Tidy time series forecasting with fable

2019-05-23

Contents

Chapter 1

Forecasting with fable

The `fable` package is a tidy renovation of the forecast package, and it explores new interfaces for modelling and subsequent analysis in R. For users experienced with the tidyverse, modelling in R can be a jarring experience. Models in R can be difficult to work with as there is little standardisation in model object structures and interfaces. This is partially alleviated from tidy modelling packages such as broom (and sweep for time series) which can be used to extract key features from model objects in suitable formats for tidy workflows.

The existing integration of modelling and the tidyverse is especially problematic for time series, as the input data structure (ts) is inherently untidy. Much like tsibble implements tidy time series data, the `fable` package applies tidyverse principles to time series modelling, making the forecasting workflow seamlessly integrate with other `tidyverse` packages.

The design choices around `fable` prioritise the user interface, allowing complicated models to be estimated using common statistical terminology. It intentionally abstracts away from the complications of programming to allow users to focus on the model(s) that they are estimating rather than how they are coding it.

1.1 Background

The ideas presented in this resource are subject to revision, as the development of these tools are still experimental. This text aims to consolidate the information currently dispersed in the `fable` package documentation, wiki pages, issues, and emails in a way that is more accessible to the wider community. Hopefully this firstly helps us better design and plan the workflow of tidy forecasting, and invite external opinions to ensure that `fable` works well with the existing tidy modelling developments.

Chapter 2

Tidy time series

2.1 ts

Time series data structures in R vary substantially, however most time series models make use of the `ts` object structure from the `stats` package. This object concisely stores the time series index using three 'time series parameters' (`tsp`): `start`, `frequency`, and `end`. For most time series tools (such as `arima`, `ets`, `stl`) this structural information is sufficient, however it lacks details that are present in modern time series datasets:

- Multiple seasonality
- Irregular observations
- Exogenous information
- Many time series (that differ in length)

In many senses this structure is limited, and inconsistent with the tidy data principles:

- Non-rectangular index structure
- Wide-format for keyed data (`mts`)
- Unnatural index format for importing data
- Difficulty working with tidyverse tools

2.2 tsibble

The tsibble package by Earo Wang provides a tidy data structure for time series, and is well described in her introductory vignette.

This data structure is sufficiently flexible to support the future of time series modelling tools (such as `tbats`, `fasster` and `prophet`). Beyond the data tidying and transformation tools that the package provides, the object also includes valuable structural information (`index` and `key`) for time series modelling.

2.2.1 index

The index is essential for modelling as it can be used to identify the frequency and regularity of the observations. By storing a standard `datetime` object within the dataset, it makes irregular time series modelling possible. It also allows a more flexible specification of seasonal frequency (see seasonal period) that is easier to specify for the end user (a very common difficulty when constructing `ts` objects).

2.2.2 key

Keys are used within tsibble to uniquely identify related time series in a tidy structure. They are also useful for identifying relational structures between each time series. This is especially useful for forecast reconciliation, where a hierarchical or grouped structure is imposed on a set of forecasts to impose relational constraints (typically aggregation). Keys within tsibble can be either nested (hierarchical) or crossed (grouped), and can be directly used to reconcile forecasts. This structure also has purpose for univariate models, as it allows batch forecasting to be applied across many time series.

Chapter 3

Model basics

The fablelite package provides a set of tools for creating models that work well together. These tools aim to simplify model development and encourage a consistent interface across many model types. By developing a model with fablelite, complexity introduced by batch forecasting and advanced functionality is handled automatically. This should allow model developers to focus on implementing model specific functionality.

The model function is expected to accept a tsibble and model formula, and return a fitted model stored as a mable.

3.1 Model specification

A consistent interface across models is essential.

3.2 Model formula

fable introduces the formula based model specification that is familiar in cross-sectional models to time series. This allows an interface for concise and human readable model specifications. Additionally, this change allows a more flexible specification of transformations, and model elements. A typical model formula may look like this:

```r
log(y) ~ trend() + season(period = "day") + x
```

Like `lm()` and other cross-sectional model interfaces, the left hand side of the formula defines the response. fable extends this standard usage by supporting transformations of any type. In the above example, a `log` transformation has been used on the response variable, `y`. Unlike other models, the transformation specified in formula's LHS will be automatically inverted, and used for automatic back-transformation of your forecasts and fitted values. For more details on how transformations work within fable, you can refer to the transformations vignette: `vignette("transformations", package="fable")`

On the right hand side of the formula, we define the specials used for the model. Specifying the RHS is optional, and any required specials that are omitted will be included using their defaults (allowing for automatic model selection). In the above formula, we have included a trend, a daily seasonal pattern and exogenous regressor `x`. The specials that are supported will vary between each model function, and the available specials are (should be) documented under the "Model Formula" section of the help file.

3.3 Selection parameters

3.4 Optimisation parameters

3.5 Computational parameters

Chapter 4

Accessing model elements

4.1 Display

The `print` and `summary` methods are standard displays for fitted models. The `print` method typically displays a limited amount of key information, such as the model that was fit, and the estimated coefficients. The `summary` function extends the `print` method with a more detailed summary of fit, which may include measures for goodness of fit, and significance of model terms.

As fable naturally supports batch/multiple forecasting, the print method is standardised for any number of models. A very short model specific display can be defined using the `model_sum` generic, which is shown in the mable.

```r
library(tsibbledata)
UKLungDeaths <- as_tsibble(cbind(mdeaths, fdeaths), gather = FALSE)
```

```
## Warning: Argument `gather` is deprecated, please use `pivot_longer`
## instead.
```

```r
ets_fit <- UKLungDeaths %>%
  model(ETS(mdeaths))
```

```
## # A mable: 1 x 1
##   `ETS(mdeaths)`
##   <model>
## 1 <ETS(M,A,A)>
```

The summary method can then be used to reveal more information about this model, such as fitted parameters and goodness of fit. Ideally this information would also be standardised into a tabular form for batch modelling, although this is currently not the case.

```r
ets_fit %>%
  summary
```

```
##        Length Class  Mode
## par     2     tbl_df list
## est     4     tbl_ts list
## fit     8     tbl_df list
## states 15     tbl_ts list
## spec    5     tbl_df list
```

4.2 Fitted values and residuals

Accessors for fitted values and residuals return a tsibble containing the index from the original data, with a measured variables for `.fitted` values and `.resids`. If the mable contains more than one model, the resulting object maintains and respects the key structure.

```
ets_fit %>%
  fitted
```

```
## # A tsibble: 72 x 3 [1M]
## # Key:       .model [1]
##    .model          index .fitted
##    <chr>           <mth>   <dbl>
##  1 ETS(mdeaths) 1974 Jan   2278.
##  2 ETS(mdeaths) 1974 Feb   2283.
##  3 ETS(mdeaths) 1974 Mar   2142.
##  4 ETS(mdeaths) 1974 Apr   1776.
##  5 ETS(mdeaths) 1974 May   1442.
##  6 ETS(mdeaths) 1974 Jun   1341.
##  7 ETS(mdeaths) 1974 Jul   1270.
##  8 ETS(mdeaths) 1974 Aug   1160.
##  9 ETS(mdeaths) 1974 Sep   1147.
## 10 ETS(mdeaths) 1974 Oct   1381.
## # ... with 62 more rows
```

```
ets_fit %>%
  residuals
```

```
## # A tsibble: 72 x 3 [1M]
## # Key:       .model [1]
##    .model          index   .resid
##    <chr>           <mth>    <dbl>
##  1 ETS(mdeaths) 1974 Jan -0.0633
##  2 ETS(mdeaths) 1974 Feb -0.184
##  3 ETS(mdeaths) 1974 Mar -0.124
##  4 ETS(mdeaths) 1974 Apr  0.0569
##  5 ETS(mdeaths) 1974 May  0.0346
##  6 ETS(mdeaths) 1974 Jun -0.0689
##  7 ETS(mdeaths) 1974 Jul  0.00764
##  8 ETS(mdeaths) 1974 Aug -0.0248
##  9 ETS(mdeaths) 1974 Sep  0.0544
## 10 ETS(mdeaths) 1974 Oct  0.0805
## # ... with 62 more rows
```

