Chapter 13 Some practical forecasting issues

Ankit Gupta

10/04/2023'

13.1 Weekly, daily and sub-daily data

Weekly, daily and sub-daily data can be challenging for forecasting, although for different reasons.

Weekly data

Weekly data is difficult to work with because the seasonal period (the number of weeks in a year) is both large and non-integer. The average number of weeks in a year is 52.18. Most of the methods we have considered require the seasonal period to be an integer. Even if we approximate it by 52, most of the methods will not handle such a large seasonal period efficiently.

The simplest approach is to use an STL decomposition along with a non-seasonal method applied to the seasonally adjusted data (as discussed in Chapter 3). Here is an example using weekly data on US finished motor gasoline products supplied (in millions of barrels per day) from February 1991 to May 2005.

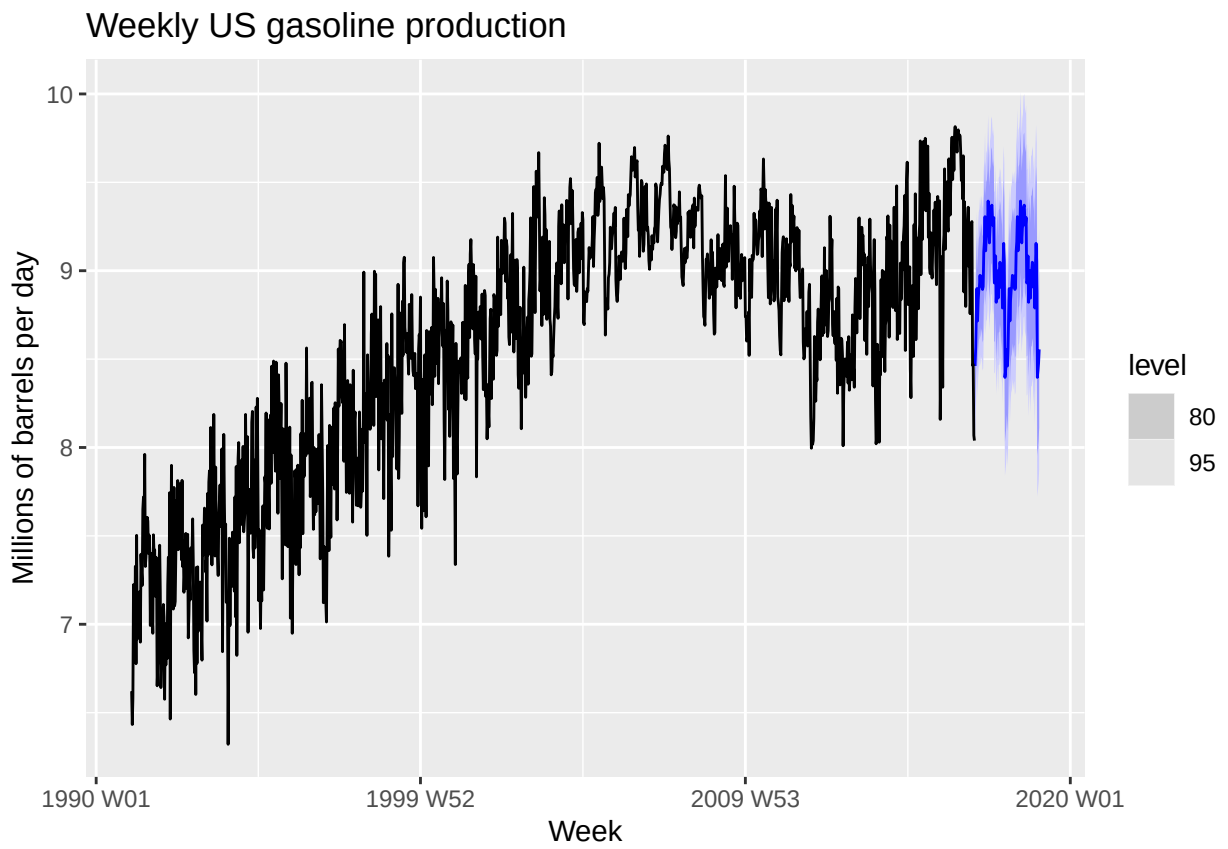
```
library(fpp3)
```

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

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

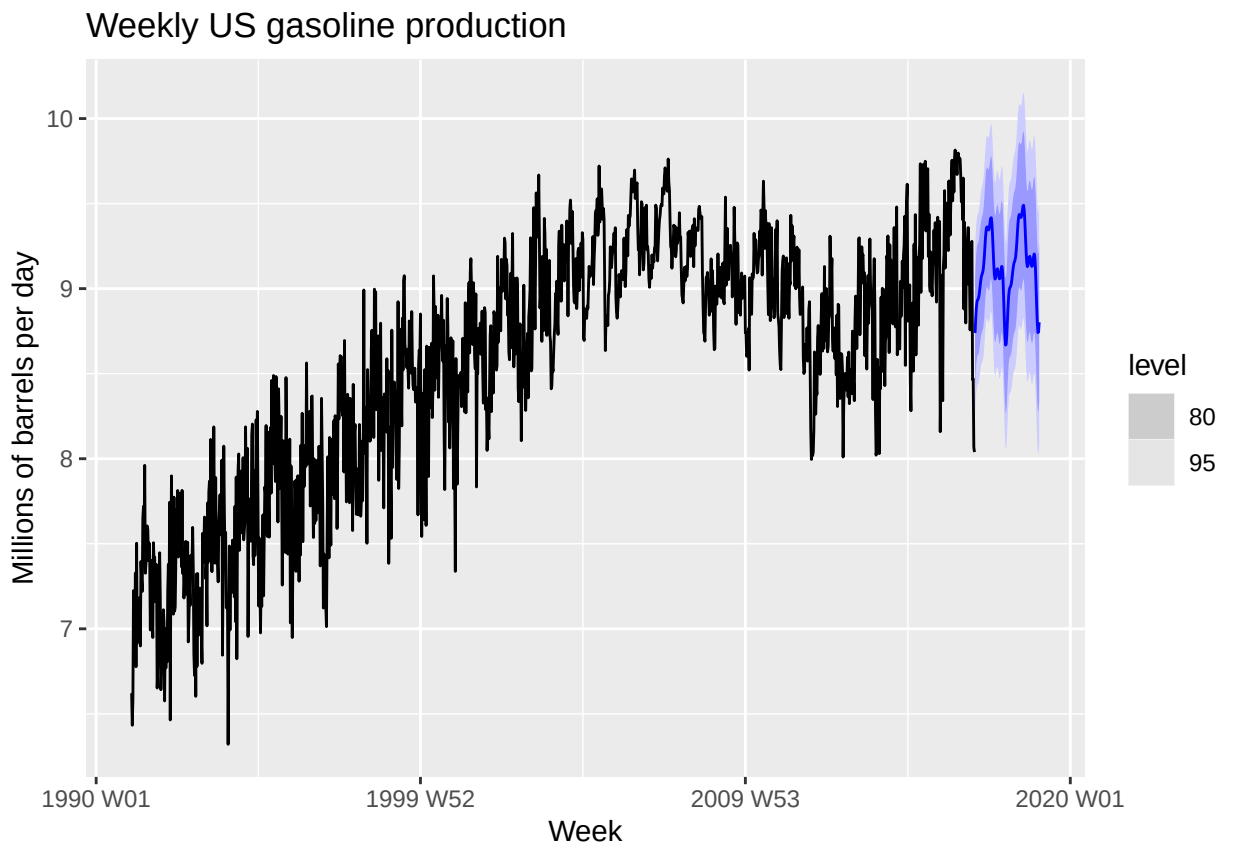
```
## -- Conflicts ------------------------------------------------ fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()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

```
my_dcmp_spec <- decomposition_model(
  STL(Barrels),
  ETS(season_adjust ~ season("N"))
)
us_gasoline |>
  model(stl_ets = my_dcmp_spec) |>
  forecast(h = "2 years") |>
  autoplot(us_gasoline) +
  labs(y = "Millions of barrels per day",
       title = "Weekly US gasoline production")
```



An alternative approach is to use a dynamic harmonic regression model, as discussed in Section 10.5. In the following example, the number of Fourier terms was selected by minimising the AICc. The order of the ARIMA model is also selected by minimising the AICc, although that is done within the ARIMA() function. We use PDQ(0,0,0) to prevent ARIMA() trying to handle the seasonality using seasonal ARIMA components.

