOOG       1  512.  513.  498.  499.      499. 2899900     3
##  4 2015-01-02 GOOG       2  526.  528.  521.  522.      522. 1447600     1
##  5 2015-01-05 GOOG       2  520.  521.  510.  511.      511. 2059800     2
##  6 2015-01-06 GOOG       2  512.  513.  498.  499.      499. 2899900     3
##  7 2015-01-07 GOOG       2  504.  504.  497.  498.      498. 2065100     4
##  8 2015-01-02 GOOG       3  526.  528.  521.  522.      522. 1447600     1
##  9 2015-01-05 GOOG       3  520.  521.  510.  511.      511. 2059800     2
## 10 2015-01-06 GOOG       3  512.  513.  498.  499.      499. 2899900     3
## # ... with 31,865 more rows
```

```
# TSCV accuracy
google_2015_tr %>%
  model(RW(Close ~ drift())) %>%
  forecast(h = 1) %>%
  accuracy(google_2015)
```

```
## Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missing.
## 1 observation is missing at 253
```

```
## # A tibble: 1 x 11
##   .model           Symbol .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE   ACF1
##   <chr>            <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 RW(Close ~ drif~ GOOG   Test  0.726  11.3  7.26 0.112  1.19  1.02  1.01 0.0985
```

```
# Training set accuracy
google_2015 %>%
  model(RW(Close ~ drift())) %>%
  accuracy()
```

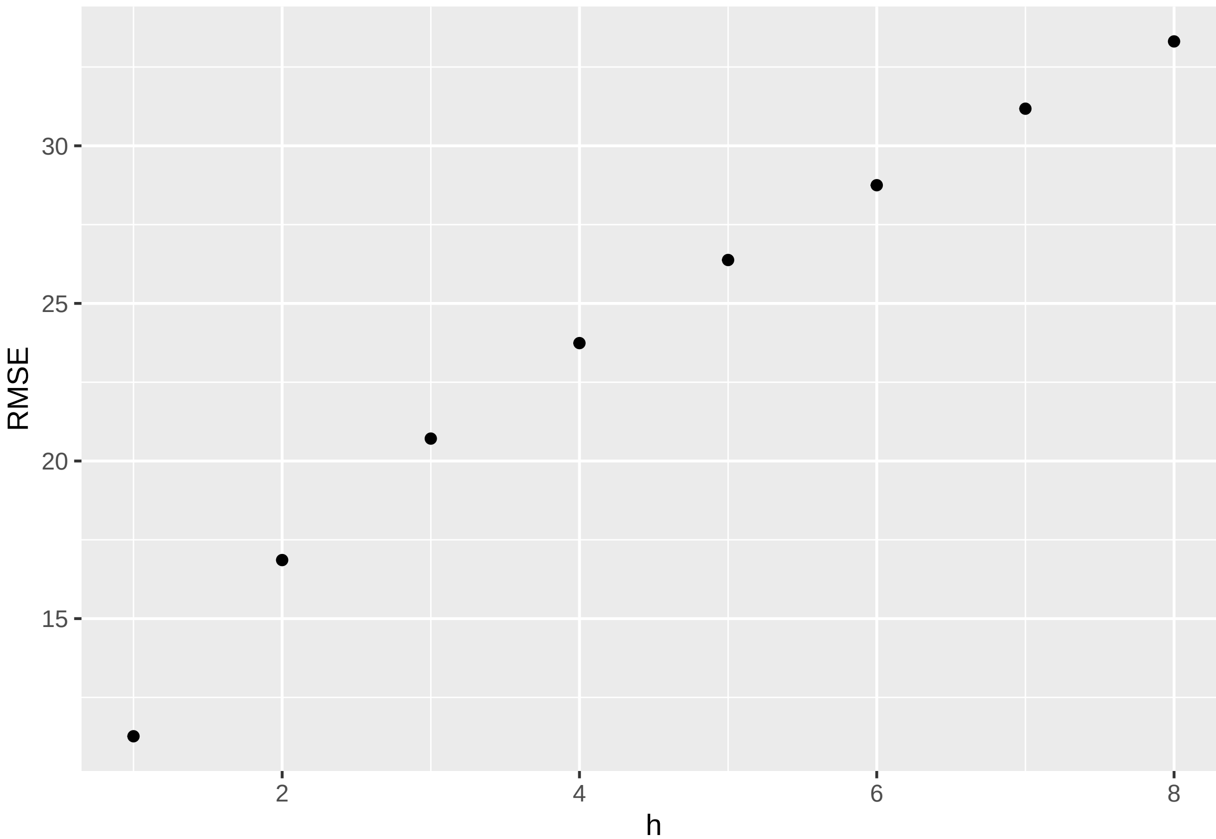
```
## # A tibble: 1 x 11
##   Symbol .model     .type         ME  RMSE   MAE     MPE  MAPE  MASE RMSSE   ACF1
##   <chr>  <chr>      <chr>      <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 GOOG   RW(Close ~ Trai~ -2.97e-14  11.1  7.16 -0.0267  1.18  1.00 0.996 0.0976
```

Example: Forecast horizon accuracy with cross-validation

```
google_2015_tr <- google_2015 %>%
  stretch_tsibble(.init = 3, .step = 1)
fc <- google_2015_tr %>%
  model(RW(Close ~ drift())) %>%
  forecast(h = 8) %>%
  group_by(.id) %>%
  mutate(h = row_number()) %>%
  ungroup()
```

```r
fc %>%
  accuracy(google_2015, by = c("h", ".model")) %>%
  ggplot(aes(x = h, y = RMSE)) +
  geom_point()
```