4.3 Broom functionality

Common features from a model can also be accessed using verbs from the broom package. Again, key structures that exist within the mable are respected.

```
ets_fit %>%
  augment
```

```
## # A tsibble: 72 x 5 [1M]
## # Key:       .model [1]
##    .model          index mdeaths .fitted   .resid
##    <chr>           <mth>   <dbl>   <dbl>    <dbl>
##  1 ETS(mdeaths) 1974 Jan    2134   2278. -0.0633
##  2 ETS(mdeaths) 1974 Feb    1863   2283. -0.184
##  3 ETS(mdeaths) 1974 Mar    1877   2142. -0.124
##  4 ETS(mdeaths) 1974 Apr    1877   1776.  0.0569
##  5 ETS(mdeaths) 1974 May    1492   1442.  0.0346
##  6 ETS(mdeaths) 1974 Jun    1249   1341. -0.0689
##  7 ETS(mdeaths) 1974 Jul    1280   1270.  0.00764
##  8 ETS(mdeaths) 1974 Aug    1131   1160. -0.0248
##  9 ETS(mdeaths) 1974 Sep    1209   1147.  0.0544
## 10 ETS(mdeaths) 1974 Oct    1492   1381.  0.0805
## # ... with 62 more rows
```

```
ets_fit %>%
  tidy
```

```
ets_fit %>%
  glance
```

```
## # A tibble: 1 x 9
##   .model        sigma2 logLik   AIC  AICc   BIC    MSE   AMSE    MAE
##   <chr>          <dbl>  <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>
## 1 ETS(mdeaths) 0.00905  -500. 1033. 1045. 1072. 24137. 23441. 0.0657
```

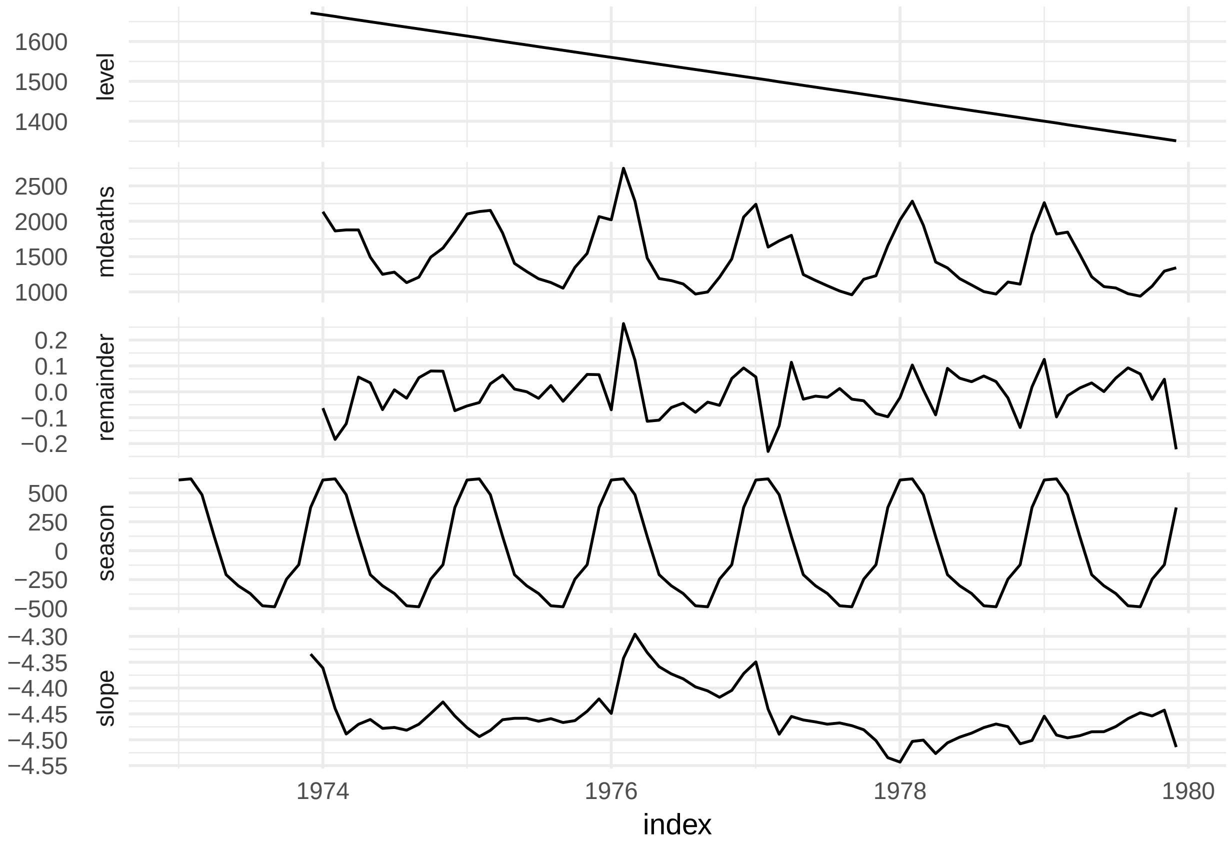
4.4 Components

In many cases, a model can be used to extract features or components from data in a similar way to decomposition methods. We use the `components` verb to extract a tsibble of data features that have been extracted via modelling or decomposition.

State space models such as `ETS` are well suited to this functionality as the states often represent features of interest.

```
UKLungDeaths %>%
  model(ETS(mdeaths)) %>%
  components
```

```
## # A dable:                  84 x 7 [1M]
## # Key:                      .model [1]
## # ETS(M,A,A) Decomposition: mdeaths = (lag(level, 1) + lag(slope, 1) +
## #   lag(season, 12)) * (1 + remainder)
##    .model          index mdeaths level slope season remainder
##    <chr>           <mth>   <dbl> <dbl> <dbl>  <dbl>     <dbl>
##  1 ETS(mdeaths) 1973 Jan      NA    NA    NA   611.        NA
##  2 ETS(mdeaths) 1973 Feb      NA    NA    NA   620.        NA
##  3 ETS(mdeaths) 1973 Mar      NA    NA    NA   483.        NA
##  4 ETS(mdeaths) 1973 Apr      NA    NA    NA   122.        NA
##  5 ETS(mdeaths) 1973 May      NA    NA    NA  -207.        NA
##  6 ETS(mdeaths) 1973 Jun      NA    NA    NA  -304.        NA
##  7 ETS(mdeaths) 1973 Jul      NA    NA    NA  -370.        NA
##  8 ETS(mdeaths) 1973 Aug      NA    NA    NA  -476.        NA
##  9 ETS(mdeaths) 1973 Sep      NA    NA    NA  -485.        NA
## 10 ETS(mdeaths) 1973 Oct      NA    NA    NA  -246.        NA
## # ... with 74 more rows
## Warning: Removed 11 rows containing missing values (geom_path).
```



It may also be worth storing how these components can be used to produce the response, which can be used for decomposition modelling.