```
gas_dhr <- us_gasoline |>
  model(dhr = ARIMA(Barrels ~ PDQ(0, 0, 0) + fourier(K = 6)))
gas_dhr |>
  forecast(h = "2 years") |>
  autoplot(us_gasoline) +
  labs(y = "Millions of barrels per day",
       title = "Weekly US gasoline production")
```



The fitted model has 6 pairs of Fourier terms and can be written as $y_t = b_t + \Sigma_{j=1}^{6}[\alpha_j \sin(\frac{2\pi jt}{52.18}) + \beta_j \cos(\frac{2\pi jt}{52.18})] + \eta_t$

where η_t is an ARIMA(0,1,1) process. Because η_t is non-stationary, the model is actually estimated on the differences of the variables on both sides of this equation. There are 12 parameters to capture the seasonality, while the total number of degrees of freedom is 14 (the other two coming from the MA parameter and the drift parameter).

The STL approach is preferable when the seasonality changes over time. The dynamic harmonic regression approach is preferable if there are covariates that are useful predictors as these can be added as additional regressors.

Daily and sub-daily data

Daily and sub-daily (such as hourly) data are challenging for a different reason — they often involve multiple seasonal patterns, and so we need to use a method that handles such complex seasonality.

Of course, if the time series is relatively short so that only one type of seasonality is present, then it will be possible to use one of the single-seasonal methods we have discussed in previous chapters (e.g., ETS or a seasonal ARIMA model). But when the time series is long enough so that some of the longer seasonal periods become apparent, it will be necessary to use STL, dynamic harmonic regression or Prophet, as discussed in Section 12.1.

However, these methods only allow for regular seasonality. Capturing seasonality associated with moving events such as Easter, Eid, or the Chinese New Year is more difficult. Even with monthly data, this can be tricky as the festivals can fall in either March or April (for Easter), in January or February (for the Chinese New Year), or at any time of the year (for Eid).

The best way to deal with moving holiday effects is to include dummy variables in the model. This can be

done within the ARIMA() or prophet() functions, for example, but not within ETS(). In fact, prophet() has a holiday() special to easily incorporate holiday effects.

13.2 Time series of counts

All of the methods discussed in this book assume that the data have a continuous sample space. But often data comes in the form of counts. For example, we may wish to forecast the number of customers who enter a store each day. We could have 0,1,2,..., customers, but we cannot have 3.45693 customers.

In practice, this rarely matters provided our counts are sufficiently large. If the minimum number of customers is at least 100, then the difference between a continuous sample space $[100, \infty)$ and the discrete sample space $\{100, 101, 102, ...\}$ has no perceivable effect on our forecasts. However, if our data contains small counts $(0, 1, 2, ...)$, then we need to use forecasting methods that are more appropriate for a sample space of non-negative integers.

Such models are beyond the scope of this book. However, there is one simple method which gets used in this context, that we would like to mention. It is "Croston's method", named after its British inventor, John Croston, and first described in Croston (1972). Actually, this method does not properly deal with the count nature of the data either, but it is used so often, that it is worth knowing about it.

With Croston's method, we construct two new series from our original time series by noting which time periods contain zero values, and which periods contain non-zero values. Let q_i be the i^{th} non-zero quantity, and let a_i be the time between q_{i-1} and q_i. Croston's method involves separate simple exponential smoothing forecasts on the two new series a and q . Because the method is usually applied to time series of demand for items, q is often called the "demand" and a the "inter-arrival time".

If $\hat{q}_{i+1|i}$ and $\hat{a}_{i+1|i}$ are the one-step forecasts of the $(i+1)^{th}$ demand and inter-arrival time respectively, based on data up to demand i , then Croston's method gives

$\hat{q}_{i+1|i} = (1-\alpha_q)\hat{q}_{i|i-1} + \alpha_q q_i$, (13.1)

$\hat{a}_{i+1|i} = (1-\alpha_a)\hat{a}_{i|i-1} + \alpha_a a_i$. (13.2)

The smoothing parameters α_a and α_q take values between 0 and 1. Let j be the time for the last observed positive observation. Then the h-step ahead forecast for the demand at time T+h, is given by the ratio

$\hat{y}_{T+h|T} = \frac{\hat{q}_{j+1|j}}{\hat{a}_{j+1|j}}$. (1)

There are no algebraic results allowing us to compute prediction intervals for this method, because the method does not correspond to any statistical model (Shenstone & Hyndman, 2005).

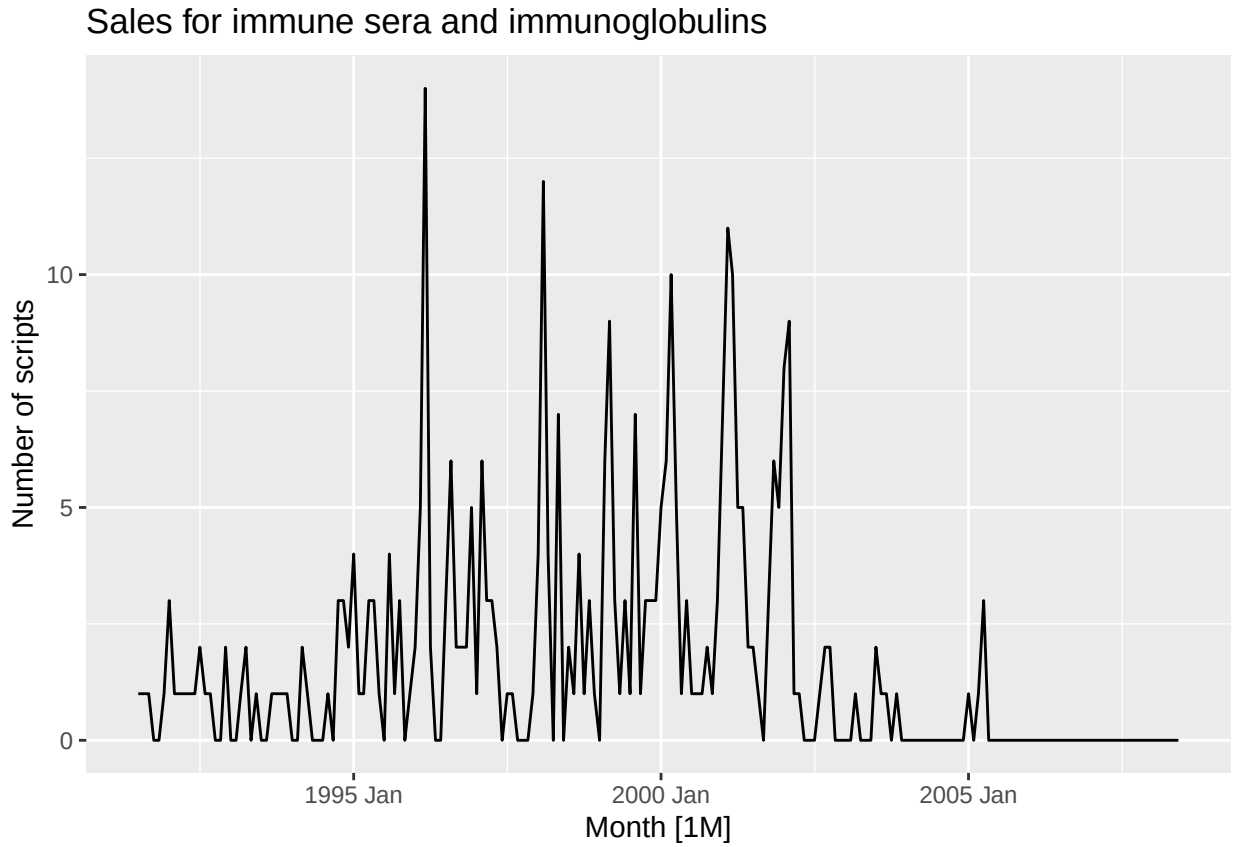
The CROSTON() function produces forecasts using Croston's method. The two smoothing parameters α_a and α_q are estimated from the data. This is different from the way Croston envisaged the method being used. He would simply use $\alpha_a = \alpha_q = 0.1$, and set a_0 and q_0 to be equal to the first observation in each of the series.

Example: Pharmaceutical sales

Figure 13.3 shows the numbers of scripts sold each month for immune sera and immunoglobulin products in Australia. The data contain small counts, with many months registering no sales at all, and only small numbers of items sold in other months.

```
j06 <- PBS |>
  filter(ATC2 == "J06") |>
  summarise(Scripts = sum(Scripts))

j06 |> autoplot(Scripts) +
  labs(y="Number of scripts",
       title = "Sales for immune sera and immunoglobulins")
```



In this example, the smoothing parameters are estimated to be $\alpha_a = 0.08, \alpha_q = 0.71$, $\hat{q}_{1|0} = 4.17$, and $\hat{a}_{1|0} = 3.52$.

The final forecasts for the two series are $\hat{q}_{T+1|T} = 2.419$ and $\hat{a}_{T+1|T} = 2.484$. So the forecasts are all equal to $\hat{y}_{T+h|T} = 2.419/2.484 = 0.974$.

In practice, fable does these calculations for you:

```
j06 |>
  model(CROSTON(Scripts)) |>
  forecast(h = 6)
```

```
## # A fable: 6 x 4 [1M]
## # Key:     .model [1]
##   .model              Month   Scripts .mean
##   <chr>               <mth>    <dist> <dbl>
## 1 CROSTON(Scripts) 2008 Jul 0.9735062 0.974
## 2 CROSTON(Scripts) 2008 Aug 0.9735062 0.974
## 3 CROSTON(Scripts) 2008 Sep 0.9735062 0.974
## 4 CROSTON(Scripts) 2008 Oct 0.9735062 0.974
## 5 CROSTON(Scripts) 2008 Nov 0.9735062 0.974
## 6 CROSTON(Scripts) 2008 Dec 0.9735062 0.974
```

```
#> # A fable: 6 x 4 [1M]
#> # Key:     .model [1]
#>   .model              Month Scripts .mean
#>   <chr>               <mth>  <dist> <dbl>
#> 1 CROSTON(Scripts) 2008 Jul  0.9735 0.974
```