```
## Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missing.
## 8 observations are missing between 253 and 260
```

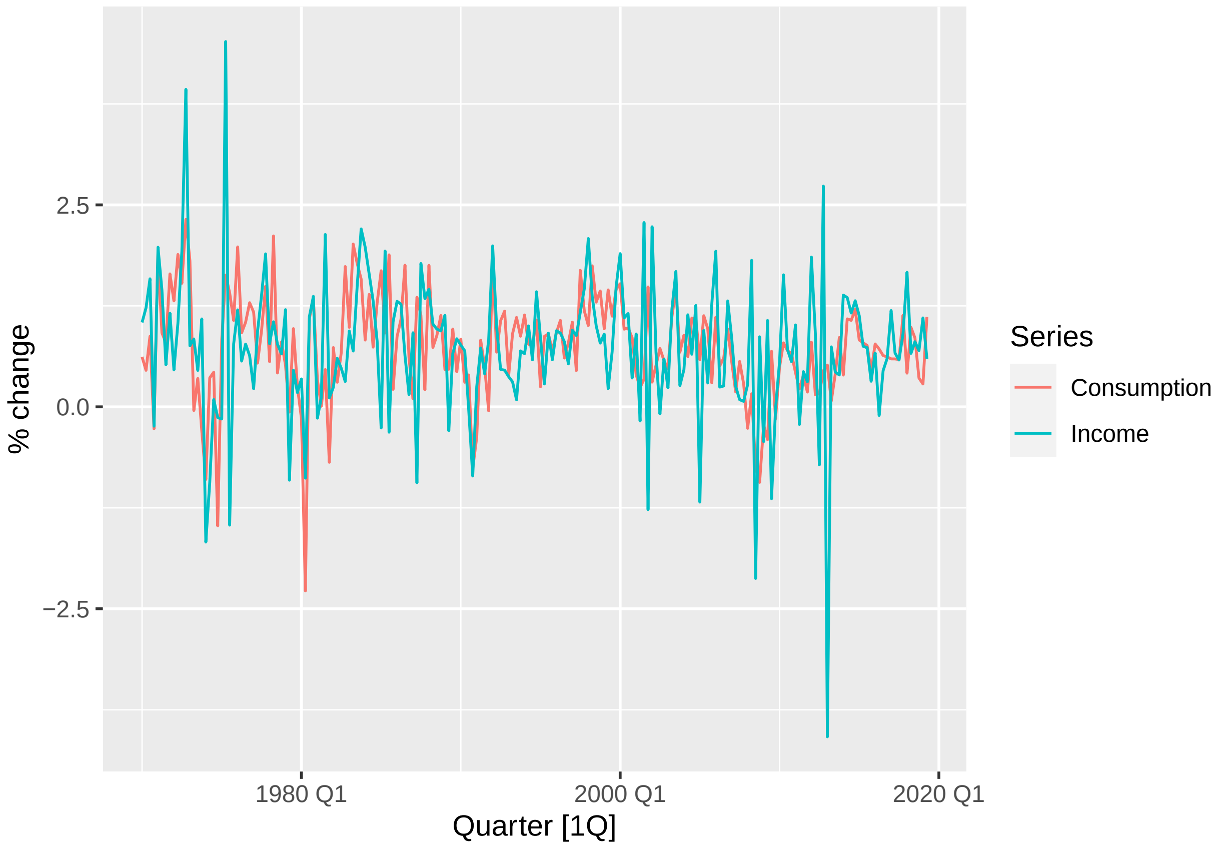


Chapter 7 Time series regression models

7.1 The linear model

Simple linear regression

```r
us_change %>%
  pivot_longer(c(Consumption, Income), names_to="Series") %>%
  autoplot(value) +
    labs(y = "% change")
```



7.2 Least squares estimation

Example: US consumption expenditure

```
fit.consMR <- us_change %>%
  model(tslm = TSLM(Consumption ~ Income + Production +
                                    Unemployment + Savings))
report(fit.consMR)
```

```
## Series: Consumption
## Model: TSLM
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.90555 -0.15821 -0.03608  0.13618  1.15471
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.253105   0.034470   7.343 5.71e-12 ***
## Income         0.740583   0.040115  18.461  < 2e-16 ***
## Production     0.047173   0.023142   2.038   0.0429 *
## Unemployment -0.174685   0.095511  -1.829   0.0689 .
## Savings      -0.052890   0.002924 -18.088  < 2e-16 ***
## ---
```

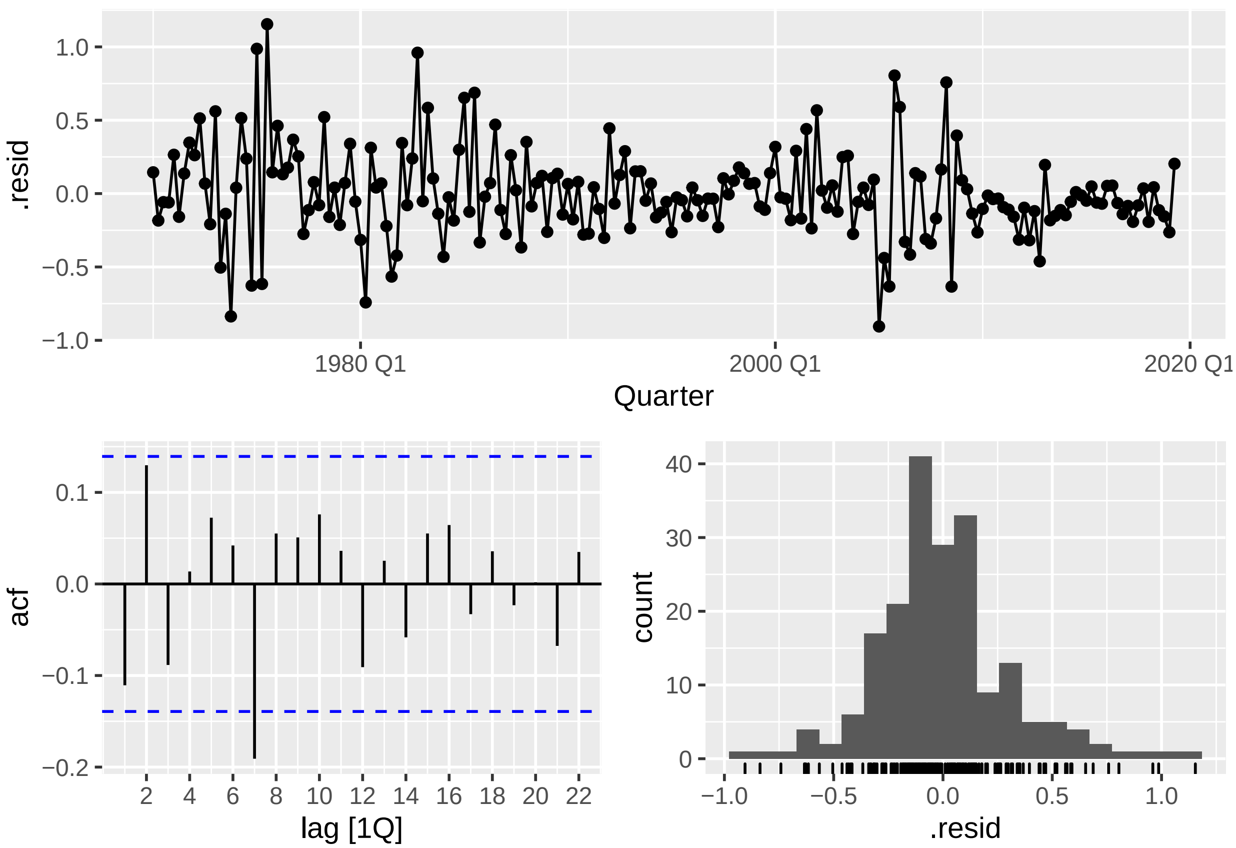
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3102 on 193 degrees of freedom
## Multiple R-squared: 0.7683,  Adjusted R-squared: 0.7635
## F-statistic:   160 on 4 and 193 DF, p-value: < 2.22e-16
```