Chapter 5

Model methods

5.1 Interpolation

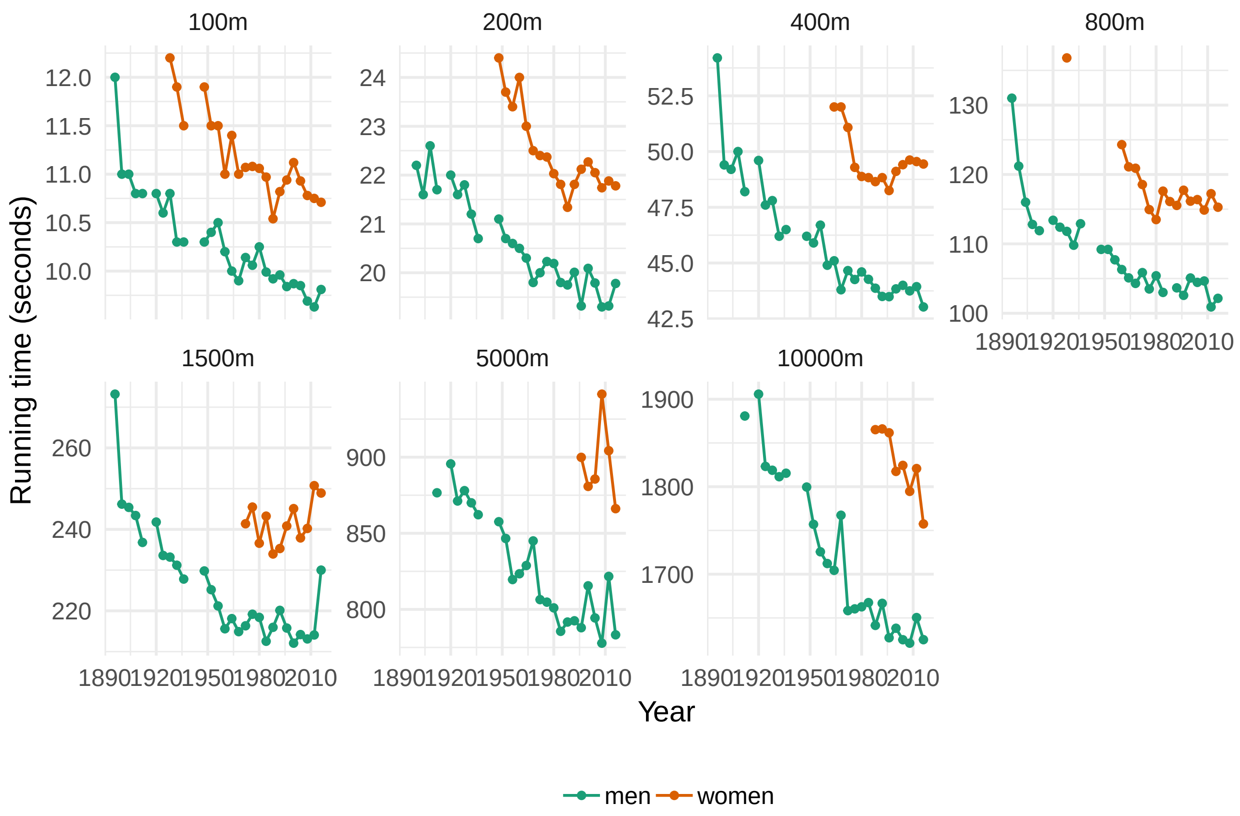
Models that can be estimated in the presence of missing values can often be used to interpolate the unknown values. Often interpolated values can be taken from model's fitted values, and some models may support more sophisticated interpolation methods.

The forecast package provides the `na.interp` function for interpolating time series data, which uses linear interpolation for non-seasonal data, and STL decomposition for seasonal data.

Tidy time series tools should allow users to interpolate missing values using any appropriate model.

For example, the `tsibbledata::olympic_running` dataset contains Olympic men's 400m track final winning times. The winning times for the 1916, 1940 and 1944 Olympics are missing from the dataset due to the World Wars.

```
## Warning: Removed 31 rows containing missing values (geom_point).
```

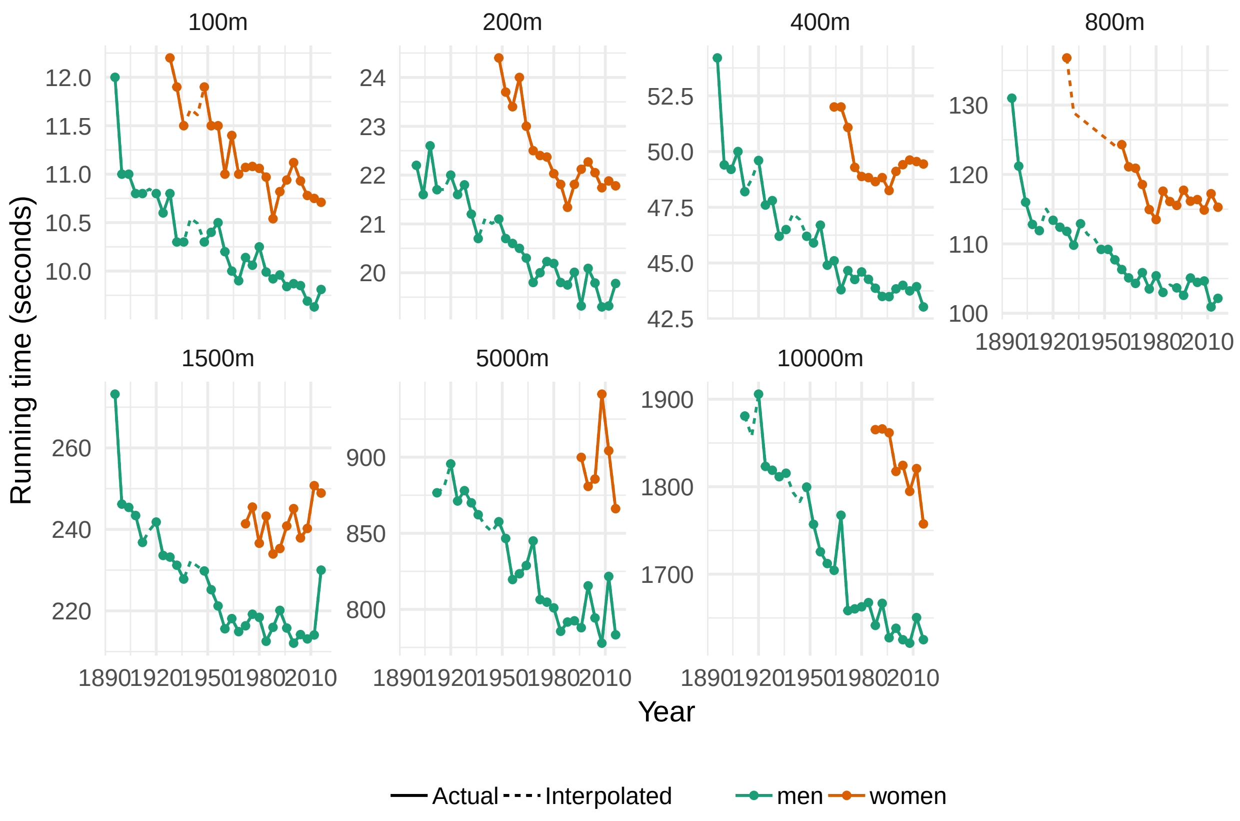


We could then interpolate these missing values using the fitted values from a linear model with a trend:

```r
olympic_running %>%
  model(lm = TSLM(Time ~ trend())) %>%
  interpolate(olympic_running)
```

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

```
## Warning: Removed 31 rows containing missing values (geom_point).
```



5.2 Re-estimation

https://github.com/tidyverts/fable/issues/43

5.2.1 refit()

The refitting a model allows the same model to be applied to a new dataset. This is similar to the `model` argument available in most modelling functions from the forecast package.

