

Package: fabletools (via r-universe)

October 11, 2026

Title Core Tools for Packages in the 'fable' Framework

Version 0.8.0.9000

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.

License GPL-3

URL <https://fabletools.tidyverts.org/>,
<https://github.com/tidyverts/fabletools>

BugReports <https://github.com/tidyverts/fabletools/issues>

Depends R (>= 3.1.3)

Imports tsibble (>= 0.9.0), tibble (>= 1.4.1), ggplot2 (>= 3.0.0), tidyselect, rlang (>= 0.4.5), stats, dplyr (>= 1.0.0), tidyr (>= 1.1.0), generics, R6, utils, vctrs (>= 0.2.2), distributional (>= 0.6.0), progressr, lifecycle, ggdist, scales, cli

Suggests covr, crayon, fable (>= 0.2.0), future, future.apply, knitr, pillar (>= 1.0.1), feasts (>= 0.1.2), rmarkdown, spelling, testthat, tsibbledata (>= 0.2.0), lubridate, mixtime, urca, mvtnorm, Matrix, ggtime

VignetteBuilder knitr

RdMacros lifecycle

ByteCompile true

Encoding UTF-8

Language en-GB

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

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev

Repository <https://tidyverts.r-universe.dev>

Date/Publication 2026-10-11 04:33:18 UTC

RemoteUrl <https://github.com/tidyverts/fabletools>

RemoteRef HEAD

RemoteSha 00d3eaf51d39b11455c56cb935b79651f4bdf9d9

Contents

fabletools-package	3
accuracy.mbl_df	4
agg_vec	6
aggregate_key	6
as_dable	7
as_fable	8
as_mable	9
augment.mbl_df	9
autoplot.dcmp_ts	10
autoplot.fbl_ts	11
autoplot.tbl_ts	12
bootstrap_iid	13
box_cox	14
coherent_cmat	15
coherent_smat	16
combination_ensemble	17
combination_model	18
combination_weighted	19
common_periods	19
common_xregs	20
components.mbl_df	21
conformal_scp	22
dable	23
decomposition_model	24
distribution_var	25
estimate	25
fable	26
feature_set	26
features	27
fitted.mbl_df	28
forecast.mbl_df	29
generate.mbl_df	31
glance.mbl_df	33
hypothesize.mbl_df	34
interpolate.mbl_df	34
IRF	35
is_aggregated	36
is_dable	37
is_fable	37
is_mable	37
is_model	38

MAAPE	38
mable	39
mable_vars	39
MDA	40
ME	41
model	42
model_lhs	43
model_rhs	44
model_sum	44
new_model_class	44
new_specials	46
new_transformation	46
outliers	47
percentile_score	48
reconcile	49
reconcile_bu	50
reconcile_mint	51
reconcile_mo	52
reconcile_td	53
refit.mbl_df	53
register_feature	54
report	55
residuals.mbl_df	55
response	56
response_vars	56
scenarios	57
simulate_iid	57
skill_score	58
special_xreg	59
stream	59
tidy.mbl_df	60
winkler_score	61
Index	63

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> ([ORCID](#))

Authors:

- Mitchell O’Hara-Wild <mail@mitchelloharawild.com> ([ORCID](#))
- Rob Hyndman
- Earo Wang ([ORCID](#))

Other contributors:

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

See Also

Useful links:

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

accuracy.mbl_df

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

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

## S3 method for class 'mdl_lst'
accuracy(object, ...)

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

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

Arguments

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

```
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)
  )
```

agg_vec	Create an aggregation vector
---------	------------------------------

Description

[Maturing]

Usage

```
agg_vec(x = character(), aggregated = logical(vec_size(x)))
```

Arguments

x	The vector of values.
aggregated	A logical vector to identify which values are <aggregated>.

Details

An aggregation vector extends usual vectors by adding <aggregated> values. These vectors are typically produced via the [aggregate_key\(\)](#) function, however it can be useful to create them manually to produce more complicated hierarchies (such as unbalanced hierarchies).

