# Package ‘fabletools’

January 20, 2020

**Version** 0.1.2

**Title** Core Tools for Packages in the 'fable' Framework

**Description** Provides tools, helpers and data structures for developing models and time series functions for 'fable' and extension packages. These tools support a consistent and tidy interface for time series modelling and analysis.

**Depends** R (>= 3.1.3)

**Imports** tsibble (>= 0.8.0),
tibble (>= 1.4.1),
ggplot2 (>= 3.0.0),
tidyselect,
rlang (>= 0.2.0),
stats,
dplyr (>= 0.8.0),
tidyr (>= 0.8.3),
generics,
R6,
utils,
vctrs

**Suggests** colorspace,
covr,
crayon,
digest,
fable,
future.apply,
knitr,
methods,
pillar (>= 1.0.1),
feasts (>= 0.1.2),
rmarkdown,
scales,
spelling,
testthat,
tsibbledata,
lubridate,
SparseM

**ByteCompile** true

**License** GPL-3

**URL** http://fabletools.tidyverts.org/

**BugReports** https://github.com/tidyverts/fabletools/issues

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.0.2

**Roxygen** list(markdown = TRUE, roclets=c('rd', 'collate', 'namespace'))

**Language** en-GB

# R topics documented:

---

`fabletools-package` *fabletools: Core Tools for Packages in the 'fable' Framework*

---

## Description

Provides tools, helpers and data structures for developing models and time series functions for 'fable' and extension packages. These tools support a consistent and tidy interface for time series modelling and analysis.

### Author(s)

**Maintainer**: Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Authors:

- Rob Hyndman
- Earo Wang

Other contributors:

- Di Cook [contributor]
- George Athanasopoulos [contributor]

### See Also

Useful links:

- http://fabletools.tidyverts.org/
- Report bugs at https://github.com/tidyverts/fabletools/issues

## accaracy *Evaluate accuracy of a forecast or model*

### Description

Summarise the performance of the model using accuracy measures. Accuracy measures can be computed directly from models as the one-step-ahead fitted residuals are available. When evaluating accuracy on forecasts, you will need to provide a complete dataset that includes the future data and data used to train the model.

### Usage

```
accuracy(object, ...)

## S3 method for class 'mdl_df'
accuracy(object, measures = point_accuracy_measures, ...)

## S3 method for class 'fbl_ts'
accuracy(object, data, measures = point_accuracy_measures, ..., by = NULL)
```

### Arguments

| | |
|---|---|
| object | A model or forecast object |
| ... | Additional arguments to be passed to measures that use it. |
| measures | A list of accuracy measure functions to compute (such as point_accuracy_measures, interval_accuracy_measures, or distribution_accuracy_measures) |

`data` A dataset containing the complete model dataset (both training and test data). The training portion of the data will be used in the computation of some accuracy measures, and the test data is used to compute the forecast errors.

`by` Variables over which the accuracy is computed (useful for computing across forecast horizons in cross-validation). If `by` is `NULL`, groups will be chosen automatically from the key structure.

### See Also

Evaluating forecast accuracy

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibble)
library(tsibbledata)
library(dplyr)

fit <- aus_production %>%
  filter(Quarter < yearquarter("2006 Q1")) %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A")))

# In-sample training accuracy does not require extra data provided.
accuracy(fit)

# Out-of-sample forecast accuracy requires the future values to compare with.
# All available future data will be used, and a warning will be given if some
# data for the forecast window is unavailable.
fc <- fit %>%
  forecast(h = "5 years")
fc %>%
  accuracy(aus_production)

# It is also possible to compute interval and distributional measures of
# accuracy for models and forecasts which give forecast distributions.
fc %>%
  accuracy(
    aus_production,
    measures = list(interval_accuracy_measures, distribution_accuracy_measures)
  )
}
```

## `aggregate_key` *Expand a dataset to include other levels of aggregation*

### Description

Uses the structural specification given in .spec to aggregate a time series. A grouped structure is specified using grp1 * grp2, and a nested structure is specified via parent / child. Aggregating the key structure is commonly used with forecast reconciliation to produce coherent forecasts over some hierarchy.

### Usage

```
aggregate_key(.data, .spec, ...)
```

### Arguments

| | |
|---|---|
| .data | A tsibble. |
| .spec | The specification of aggregation structure. |
| ... | Name-value pairs of summary functions. The name will be the name of the variable in the result. The value should be an expression that returns a single value like min(x), n(), or sum(is.na(y)).<br>The arguments in ... are automatically quoted and evaluated in the context of the data frame. They support unquoting and splicing. See vignette("programming") for an introduction to these concepts. |

### Details

This function is experimental, and is subject to change in the future.

The way in which the measured variables are aggregated is specified in a similar way to how [dplyr::summarise()] is used.

### See Also

reconcile(), is_aggregated()

### Examples

```
library(tsibble)
tourism %>%
  aggregate_key(Purpose * (State / Region), Trips = sum(Trips))
```

## as_dable *Coerce to a dable object*

### Description

Coerce to a dable object

### Usage

```
as_dable(x, ...)

## S3 method for class 'tbl_df'
as_dable(x, response, method = NULL, seasons = list(), aliases = list(), ...)

## S3 method for class 'tbl_ts'
as_dable(x, response, method = NULL, seasons = list(), aliases = list(), ...)
```

### Arguments

`x` Object to be coerced to a dable (`dcmp_ts`)

`...` Additional arguments passed to methods

`response` The response variable(s). A single response can be specified directly via `response = y`, multiple responses should be use `response = c(y,z)`.

`method` The name of the decomposition method.

`seasons` A named list describing the structure of seasonal components (such as `period`, and `base`).

`aliases` A named list of calls describing common aliases computed from components.

## `as_fable` *Coerce to a fable object*

### Description

Coerce to a fable object

### Usage

```
as_fable(x, ...)

## S3 method for class 'tbl_ts'
as_fable(x, response, distribution, ...)

## S3 method for class 'grouped_ts'
as_fable(x, response, distribution, ...)

## S3 method for class 'tbl_df'
as_fable(x, response, distribution, ...)

## S3 method for class 'fbl_ts'
as_fable(x, response, distribution, ...)

## S3 method for class 'grouped_df'
as_fable(x, response, distribution, ...)
```

### Arguments

| | |
|---|---|
| `x` | Object to be coerced to a fable (`fbl_ts`) |
| `...` | Additional arguments passed to methods |
| `response` | The response variable(s). A single response can be specified directly via `response = y`, multiple responses should be use `response = c(y,z)`. |
| `distribution` | The distribution variable (given as a bare or unquoted variable). |

## `as_mable` *Coerce a dataset to a mable*

### Description

Coerce a dataset to a mable

### Usage

```
as_mable(x, ...)