```
#> 2 CROSTON(Scripts) 2008 Aug  0.9735 0.974
#> 3 CROSTON(Scripts) 2008 Sep  0.9735 0.974
#> 4 CROSTON(Scripts) 2008 Oct  0.9735 0.974
#> 5 CROSTON(Scripts) 2008 Nov  0.9735 0.974
#> 6 CROSTON(Scripts) 2008 Dec  0.9735 0.974
```

The Scripts column repeats the mean rather than provide a full distribution, because there is no underlying stochastic model.

Forecasting models that deal more directly with the count nature of the data, and allow for a forecasting distribution, are described in Christou & Fokianos (2015).

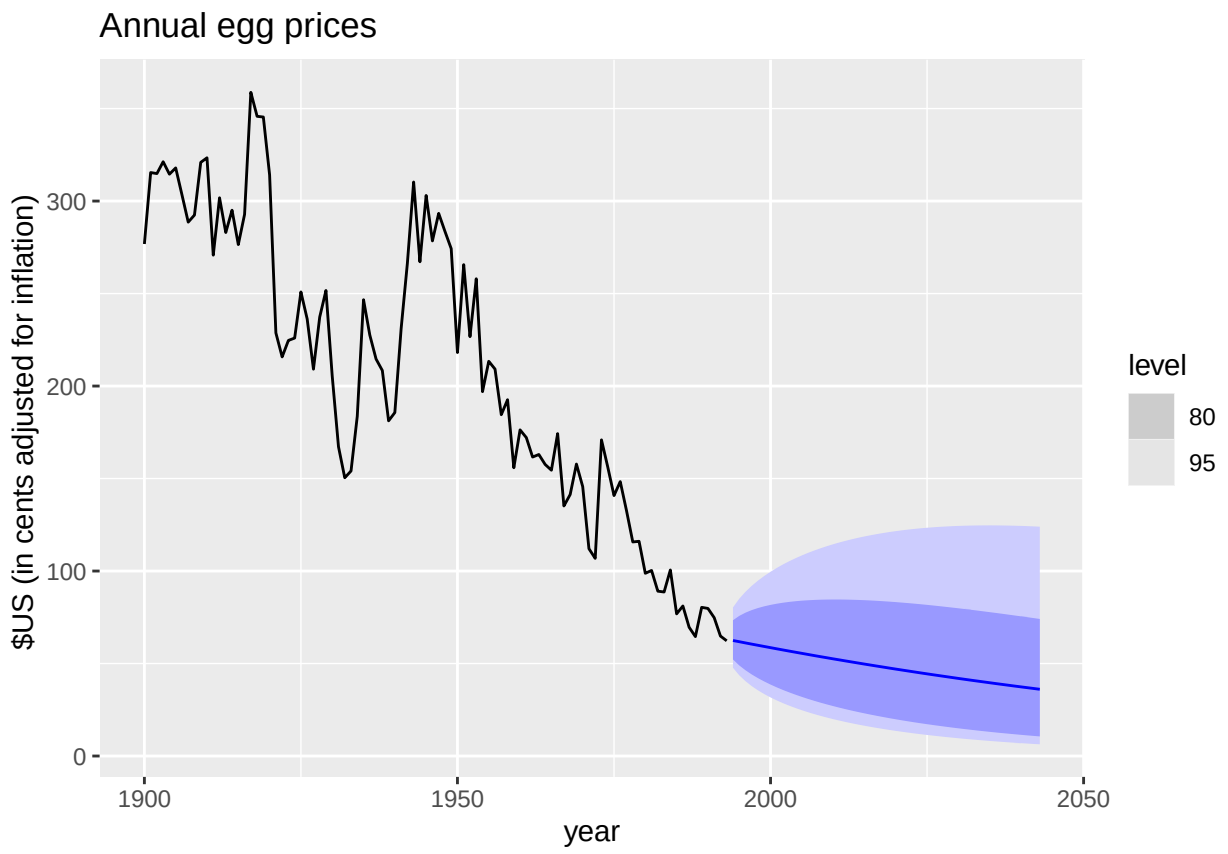
13.3 Ensuring forecasts stay within limits

It is common to want forecasts to be positive, or to require them to be within some specified range [a,b]. Both of these situations are relatively easy to handle using transformations.

Positive forecasts

To impose a positivity constraint, we can simply work on the log scale. For example, consider the real price of a dozen eggs (1900-1993; in cents) shown in Figure 13.4. Because of the log transformation, the forecast distributions are constrained to stay positive, and so they will become progressively more skewed as the mean decreases.

```
egg_prices <- prices |> filter(!is.na(eggs))
egg_prices |>
  model(ETS(log(eggs) ~ trend("A"))) |>
  forecast(h = 50) |>
  autoplot(egg_prices) +
  labs(title = "Annual egg prices",
       y = "$US (in cents adjusted for inflation) ")
```



Forecasts constrained to an interval

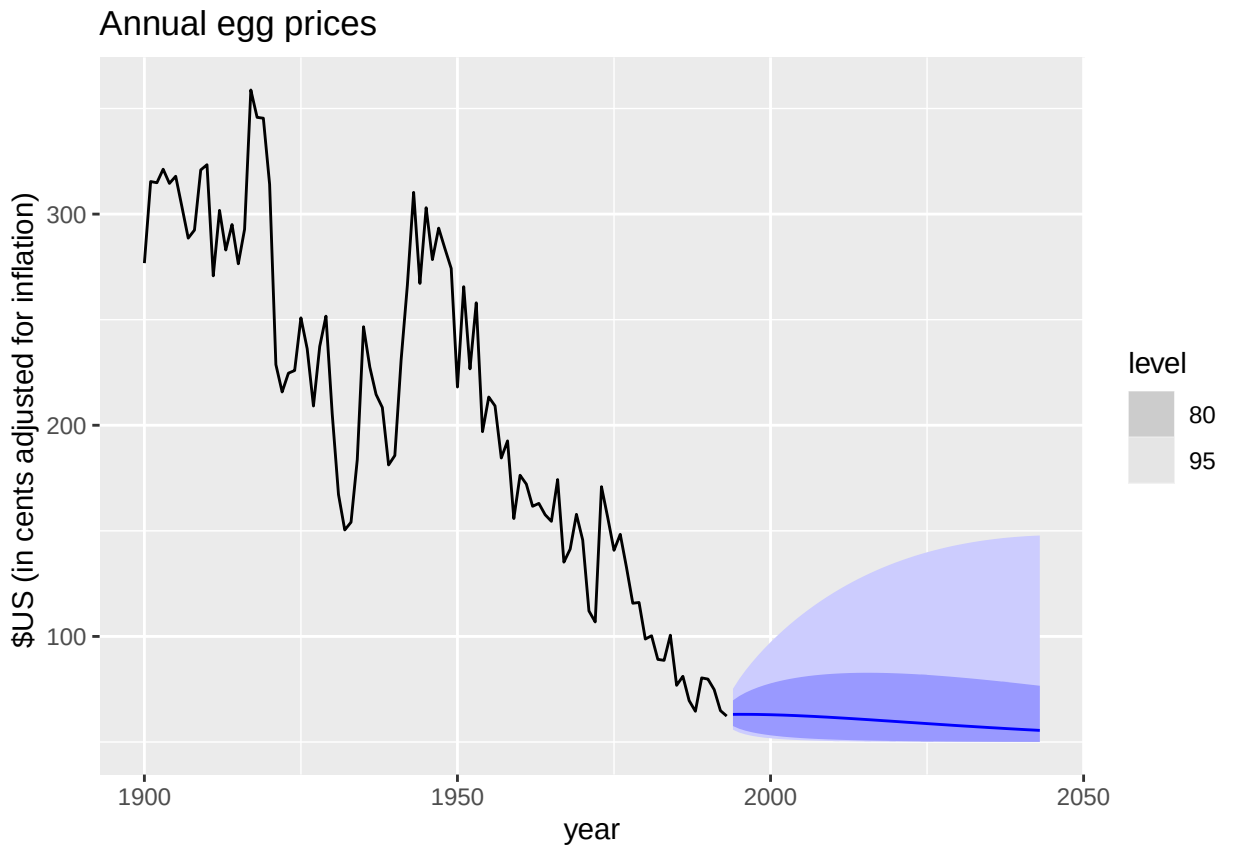
To see how to handle data constrained to an interval, imagine that the egg prices were constrained to lie within a=50 and b=400. Then we can transform the data using a scaled logit transform which maps (a, b) to the whole real line: $y = \log(\frac{x-a}{b-x})$,

where x is on the original scale and y is the transformed data. To reverse the transformation, we will use $x = \frac{(b-a)e^y}{1+e^y} + a$.

This is not a built-in transformation, so we will need to first setup the transformation functions.