Examples

```
agg_vec(
  x = c(NA, "A", "B"),
  aggregated = c(TRUE, FALSE, FALSE)
)
```

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. |
| ... | <p><data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result.</p> <p>The value can be:</p> <ul style="list-style-type: none"> • A vector of length 1, e.g. <code>min(x)</code>, <code>n()</code>, or <code>sum(is.na(y))</code>. • A data frame with 1 row, to add multiple columns from a single expression. |

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_bu\(\)](#), [reconcile_td\(\)](#), [reconcile_mo\(\)](#), [reconcile_mint\(\)](#), [is_aggregated\(\)](#)

Examples

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

as_dable	<i>Coerce to a dable object</i>
----------	---------------------------------

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 character vector of response variable(s).
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	<i>Coerce to a fable object</i>
----------	---------------------------------

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_df'
as_fable(x, response, distribution, ...)

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

## S3 method for class 'forecast'
as_fable(x, ..., point_forecast = list(.mean = mean))
```

Arguments

x	Object to be coerced to a fable (fbl_df)
...	Additional arguments passed to methods
response	The character vector of response variable(s).
distribution	The name of the distribution column (can be provided using a bare expression).

`point_forecast` The point forecast measure(s) which should be returned in the resulting fable. Specified as a named list of functions which accept a distribution and return a vector. To compute forecast medians, you can use `list(.median = median)`.

as_mable	<i>Coerce a dataset to a mable</i>
----------	------------------------------------

Description

Coerce a dataset to a mable

Usage

```
as_mable(x, ...)
```

```
## S3 method for class 'data.frame'
```

```
as_mable(x, key = NULL, model = NULL, response = NULL, ...)
```

Arguments

<code>x</code>	A dataset containing a list model column.
<code>...</code>	Additional arguments passed to other methods.
<code>key</code>	Structural variable(s) that identify each model.
<code>model</code>	Identifiers for the columns containing model(s).
<code>response</code>	A character vector of the response variable(s) of the models. If <code>NULL</code> , the response variable(s) are taken from the models (and must be provided if the mable contains no models).

augment.mbl_df	<i>Augment a mable</i>
----------------	------------------------

Description

Uses a fitted model to augment the response variable with fitted values and residuals. Response residuals (back-transformed) are stored in the `.resid` column, while innovation residuals (transformed) are stored in the `.innov` column.

Usage

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

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

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

Arguments

x	A dable.
...	Arguments for model methods.
type	Deprecated.

Examples

```
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()
```

autoplot.dcmp_ts	<i>Decomposition plots</i>
------------------	----------------------------

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, scaleBars = TRUE, level = c(80, 95), ...)
```

Arguments

object	A dable.
.vars	The column of the dable used to plot. By default, this will be the response variable of the decomposition.
scaleBars	If TRUE, each facet will include a scale bar which represents the same units across each facet.
level	If the decomposition contains distributions, which levels should be used to display intervals?
...	Further arguments passed to <code>ggplot2::geom_line()</code> , which can be used to specify fixed aesthetics such as <code>colour = "red"</code> or <code>linewidth = 3</code> .

Examples

```
library(feasts)
library(tsibbledata)
aus_production %>%
  model(STL(Beer)) %>%
  components() %>%
  autoplot()
```

autoplot.fbl_ts	<i>Plot a set of forecasts</i>
-----------------	--------------------------------

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),
  point_forecast = list(mean = mean),
  show_gap = TRUE,
  ...
)
```

Arguments

object	A fable.
data	A tsibble with the same key structure as the fable.
level	The confidence level(s) for the plotted intervals.
show_gap	Setting this to FALSE will connect the most recent value in data with the forecasts.
...	Further arguments passed used to specify fixed aesthetics for the forecasts such as colour = "red" or linewidth = 3.
point_forecast	The point forecast measure to be displayed in the plot.

Examples

```
library(fable)
library(tsibbledata)