7.3 Evaluating the regression model

ACF plot of residuals

Histogram of residuals

```
fit.consMR %>% gg_tsresiduals()
```

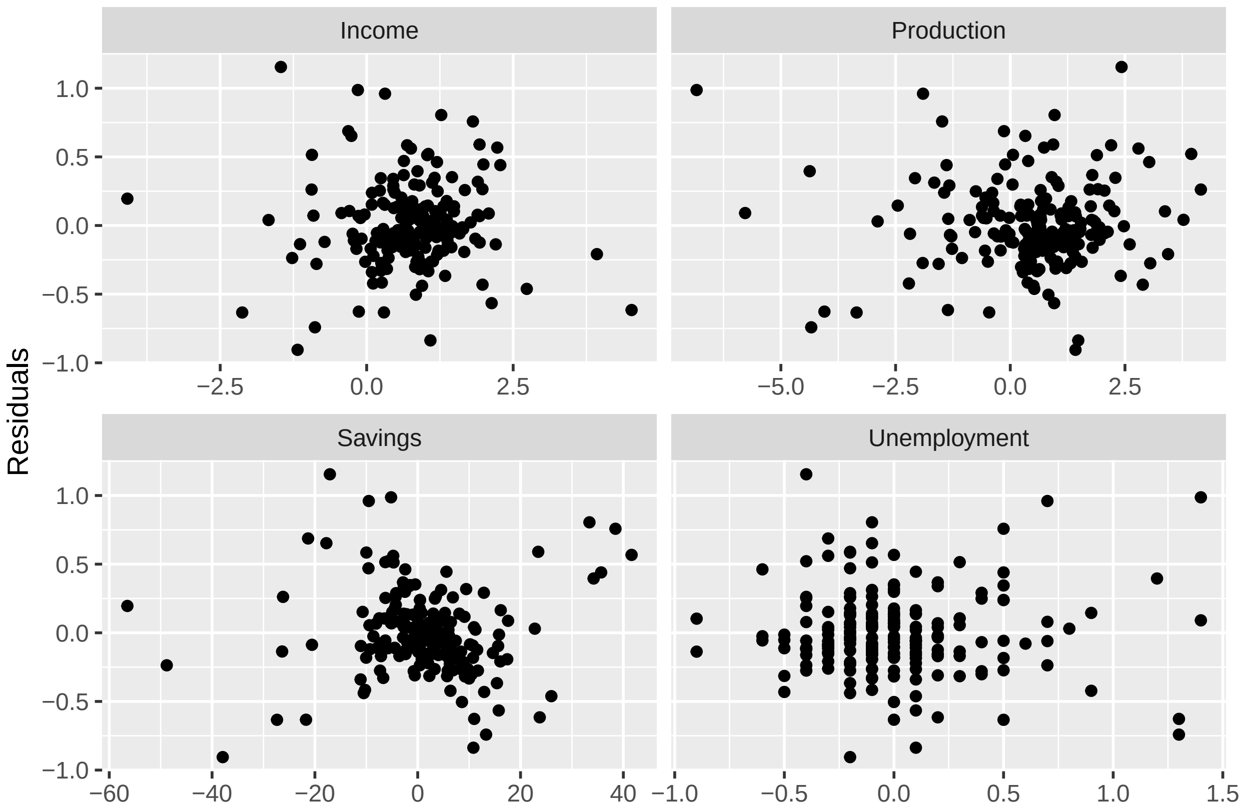



```
augment(fit.consMR) %>%
  features(.innov, ljung_box, lag = 10, dof = 5)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 tslm      18.9   0.00204
```