```
scaled_logit <- function(x, lower = 0, upper = 1) {
  log((x - lower) / (upper - x))
}
inv_scaled_logit <- function(x, lower = 0, upper = 1) {
  (upper - lower) * exp(x) / (1 + exp(x)) + lower
}
my_scaled_logit <- new_transformation(
                    scaled_logit, inv_scaled_logit)
egg_prices |>
  model(
    ETS(my_scaled_logit(eggs, lower = 50, upper = 400)
          ~ trend("A"))
  ) |>
  forecast(h = 50) |>
  autoplot(egg_prices) +
  labs(title = "Annual egg prices",
```

```
       y = "$US (in cents adjusted for inflation) ")
```



The bias-adjustment is automatically applied here, and the prediction intervals from these transformations have the same coverage probability as on the transformed scale, because quantiles are preserved under monotonically increasing transformations.

The prediction intervals lie above 50 due to the transformation. As a result of this artificial (and unrealistic) constraint, the forecast distributions have become extremely skewed.

13.4 Forecast combinations

An easy way to improve forecast accuracy is to use several different methods on the same time series, and to average the resulting forecasts. Over 50 years ago, John Bates and Clive Granger wrote a famous paper (Bates & Granger, 1969), showing that combining forecasts often leads to better forecast accuracy. Twenty years later, Clemen (1989) wrote:

"The results have been virtually unanimous: combining multiple forecasts leads to increased forecast accuracy. In many cases one can make dramatic performance improvements by simply averaging the forecasts."

While there has been considerable research on using weighted averages, or some other more complicated combination approach, using a simple average has proven hard to beat.

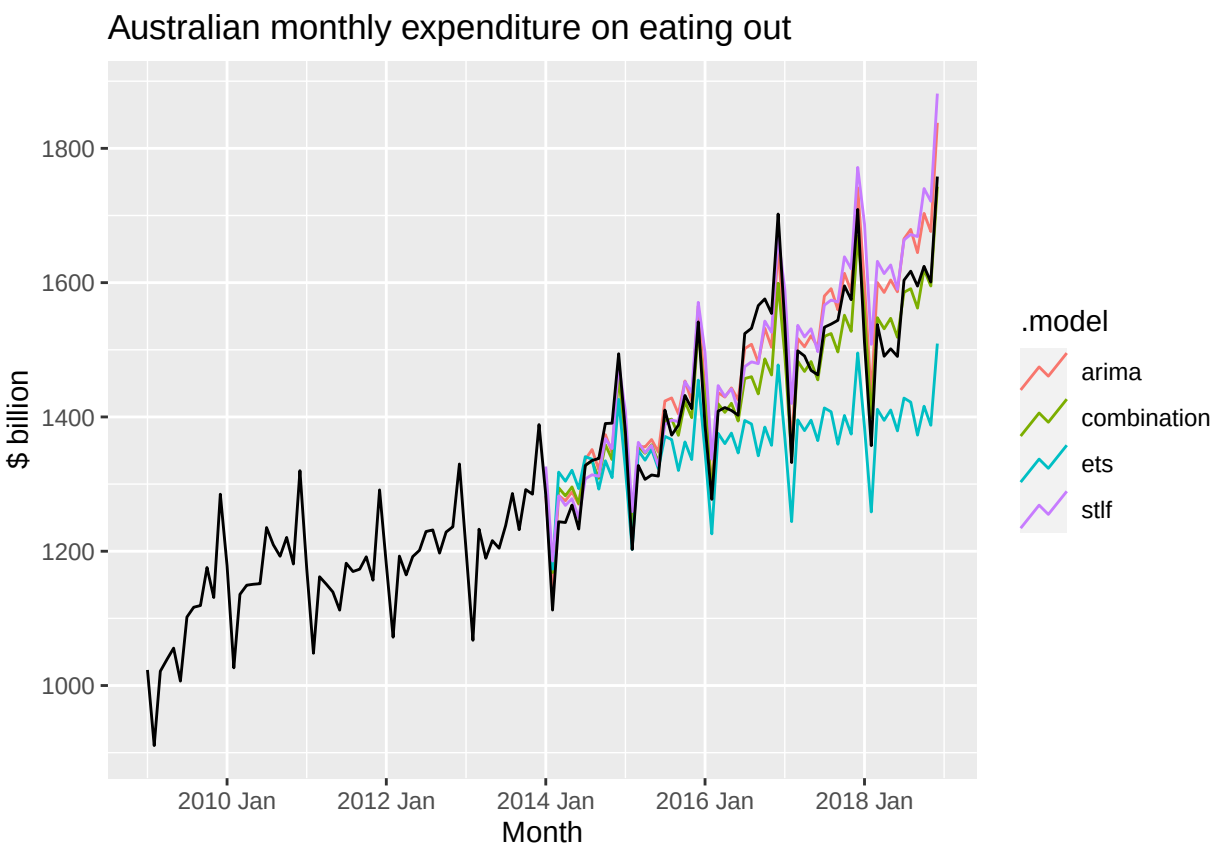
Here is an example using monthly revenue from take-away food in Australia, from April 1982 to December 2018. We use forecasts from the following models: ETS, STL-ETS, and ARIMA; and we compare the results using the last 5 years (60 months) of observations.

```
auscafe <- aus_retail |>
  filter(stringr::str_detect(Industry, "Takeaway")) |>
  summarise(Turnover = sum(Turnover))
```

```
train <- auscafe |>
  filter(year(Month) <= 2013)
STLF <- decomposition_model(
  STL(log(Turnover) ~ season(window = Inf)),
  ETS(season_adjust ~ season("N"))
)
cafe_models <- train |>
  model(
    ets = ETS(Turnover),
    stlf = STLF,
    arima = ARIMA(log(Turnover))
  ) |>
  mutate(combination = (ets + stlf + arima) / 3)
cafe_fc <- cafe_models |>
  forecast(h = "5 years")
```

Notice that we form a combination in the mutate() function by simply taking a linear function of the estimated models. This very simple syntax will automatically handle the forecast distribution appropriately by taking account of the correlation between the forecast errors of the models that are included. However, to keep the next plot simple, we will omit the prediction intervals.

```
cafe_fc |>
  autoplot(auscafe |> filter(year(Month) > 2008),
           level = NULL) +
  labs(y = "$ billion",
       title = "Australian monthly expenditure on eating out")
```



```
cafe_fc |>
  accuracy(auscafe) |>
  arrange(RMSE)
```

```
## # 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 combination Test    8.09  41.0  31.8  0.401  2.19 0.776 0.790 0.747
## 2 arima       Test  -25.4   46.2  38.9 -1.77   2.65 0.949 0.890 0.786
## 3 stlf        Test  -36.9   64.1  51.7 -2.55   3.54 1.26  1.23  0.775
## 4 ets         Test   86.5  122.  101.   5.51   6.66 2.46  2.35  0.880
```

```
#> # A tibble: 4 × 10
#>   .model      .type     ME  RMSE   MAE    MPE  MAPE  MASE RMSSE  ACF1
#>   <chr>       <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 combination Test    8.09  41.0  31.8  0.401  2.19 0.776 0.790 0.747
#> 2 arima       Test  -25.4   46.2  38.9 -1.77   2.65 0.949 0.890 0.786
#> 3 stlf        Test  -36.9   64.1  51.7 -2.55   3.54 1.26  1.23  0.775
#> 4 ets         Test   86.5  122.  101.   5.51   6.66 2.46  2.35  0.880
```

ARIMA does particularly well with this series, while the combination approach does even better (based on most measures including RMSE and MAE). For other data, ARIMA may be quite poor, while the combination approach is usually not far off, or better than, the best component method.

Forecast combination distributions

The cafe_fc object contains forecast distributions, from which any prediction interval can usually be computed. Let's look at the intervals for the first period.

```
cafe_fc |> filter(Month == min(Month))
```

```
## # A fable: 4 x 4 [1M]
## # Key:     .model [4]
##   .model         Month           Turnover .mean
##   <chr>          <mth>             <dist> <dbl>
## 1 ets         2014 Jan      N(1289, 1118) 1289.
## 2 stlf        2014 Jan t(N(7.2, 0.00063)) 1326.
## 3 arima       2014 Jan t(N(7.2, 0.00061)) 1283.
## 4 combination 2014 Jan            1299.38 1299.
```

```
#> # A fable: 4 x 4 [1M]
#> # Key:     .model [4]
#>   .model         Month           Turnover .mean
#>   <chr>          <mth>             <dist> <dbl>
#> 1 ets         2014 Jan      N(1289, 1118) 1289.
#> 2 stlf        2014 Jan t(N(7.2, 0.00063)) 1326.
#> 3 arima       2014 Jan t(N(7.2, 0.00061)) 1283.
#> 4 combination 2014 Jan               1299 1299.
```

The first three are a mixture of normal and transformed normal distributions. The package does not yet combine such diverse distributions, so the combination output is simply the mean instead.

However, if we work with simulated sample paths, it is possible to create forecast distributions for the combination forecast as well.