The refitted model should maintain the same structure and coefficients of the original model, with fitted information updated to reflect the model's behaviour on the new dataset. It should also be possible to allow re-estimation of parameters using the `reestimate` argument, which keeps the selected model terms but updates the model coefficients/parameters.

It is expected that a refit method uses a fitted model and replacement data to return a mable.

For the ETS model for `mdeaths` estimated above:

```
library(fable)
ets_fit <- as_tsibble(mdeaths) %>%
  model(ETS(value))
```

We may be interested in using the same model with the same coefficients to estimate the `fdeaths` series:

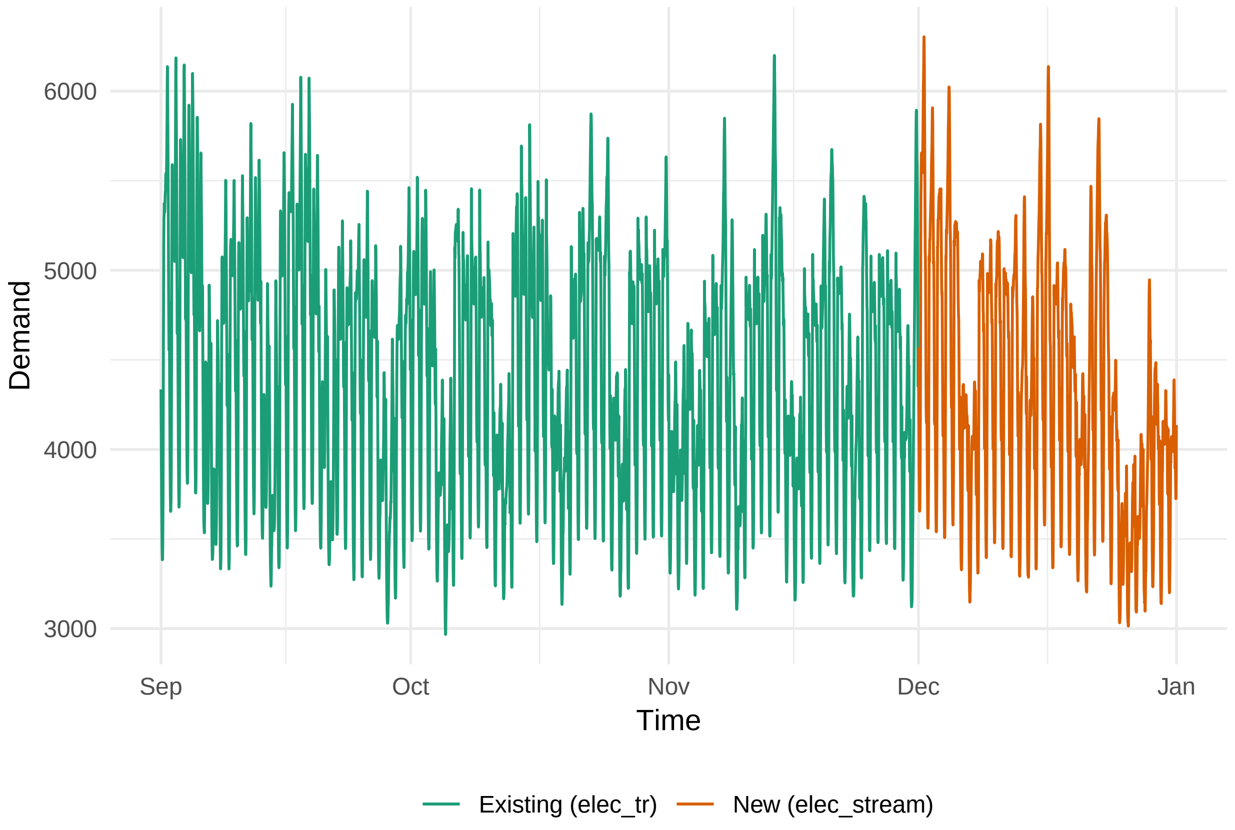
```
refit(ets_fit, as_tsibble(fdeaths))
```

```
## # A mable: 1 x 1
##   `ETS(value)`
##    <model>
## 1 <ETS(M,A,A)>
```

5.2.2 stream()

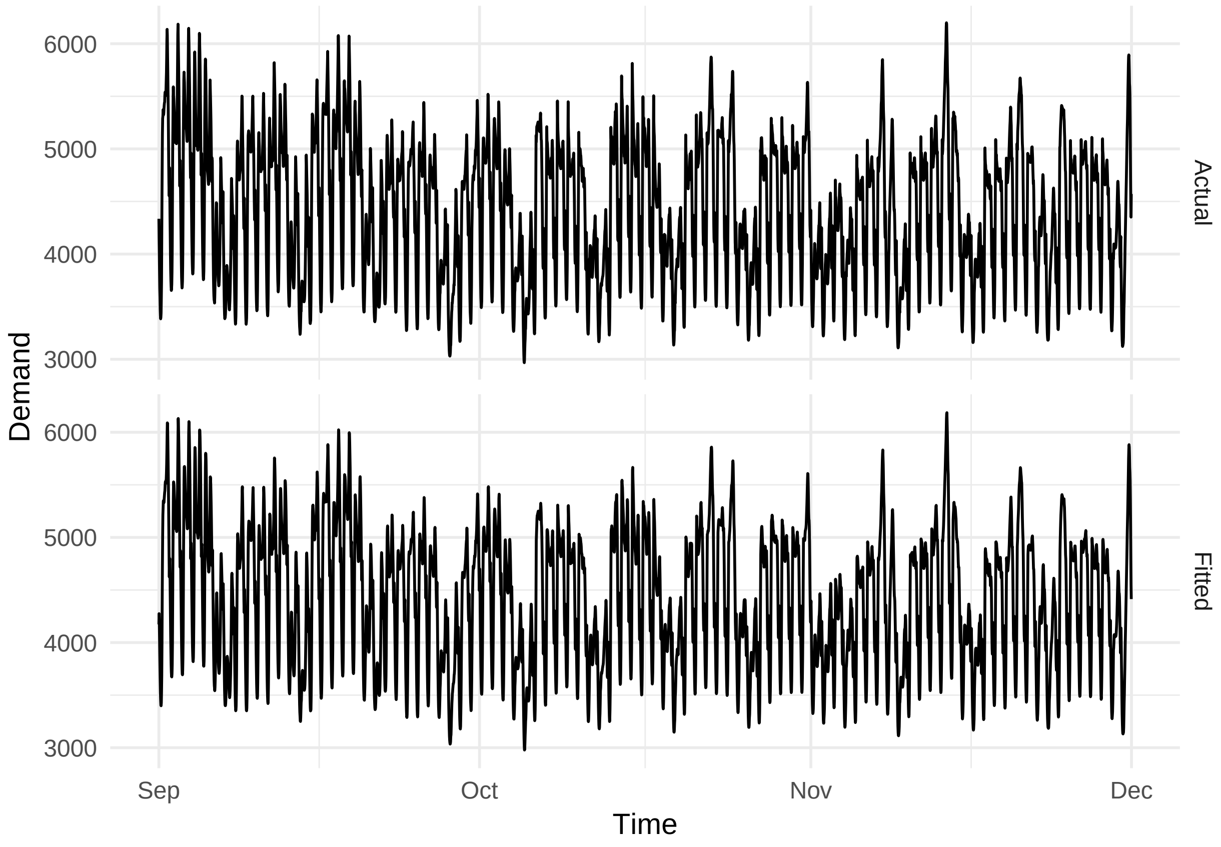
Streaming data into a model allows a model to be extended to accomodate new, future data. Like `refit`, `stream` should allow re-estimation of the model parameters. As this can be a costly operation for some models, in most cases updating the parameters should not occur. However it is recommended that the model parameters are updated on a regular basis.