## S3 method for class 'tbl_df'
as_mable(x, key = NULL, models = NULL, ...)
```

### Arguments

| | |
|---|---|
| `x` | A dataset containing a list model column. |
| `...` | Additional arguments passed to other methods. |
| `key` | Structural variable(s) that identify each model. |
| `models` | Identifiers for the columns containing model(s). |

## `augment.mdl_df` *Augment a mable*

### Description

Uses a fitted model to augment the response variable with fitted values and residuals.

### Usage

```
## S3 method for class 'mdl_df'
augment(x, ...)

## S3 method for class 'mdl_ts'
augment(x, ...)
```

### Arguments

| | |
|---|---|
| x | A mable. |
| ... | Arguments for model methods. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)

# Forecasting with an ETS(M,Ad,A) model to Australian beer production
aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A"))) %>%
  augment(type = "response")
}
```

## autoplot.dcmp_ts *Decomposition plots*

### Description

Produces a faceted plot of the components used to build the response variable of the dable. Useful for visualising how the components contribute in a decomposition or model.

### Usage

```
## S3 method for class 'dcmp_ts'
autoplot(object, .vars = NULL, scale_bars = TRUE, ...)
```

### Arguments

| | |
|---|---|
| object | A dable. |
| .vars | The column of the dable used to plot. By default, this will be the response variable of the decomposition. |
| scale_bars | If TRUE, each facet will include a scale bar which represents the same units across each facet. |
| ... | Further arguments passed to ggplot2::geom_line(), which can be used to specify fixed aesthetics such as colour = "red" or size = 3. |

### Examples

```
if (requireNamespace("feasts", quietly = TRUE)) {
library(feasts)
library(tsibbledata)
aus_production %>%
  model(STL(Beer)) %>%
  components() %>%
  autoplot()
}
```

## autoplot.fbl_ts *Plot a set of forecasts*

### Description

Produces a forecast plot from a fable. As the original data is not included in the fable object, it will need to be specified via the `data` argument. The `data` argument can be used to specify a shorter period of data, which is useful to focus on the more recent observations.

### Usage

```
## S3 method for class 'fbl_ts'
autoplot(object, data = NULL, level = c(80, 95), show_gap = TRUE, ...)

## S3 method for class 'fbl_ts'
autolayer(object, data = NULL, level = c(80, 95), show_gap = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `object` | A fable. |
| `data` | A tsibble with the same key structure as the fable. |
| `level` | The confidence levels for the plotted prediction intervals. |
| `show_gap` | Setting this to `FALSE` will connect the historical observations with the forecasts. |
| `...` | Further arguments passed used to specify fixed aesthetics for the forecasts such as `colour = "red"` or `size = 3`. |

### Examples

```
library(tsibbledata)
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)

fc <- aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A"))) %>%
  forecast(h = "3 years")

fc %>%
  autoplot(aus_production)
}


if (requireNamespace("fable", quietly = TRUE)) {
aus_production %>%
  autoplot(Beer) +
  autolayer(fc)
}
```

## autoplot.tbl_ts *Plot time series from a tsibble*

### Description

Produces a time series plot of one or more variables from a tsibble. If the tsibble contains a multiple keys, separate time series will be identified by colour.

### Usage

```
## S3 method for class 'tbl_ts'
autoplot(object, .vars = NULL, ...)

## S3 method for class 'tbl_ts'
autolayer(object, .vars = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | A tsibble. |
| .vars | A bare expression containing data you wish to plot. Multiple variables can be plotted using ggplot2::vars(). |
| ... | Further arguments passed to ggplot2::geom_line(), which can be used to specify fixed aesthetics such as colour = "red" or size = 3. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)
library(tsibble)

tsibbledata::gafa_stock %>%
 autoplot(vars(Close, log(Close)))
}
```

## bias_adjust *Bias adjust back-transformation functions*

### Description

To produce forecast means (instead of forecast medians) it is necessary to adjust the back-transformation function relative to the forecast variance.

### Usage

```
bias_adjust(bt, sd)
```

### Arguments

| | |
|---|---|
| bt | The back-transformation function |
| sd | The forecast standard deviation |

### Details

More details about bias adjustment can be found in the transformations vignette: read the vignette: `vignette("transformations",package = "fable")`

### Examples

```
adj_fn <- bias_adjust(function(x) exp(x), 1:10)
y <- rnorm(10)
exp(y)
adj_fn(y)
```

## box_cox *Box Cox Transformation*

### Description

`box_cox()` returns a transformation of the input variable using a Box-Cox transformation. `inv_box_cox()` reverses the transformation.

### Usage

```
box_cox(x, lambda)

inv_box_cox(x, lambda)
```

## Arguments

| | |
|---|---|
| `x` | a numeric vector. |
| `lambda` | a numeric value for the transformation parameter. |

## Details

The Box-Cox transformation is given by

$$f_\lambda(x) = \frac{x^\lambda - 1}{\lambda}$$

if $\lambda \neq 0$. For $\lambda = 0$,

$$f_0(x) = \log(x)$$

.

## Value

a transformed numeric vector of the same length as x.

## Author(s)

Rob J Hyndman & Mitchell O'Hara-Wild

## References

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. *JRSS B* **26** 211–246.

## Examples

```
library(tsibble)
library(dplyr)
airmiles %>%
  as_tsibble() %>%
  mutate(box_cox = box_cox(value, lambda = 0.3))
```

---

# `combination_ensemble` *Ensemble combination*

---

## Description

Ensemble combination

## Usage

```
combination_ensemble(..., weights = c("equal", "inv_var"))
```

### Arguments

| | |
|---|---|
| `...` | Estimated models used in the ensemble. |
| `weights` | The method used to weight each model in the ensemble. |

## combination_model *Combination modelling*

### Description

Combines multiple model definitions (passed via `...`) to produce a model combination definition using some combination function (`cmbn_fn`). Currently distributional forecasts are only supported for models producing normally distributed forecasts.

### Usage