```
cafe_futures <- cafe_models |>
  # Generate 1000 future sample paths
```

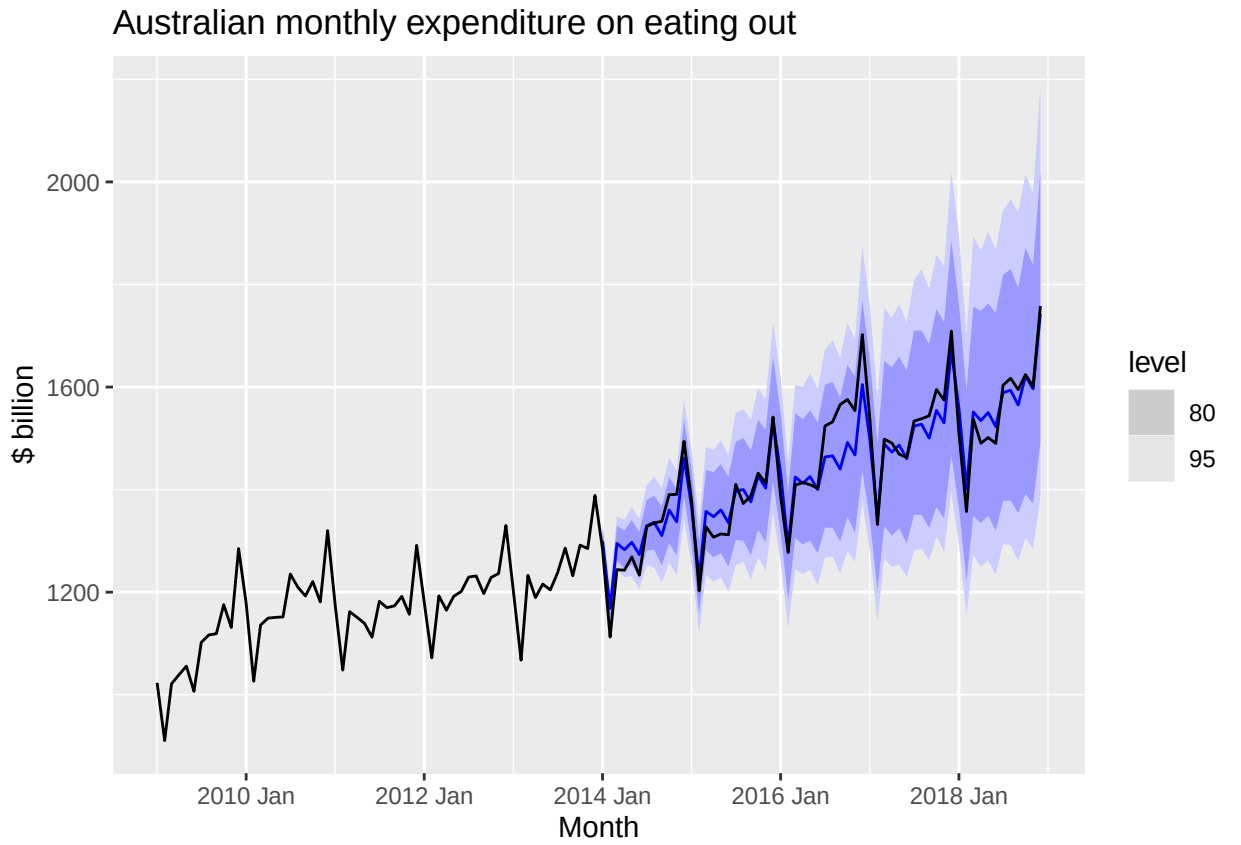
```
  generate(h = "5 years", times = 1000) |>
  # Compute forecast distributions from future sample paths
  as_tibble() |>
  group_by(Month, .model) |>
  summarise(
    dist = distributional::dist_sample(list(.sim))
  ) |>
  ungroup() |>
  # Create fable object
  as_fable(index = Month, key = .model,
           distribution = dist, response = "Turnover")
```

```
## `summarise()` has grouped output by 'Month'. You can override using the
## `.groups` argument.
```

```
## Warning: The dimnames of the fable's distribution are missing and have been set
## to match the response variables.
```

Now all four models, including the combination, are stored as empirical distributions, and we can plot prediction intervals for the combination forecast, as shown in Figure 13.7.

```
cafe_futures |>
  filter(.model == "combination") |>
  autoplot(auscafe |> filter(year(Month) > 2008)) +
  labs(y = "$ billion",
       title = "Australian monthly expenditure on eating out")
```



To check the accuracy of the 95% prediction intervals, we can use a Winkler score (defined in Section 5.9).

```
cafe_futures |>
  accuracy(auscafe, measures = interval_accuracy_measures,
    level = 95) |>
  arrange(winkler)
```

```
## # A tibble: 4 x 3
##   .model      .type winkler
##   <chr>       <chr>   <dbl>
## 1 combination Test     412.
## 2 stlf        Test     584.
## 3 ets         Test     751.
## 4 arima       Test     776.
```

```
#> # A tibble: 4 × 5
#>   .model      .type winkler pinball scaled_pinball
#>   <chr>       <chr>   <dbl>   <dbl>          <dbl>
#> 1 combination Test     420.    17.6          0.214
#> 2 stlf        Test     596.    30.2          0.369
#> 3 ets         Test     731.    23.9          0.292
#> 4 arima       Test     766.    38.6          0.471
```

Lower is better, so the combination forecast is again better than any of the component models.

13.5 Prediction intervals for aggregates

A common problem is to forecast the aggregate of several time periods of data, using a model fitted to the disaggregated data. For example, we may have monthly data but wish to forecast the total for the next year. Or we may have weekly data, and want to forecast the total for the next four weeks.

If the point forecasts are means, then adding them up will give a good estimate of the total. But prediction intervals are more tricky due to the correlations between forecast errors.

A general solution is to use simulations. Here is an example using ETS models applied to Australian take-away food sales, assuming we wish to forecast the aggregate revenue in the next 12 months.

```
fit <- auscafe |>
  # Fit a model to the data
  model(ETS(Turnover))
futures <- fit |>
  # Simulate 10000 future sample paths, each of length 12
  generate(times = 10000, h = 12) |>
  # Sum the results for each sample path
  as_tibble() |>
  group_by(.rep) |>
  summarise(.sim = sum(.sim)) |>
  # Store as a distribution
  summarise(total = distributional::dist_sample(list(.sim)))
```

We can compute the mean of the simulations, along with prediction intervals:

```
futures |>
  mutate(
    mean = mean(total),
    pi80 = hilo(total, 80),
    pi95 = hilo(total, 95)
  )
```

```
## # A tibble: 1 x 4
##           total   mean                  pi80                  pi95
##          <dist>  <dbl>                <hilo>                <hilo>
## 1 sample[10000] 19213. [18320.8, 20110.99]80 [17859.19, 20639.4]95
```

```
#> # A tibble: 1 × 4
#>           total   mean             pi80             pi95
#>          <dist>  <dbl>           <hilo>           <hilo>
#> 1 sample[10000] 19212. [18330, 20118]80 [17851, 20628]95
```

As expected, the mean of the simulated data is close to the sum of the individual forecasts.

```
forecast(fit, h = 12) |>
  as_tibble() |>
  summarise(total = sum(.mean))
```

```
## # A tibble: 1 x 1
##    total
##    <dbl>
## 1 19212.
```

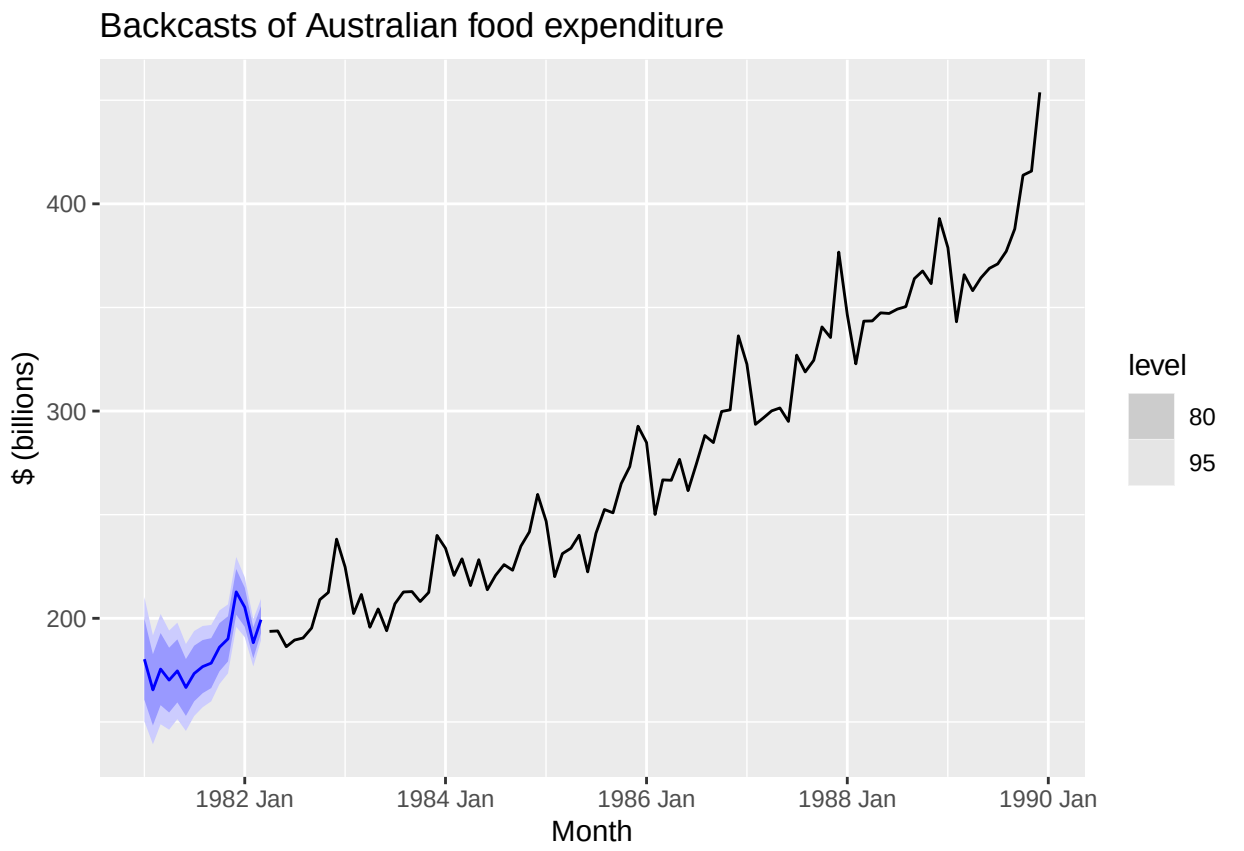
```
#> # A tibble: 1 × 1
#>    total
#>    <dbl>
#> 1 19212.
```

13.6 Backcasting

Sometimes it is useful to "backcast" a time series — that is, forecast in reverse time. Although there are no in-built R functions to do this, it is easy to implement by creating a new time index.

Suppose we want to extend our Australian takeaway to the start of 1981 (the actual data starts in April 1982)