Suppose we are estimating electricity demand data (`tsibbledata::aus_elec`), and after fitting a model to the existing data, a new set of data from the next month becomes available.



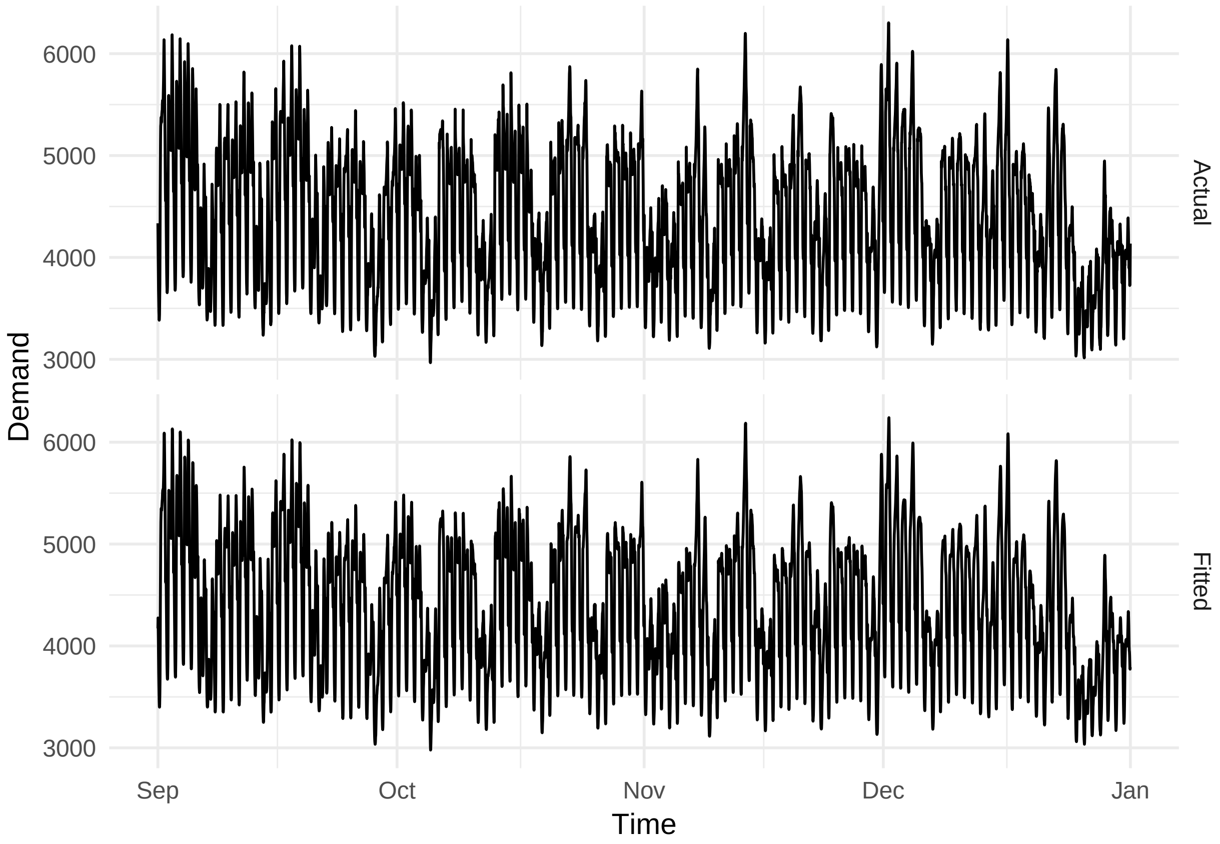
A (minimal) model for the electricity demand above can be estimated using fasster.

```
fit <- elec_tr %>%
  model(fasster = fasster(Demand ~ Holiday %S% (poly(1) + trig(10))))
```



To extend these fitted values to include December's electricity data, we can use the `stream` functionality:

```
fit <- fit %>%
  stream(elec_stream)
```



5.3 Simulation

Much like the tidymodels opinion toward `predict`, `generate` should not default to an archived version of the training set. This allows models to be used for simulating new data sets, which is especially relevant for time series as often future paths beyond the training set are simulated.

The generate method for a fable model should accept these arguments (names chosen for consistency with `tidymodels`):

- object: The model itself
- new_data: The data used for simulation
- ~~times~~: The number of simulated series (handled by fablelite)
- ~~seed~~: Random generator initialisation (handled by fablelite)

The `new_data` dataset extends existing `stats::simulate` functionality by allowing the simulation to accept a new time index for simulating beyond the sample (`.idx`), and allows the simulation to work with a new set of exogenous regressors (say `x1` and `x2`).

It is expected that the innovations (`.innov`) for the simulation are randomly generated for each repition number (`rep`), which can be achieved using the `times` argument. However, users should also be able to provide a set of pre-generated innovations (`.innov`) for each repition (`.rep`). If these columns are provided in the `new_data`, then this data will be passed directly to the simulation method (without generating new numbers over `times` replications).