```
combination_model(..., cmbn_fn = combination_ensemble, cmbn_args = list())
```

### Arguments

| | |
|---|---|
| `...` | Model definitions used in the combination. |
| `cmbn_fn` | A function used to produce the combination. |
| `cmbn_args` | Additional arguments passed to `cmbn_fn`. |

### Details

A combination model can also be produced using mathematical operations.

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibble)
library(tsibbledata)

# cmbn1 and cmbn2 are equivalent and equally weighted.
aus_production %>%
  model(
    cmbn1 = combination_model(SNAIVE(Beer), TSLM(Beer ~ trend() + season())),
    cmbn2 = (SNAIVE(Beer) + TSLM(Beer ~ trend() + season()))/2
  )

# An inverse variance weighted ensemble.
aus_production %>%
  model(
    cmbn1 = combination_model(
      SNAIVE(Beer), TSLM(Beer ~ trend() + season()),
      cmbn_args = list(weights = "inv_var")
    )
  )
}
```

## common_periods *Extract frequencies for common seasonal periods*

### Description

Extract frequencies for common seasonal periods

### Usage

```
common_periods(x)

## Default S3 method:
common_periods(x)

## S3 method for class 'tbl_ts'
common_periods(x)

## S3 method for class 'interval'
common_periods(x)

get_frequencies(period, ...)

## S3 method for class 'numeric'
get_frequencies(period, ...)

## S3 method for class '`NULL`'
get_frequencies(period, data, ..., .auto = c("smallest", "largest", "all"))

## S3 method for class 'character'
get_frequencies(period, data, ...)

## S3 method for class 'Period'
get_frequencies(period, data, ...)
```

### Arguments

| | |
|---|---|
| x | An object containing temporal data (such as a tsibble, interval, datetime and others.) |
| period | Specification of the time-series period |
| ... | Other arguments to be passed on to methods |
| data | A tsibble |
| .auto | The method used to automatically select the appropriate seasonal periods |

## Value

A named vector of frequencies appropriate for the provided data.

## References

https://robjhyndman.com/hyndsight/seasonal-periods/

## Examples

```
common_periods(tsibble::pedestrian)
```

---

# components.mdl_df *Extract components from a fitted model*

---

## Description

Allows you to extract elements of interest from the model which can be useful in understanding how they contribute towards the overall fitted values.

## Usage

```
## S3 method for class 'mdl_df'
components(object, ...)

## S3 method for class 'mdl_ts'
components(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A mable. |
| `...` | Other arguments passed to methods. |

## Details

A dable will be returned, which will allow you to easily plot the components and see the way in which components are combined to give forecasts.

## Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)

# Forecasting with an ETS(M,Ad,A) model to Australian beer production
aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A"))) %>%
  components() %>%
  autoplot()
}
```

## construct_fc *Construct a new set of forecasts*

### Description

Will be deprecated in the future, forecast objects should be produced with either `fable` or `as_fable` functions.

### Usage

```
construct_fc(point, sd, dist)
```

### Arguments

| | |
|---|---|
| `point` | The transformed point forecasts |
| `sd` | The standard deviation of the transformed forecasts |
| `dist` | The forecast distribution (typically produced using `new_fcdist`) |

### Details

Backtransformations are automatically handled, and so no transformations should be specified here.

## dable *Create a dable object*

### Description

A dable (decomposition table) data class (`dcmp_ts`) which is a tsibble-like data structure for representing decompositions. This data class is useful for representing decompositions, as its print method describes how its columns can be combined to produce the original data, and has a more appropriate `autoplot()` method for displaying decompositions. Beyond this, a dable (`dcmp_ts`) behaves very similarly to a tsibble (`tbl_ts`).

### Usage

```
dable(..., response, method = NULL, seasons = list(), aliases = list())
```

### Arguments

| | |
|---|---|
| ... | Arguments passed to `tsibble::tsibble()`. |
| response | The response variable(s). A single response can be specified directly via `response = y`, multiple responses should be use `response = c(y,z)`. |
| method | The name of the decomposition method. |
| seasons | A named list describing the structure of seasonal components (such as `period`, and `base`). |
| aliases | A named list of calls describing common aliases computed from components. |

## decomposition_model *Decomposition modelling*

### Description

This function allows you to specify a decomposition combination model using any additive decomposition. It works by first decomposing the data using the decomposition method provided to `dcmp_fn` with the given formula. Secondary models are used to fit each of the components from the resulting decomposition. These models are specified after the decomposition formula. All non-seasonal decomposition components must be specified, and any unspecified seasonal components will be forecasted using seasonal naive. These component models will be combined according to the decomposition method, giving a combination model for the response of the decomposition.

### Usage

```
decomposition_model(dcmp, ...)
```

### Arguments

| | |
|---|---|
| dcmp | A model definition which supports extracting decomposed `components()`. |
| ... | Model definitions used to model the components |

### See Also

*Forecasting: Principles and Practice* - Forecasting Decomposition

### Examples

```
if (requireNamespace("fable", quietly = TRUE) && requireNamespace("feasts", quietly = TRUE)) {
library(fable)
library(feasts)
library(tsibble)
library(dplyr)

vic_food <- tsibbledata::aus_retail %>%
  filter(State == "Victoria", Industry == "Food retailing")
# Identify an appropriate decomposition
vic_food %>%
  model(STL(log(Turnover) ~ season(window = Inf))) %>%
  components() %>%
  autoplot()

# Use an ARIMA model to seasonally adjusted data, and SNAIVE to season_year
# Any model can be used, and seasonal components will default to use SNAIVE.
my_dcmp_spec <- decomposition_model(
  STL(log(Turnover) ~ season(window = Inf)),
  ETS(season_adjust ~ season(”N”)), SNAIVE(season_year)
)

vic_food %>%
  model(my_dcmp_spec) %>%
  forecast(h=”5 years”) %>%
  autoplot(vic_food)
}
```

## `dist_normal` *Distributions for intervals*

### Description

Distributions for intervals

### Usage

```
dist_normal(mean, sd, ...)

dist_mv_normal(mean, sd, ...)

dist_sim(sample, ...)

dist_unknown(n, ...)
```

### Arguments

| | |
|---|---|
| `mean` | vector of distributional means. |
| `sd` | vector of distributional standard deviations. |
| `...` | Additional arguments passed on to quantile methods. |
| `sample` | a list of simulated values |
| `n` | The number of distributions. |

## Examples

```
dist_normal(rep(3, 10), seq(0, 1, length.out=10))

dist_sim(list(rnorm(100), rnorm(100), rnorm(100)))

dist_unknown(10)
```

---

## estimate *Estimate a model*

---

### Description

Estimate a model

### Usage

```
estimate(.data, ...)

## S3 method for class 'tbl_ts'
estimate(.data, .model, ...)
```

### Arguments

| | |
|---|---|
| .data | A data structure suitable for the models (such as a tsibble). |
| ... | Further arguments passed to methods. |
| .model | Definition for the model to be used. |

---

## fable *Create a fable object*

---

### Description

A fable (forecast table) data class (fbl_ts) which is a tsibble-like data structure for representing forecasts. In extension to the key and index from the tsibble (tbl_ts) class, a fable (fbl_ts) must contain columns of point forecasts for the response variable(s), and a single distribution column (fcdist).