```
backcasts <- auscafe |>
  mutate(reverse_time = rev(row_number())) |>
  update_tsibble(index = reverse_time) |>
  model(ets = ETS(Turnover ~ season(period = 12))) |>
  forecast(h = 15) |>
  mutate(Month = auscafe$Month[1] - (1:15)) |>
  as_fable(index = Month, response = "Turnover",
    distribution = "Turnover")
backcasts |>
  autoplot(auscafe |> filter(year(Month) < 1990)) +
  labs(title = "Backcasts of Australian food expenditure",
       y = "$ (billions)")
```



Most of the work here is in re-indexing the tsibble object and then re-indexing the fable object.

13.7 Very long and very short time series

Forecasting very short time series

We often get asked how few data points can be used to fit a time series model. As with almost all sample size questions, there is no easy answer. It depends on the number of model parameters to be estimated and the amount of randomness in the data. The sample size required increases with the number of parameters to be estimated, and the amount of noise in the data.

Some textbooks provide rules-of-thumb giving minimum sample sizes for various time series models. These are misleading and unsubstantiated in theory or practice. Further, they ignore the underlying variability of the data and often overlook the number of parameters to be estimated as well. There is, for example, no justification for the magic number of 30 often given as a minimum for ARIMA modelling. The only theoretical limit is that we need more observations than there are parameters in our forecasting model. However, in practice, we usually need substantially more observations than that.

Ideally, we would test if our chosen model performs well out-of-sample compared to some simpler approaches. However, with short series, there is not enough data to allow some observations to be withheld for testing purposes, and even time series cross validation can be difficult to apply. The AICc is particularly useful here, because it is a proxy for the one-step forecast out-of-sample MSE. Choosing the model with the minimum AICc value allows both the number of parameters and the amount of noise to be taken into account.

What tends to happen with short series is that the AICc suggests simple models because anything with more than one or two parameters will produce poor forecasts due to the estimation error. We will fit an ARIMA model to the annual series from the M3-competition with fewer than 20 observations. First we need to create a tsibble, containing the relevant series.

```
m3totsibble <- function(z) {
  bind_rows(
    as_tsibble(z$x) |> mutate(Type = "Training"),
    as_tsibble(z$xx) |> mutate(Type = "Test")
  ) |>
    mutate(
      st = z$st,
      type = z$type,
      period = z$period,
      description = z$description,
      sn = z$sn
    ) |>
    as_tibble()
}
short <- Mcomp::M3 |>
  subset("yearly") |>
  purrr::map_dfr(m3totsibble) |>
  group_by(sn) |>
  mutate(n = max(row_number())) |>
  filter(n <= 20) |>
  ungroup() |>
  as_tsibble(index = index, key = c(sn, period, st))
```

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

Now we can apply an ARIMA model to each series.

```
short_fit <- short |>
  model(arima = ARIMA(value))
```

Of the 152 series, 21 had models with zero parameters (white noise and random walks), 86 had models with one parameter, 31 had models with two parameters, 13 had models with three parameters, and only 1 series had a model with four parameters.

Forecasting very long time series

Most time series models do not work well for very long time series. The problem is that real data do not come from the models we use. When the number of observations is not large (say up to about 200) the models often work well as an approximation to whatever process generated the data. But eventually we will have enough data that the difference between the true process and the model starts to become more obvious. An additional problem is that the optimisation of the parameters becomes more time consuming because of the number of observations involved.

What to do about these issues depends on the purpose of the model. A more flexible and complicated model could be used, but this still assumes that the model structure will work over the whole period of the data. A better approach is usually to allow the model itself to change over time. ETS models are designed to handle this situation by allowing the trend and seasonal terms to evolve over time. ARIMA models with differencing have a similar property. But dynamic regression models do not allow any evolution of model components.

If we are only interested in forecasting the next few observations, one simple approach is to throw away the earliest observations and only fit a model to the most recent observations. Then an inflexible model can work well because there is not enough time for the relationships to change substantially.

For example, we fitted a dynamic harmonic regression model to 26 years of weekly gasoline production in

Section 13.1. It is, perhaps, unrealistic to assume that the seasonal pattern remains the same over nearly three decades. So we could simply fit a model to the most recent years instead.

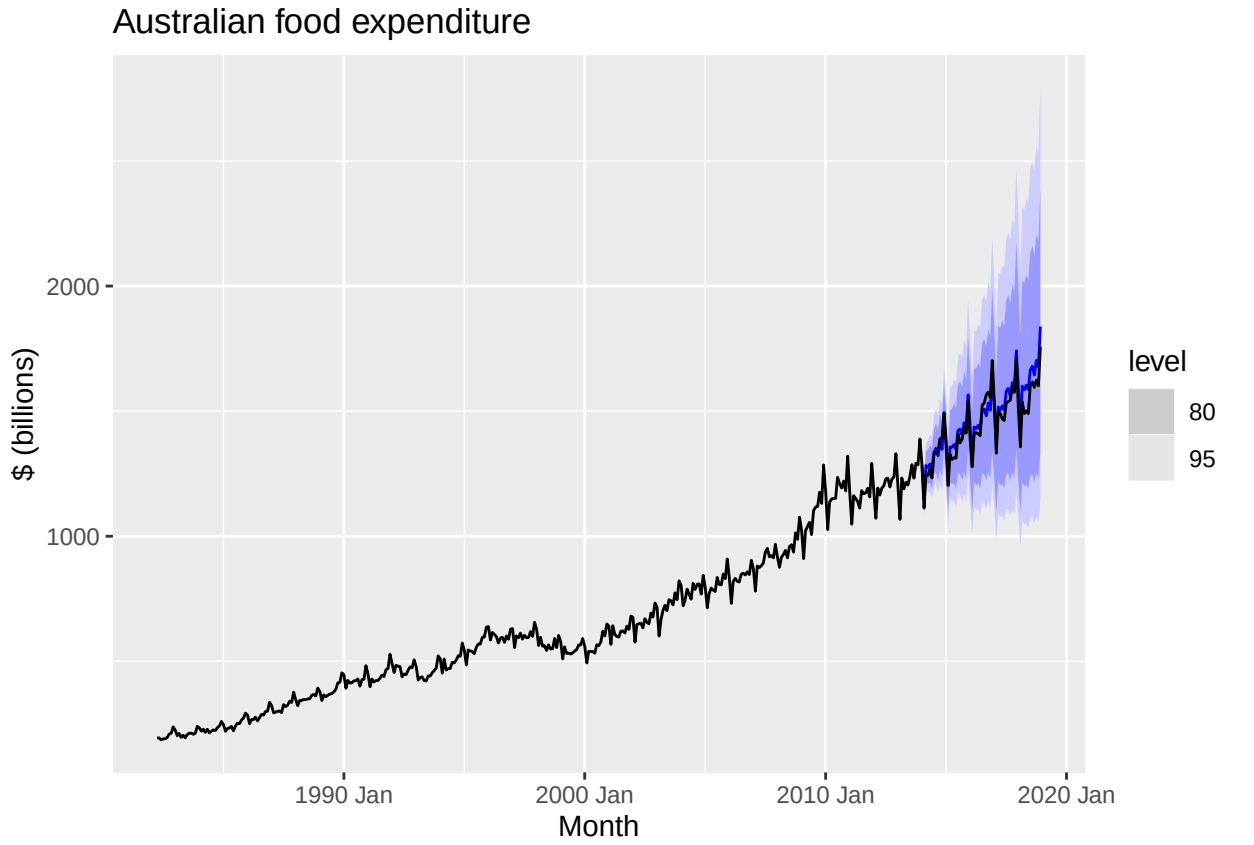
13.8 Forecasting on training and test sets

Typically, we compute one-step forecasts on the training data (the "fitted values") and multi-step forecasts on the test data. However, occasionally we may wish to compute multi-step forecasts on the training data, or one-step forecasts on the test data.

Multi-step forecasts on training data

We normally define fitted values to be one-step forecasts on the training set (see Section 5.3), but a similar idea can be used for multi-step forecasts. We will illustrate the method using an ARIMA model for the Australian take-away food expenditure. The last five years are used for a test set, and the forecasts are plotted in Figure 13.9.