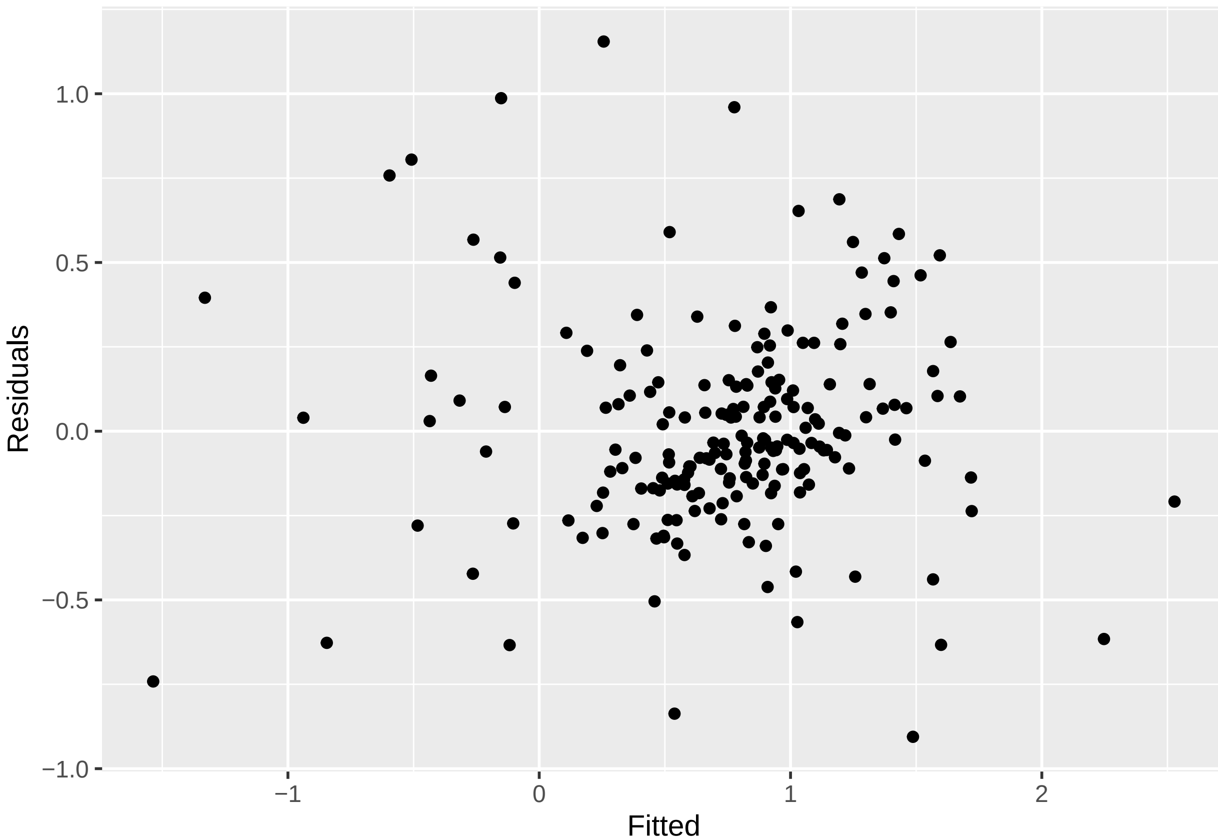
Residual plots against predictors

```
us_change %>%
  left_join(residuals(fit.consMR), by = "Quarter") %>%
  pivot_longer(Income:Unemployment,
               names_to = "regressor", values_to = "x") %>%
  ggplot(aes(x = x, y = .resid)) +
  geom_point() +
  facet_wrap(. ~ regressor, scales = "free_x") +
  labs(y = "Residuals", x = "")
```



Residual plots against fitted values

```
augment(fit.consMR) %>%
  ggplot(aes(x = .fitted, y = .resid)) +
  geom_point() + labs(x = "Fitted", y = "Residuals")
```