### Usage

```
fable(..., response, distribution)
```

### Arguments

`...` Arguments passed to `tsibble::tsibble()`.

`response` The response variable(s). A single response can be specified directly via `response = y`, multiple responses should be use `response = c(y,z)`.

`distribution` The distribution variable (given as a bare or unquoted variable).

---

## `features` *Extract features from a dataset*

---

### Description

Create scalar valued summary features for a dataset from feature functions.

### Usage

```
features(.tbl, .var, features, ...)

features_at(.tbl, .vars, features, ...)

features_all(.tbl, features, ...)

features_if(.tbl, .predicate, features, ...)
```

### Arguments

`.tbl` A dataset

`.var, .vars` The variable(s) to compute features on

`features` A list of functions (or lambda expressions) for the features to compute. `feature_set()` is a useful helper for building sets of features.

`...` Additional arguments to be passed to each feature. These arguments will only be passed to features which use it in their formal arguments (`base::formals()`), and not via their `...`. While passing `na.rm = TRUE` to `stats::var()` will work, it will not for `base::mean()` as its formals are `x` and `...`. To more precisely pass inputs to each function, you can use lambdas in the list of features (`~ mean(.,na.rm = TRUE)`).

`.predicate` A predicate function (or lambda expression) to be applied to the columns or a logical vector. The variables for which .predicate is or returns TRUE are selected.

### Details

Lists of available features can be found in the following pages:

- Features by package
- Features by tag

## features_by_pkg *Features by package*

### Description

This documentation lists all available in currently loaded packages. This is a useful reference for making a `feature_set()` from particular package(s).

### Details

No features found in currently loaded packages.

### See Also

features_by_tag

## features_by_tag *Features by tag*

### Description

This documentation lists all available in currently loaded packages. This is a useful reference for making a `feature_set()` from particular tag(s).

### Details

No features found in currently loaded packages.

### See Also

features_by_pkg

## feature_set *Create a feature set from tags*

### Description

Construct a feature set from features available in currently loaded packages. Lists of available features can be found in the following pages:

- Features by package
- Features by tag

### Usage

```
feature_set(pkgs = NULL, tags = NULL)
```

### Arguments

| | |
|---|---|
| pkgs | The package(s) from which to search for features. If `NULL`, all registered features from currently loaded packages will be searched. |
| tags | Tags used to identify similar groups of features. If `NULL`, all tags will be included. |

### Registering features

Features can be registered for use with the `feature_set()` function using `register_feature()`. This function allows you to register a feature along with the tags associated with it. If the features are being registered from within a package, this feature registration should happen at load time using [.onLoad()].

## fitted.mdl_df *Extract fitted values from models*

### Description

Extracts the fitted values from each of the models in a mable. A tsibble will be returned containing these fitted values. Fitted values will be automatically back-transformed if a transformation was specified.

### Usage

```
## S3 method for class 'mdl_df'
fitted(object, ...)

## S3 method for class 'mdl_ts'
fitted(object, ...)
```

### Arguments

object A mable or time series model.

... Other arguments passed to the model method for fitted()

## forecast *Produce forecasts*

### Description

The forecast function allows you to produce future predictions of a time series from fitted models. If the response variable has been transformed in the model formula, the transformation will be automatically back-transformed (and bias adjusted if bias_adjust is TRUE). More details about transformations in the fable framework can be found in vignette("transformations",package = "fable").

### Usage

```
forecast(object, ...)

## S3 method for class 'mdl_df'
forecast(object, new_data = NULL, h = NULL, bias_adjust = TRUE, ...)
```

### Arguments

object The time series model used to produce the forecasts

... Additional arguments for forecast model methods.

new_data A tsibble containing future information used to forecast.

h The forecast horison (can be used instead of new_data for regular time series with no exogenous regressors).

bias_adjust Use adjusted back-transformed mean for transformations. Refer to vignette("transformations",pack = "fable") for more details.

### Details

The forecasts returned contain both point forecasts and their distribution. A specific forecast interval can be extracted from the distribution using the hilo() function, and multiple intervals can be obtained using report(). These intervals are stored in a single column using the hilo class, to extract the numerical upper and lower bounds you can use tidyr::unnest().

## Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibble)
library(tsibbledata)
library(dplyr)
library(tidyr)

# Forecasting with an ETS(M,Ad,A) model to Australian beer production
beer_fc <- aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A"))) %>%
  forecast(h = "3 years")

# Compute 80% and 95% forecast intervals
beer_fc %>%
  hilo(level = c(80, 95))

beer_fc %>%
  autoplot(aus_production)

# Forecasting with a seasonal naive and linear model to the monthly
# "Food retailing" turnover for each Australian state/territory.
library(dplyr)
aus_retail %>%
  filter(Industry == "Food retailing") %>%
  model(
    snaive = SNAIVE(Turnover),
    ets = TSLM(log(Turnover) ~ trend() + season()),
  ) %>%
  forecast(h = "2 years 6 months") %>%
  autoplot(filter(aus_retail, Month >= yearmonth("2000 Jan")), level = 90)

# Forecast GDP with a dynamic regression model on log(GDP) using population and
# an automatically chosen ARIMA error structure. Assume that population is fixed
# in the future.
aus_economy <- global_economy %>%
  filter(Country == "Australia")
fit <- aus_economy %>%
  model(lm = ARIMA(log(GDP) ~ Population))

future_aus <- new_data(aus_economy, n = 10) %>%
  mutate(Population = last(aus_economy$Population))

fit %>%
  forecast(new_data = future_aus) %>%
  autoplot(aus_economy)
}
```

## generate.mdl_df *Generate responses from a mable*

## Description

Use a model's fitted distribution to simulate additional data with similar behaviour to the response. This is a tidy implementation of \link[stats]{simulate}.

## Usage

```
## S3 method for class 'mdl_df'
generate(x, new_data = NULL, h = NULL, times = 1, seed = NULL, ...)