```
training <- auscafe |> filter(year(Month) <= 2013)
test <- auscafe |> filter(year(Month) > 2013)
cafe_fit <- training |>
  model(ARIMA(log(Turnover)))
cafe_fit |>
  forecast(h = 60) |>
  autoplot(auscafe) +
  labs(title = "Australian food expenditure",
       y = "$ (billions)")
```

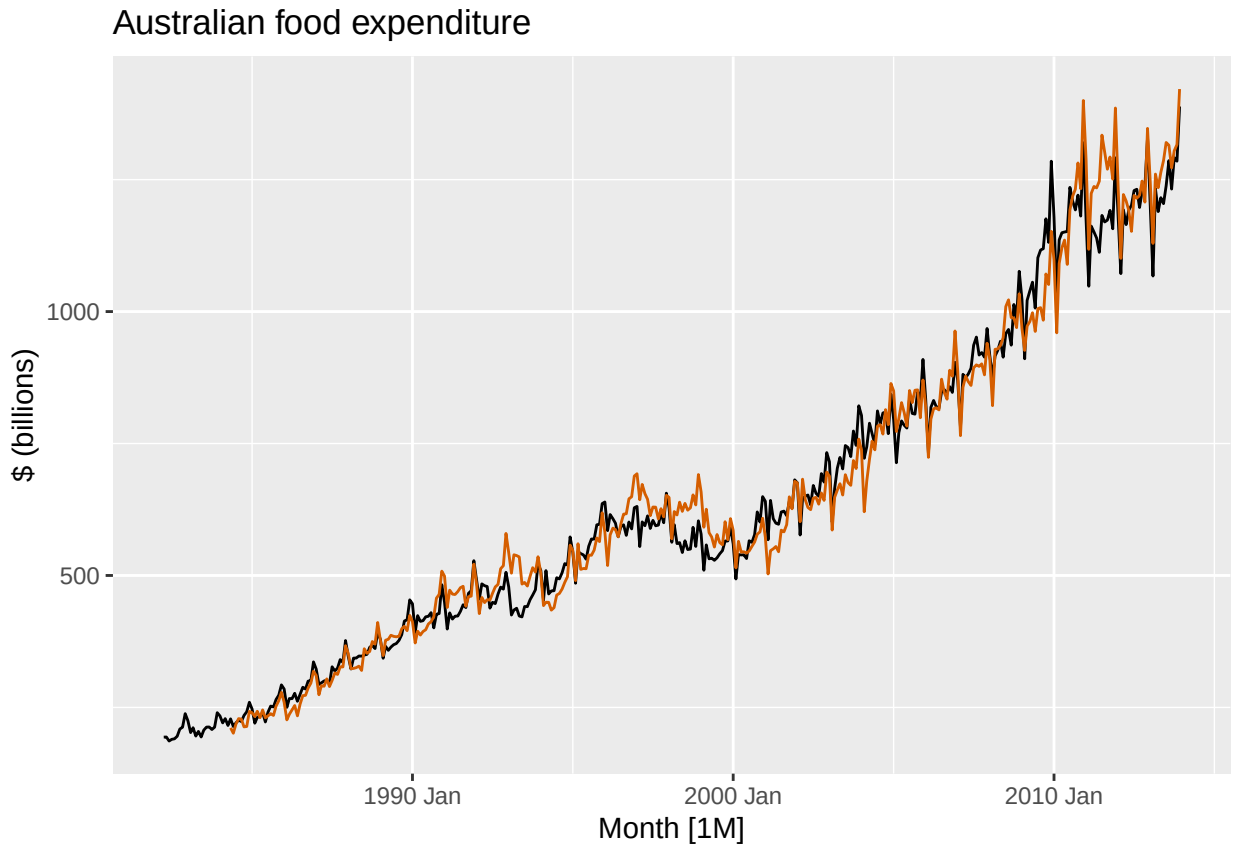


The fitted() function has an h argument to allow for h-step "fitted values" on the training set. Figure 13.10

is a plot of 12-step (one year) forecasts on the training set. Because the model involves both seasonal (lag 12) and first (lag 1) differencing, it is not possible to compute these forecasts for the first few observations.

```
fits12 <- fitted(cafe_fit, h = 12)
training |>
  autoplot(Turnover) +
  autolayer(fits12, .fitted, col = "#D55E00") +
  labs(title = "Australian food expenditure",
       y = "$ (billions)")
```

```
## Warning: Removed 25 rows containing missing values (`geom_line()`).
```



One-step forecasts on test data

It is common practice to fit a model using training data, and then to evaluate its performance on a test data set. The way this is usually done means the comparisons on the test data use different forecast horizons. In the above example, we have used the last sixty observations for the test data, and estimated our forecasting model on the training data. Then the forecast errors will be for 1-step, 2-steps, ..., 60-steps ahead. The forecast variance usually increases with the forecast horizon, so if we are simply averaging the absolute or squared errors from the test set, we are combining results with different variances.

One solution to this issue is to obtain 1-step errors on the test data. That is, we still use the training data to estimate any parameters, but when we compute forecasts on the test data, we use all of the data preceding each observation (both training and test data). So our training data are for times $1, 2, ..., T-60$. We estimate the model on these data, but then compute $\hat{y}_{T-60+h|T-61+h}$, for h=1,...,T−1. Because the test data are not used to estimate the parameters, this still gives us a "fair" forecast.

Using the same ARIMA model used above, we now apply the model to the test data.

```
cafe_fit |>
  refit(test) |>
  accuracy()
```

```
## # A tibble: 1 x 10
##   .model               .type     ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
##   <chr>                <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 ARIMA(log(Turnover)) Train~ -2.49  20.5  15.4 -0.169  1.06 0.236 0.259 -0.0502
```

```
#> # A tibble: 1 × 10
#>   .model               .type    ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
#>   <chr>                <chr> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 ARIMA(log(Turnover… Trai… -2.49  20.5  15.4 -0.169  1.06 0.236 0.259 -0.0502
```

Note that model is not re-estimated in this case. Instead, the model obtained previously (and stored as cafe_fit) is applied to the test data. Because the model was not re-estimated, the "residuals" obtained here are actually one-step forecast errors. Consequently, the results produced from the accuracy() command are actually on the test set (despite the output saying "Training set"). This approach can be used to compare one-step forecasts from different models.

13.9 Dealing with outliers and missing values

Real data often contains missing values, outlying observations, and other messy features. Dealing with them can sometimes be troublesome.

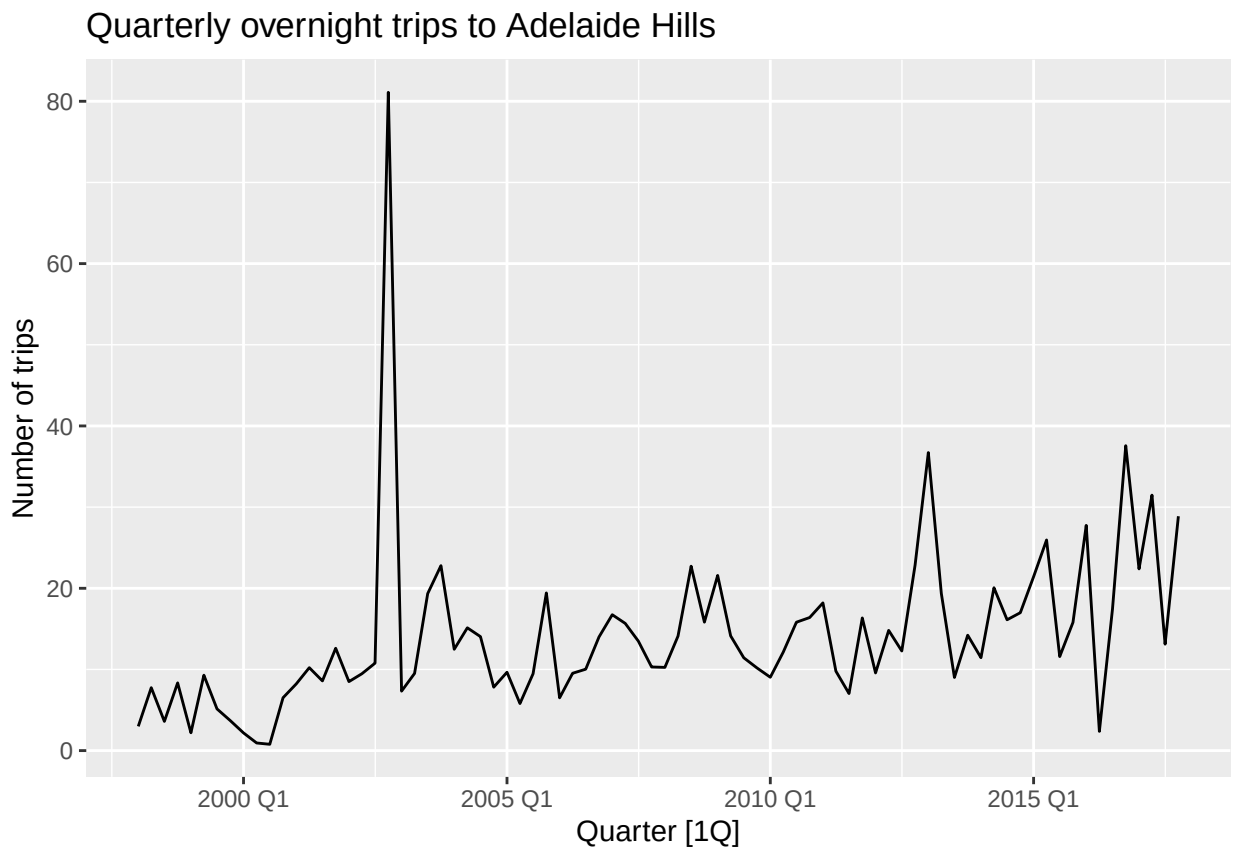
Outliers

Outliers are observations that are very different from the majority of the observations in the time series. They may be errors, or they may simply be unusual. (See Section 7.3 for a discussion of outliers in a regression context.) None of the methods we have considered in this book will work well if there are extreme outliers in the data. In this case, we may wish to replace them with missing values, or with an estimate that is more consistent with the majority of the data.

Simply replacing outliers without thinking about why they have occurred is a dangerous practice. They may provide useful information about the process that produced the data, which should be taken into account when forecasting. However, if we are willing to assume that the outliers are genuinely errors, or that they won't occur in the forecasting period, then replacing them can make the forecasting task easier.

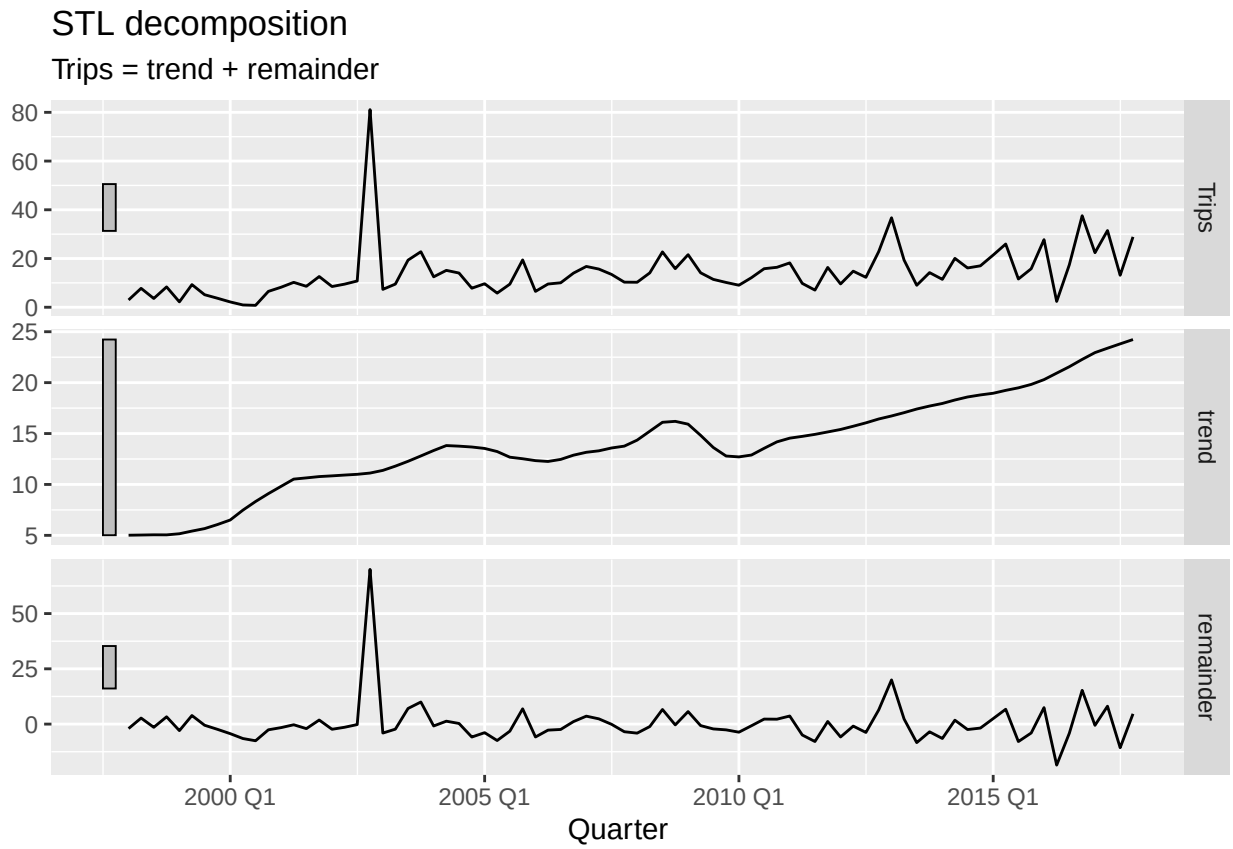
Figure 13.11 shows the number of visitors to the Adelaide Hills region of South Australia. There appears to be an unusual observation in 2002 Q4.