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

fc %>%
  autoplot(aus_production)

aus_production %>%
  autoplot(Beer) +
  autolayer(fc)
```

autoplot.tbl_ts	<i>Plot time series from a tsibble</i>
-----------------	--

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 <code>ggplot2::vars()</code> .
...	Further arguments passed to <code>ggplot2::geom_line()</code> , which can be used to specify fixed aesthetics such as <code>colour = "red"</code> or <code>linewidth = 3</code> .

Examples

```
library(fable)
library(tsibbledata)
library(tsibble)

tsibbledata::gafa_stock %>%
```

```
autoplot(vars(Close, log(Close)))
```

bootstrap_iid	<i>Bootstrap forecast sample paths</i>
---------------	--

Description

[Experimental]

Wraps a fitted model (or a model column from a mable) so that `forecast()` produces its distribution from bootstrapped sample paths instead of closed-form/analytical results, and so that `generate()` draws its innovations from the model's own fitted residuals instead of the model's assumed error distribution. This allows the forecast distribution to take a different shape to the model's theoretical one, which can better reflect asymmetry or heavy tails in the data.

`bootstrap_iid()` resamples residuals independently (with replacement) for each simulated time point. `bootstrap_block()` instead resamples contiguous blocks of residuals, preserving any residual autocorrelation the model hasn't captured.

Applied to a `mdl_1st` (a model column spanning multiple series/keys), `bootstrap_iid()/bootstrap_block()` sample **jointly** across the series: the same historical time point is drawn for every series at a given (replicate, forecast-step) position, so the historical cross-sectional relationship between series' residuals carries over into the simulated innovations, rather than resampling each series as if it were independent of the others. This matters most for hierarchies (see `aggregate_key()/reconcile_mint()`): jointly bootstrapping respects the observed correlation between a hierarchy's nodes, instead of treating every node as unrelated. If the series don't all share the same historical time domain, sampling is restricted to their overlapping period and a warning is raised.

Usage

```
bootstrap_iid(model, times = 5000)
```

```
bootstrap_block(model, times = 5000, block_size = NULL)
```

Arguments

model	A fitted model, typically a model column from a mable modified with <code>dplyr::mutate()</code> (e.g. <code>mutate(ets = bootstrap_iid(ets))</code>). Alternatively, an unfitted model specification can be wrapped directly inside <code>model()</code> (e.g. <code>model(ets = bootstrap_iid(ETS(Trips)))</code>), in which case each series is fitted and bootstrapped independently - unlike wrapping an already-fitted <code>mdl_1st</code> column (which samples jointly across series, see Description).
times	The number of bootstrapped sample paths to simulate.
block_size	The number of contiguous residuals resampled together in each bootstrap draw. Defaults to the data's seasonal period (via <code>stats::frequency()</code>), which preserves within-season residual autocorrelation; use <code>block_size = 1</code> for the same behaviour as <code>bootstrap_iid()</code> .

See Also

`simulate_iid()`, `forecast()`, `generate()`

Examples

```
library(fable)
library(dplyr)
aus_production %>%
  model(ets = ETS(Beer)) %>%
  mutate(ets = bootstrap_iid(ets, times = 1000)) %>%
  forecast()

# Applied to the model specification before fitting, each series is
# bootstrapped independently (no joint cross-sectional sampling):
aus_production %>%
  model(ets = bootstrap_iid(ETS(Beer), times = 1000)) %>%
  forecast()
```

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))
```

coherent_cmat

Zero-constraint matrix for coherent time series

Description

Constructs the $n_a \times n$ zero-constraint matrix **C** for a set of linearly related time series, where $n_a = n - n_b$ is the number of aggregated series.

Usage

```
coherent_cmat(data, sparse = FALSE, ...)
```

Arguments

<code>data</code>	A data object which contains linearly related coherent structures.
<code>sparse</code>	If TRUE, a sparse matrix (class "dgCMatrix" from the Matrix package) is returned. Defaults to FALSE.
<code>...</code>	Arguments passed to methods.