## S3 method for class 'mdl_ts'
generate(x, new_data = NULL, h = NULL, times = 1, seed = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | A mable. |
| `new_data` | The data to be generated (time index and exogenous regressors) |
| `h` | The simulation horizon (can be used instead of `new_data` for regular time series with no exogenous regressors). |
| `times` | The number of replications. |
| `seed` | The seed for the random generation from distributions. |
| `...` | Additional arguments for individual simulation methods. |

## Details

Innovations are sampled by the model's assumed error distribution. If `bootstrap` is `TRUE`, innovations will be sampled from the model's residuals. If `new_data` contains the `.innov` column, those values will be treated as innovations for the simulated paths..

## Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(dplyr)
UKLungDeaths <- as_tsibble(cbind(mdeaths, fdeaths), pivot_longer = FALSE)
UKLungDeaths %>%
  model(lm = TSLM(mdeaths ~ fourier("year", K = 4) + fdeaths)) %>%
  generate(UKLungDeaths, times = 5)
}
```

---

# `GeomForecast` *Forecast plot*

---

## Description

Generates forecasts from the given model and adds them to the plot.

### Usage

```
GeomForecast

geom_forecast(
  mapping = NULL,
  data = NULL,
  stat = "forecast",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  level = c(80, 95),
  h = NULL,
  model = fable::ETS(y),
  fc_args = list(),
  ...
)

StatForecast
```

### Arguments

`mapping` Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).

`stat` Use to override the default connection between `geom_smooth()` and `stat_smooth()`.

`position` Position adjustment, either as a string, or the result of a call to a position adjustment function.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`level` A vector of numbers between 0 and 100 which define the confidence range to be plotted. If `NULL`, confidence intervals will not be plotted, giving only the forecast line.

`h` The forecast horison (can be used instead of `new_data` for regular time series with no exogenous regressors).

`model` The time-series model used to produce the forecast. The data must be `y` (indicating aesthetic `y`), and the time index for `y` is determined from the `x` aesthetic.

`fc_args` A list of arguments to be used in the `forecast` function

`...` Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

## Format

An object of class `GeomForecast` (inherits from `Geom`, `ggproto`, `gg`) of length 7.

## Details

The aesthetics required for the forecasting to work includes forecast observations on the y axis, and the `time` of the observations on the x axis. Refer to the examples below. To automatically set up aesthetics, use `autoplot`.

## Value

A layer for a ggplot graph.

## Author(s)

Mitchell O'Hara-Wild

## See Also

`forecast`, `ggproto`

## Examples

```
## Not run:
library(ggplot2)
library(tsibble)
as_tsibble(cbind(mdeaths, fdeaths)) %>%
 autoplot() +
 geom_forecast()

## End(Not run)
```

## glance.mdl_df *Glance a mable*

### Description

Uses the models within a mable to produce a one row summary of their fits. This typically contains information about the residual variance, information criterion, and other relevant summary statistics. Each model will be represented with a row of output.

### Usage

```
## S3 method for class 'mdl_df'
glance(x, ...)

## S3 method for class 'mdl_ts'
glance(x, ...)
```

### Arguments

| | |
|---|---|
| x | A mable. |
| ... | Arguments for model methods. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)

olympic_running %>%
  model(lm = TSLM(log(Time) ~ trend())) %>%
  glance()
}
```

## guide_level *Level shade bar guide*

### Description

The level guide shows the colour from the forecast intervals which is blended with the series colour.

### Usage

```
guide_level(title = waiver(), max_discrete = 5, ...)
```

### Arguments

`title` A character string or expression indicating a title of guide. If `NULL`, the title is not shown. By default (`waiver()`), the name of the scale object or the name specified in `labs()` is used for the title.

`max_discrete` The maximum number of levels to be shown using `guide_legend`. If the number of levels exceeds this value, level shades are shown with `guide_colourbar`.

`...` Further arguments passed onto either `guide_colourbar` or `guide_legend`

---

## `hilo.fbl_ts` *Compute hilo intervals*

---

### Description

Used to extract a specified prediction interval at a particular confidence level from a distribution or fable.

### Usage

```
## S3 method for class 'fbl_ts'
hilo(x, level = c(80, 95), ...)

hilo(x, ...)

## S3 method for class 'fcdist'
hilo(x, level = 95, ...)
```

### Arguments

`x` Object to create hilo from

`level` The confidence levels for the plotted prediction intervals.

`...` Additional arguments for the distribution’s quantile function.

### Examples

```

dist_normal(10, 3) %>% hilo(95)

if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)
library(dplyr)
aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A"))) %>%
  forecast(h = "3 years") %>%
  mutate(interval = hilo(.distribution, 95))
}
```

## interpolate.mdl_df *Interpolate missing values*

### Description

Uses a fitted model to interpolate missing values from a dataset.

### Usage

```
## S3 method for class 'mdl_df'
interpolate(object, new_data, ...)

## S3 method for class 'mdl_ts'
interpolate(object, new_data, ...)
```

### Arguments

| | |
|---|---|
| `object` | A mable containing a single model column. |
| `new_data` | A dataset with the same structure as the data used to fit the model. |
| `...` | Other arguments passed to interpolate methods. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)

# The fastest running times for the olympics are missing for years during
# world wars as the olympics were not held.
olympic_running

olympic_running %>%
  model(TSLM(Time ~ trend())) %>%
  interpolate(olympic_running)
}
```

## is_aggregated *Is the element an aggregation of smaller data*

### Description

Is the element an aggregation of smaller data

### Usage

```
is_aggregated(x)
```

## Arguments

x An object.

## See Also

aggregate_key

---

## is_fable *Is the object a fable*

---

## Description

Is the object a fable

## Usage

```
is_fable(x)
```

## Arguments

x An object.

---

## is_hilo *Is the object a hilo*

---

## Description

Is the object a hilo

## Usage

```
is_hilo(x)
```

## Arguments

x An object.

## `is_mable` *Is the object a mable*

### Description

Is the object a mable

### Usage

```
is_mable(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

## `is_model` *Is the object a model*

### Description

Is the object a model

### Usage

```
is_model(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

## `MAAPE` *Mean Arctangent Absolute Percentage Error*

### Description

Mean Arctangent Absolute Percentage Error

### Usage

```
MAAPE(.resid, .actual, na.rm = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `.resid` | A vector of residuals from either the training (model accuracy) or test (forecast accuracy) data. |
| `.actual` | A vector of responses matching the fitted values (for forecast accuracy, `new_data` must be provided). |
| `na.rm` | Remove the missing values before calculating the accuracy measure |
| `...` | Additional arguments for each measure. |

### References

Kim, Sungil and Heeyoung Kim (2016) "A new metric of absolute percentage error for intermittent demand forecasts". *International Journal of Forecasting*, **32**(3), 669-679.