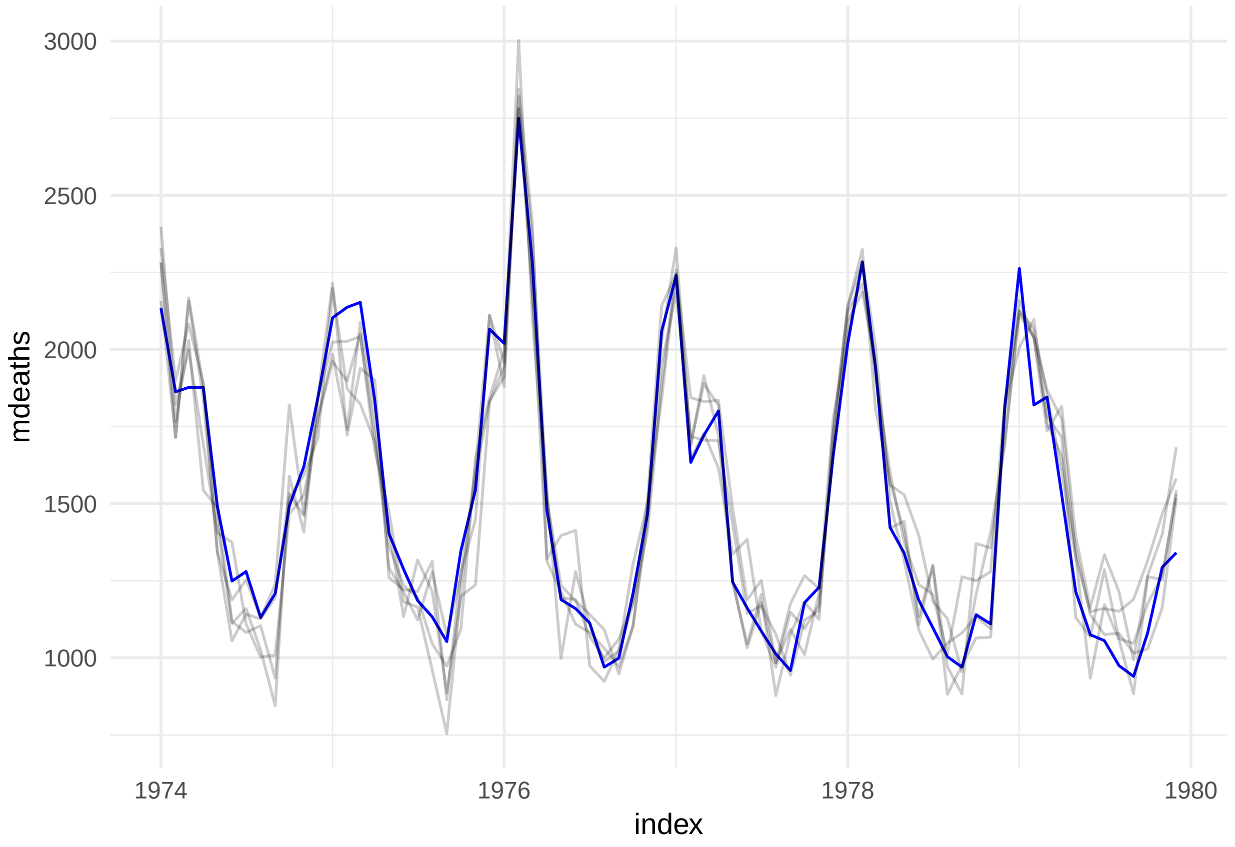
```
## Warning: `id()` is deprecated for creating key.
## Please use `key = .rep`.
```

```
## # A tsibble: 9 x 5 [1M]
## # Key:       .rep [3]
##    .rep     .idx .innov      x1    x2
##   <int>    <mth>  <dbl>   <dbl> <dbl>
## 1     1 2017 Jan  0.403  0.336  -1.23
## 2     1 2017 Feb  2.36   0.0252 -1.65
## 3     1 2017 Mar  0.508  4.58   -2.40
## 4     2 2017 Jan -1.52   3.28   -1.89
## 5     2 2017 Feb -0.891  0.182  -2.61
## 6     2 2017 Mar -1.47   3.73   -2.19
## 7     3 2017 Jan  1.34  -1.62   -3.02
## 8     3 2017 Feb  0.153  2.96   -1.22
## 9     3 2017 Mar -1.64   2.58   -1.66
```

For the end user, creating simulations would work like this:

```
library(fable)
library(tsibbledata)
UKLungDeaths %>%
  model(lm = TSLM(mdeaths ~ fourier("year", K = 4) + fdeaths)) %>%
  generate(UKLungDeaths, times = 5)
```

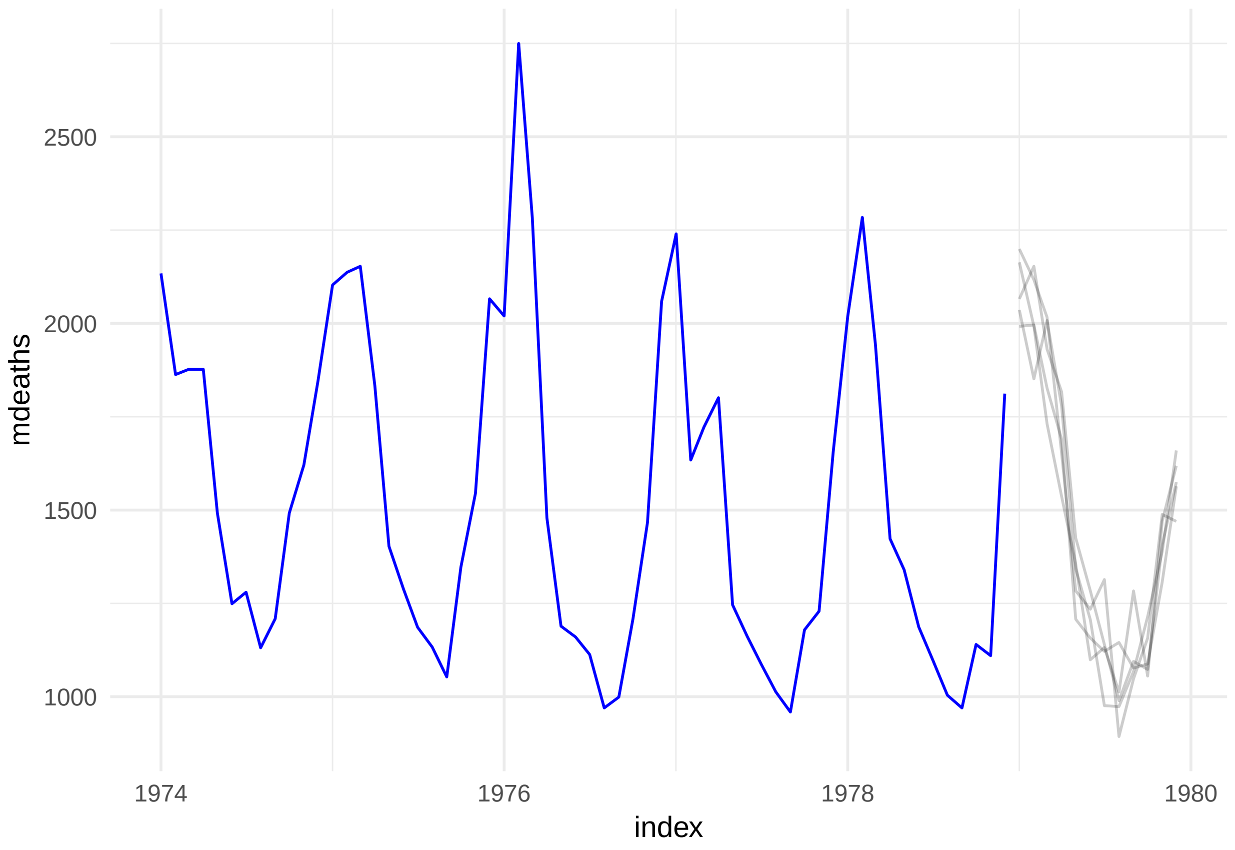
```
## # A tsibble: 360 x 4 [1M]
## # Key:       .rep, .model [5]
##    .model  .rep    index  .sim
##    <chr>  <int>    <mth> <dbl>
##  1 lm         1 1974 Jan 2398.
##  2 lm         1 1974 Feb 1821.
##  3 lm         1 1974 Mar 1996.
##  4 lm         1 1974 Apr 1692.
##  5 lm         1 1974 May 1406.
##  6 lm         1 1974 Jun 1375.
##  7 lm         1 1974 Jul 1110.
##  8 lm         1 1974 Aug 1003.
##  9 lm         1 1974 Sep 1009.
## 10 lm         1 1974 Oct 1466.
## # ... with 350 more rows
```



Or, to generate data beyond the sample:

```
library(lubridate)
UKLungDeaths %>%
  filter(year(index) <= 1978) %>%
  model(lm = TSLM(mdeaths ~ fourier("year", K = 4) + fdeaths)) %>%
  generate(
    UKLungDeaths %>% filter(year(index) > 1978),
    times = 5
  )
```

```
## # A tsibble: 60 x 4 [1M]
## # Key:       .rep, .model [5]
##    .model .rep    index  .sim
##    <chr>  <int>   <mth> <dbl>
##  1 lm         1 1979 Jan 2036.
##  2 lm         1 1979 Feb 1851.
##  3 lm         1 1979 Mar 2008.
##  4 lm         1 1979 Apr 1779.
##  5 lm         1 1979 May 1343.
##  6 lm         1 1979 Jun 1207.
##  7 lm         1 1979 Jul  976.
##  8 lm         1 1979 Aug  974.
##  9 lm         1 1979 Sep 1068.
## 10 lm         1 1979 Oct 1215.
## # ... with 50 more rows
```



5.4 Visualisation

Different plots are appropriate for visualising each type of model. For example, a plot of an ARIMA model may show the AR and/or MA roots from the model on a unit circle. A linear model has several common plots, including plots showing "Residuals vs Fitted" values, normality via a Q-Q plot, and measures of leverage. These model plots are further extended by the visreg package to show the affects of terms on the model's response. Some models currently have no model-specific plots, such as ETS, which defaults to showing a components plot using the estimated states.