```
tourism |>
  filter(
    Region == "Adelaide Hills", Purpose == "Visiting"
  ) |>
  autoplot(Trips) +
  labs(title = "Quarterly overnight trips to Adelaide Hills",
       y = "Number of trips")
```



One useful way to find outliers is to apply STL() to the series with the argument robust=TRUE. Then any outliers should show up in the remainder series. The data in Figure 13.11 have almost no visible seasonality, so we will apply STL without a seasonal component by setting period=1

```
ah_decomp <- tourism |>
  filter(
    Region == "Adelaide Hills", Purpose == "Visiting"
  ) |>
  # Fit a non-seasonal STL decomposition
  model(
    stl = STL(Trips ~ season(period = 1), robust = TRUE)
  ) |>
  components()
ah_decomp |> autoplot()
```



In the above example the outlier was easy to identify. In more challenging cases, using a boxplot of the remainder series would be useful. We can identify as outliers those that are greater than 1.5 interquartile ranges (IQRs) from the central 50% of the data. If the remainder was normally distributed, this would show 7 in every 1000 observations as "outliers". A stricter rule is to define outliers as those that are greater than 3 interquartile ranges (IQRs) from the central 50% of the data, which would make only 1 in 500,000 normally distributed observations to be outliers. This is the rule we prefer to use.

```
outliers <- ah_decomp |>
  filter(
    remainder < quantile(remainder, 0.25) - 3*IQR(remainder) |
    remainder > quantile(remainder, 0.75) + 3*IQR(remainder)
  )
outliers
```

```
## # A dable: 1 x 9 [1Q]
## # Key:     Region, State, Purpose, .model [1]
## # :        Trips = trend + remainder
##   Region        State Purpose .model Quarter Trips trend remainder season_adjust
##   <chr>         <chr> <chr>   <chr>    <qtr> <dbl> <dbl>     <dbl>         <dbl>
## 1 Adelaide Hil~ Sout~ Visiti~ stl    2002 Q4  81.1  11.1      70.0          81.1
```

```
#> # A dable: 1 x 9 [1Q]
#> # Key:     Region, State, Purpose, .model [1]
#> # :        Trips = trend + remainder
#>   Region      State Purpose .model Quarter Trips trend remainder season_adjust
#>   <chr>       <chr> <chr>   <chr>    <qtr> <dbl> <dbl>     <dbl>         <dbl>
#> 1 Adelaide H… Sout… Visiti… stl    2002 Q4  81.1  11.1      70.0          81.1
```

This finds the one outlier that we suspected from Figure 13.11. Something similar could be applied to the full data set to identify unusual observations in other series.

Missing values

Missing data can arise for many reasons, and it is worth considering whether the missingness will induce bias in the forecasting model. For example, suppose we are studying sales data for a store, and missing values occur on public holidays when the store is closed. The following day may have increased sales as a result. If we fail to allow for this in our forecasting model, we will most likely under-estimate sales on the first day after the public holiday, but over-estimate sales on the days after that. One way to deal with this kind of situation is to use a dynamic regression model, with dummy variables indicating if the day is a public holiday or the day after a public holiday. No automated method can handle such effects as they depend on the specific forecasting context.

In other situations, the missingness may be essentially random. For example, someone may have forgotten to record the sales figures, or the data recording device may have malfunctioned. If the timing of the missing data is not informative for the forecasting problem, then the missing values can be handled more easily.

Finally, we might remove some unusual observations, thus creating missing values in the series.

Some methods allow for missing values without any problems. For example, the naïve forecasting method continues to work, with the most recent non-missing value providing the forecast for the future time periods. Similarly, the other benchmark methods introduced in Section 5.2 will all produce forecasts when there are missing values present in the historical data. The fable functions for ARIMA models, dynamic regression models and NNAR models will also work correctly without causing errors. However, other modelling functions do not handle missing values including ETS() and STL().

When missing values cause errors, there are at least two ways to handle the problem. First, we could just take the section of data after the last missing value, assuming there is a long enough series of observations to produce meaningful forecasts. Alternatively, we could replace the missing values with estimates. To do this, we first fit an ARIMA model to the data containing missing values, and then use the model to interpolate the missing observations.

We will replace the outlier identified in Figure 13.12 by an estimate using an ARIMA model.

```
ah_miss <- tourism |>
  filter(
    Region == "Adelaide Hills",
    Purpose == "Visiting"
  ) |>
  # Remove outlying observations
  anti_join(outliers) |>
  # Replace with missing values
  fill_gaps()
```

```
## Joining with `by = join_by(Quarter, Region, State, Purpose, Trips)`
```

```
ah_fill <- ah_miss |>
  # Fit ARIMA model to the data containing missing values
  model(ARIMA(Trips)) |>
  # Estimate Trips for all periods
  interpolate(ah_miss)
ah_fill |>
  # Only show outlying periods
  right_join(outliers |> select(-Trips))
```

```
## Joining with `by = join_by(Region, State, Purpose, Quarter)`
```

```
## # A tsibble: 1 x 9 [?]
```

```
## # Key:       Region, State, Purpose [1]
##   Region        State Purpose Quarter Trips .model trend remainder season_adjust
##   <chr>         <chr> <chr>     <qtr> <dbl> <chr>  <dbl>     <dbl>         <dbl>
## 1 Adelaide Hil~ Sout~ Visiti~ 2002 Q4  8.50 stl     11.1      70.0          81.1
```

```
#> # A tsibble: 1 x 9 [?]
#> # Key:       Region, State, Purpose [1]
#>   Region      State Purpose Quarter Trips .model trend remainder season_adjust
#>   <chr>       <chr> <chr>     <qtr> <dbl> <chr>  <dbl>     <dbl>         <dbl>
#> 1 Adelaide H… Sout… Visiti… 2002 Q4  8.50 stl     11.1      70.0          81.1
#>
```

The interpolate() function uses the ARIMA model to estimate any missing values in the series. In this case, the outlier of 81.1 has been replaced with 8.5. The resulting series is shown in Figure 13.13.

The ah_fill data could now be modeled with a function that does not allow missing values.

```
ah_fill |>
  autoplot(Trips) +
  autolayer(ah_fill |> filter_index("2002 Q3"~"2003 Q1"),
    Trips, colour="#D55E00") +
  labs(title = "Quarterly overnight trips to Adelaide Hills",
       y = "Number of trips")
```