## `mable` *Create a new mable*

### Description

A mable (model table) data class (`mdl_df`) is a tibble-like data structure for applying multiple models to a dataset. Each row of the mable refers to a different time series from the data (identified by the key columns). A mable must contain at least one column of time series models (`mdl_ts`), where the list column itself (`lst_mdl`) describes how these models are related.

### Usage

```
mable(..., key = NULL, models = NULL)
```

### Arguments

| | |
|---|---|
| `...` | A set of name-value pairs. Arguments are evaluated sequentially, so you can refer to previously created elements. These arguments are processed with `rlang::quos()` and support unquote via `!!` and unquote-splice via `!!!`. Use `:=` to create columns that start with a dot. |
| `key` | Structural variable(s) that identify each model. |
| `models` | Identifiers for the columns containing model(s). |

ME *Point estimate accuracy measures*

## Description

Point estimate accuracy measures

## Usage

```
ME(.resid, na.rm = TRUE, ...)

MSE(.resid, na.rm = TRUE, ...)

RMSE(.resid, na.rm = TRUE, ...)

MAE(.resid, na.rm = TRUE, ...)

MPE(.resid, .actual, na.rm = TRUE, ...)

MAPE(.resid, .actual, na.rm = TRUE, ...)

MASE(
  .resid,
  .train,
  demean = FALSE,
  na.rm = TRUE,
  .period,
  d = .period == 1,
  D = .period > 1,
  ...
)

ACF1(.resid, na.action = stats::na.pass, demean = TRUE, ...)

point_accuracy_measures
```

## Arguments

| | |
|---|---|
| `.resid` | A vector of residuals from either the training (model accuracy) or test (forecast accuracy) data. |
| `na.rm` | Remove the missing values before calculating the accuracy measure |
| `...` | Additional arguments for each measure. |
| `.actual` | A vector of responses matching the fitted values (for forecast accuracy, `new_data` must be provided). |
| `.train` | A vector of responses used to train the model (for forecast accuracy, the `orig_data` must be provided). |

| | |
|---|---|
| demean | Should the response be demeaned (MASE) |
| .period | The seasonal period of the data (defaulting to 'smallest' seasonal period). from a model, or forecasted values from the forecast. |
| d | Should the response model include a first difference? |
| D | Should the response model include a seasonal difference? |
| na.action | Function to handle missing values. |

### Format

An object of class `list` of length 7.

## min_trace *Minimum trace forecast reconciliation*

### Description

Reconciles a hierarchy using the minimum trace combination method. The response variable of the hierarchy must be aggregated using sums.

### Usage

```
min_trace(
  models,
  method = c("wls_var", "ols", "wls_struct", "mint_cov", "mint_shrink"),
  sparse = NULL
)
```

### Arguments

| | |
|---|---|
| models | A column of models in a mable. |
| method | The reconciliation method to use. |
| sparse | If TRUE, the reconciliation will be computed using sparse matrix algebra? By default, sparse matrices will be used if the MatrixM package is installed. |

### References

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2019). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. Journal of the American Statistical Association, 1-45. https://doi.org/10.1080/01621459.2018.1448825

### See Also

`reconcile()`, `aggregate_key()`

## `model` *Estimate models*

### Description

Trains specified model definition(s) to a dataset. This function will estimate the a set of model definitions (passed via `...`) to each series within `.data` (as identified by the key structure). The result will be a mable (a model table), which neatly stores the estimated models in a tabular structure. Rows of the data identify different series within the data, and each model column contains all models from that model definition. Each cell in the mable identifies a single model.

### Usage

```
model(.data, ...)

## S3 method for class 'tbl_ts'
model(.data, ..., .safely = TRUE)
```

### Arguments

| | |
|---|---|
| `.data` | A data structure suitable for the models (such as a `tsibble`) |
| `...` | Definitions for the models to be used |
| `.safely` | If a model encounters an error, rather than aborting the process a NULL model will be returned instead. This allows for an error to occur when computing many models, without losing the results of the successful models. |

### Parallel

It is possible to estimate models in parallel using the future package. By specifying a `future::plan()` before estimating the models, they will be computed according to that plan.

### Examples

```
if (requireNamespace("fable", quietly = TRUE) && requireNamespace("tsibbledata", quietly = TRUE)) {
library(fable)
library(tsibbledata)

# Training an ETS(M,Ad,A) model to Australian beer production
aus_production %>%
  model(ets = ETS(log(Beer) ~ error("M") + trend("Ad") + season("A")))

# Training a seasonal naive and ETS(A,A,A) model to the monthly
# "Food retailing" turnover for selected Australian states.
library(dplyr)
aus_retail %>%
  filter(
    Industry == "Food retailing",
    State %in% c("Victoria", "New South Wales", "Queensland")
  ) %>%
  model(
    snaive = SNAIVE(Turnover),
    ets = ETS(log(Turnover) ~ error("A") + trend("A") + season("A")),
  )
}
```

## model_lhs *Extract the left hand side of a model*

### Description

Extract the left hand side of a model

### Usage

```
model_lhs(model)
```

### Arguments

| | |
|---|---|
| model | A formula |

## model_rhs *Extract the right hand side of a model*

### Description

Extract the right hand side of a model

### Usage

```
model_rhs(model)
```

### Arguments

| | |
|---|---|
| model | A formula |

## `model_sum` *Provide a succinct summary of a model*

### Description

Similarly to pillar's type_sum and obj_sum, model_sum is used to provide brief model summaries.