Visualising these models poses a substantial challenge for consistency across models, and is made more difficult as batch modelling becomes commonplace.

Chapter 6

Advanced modelling

6.1 Batch

https://github.com/tidyverts/fable/wiki/Tidy-forecasting-with-the-fable-package

Estimating multiple models is a **key** feature of fable. Most time series can be naturally disaggregated using a series of factors known as keys. These keys are used to uniquely identify separate time series, each of which can be modelled separately.

```
UKLungDeaths %>%
  gather("sex", "deaths") %>%
  model(ETS(deaths))
```

```
## # A mable: 2 x 2
## # Key:     sex [2]
##   sex     `ETS(deaths)`
##   <chr>   <model>
## 1 mdeaths <ETS(M,A,A)>
## 2 fdeaths <ETS(M,N,M)>
```

6.2 Decomposition

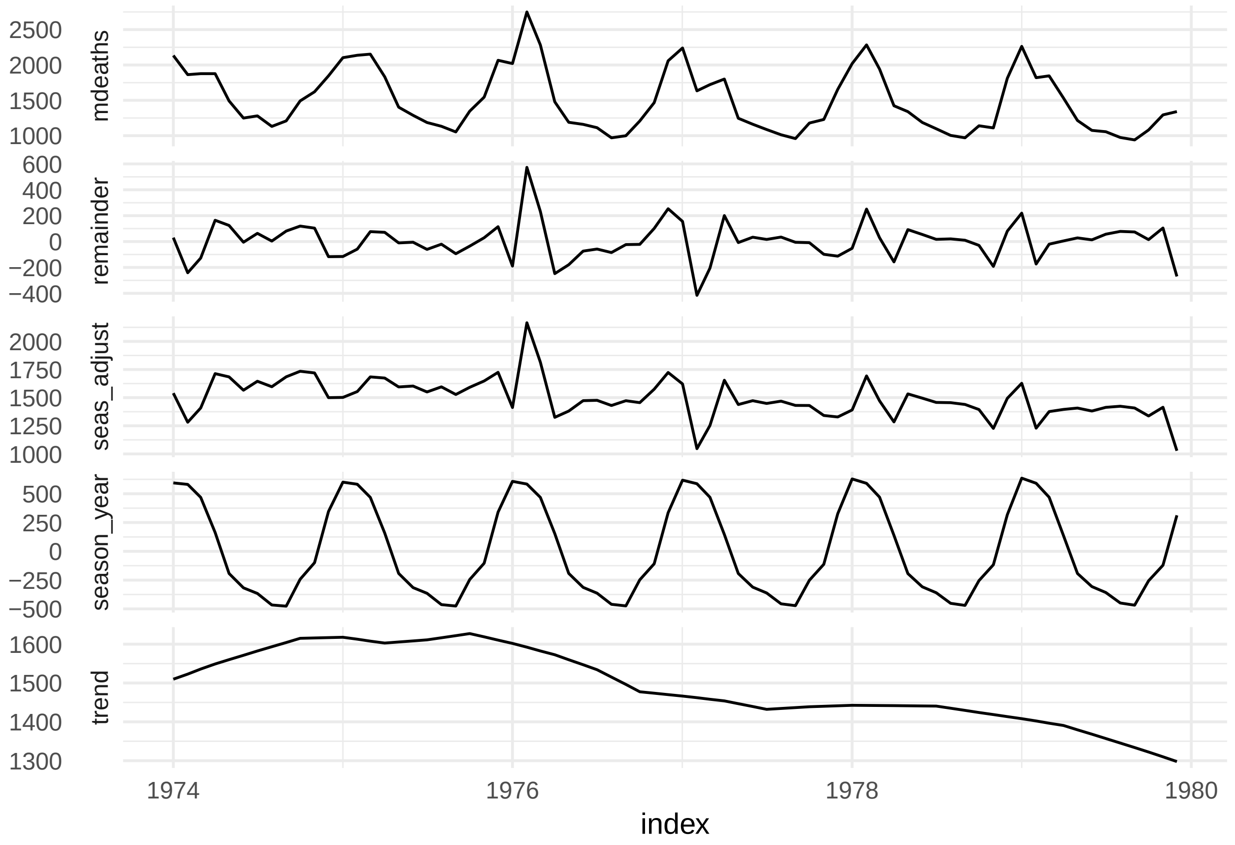
https://github.com/tidyverts/fable/wiki/Combining-models

Objects which support a `components` method can then have their components modelled separately. The working name for this functionality is `model_components`, however a shorter (single word) verb is preferred.

The user should be able to specify how each of the components are modelled, and the `components` method should define how each component is combined (and perhaps some default models that can be used).

```
library(feasts)
md_decomp <- UKLungDeaths %>%
  STL(mdeaths ~ season(window = 12))
md_decomp
```

```
## # A dable:           72 x 6 [1M]
## # STL Decomposition: mdeaths = trend + season_year + remainder
##       index mdeaths trend season_year remainder seas_adjust
##       <mth>   <dbl> <dbl>       <dbl>     <dbl>       <dbl>
##  1 1974 Jan    2134 1510.        594.     30.0        1540.
##  2 1974 Feb    1863 1523.        581.   -241.         1282.
##  3 1974 Mar    1877 1536.        469.   -127.         1408.
##  4 1974 Apr    1877 1549.        163.    165.         1714.
##  5 1974 May    1492 1560.       -193.    124.         1685.
##  6 1974 Jun    1249 1571.       -317.     -5.09       1566.
##  7 1974 Jul    1280 1583.       -366.     63.4        1646.
##  8 1974 Aug    1131 1593.       -466.      3.69       1597.
##  9 1974 Sep    1209 1604.       -476.     81.0        1685.
## 10 1974 Oct    1492 1615.       -243.    120.         1735.
## # ... with 62 more rows
```



```
md_decomp %>%
  model_components(???)
```