Outliers and influential observations

Spurious regression

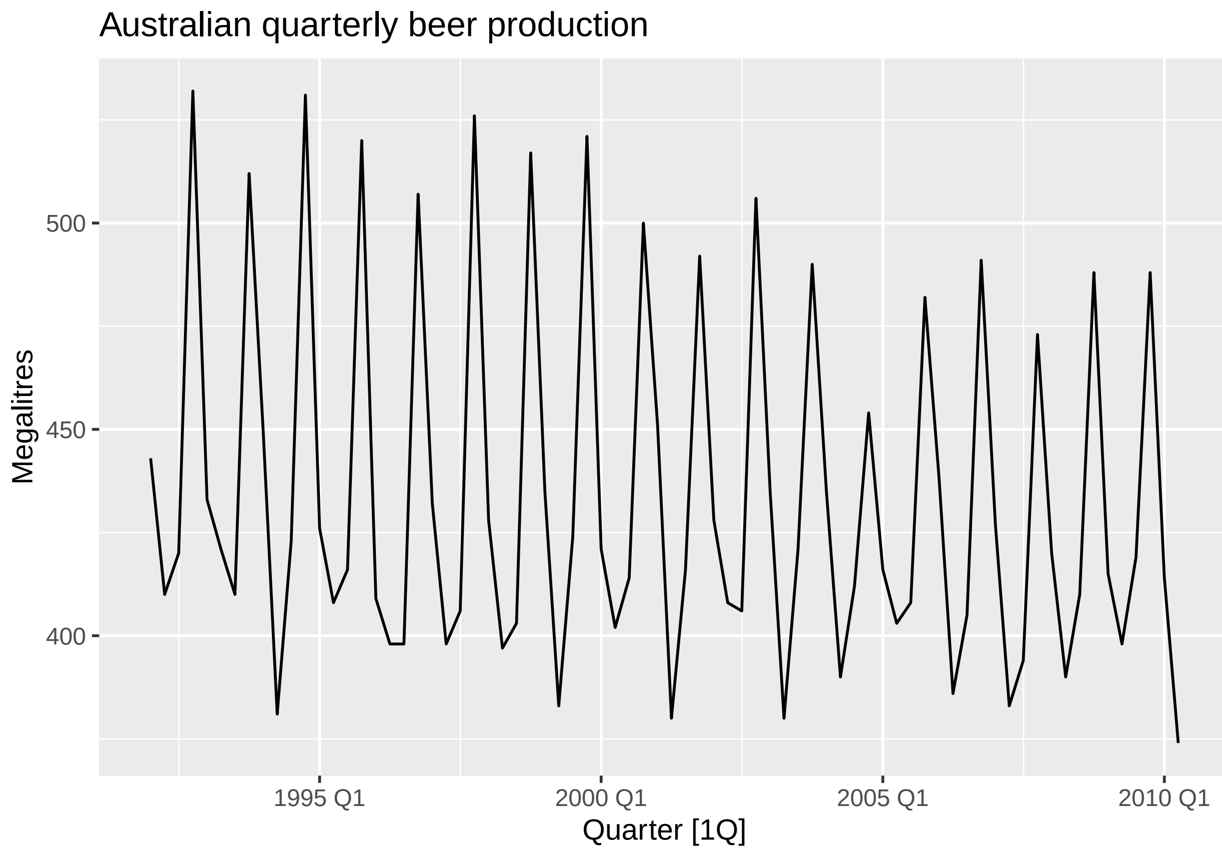
7.4 Some useful predictors

Trend

Dummy variables

Seasonal dummy variables

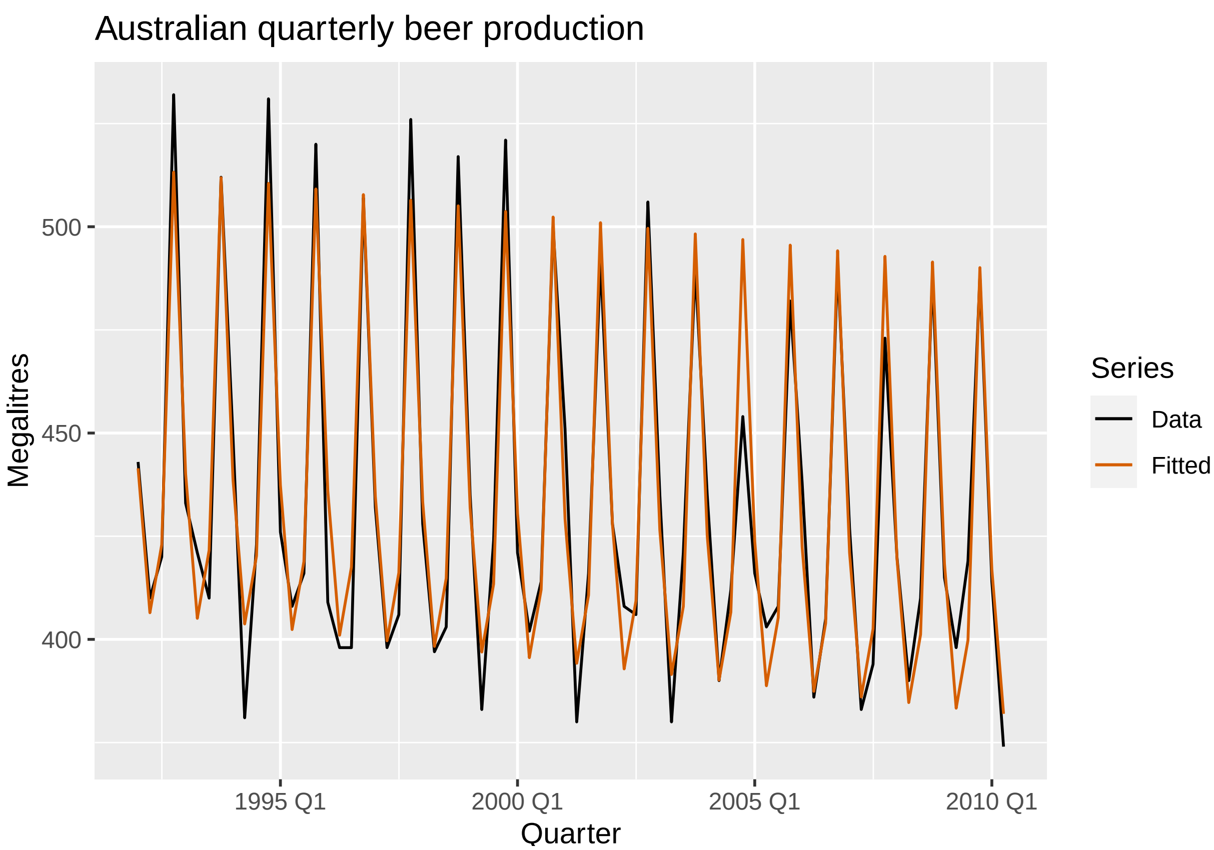
```
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
recent_production %>%
  autoplot(Beer) +
  labs(y = "Megalitres",
       title = "Australian quarterly beer production")
```



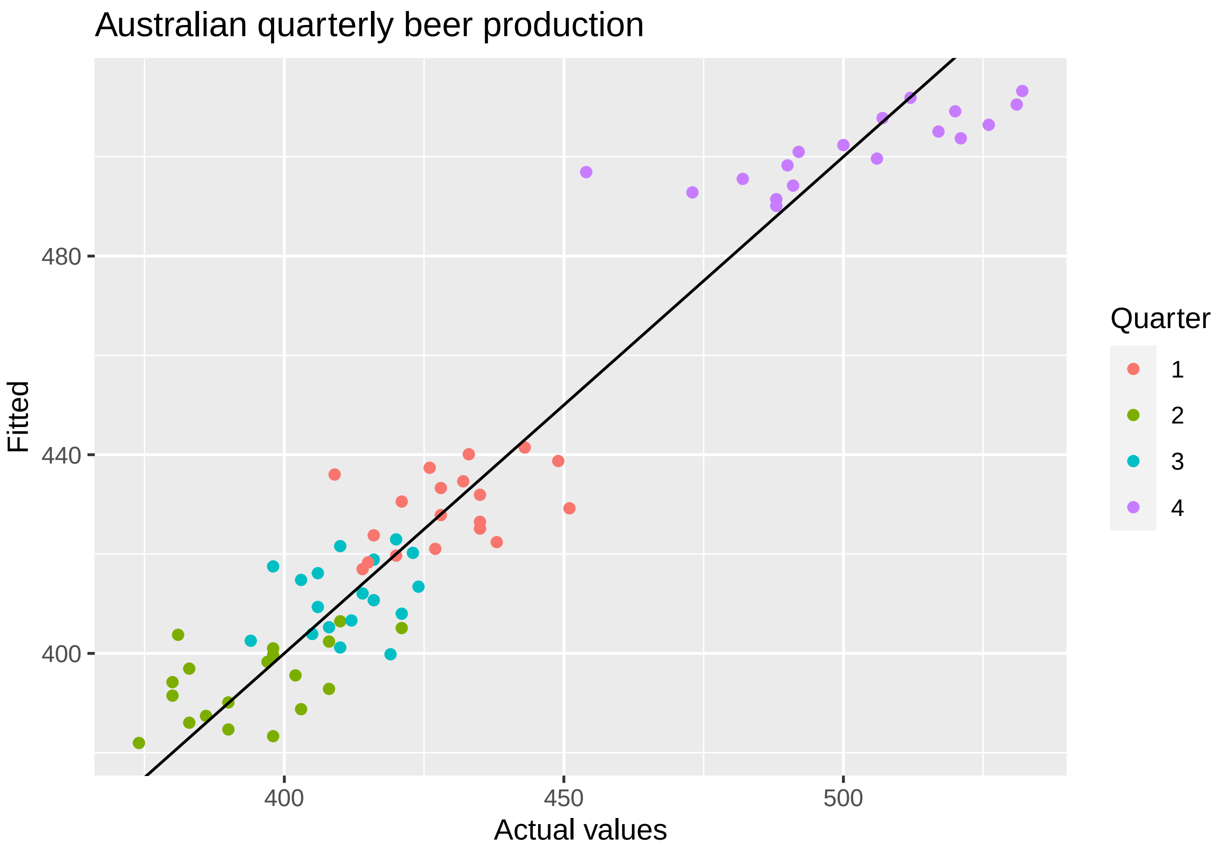

```
fit_beer <- recent_production %>%
  model(TSLM(Beer ~ trend() + season()))
report(fit_beer)
```

```
## Series: Beer
## Model: TSLM
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -42.9029  -7.5995  -0.4594   7.9908  21.7895
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   441.80044    3.73353 118.333  < 2e-16 ***
## trend()        -0.34027    0.06657  -5.111 2.73e-06 ***
## season()year2 -34.65973    3.96832  -8.734 9.10e-13 ***
## season()year3 -17.82164    4.02249  -4.430 3.45e-05 ***
## season()year4  72.79641    4.02305  18.095  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.23 on 69 degrees of freedom
## Multiple R-squared: 0.9243,  Adjusted R-squared: 0.9199
## F-statistic: 210.7 on 4 and 69 DF, p-value: < 2.22e-16
```

```
augment(fit_beer) %>%
  ggplot(aes(x = Quarter)) +
  geom_line(aes(y = Beer, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Fitted")) +
  scale_color_manual(
    values = c(Data = "black", Fitted = "#D55E00")
  ) +
  labs(y = "Megalitres",
       title = "Australian quarterly beer production") +
  guides(colour = guide_legend(title = "Series"))
```




```
augment(fit_beer) %>%
  ggplot(aes(x = Beer, y = .fitted,
             colour = factor(quarter(Quarter)))) +
  geom_point() +
  labs(y = "Fitted", x = "Actual values",
       title = "Australian quarterly beer production") +
  geom_abline(intercept = 0, slope = 1) +
  guides(colour = guide_legend(title = "Quarter"))
```