### Usage

```
model_sum(x)
```

### Arguments

| | |
|---|---|
| `x` | The model to summarise |

## `new_fcdist` *Create a forecast distribution object*

### Description

Create a forecast distribution object

### Usage

```
new_fcdist(..., .env)

new_fcdist_env(quantile, transformation = list(identity), display = NULL)
```

### Arguments

| | |
|---|---|
| `...` | Arguments for `f` function |
| `.env` | An environment produced using `new_fcdist_env` |
| `quantile` | A distribution function producing quantiles (such as `qnorm`) |
| `transformation` | Transformation to be applied to resulting quantiles from `quantile` |
| `display` | Function that is used to format the distribution display |

## new_hilo *Construct hilo intervals*

### Description

Construct hilo intervals

### Usage

```
new_hilo(lower, upper, level = NULL)
```

### Arguments

| | |
|---|---|
| `lower, upper` | A numeric vector of values for lower and upper limits. |
| `level` | Default `NULL` does not include 'level'. Otherwise values of length 1 or as length of `lower`, expected between 0 and 100. |

### Value

A "hilo" object

### Author(s)

Earo Wang & Mitchell O'Hara-Wild

### Examples

```
new_hilo(lower = rnorm(10), upper = rnorm(10) + 5, level = 95L)
```

## new_model_class *Create a new class of models*

### Description

Suitable for extension packages to create new models for fable.

### Usage

```
new_model_class(
  model = "Unknown model",
  train = function(.data, formula, specials, ...)
    abort("This model has not defined a training method."),
  specials = new_specials(),
  check = function(.data) { },
  prepare = function(...) { },
  ...,
  .env = caller_env(),
  .inherit = model_definition
)

new_model_definition(.class, ..., .env = caller_env(n = 2))
```

### Arguments

| | |
|---|---|
| `model` | The name of the model |
| `train` | A function that trains the model to a dataset. `.data` is a tsibble containing the data's index and response variables only. `formula` is the user's provided formula. `specials` is the evaluated specials used in the formula. |
| `specials` | Special functions produced using `new_specials()` |
| `check` | A function that is used to check the data for suitability with the model. This can be used to check for missing values (both implicit and explicit), regularity of observations, ordered time index, and univariate responses. |
| `prepare` | This allows you to modify the model class according to user inputs. `...` is the arguments passed to `new_model_definition`, allowing you to perform different checks or training procedures according to different user inputs. |
| `...` | Further arguments to `R6::R6Class()`. This can be useful to set up additional elements used in the other functions. For example, to use `common_xregs`, an `origin` element in the model is used to store the origin for `trend()` and `fourier()` specials. To use these specials, you must add an `origin` element to the object (say with `origin` = NULL). |
| `.env` | The environment from which functions should inherit from. |
| `.inherit` | A model class to inherit from. |
| `.class` | A model class (typically created with `new_model_class()`) |

### Details

This function produces a new R6 model definition. An understanding of R6 is not required, however could be useful to provide more sophisticated model interfaces. All functions have access to `self`, allowing the functions for training the model and evaluating specials to access the model class itself. This can be useful to obtain elements set in the %TODO

---

## `new_specials` *Create evaluation environment for specials*

---

### Description

Allows extension packages to make use of the formula parsing of specials.

### Usage

```
new_specials(..., .required_specials = NULL, .xreg_specials = NULL)
```

### Arguments

... A named set of functions which used to parse formula inputs

.required_specials
The names of specials which must be provided (and if not, are included with no inputs).

.xreg_specials The names of specials which will be only used as inputs to other specials (most commonly xreg).

## new_transformation *Create a new modelling transformation*

### Description

Produces a new transformation for fable modelling functions which will be used to transform, back-transform, and adjust forecasts.

### Usage

```
new_transformation(transformation, inverse)

invert_transformation(x, ...)
```

### Arguments

transformation A function which transforms the data

inverse A function which is the inverse of a transformation

x A transformation (such as one created with new_transformation).

... Further arguments passed to other methods.

### Details

For more details about transformations, read the vignette: vignette("transformations",package = "fable")

### Examples

```
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)

t_vals <- my_scaled_logit(1:10, 0, 100)
t_vals
```

## `parse_model` *Parse the model specification for specials*

### Description

Using a list of defined special functions, the user's formula specification and data is parsed to extract important modelling components.

### Usage

```
parse_model(model)
```

### Arguments

| | |
|---|---|
| `model` | A model definition |

## `parse_model_lhs` *Parse the RHS of the model formula for transformations*

### Description

Parse the RHS of the model formula for transformations

### Usage

```
parse_model_lhs(model)
```

### Arguments

| | |
|---|---|
| `model` | A model definition |

## `parse_model_rhs` *Parse the RHS of the model formula for specials*

### Description

Parse the RHS of the model formula for specials

### Usage

```
parse_model_rhs(model)
```

### Arguments

| | |
|---|---|
| `model` | A model definition |

## percentile_score *Distribution accuracy measures*

### Description

Distribution accuracy measures

### Usage

```
percentile_score(.dist, .actual, na.rm = TRUE, ...)

CRPS(.dist, .actual, n_quantiles = 1000, na.rm = TRUE, ...)

distribution_accuracy_measures
```

### Arguments

| | |
|---|---|
| `.dist` | The distribution of fitted values from the model, or forecasted values from the forecast. |
| `.actual` | A vector of responses matching the fitted values (for forecast accuracy, `new_data` must be provided). |
| `na.rm` | Remove the missing values before calculating the accuracy measure |
| `...` | Additional arguments for each measure. |
| `n_quantiles` | The number of quantiles to use in approximating CRPS when an exact solution is not available. |

### Format

An object of class `list` of length 2.

## reconcile *Forecast reconciliation*

### Description

This function allows you to specify the method used to reconcile forecasts in accordance with its key structure.

### Usage

```
reconcile(.data, ...)

## S3 method for class 'mdl_df'
reconcile(.data, ...)
```

### Arguments

| | |
|---|---|
| `.data` | A mable. |
| `...` | Reconciliation methods applied to model columns within `.data`. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
lung_deaths_agg <- as_tsibble(cbind(mdeaths, fdeaths)) %>%
  aggregate_key(key, value = sum(value))

lung_deaths_agg %>%
  model(lm = TSLM(value ~ trend() + season())) %>%
  reconcile(lm = min_trace(lm)) %>%
  forecast()
}
```

---

## `refit.mdl_df` *Refit a mable to a new dataset*

---

### Description

Applies a fitted model to a new dataset. For most methods this can be done with or without re-estimation of the parameters.

### Usage

```
## S3 method for class 'mdl_df'
refit(object, new_data, ...)