Details

Given the aggregation matrix **A** (see `coherent_smat()`), the constraint matrix is defined as

$$\mathbf{C} = [\mathbf{I}_{n_a} \quad -\mathbf{A}],$$

so that $\mathbf{C}\mathbf{y}_t = \mathbf{0}_{n_a}$ for all coherent vectors $\mathbf{y}_t$. This zero-constrained representation yields the mapping matrix

$$\mathbf{M} = \mathbf{I}_n - \mathbf{W}\mathbf{C}'(\mathbf{C}\mathbf{W}\mathbf{C}')^{-1}\mathbf{C},$$

which requires inverting an $n_a \times n_a$ matrix rather than the $n_b \times n_b$ matrix in the structural form, and is therefore more efficient when $n_a < n_b$.

Value

An $n_a \times n$ matrix (dense or sparse) whose rows encode each aggregation constraint as $\mathbf{C}\mathbf{y}_t = \mathbf{0}_{n_a}$.

References

Hyndman, R. J., & Athanasopoulos, G. (2022). *Notation for forecast reconciliation*. <https://robjhyndman.com/hyndsight/reconciliation-notation.html>

Di Fonzo, T., & Girolimetto, D. (2021). Cross-temporal forecast reconciliation: Optimal combination method and heuristic alternatives. *International Journal of Forecasting*. doi:10.1016/j.ijforecast.2021.08.004

See Also

[coherent_smat\(\)](#) for the corresponding structural matrix **S** and [aggregate_key\(\)](#) for computing cross-sectional aggregations with tsibble data sets.

coherent_smat	<i>Structural (summing) matrix for coherent time series</i>
---------------	---

Description

Constructs the $n \times n_b$ structural matrix **S** for a set of linearly related time series, where n is the total number of series and n_b is the number of bottom-level series.

Usage

```
coherent_smat(data, sparse = FALSE, ...)

## S3 method for class 'tbl'
coherent_smat(data, sparse = FALSE, with_bottom = TRUE, ...)
```

Arguments

data	A data object which contains linearly related coherent structures.
sparse	If TRUE, a sparse matrix (class "dgCMatrix" from the Matrix package) is returned. Defaults to FALSE.
...	Arguments passed to methods.
with_bottom	If TRUE, the returned matrix includes the identity matrix for bottom-level series. Defaults to TRUE.

Details

Let $\mathbf{b}_t$ be the n_b -vector of bottom-level series at time t , and $\mathbf{a}_t = \mathbf{A}\mathbf{b}_t$ be the $n_a = n - n_b$ aggregated series, where $\mathbf{A}$ is the $n_a \times n_b$ aggregation matrix. The full n -vector of series is $\mathbf{y}_t = [\mathbf{a}'_t, \mathbf{b}'_t]'$, and the structural matrix satisfies

$$\mathbf{y}_t = \mathbf{S}\mathbf{b}_t, \quad \mathbf{S} = \begin{bmatrix} \mathbf{A} \\ \mathbf{I}_{n_b} \end{bmatrix}.$$

Any linear reconciliation method can be written as $\tilde{\mathbf{y}}_h = \mathbf{S}\mathbf{G}\hat{\mathbf{y}}_h$, where $\hat{\mathbf{y}}_h$ are the h -step base forecasts. Optimal reconciled forecasts use $\mathbf{G} = (\mathbf{S}'\mathbf{W}^{-1}\mathbf{S})^{-1}\mathbf{S}'\mathbf{W}^{-1}$, with $\mathbf{W}$ a positive definite weight matrix (see `min_trace()`).

Value

An $n \times n_b$ matrix (dense or sparse) encoding the structural relationships among all series.

References

Hyndman, R. J., & Athanasopoulos, G. (2022). *Notation for forecast reconciliation*. <https://robjhyndman.com/hyndsight/reconciliation-notation.html>

See Also

`coherent_cmat()` for the corresponding zero-constraint matrix $\mathbf{C}$ and `aggregate_key()` for computing cross-sectional aggregations with tsibble data sets.