Intervention variables

Trading days

Distributed lags

Easter

Fourier series

```
fourier_beer <- recent_production %>%
  model(TSLM(Beer ~ trend() + fourier(K = 2)))
report(fourier_beer)
```

```
## Series: Beer
## Model: TSLM
##
## Residuals:
##       Min       1Q   Median       3Q      Max
## -42.9029  -7.5995  -0.4594   7.9908  21.7895
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          446.87920    2.87321 155.533  < 2e-16 ***
```

```
## trend()              -0.34027    0.06657  -5.111 2.73e-06 ***
## fourier(K = 2)C1_4   8.91082    2.01125   4.430 3.45e-05 ***
## fourier(K = 2)S1_4 -53.72807    2.01125 -26.714  < 2e-16 ***
## fourier(K = 2)C2_4 -13.98958    1.42256  -9.834 9.26e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.23 on 69 degrees of freedom
## Multiple R-squared: 0.9243,  Adjusted R-squared: 0.9199
## F-statistic: 210.7 on 4 and 69 DF, p-value: < 2.22e-16
```

7.5 Selecting predictors

```
glance(fit.consMR) %>%
  select(adj_r_squared, CV, AIC, AICc, BIC)
```

```
## # A tibble: 1 x 5
##   adj_r_squared    CV   AIC  AICc   BIC
##           <dbl> <dbl> <dbl> <dbl> <dbl>
## 1         0.763 0.104 -457. -456. -437.
```

Adjusted R^2

Cross-validation

Akaike's Information Criterion

Corrected Akaike's Information Criterion

Schwarz's Bayesian Information Criterion

Best subset regression

Stepwise regression

7.6 Forecasting with regression

7.7 Nonlinear regression

7.8 Correlation, causation and forecasting

Chapter 10 Dynamic regression models

In Chapter 7 we considered regression models of the form

$$y_t = \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + \varepsilon_t,$$

Now we will allow the errors from a regression to contain autocorrelation. in order to do that we replace ε_t with η_t which is assumed to follow an ARIMA model.

$$\begin{aligned} y_t &= \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + \eta_t, \\ (1-\phi_1 B)(1-B)\eta_t &= (1+\theta_1 B)\varepsilon_t, \end{aligned}$$