## S3 method for class 'mdl_ts'
refit(object, new_data, ...)
```

### Arguments

| | |
|---|---|
| `object` | A mable. |
| `new_data` | A tsibble dataset used to refit the model. |
| `...` | Additional optional arguments for refit methods. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)

fit <- as_tsibble(mdeaths) %>%
  model(ETS(value ~ error("M") + trend("A") + season("A")))
fit %>% report()

fit %>%
  refit(as_tsibble(fdeaths)) %>%
  report(reinitialise = TRUE)
}
```

## register_feature *Register a feature function*

### Description

Allows users to find and use features from your package using `feature_set()`. If the features are being registered from within a package, this feature registration should happen at load time using [.onLoad()].

### Usage

```
register_feature(fn, tags)
```

### Arguments

| | |
|---|---|
| `fn` | The feature function |
| `tags` | Identifying tags |

### Examples

```
## Not run:
tukey_five <- function(x){
  setNames(fivenum(x), c("min", "hinge_lwr", "med", "hinge_upr", "max"))
}

register_feature(tukey_five, tags = c("boxplot", "simple"))


## End(Not run)
```

## report — *Report information about an object*

### Description

Displays the object in a suitable format for reporting.

### Usage

```
report(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | The object to report |
| `...` | Additional options for the reporting function |

## residuals.mdl_df — *Extract residuals values from models*

### Description

Extracts the residuals from each of the models in a mable. A tsibble will be returned containing these residuals.

### Usage

```
## S3 method for class 'mdl_df'
residuals(object, ...)

## S3 method for class 'mdl_ts'
residuals(object, type = "innovation", ...)
```

### Arguments

| | |
|---|---|
| `object` | A mable or time series model. |
| `...` | Other arguments passed to the model method for `residuals()` |
| `type` | The type of residuals to compute. If `type="response"`, residuals on the back-transformed data will be computed. |

## response *Extract the response variable from a model*

### Description

Returns a tsibble containing only the response variable used in the fitting of a model.

### Usage

```
response(object, ...)
```

### Arguments

| | |
|---|---|
| object | The object containing response data |
| ... | Additional parameters passed on to other methods |

## scale_level *Level colour scales*

### Description

This set of scales defines new scales for level geoms equivalent to the ones already defined by ggplot2. This allows the shade of confidence intervals to work with the legend output.

### Usage

```
scale_level_gradient(
  ...,
  low = "#888888",
  high = "#BBBBBB",
  space = "Lab",
  na.value = NA,
  guide = "level"
)

scale_level_continuous(
  ...,
  low = "#888888",
  high = "#BBBBBB",
  space = "Lab",
  na.value = NA,
  guide = "level"
)
```

### Arguments

... Arguments passed on to `continuous_scale`

**scale_name** The name of the scale

**palette** A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding values in the range the scale maps to.

**name** The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

**breaks** One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

**minor_breaks** One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

**labels** One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

**limits** One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits

**rescaler** Used by diverging and n colour gradients (i.e. `scale_colour_gradient2()`, `scale_colour_gradientn()`). A function used to scale the input values to the range [0, 1].

**oob** Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with `NA`.

**trans** Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `name_trans`, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

**position** The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

**super** The super class to use for the constructed scale

**expand** Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`low, high` Colours for low and high ends of the gradient.

`space` colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

`na.value` Colour to use for missing values

`guide` Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend.

## Value

A ggproto object inheriting from `Scale`

---

`stream` *Extend a fitted model with new data*

---

## Description

Extend the length of data used to fit a model and update the parameters to suit this new data.

## Usage

```
stream(object, ...)

## S3 method for class 'mdl_df'
stream(object, new_data, ...)
```

## Arguments

`object` An object (such as a model) which can be extended with additional data.

`...` Additional arguments passed on to stream methods.

`new_data` A dataset of the same structure as was used to fit the model.

## tidy.mdl_df *Extract model coefficients from a mable*

### Description

This function will obtain the coefficients (and associated statistics) for each model in the mable.

### Usage

```
## S3 method for class 'mdl_df'
tidy(x, ...)

## S3 method for class 'mdl_df'
coef(object, ...)

## S3 method for class 'mdl_ts'
tidy(x, ...)

## S3 method for class 'mdl_ts'
coef(object, ...)
```

### Arguments

| | |
|---|---|
| x, object | A mable. |
| ... | Arguments for model methods. |

### Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibbledata)

olympic_running %>%
  model(lm = TSLM(log(Time) ~ trend())) %>%
  tidy()
}
```

## traverse *Recursively traverse an object*

### Description

Recursively traverse an object

### Usage

```
traverse(
  x,
  .f = list,
  .g = identity,
  .h = identity,
  base = function(.x) is_syntactic_literal(.x) || is_symbol(.x)
)
```

### Arguments

| | |
|---|---|
| `x` | The object to traverse |
| `.f` | A function for combining the recursed components |
| `.g` | A function applied to the object before recursion |
| `.h` | A function applied to the base case |
| `base` | The base case for the recursion |

## `validate_formula` *Validate the user provided model*

### Description

Appropriately format the user's model for evaluation. Typically ran as one of the first steps in a model function.

### Usage

```
validate_formula(model, data = NULL)
```

### Arguments

| | |
|---|---|
| `model` | A quosure for the user's model specification |
| `data` | A dataset used for automatic response selection |

## winkler_score *Interval estimate accuracy measures*

### Description

Interval estimate accuracy measures

### Usage

```
winkler_score(.dist, .actual, level = 95, na.rm = TRUE, ...)

interval_accuracy_measures
```

### Arguments

| | |
|---|---|
| `.dist` | The distribution of fitted values from the model, or forecasted values from the forecast. |
| `.actual` | A vector of responses matching the fitted values (for forecast accuracy, `new_data` must be provided). |
| `level` | The level of the forecast interval. |
| `na.rm` | Remove the missing values before calculating the accuracy measure |
| `...` | Additional arguments for each measure. |

### Format

An object of class `list` of length 1.

# Index