6.3 Ensemble

https://github.com/tidyverts/fable/issues/34

6.4 Boosting

6.5 Reconciliation

https://github.com/tidyverts/fable/issues/36

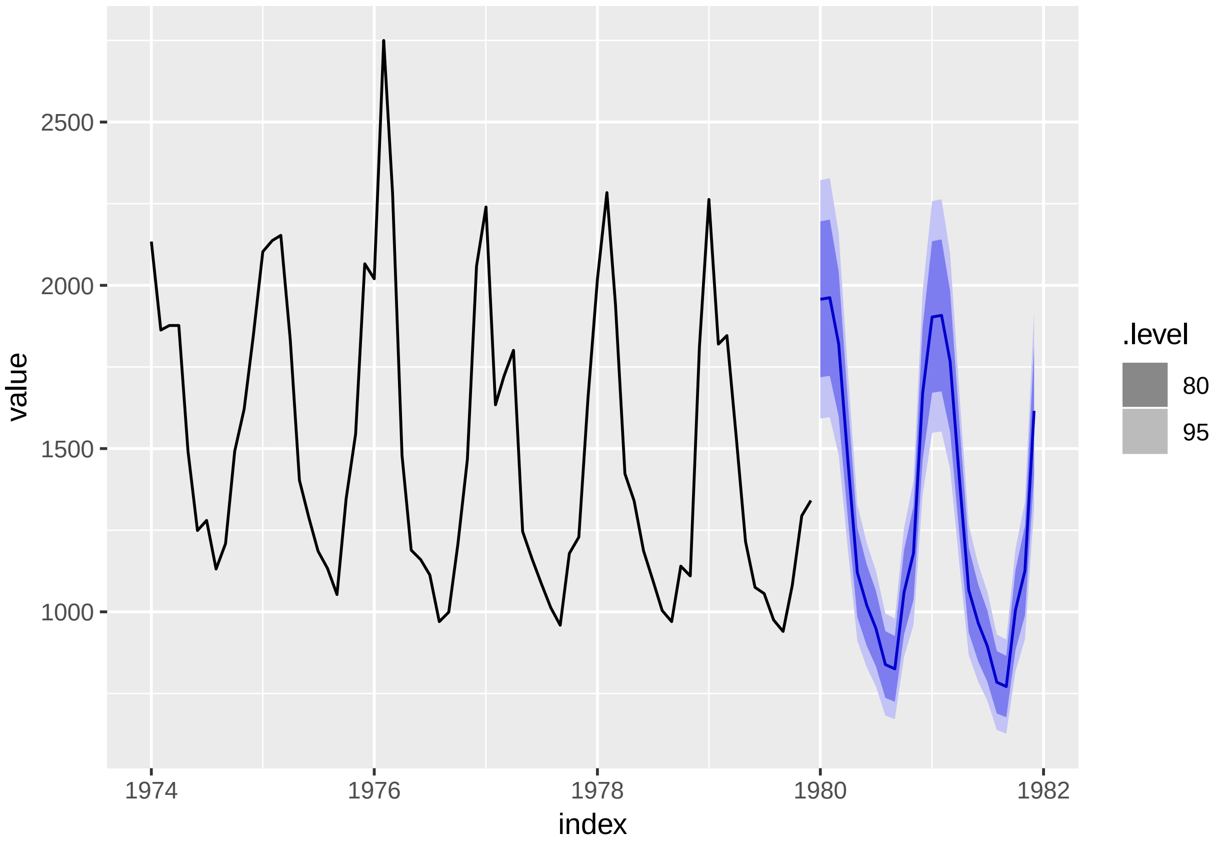
Chapter 7

Forecasting

7.1 The fable object

```
UKLungDeaths %>%
  model(ETS(mdeaths)) %>%
  forecast()
```

```
## # A fable: 24 x 4 [1M]
## # Key:     .model [1]
##    .model        index value .distribution
##    <chr>         <mth> <dbl> <dist>
##  1 ETS(value) 1980 Jan 1957. N(1957, 34666)
##  2 ETS(value) 1980 Feb 1962. N(1962, 34842)
##  3 ETS(value) 1980 Mar 1821. N(1821, 29997)
##  4 ETS(value) 1980 Apr 1455. N(1455, 19154)
##  5 ETS(value) 1980 May 1121. N(1121, 11370)
##  6 ETS(value) 1980 Jun 1020. N(1020,  9419)
##  7 ETS(value) 1980 Jul  949. N( 949,  8151)
##  8 ETS(value) 1980 Aug  839. N( 839,  6363)
##  9 ETS(value) 1980 Sep  825. N( 825,  6165)
## 10 ETS(value) 1980 Oct 1060. N(1060, 10160)
## # ... with 14 more rows
```



- Index
- Mean (backtransformed and bias adjusted)
- Standard error (may not be needed)
- Distribution

7.2 Accessing forecasts

https://github.com/tidyverts/fasster/issues/38

7.3 Bootstrapping

7.4 Visualisation

- geom_forecast
- autoplot
- autolayer

Chapter 8

Forecast evaluation

Where possible, the accuracy evaluation should be handled by existing tidymodels tools such as yardstick. It is likely that some changes or extensions will be needed for full support of time series accuracy metrics.

8.1 Accuracy

The forecast package implements accuracy as a function which is applied to a model. Out of sample accuracy can be computed by additionally providing a test set.

It is probably more transparent to compute accuracy metrics by directly providing actual response values and model predictions.

8.2 Model vs data centric

forecast is model centric

```r
# forecast
accuracy(f = forecast, x = new_ts)
```

yardstick is data centric https://github.com/r-lib/generics/pull/22

```r
# yardstick
fit_tbl %>%
  accuracy(col1, col2)
```

8.3 Proposed fable API

8.3.1 Desirable functionality

By default, `accuracy()` should provide a basic set of measures of fit for both models (`mdl_df`) and forecasts (`fbl_ts`), similarly to the `forecast` package (perhaps only MAE, RMSE/MSE, and MAPE by default).

It should be sufficiently flexible to support analysts in calculating a wide variety of accuracy measures, including:

- Point forecast accuracy measures

- Interval accuracy measures
- Distribution accuracy measures
- User specified accuracy measures

The user should be able to specify which measures they wish to compute, including measures exported by `fablelite`, measures from extension packages, and user specified measures.

8.3.2 Proposed user interface

The accuracy measures to be calculated can be specified as a list of accuracy measure functions as the `measures` argument. This input will also be flattened, allowing groups of accuracy measures to be defined.

The `...` is used to provide additional arguments that will be applied to all accuracy measures (where supported).

For models (`mdl_df`), no additional inputs are required:

```
mbl %>%
  accuracy(
    measures = list(MASE, MAE, ME),
    ...
  )
```

For forecasts (`fbl_ts`), the test set must be provided. Additionally, the dataset used for model training can be provided (interface still under consideration) to extend the inputs (required for MASE):

```
mbl %>%
  accuracy(
    new_data,
    measures = list(MASE, MAE, ME),
    training_data = NULL
    ...
  )
```

8.3.3 Implementation details

To achieve this, accuracy measure functions can expect a set of basic inputs from `accuracy()`. The measures that are required for computation should be used as formals for the function. These inputs include (list is not yet comprehensive and will be added to):

- .resid: A vector of residuals from either the training (model accuracy) or test (forecast accuracy) data.
- .resp: A vector of responses matching the residuals (for forecast accuracy, the original data must be provided).
- .fitted: The fitted values from the model, or forecasted values from the forecast.
- .dist: The distribution of fitted values from the model, or forecasted values from the forecast.
- .period: The seasonal period of the data (defaulting to 'smallest' seasonal period).
- .expr_resp: An expression for the response variable.

If a method allows more inputs than this, such as demeaning for MASE, these additional arguments are provided in the dots of the accuracy function.

8.4 Cross validation

```
CV(tsbl, mdl, h, window_type, ...)
```

8.5 Visualisation

Chapter 9

Reporting

9.1 Model equations

`equation()` displays a mathematical representation of the model that has been used.

https://github.com/tidyverts/fable/issues/4

https://github.com/tidymodels/broom/issues/492

(currently not implemented)

9.2 Object reports

`report()` returns a formatted summary of the object suitable for R Markdown documents and shiny applications.

(currently not implemented)