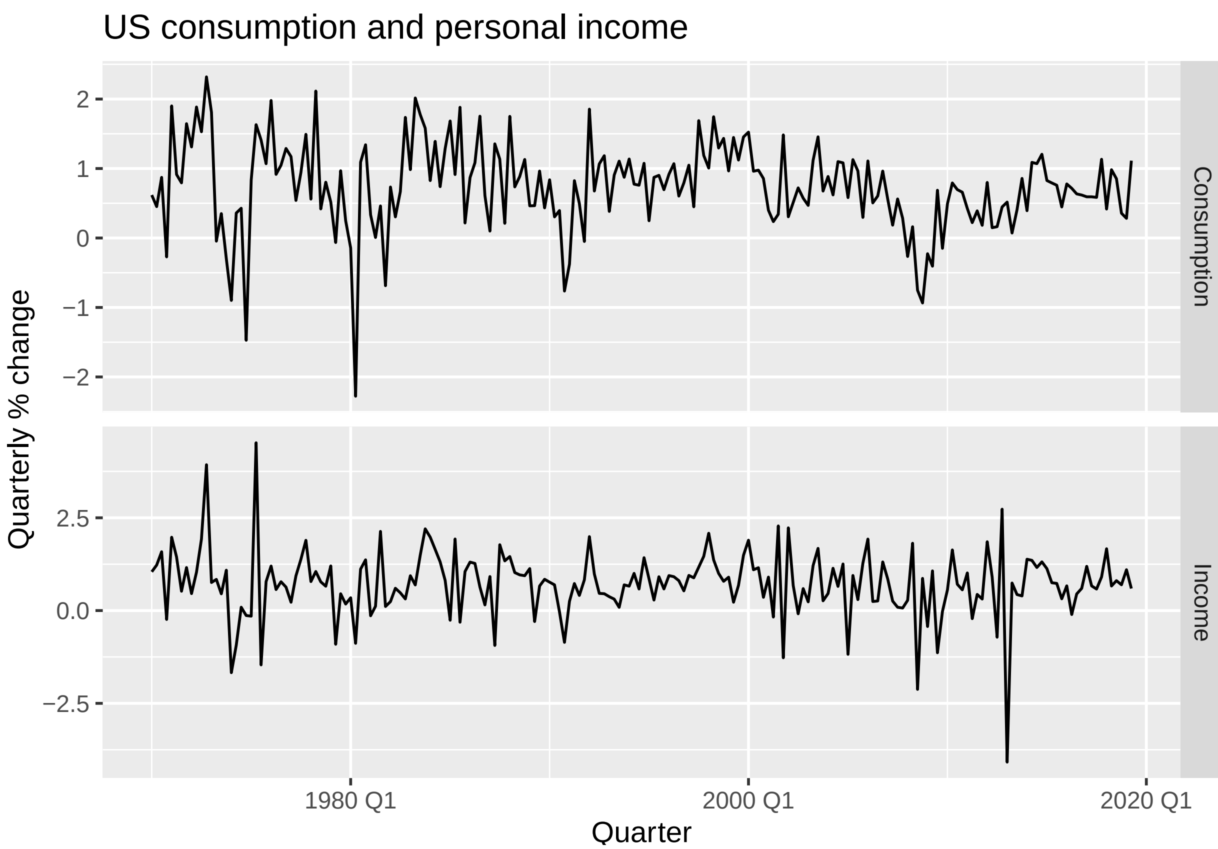
10.1 Estimation

An important consideration when estimating a **regression with ARMA errors** is that all of the variables in the model must first be stationary. We therefore first difference the non-stationary variables in the model. To maintain the form of the relationship between the response and the predictors we difference all of the variables if any of them need differencing. The resulting model is then called a *model in differences*. It is easy to see that a regression model with ARIMA errors is equivalent to a regression model in differences with ARMA errors.

10.2 Regression with ARIMA errors using fable

The `fable` function `ARIMA()` will fit a regression model with ARIMA errors if exogenous regressors are included in the formula.

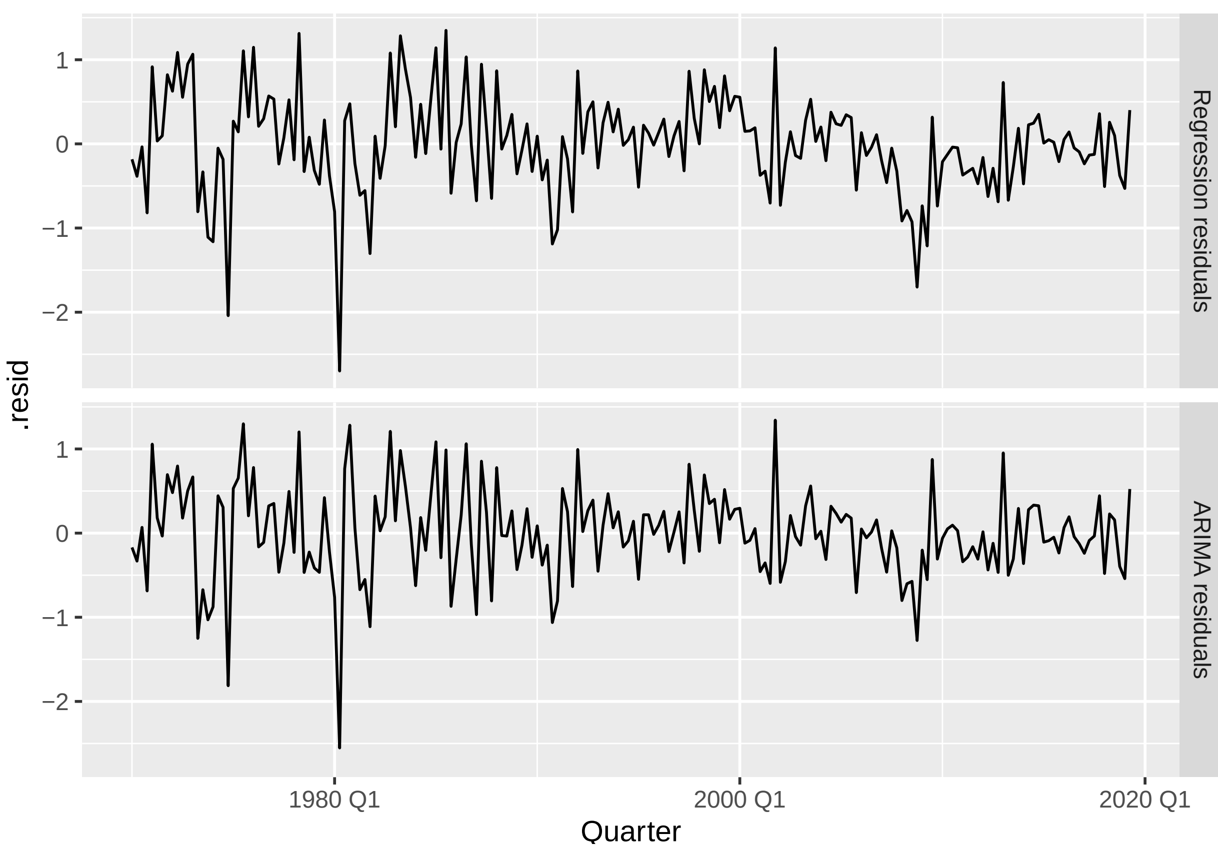
```
us_change %>%
  pivot_longer(c(Consumption, Income), names_to = "var", values_to = "value") %>%
  ggplot(aes(x = Quarter, y = value)) +
  geom_line() +
  facet_grid(vars(var), scales = "free_y") +
  labs(title = "US consumption and personal income",
       y = "Quarterly % change")
```




```
fit <- us_change %>%
  model(ARIMA(Consumption ~ Income))
report(fit)
```

```
## Series: Consumption
## Model: LM w/ ARIMA(1,0,2) errors
##
## Coefficients:
##          ar1      ma1     ma2  Income  intercept
##       0.7070  -0.6172  0.2066  0.1976     0.5949
## s.e.  0.1068   0.1218  0.0741  0.0462     0.0850
##
## sigma^2 estimated as 0.3113:  log likelihood=-163.04
## AIC=338.07   AICc=338.51   BIC=357.8
```

```
bind_rows(
    `Regression residuals` =
        as_tibble(residuals(fit, type = "regression")),
    `ARIMA residuals` =
        as_tibble(residuals(fit, type = "innovation")),
    .id = "type"
  ) %>%
  mutate(
    type = factor(type, levels=c(
      "Regression residuals", "ARIMA residuals"))
  ) %>%
  ggplot(aes(x = Quarter, y = .resid)) +
  geom_line() +
  facet_grid(vars(type))
```