Examples

```
tsibble::tourism %>%
  aggregate_key(Purpose, Trips = sum(Trips)) %>%
  coherent_smat()
```

combination_ensemble *Ensemble combination*

Description

Ensemble combination

Usage

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

Arguments

<code>...</code>	Estimated models used in the ensemble.
<code>weights</code>	The method used to weight each model in the ensemble.

See Also

[combination_weighted\(\)](#)

combination_model	<i>Combination modelling</i>
-------------------	------------------------------

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

```
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")
    )
  )
```

combination_weighted	<i>Weighted combination</i>
----------------------	-----------------------------

Description

Weighted combination

Usage

```
combination_weighted(..., weights = NULL)
```

Arguments

...	Estimated models used in the ensemble.
weights	The numeric weights applied to each model in ...

See Also

[combination_ensemble\(\)](#)

common_periods	<i>Extract frequencies for common seasonal periods</i>
----------------	--

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)

## S3 method for class '`mixtime::mt_unit`'
common_periods(x)

## S3 method for class 'list'
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, ...)

## S3 method for class '`mixtime::mixtime`'
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)
```

common_xregs

Common exogenous regressors

Description

These special functions provide interfaces to more complicated functions within the model formulae interface.

Usage

```
common_xregs
```

Specials

trend: The trend special includes common linear trend regressors in the model. It also supports piecewise linear trend via the knots argument. For irregular time series, the trend is proportional to elapsed time, measured in the units of the time index (e.g. days for Date, seconds for POSIXct).

```
trend(knots = NULL, origin = NULL)
```

knots A vector of times (same class as the data's time index) identifying the position of knots for a piecewise linear trend.
origin An optional time value (same class as the data's time index) to act as the starting time for the trend.

season: The season special includes seasonal dummy variables in the model.

```
season(period = NULL)
```

period The periodic nature of the seasonality. This can be either a number indicating the number of observations in each season.

fourier: The fourier special includes seasonal fourier terms in the model. The maximum order of the fourier terms must be specified using K.

```
fourier(period = NULL, K)
```

period The periodic nature of the seasonality. This can be either a number indicating the number of observations in each season.
K The maximum order of the fourier terms.

components.mbl_df	<i>Extract components from a fitted model</i>
-------------------	---

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 'mbl_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.

The components can also be visualised using the `autoplot()` method provided by the `ggtime` package.

Examples

```
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()
```

conformal_scp

Split conformal prediction intervals

Description

[Experimental]

Produces forecast distributions using split conformal prediction rather than the model's own distributional assumptions. This can give better calibrated intervals when the model's errors are biased or poorly described by its assumed distribution.

The forecast distribution is obtained as follows:

1. The model is refit on an expanding window of its history, forecasting up to h steps ahead from each origin to obtain out-of-sample errors.
2. The errors for each forecast horizon are used as calibration sets. Horizons with fewer than `min_calibration` errors instead use the errors from all horizons combined.
3. The quantiles of the calibration errors are added to the model's point forecast to give the forecast distribution.

Usage

```
conformal_scp(model, times = 5000, min_calibration = 20)
```

Arguments

<code>model</code>	A fitted model (e.g. <code>mutate(ets = conformal_scp(ets))</code>), or a model specification (e.g. <code>model(ets = conformal_scp(ETS(y)))</code>).
<code>times</code>	The number of quantiles used to represent each forecast distribution.
<code>min_calibration</code>	The minimum number of errors needed to calibrate a forecast horizon.

References

Wang, X., & Hyndman, R.J. (2024). **conformalForecast**: Conformal Prediction for Time Series Forecasting. <https://github.com/xqnlwang/conformalForecast>

See Also

[bootstrap_iid\(\)](#), [simulate_iid\(\)](#), [forecast\(\)](#), [refit\(\)](#)

Examples