There are two types of residual: regression and innovation (the latter are the ARIMA residuals):

```r
as_tibble(residuals(fit, type="regression")) %>%
  head
```

```
## # A tibble: 6 x 3
##   .model                      Quarter  .resid
##   <chr>                         <qtr>   <dbl>
## 1 ARIMA(Consumption ~ Income) 1970 Q1 -0.183
## 2 ARIMA(Consumption ~ Income) 1970 Q2 -0.385
## 3 ARIMA(Consumption ~ Income) 1970 Q3 -0.0353
## 4 ARIMA(Consumption ~ Income) 1970 Q4 -0.819
## 5 ARIMA(Consumption ~ Income) 1971 Q1  0.916
## 6 ARIMA(Consumption ~ Income) 1971 Q2  0.0342
```

```r
as_tibble(residuals(fit, type="innovation")) %>%
  head
```

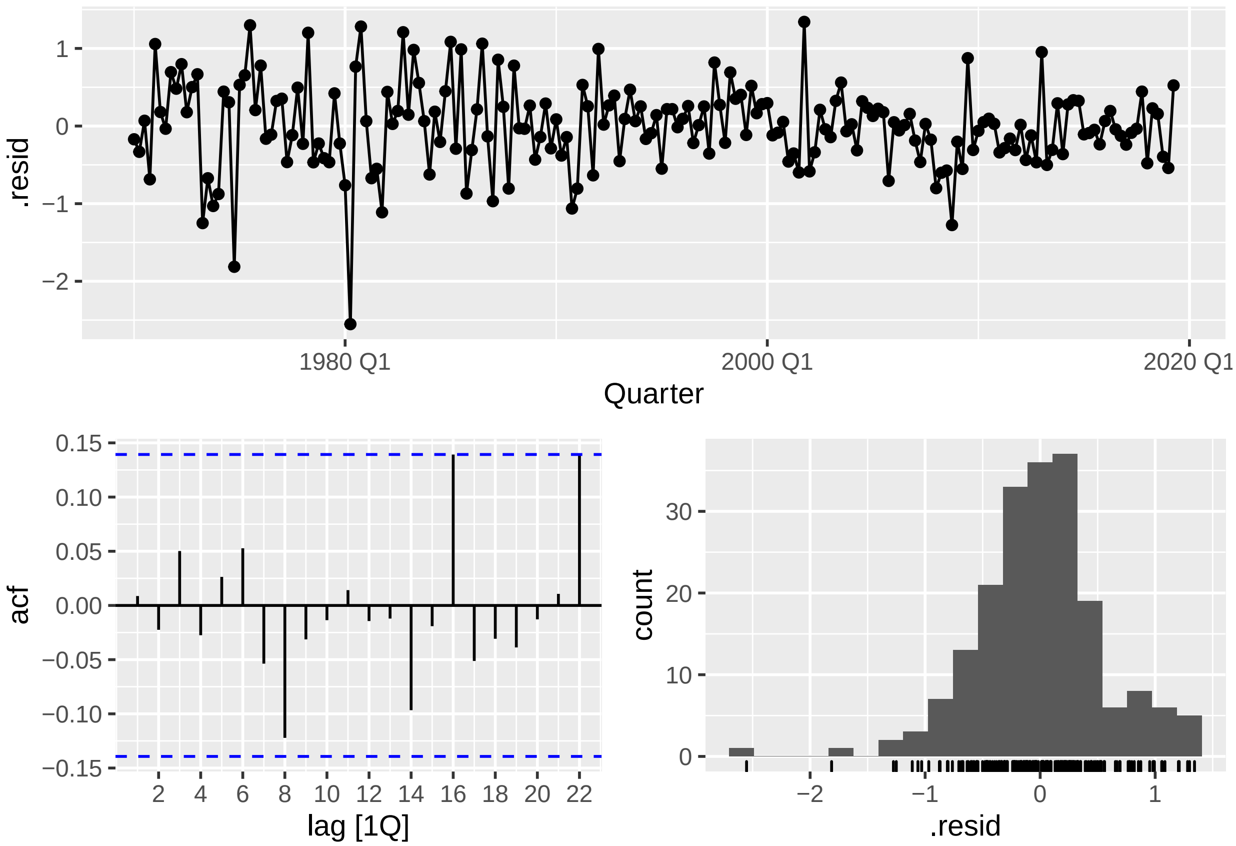
```
## # A tibble: 6 x 3
##   .model                      Quarter  .resid
##   <chr>                         <qtr>   <dbl>
## 1 ARIMA(Consumption ~ Income) 1970 Q1 -0.170
## 2 ARIMA(Consumption ~ Income) 1970 Q2 -0.332
## 3 ARIMA(Consumption ~ Income) 1970 Q3  0.0681
## 4 ARIMA(Consumption ~ Income) 1970 Q4 -0.687
## 5 ARIMA(Consumption ~ Income) 1971 Q1  1.06
## 6 ARIMA(Consumption ~ Income) 1971 Q2  0.181
```

The innovation residuals are stored in the augmented fit:

```r
augment(fit) %>%
  select(.innov)
```

```
## # A tsibble: 198 x 2 [1Q]
##     .innov Quarter
##      <dbl>   <qtr>
##  1 -0.170  1970 Q1
##  2 -0.332  1970 Q2
##  3  0.0681 1970 Q3
##  4 -0.687  1970 Q4
##  5  1.06   1971 Q1
##  6  0.181  1971 Q2
##  7 -0.0348 1971 Q3
##  8  0.695  1971 Q4
##  9  0.481  1972 Q1
## 10  0.797  1972 Q2
## # ... with 188 more rows
```

```r
fit %>% gg_tsresiduals()
```




```r
augment(fit) %>%
  features(.innov, ljung_box, dof = 5, lag = 8)
```

```
## # A tibble: 1 x 3
##   .model                       lb_stat lb_pvalue
##   <chr>                          <dbl>     <dbl>
## 1 ARIMA(Consumption ~ Income)     5.21     0.157
```