```
library(fable)
library(dplyr)
aus_production %>%
  model(ets = ETS(Beer)) %>%
  mutate(ets = conformal_scp(ets)) %>%
  forecast()
```

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 provided by ggtime. 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 name of the response variable column.
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

<code>dcmp</code>	A model definition which supports extracting decomposed <code>components()</code> .
<code>...</code>	Model definitions used to model the components

See Also

Forecasting: Principles and Practice - Forecasting Decomposition

Examples

```
library(fable)
library(feasts)
library(tsibble)
library(dplyr)
library(ggtime)

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 ETS 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)
```

distribution_var	<i>Return distribution variable</i>
------------------	-------------------------------------

Description

distribution_var() returns a character vector of the distribution variable in the data.

Usage

```
distribution_var(x)
```

Arguments

x	A dataset containing a distribution variable (such as a fable).
---	---

estimate	<i>Estimate a model</i>
----------	-------------------------

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	<i>Create a fable object</i>
-------	------------------------------

Description

A fable (forecast table) data class (fbl_df) 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_df) must also contain a single distribution column that uses values from the distributional package.

Usage

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

Arguments

...	Arguments passed to <code>tsibble::tsibble()</code> .
response	The character vector of response variable(s).
distribution	The name of the distribution column (can be provided using a bare expression).

feature_set	<i>Create a feature set from tags</i>
-------------	---------------------------------------

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()]`.

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

<code>.tbl</code>	A dataset
<code>.var</code>	An expression that produces a vector from which the features are computed.
<code>features</code>	A list of functions (or lambda expressions) for the features to compute. feature_set() is a useful helper for building sets of features.
<code>...</code>	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 <code>...</code> . While passing <code>na.rm = TRUE</code> to stats::var() will work, it will not for base::mean() as its formals are <code>x</code> and <code>...</code> . To more precisely pass inputs to each function, you should use lambdas in the list of features (<code>~ mean(., na.rm = TRUE)</code>).
<code>.vars</code>	A tidyselect compatible selection of the column(s) to compute features on.
<code>.predicate</code>	A predicate function (or lambda expression) to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected.

Details

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

- [Features by package](#)
- [Features by tag](#)

See Also

[feature_set\(\)](#)

Examples

```
# Provide a set of functions as a named list to features.
library(tsibble)
tourism %>%
  features(Trips, features = list(mean = mean, sd = sd))

# Search and use useful features with `feature_set()`.

library(feasts)

tourism %>%
  features(Trips, features = feature_set(tags = "autocorrelation"))

# Best practice is to use anonymous functions for additional arguments
tourism %>%
  features(Trips, list(~ quantile(., probs=seq(0,1,by=0.2))))

# Arguments can be passed to feature functions in two ways.
# Using `...` of `features()` passes the argument to every feature which
# accepts it. For example, `.period` sets the seasonal period used by
# `feat_stl()`, and can be specified with text like "1 year".
tourism %>%
  features(Trips, feat_stl, .period = "1 year")

# Alternatively, use a lambda function to pass arguments to a specific
# feature function.
tourism %>%
  features(Trips, ~ feat_stl(., .period = 4))
```

fitted.mbl_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 'mbl_df'
fitted(object, ...)

## S3 method for class 'mdl_ts'
fitted(object, h = 1, ...)
```

Arguments

object	A mable or time series model.
...	Other arguments passed to the model method for fitted()
h	The number of steps ahead that these fitted values are computed from.

forecast.mbl_df	<i>Produce forecasts</i>
-----------------	--------------------------

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

```
## S3 method for class 'mbl_df'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  ...
)

## S3 method for class 'mdl_ts'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  bias_adjust = NULL,
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  point_forecast = list(.mean = mean),
  ...
)

## S3 method for class 'mdl_ts_sim'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  times = NULL,
```

```
point_forecast = list(.mean = mean),
...
)
```

Arguments

object	The time series model used to produce the forecasts
new_data	A tsibble containing future information used to forecast.
h	The forecast horizon (can be used instead of new_data for regular time series with no exogenous regressors).
point_forecast	The point forecast measure(s) which should be returned in the resulting fable. Specified as a named list of functions which accept a distribution and return a vector. To compute forecast medians, you can use <code>list(.median = median)</code> .
...	Additional arguments for forecast model methods.
bias_adjust	Deprecated. Please use point_forecast to specify the desired point forecast method.
simulate	[Deprecated] Please use <code>simulate_iid()</code> to wrap the model instead.
bootstrap	[Deprecated] Please use <code>bootstrap_iid()</code> or <code>bootstrap_block()</code> to wrap the model instead.
times	The number of future paths for simulations if <code>simulate = TRUE</code> .

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 `unpack_hilo()`.

Value

A fable containing the following columns:

- `.model`: The name of the model used to obtain the forecast. Taken from the column names of models in the provided mable.
- The forecast distribution. The name of this column will be the same as the dependent variable in the model(s). If multiple dependent variables exist, it will be named `.distribution`.
- Point forecasts computed from the distribution using the functions in the `point_forecast` argument.
- All columns in `new_data`, excluding those whose names conflict with the above.

Examples

```
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.mbl_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 `stats::simulate()`.

Usage

```
## S3 method for class 'mbl_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,
  bootstrap = FALSE,
  bootstrap_block_size = 1,
  ...
)

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

## S3 method for class 'mdl_bootstrap_block'
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	[Deprecated] Call <code>set.seed()</code> before <code>generate()</code> instead.
...	Additional arguments for individual simulation methods.
bootstrap	[Deprecated] Please use <code>bootstrap_iid()</code> or <code>bootstrap_block()</code> to wrap the model instead.
bootstrap_block_size	[Deprecated] Please use <code>bootstrap_block()</code> 's <code>block_size</code> argument instead.

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

```
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)
```

glance.mbl_df	<i>Glance a mable</i>
---------------	-----------------------

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 'mbl_df'
glance(x, ...)

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

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

Arguments

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

Examples

```
library(fable)
library(tsibbledata)

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

hypothesize.mbl_df	<i>Run a hypothesis test from a mable</i>
--------------------	---

Description

This function will return the results of a hypothesis test for each model in the mable.

Usage

```
## S3 method for class 'mbl_df'
hypothesize(x, ...)

## S3 method for class 'mdl_ts'
hypothesize(x, tests = list(), ...)
```

Arguments

x	A mable.
...	Arguments for model methods.
tests	a list of test functions to perform on the model

Examples

```
library(fable)
library(tsibbledata)

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

interpolate.mbl_df	<i>Interpolate missing values</i>
--------------------	-----------------------------------

Description

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

Usage

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

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

Arguments

<code>object</code>	A mable containing a single model column.
<code>new_data</code>	A dataset with the same structure as the data used to fit the model.
<code>...</code>	Other arguments passed to interpolate methods.

Examples

```
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)
```

IRF

*Compute Impulse Response Function (IRF)***Description**

This function calculates the impulse response function (IRF) of a time series model. The IRF describes how a model's variables react to external shocks over time.

Usage

```
IRF(x, ...)
```

Arguments

<code>x</code>	A fitted model object, such as from a VAR or ARIMA model. This model is used to compute the impulse response.
<code>...</code>	Additional arguments to be passed to lower-level functions.

Details

If `new_data` contains the `.impulse` column, those values will be treated as impulses for the calculated impulse responses.

The impulse response function provides insight into the dynamic behaviour of a system in response to external shocks. It traces the effect of a one-unit change in the impulse variable on the response variable over a specified number of periods.

For a mable, the number of periods is specified using `h` (or by providing `new_data`). Model specific options are passed via `...`, for example `fable::VAR()` models accept `impulse` (the name of the variable to shock) and `orthogonal` (whether to compute orthogonalised impulse responses).

Examples

```
library(fable)
library(tsibble)

# Annual GDP growth and CPI inflation (%) for Australia
aus_economy <- tsibbledata::global_economy %>%
  dplyr::filter(Country == "Australia") %>%
  dplyr::transmute(Growth, Inflation = 100 * difference(log(CPI))) %>%
  dplyr::filter(!is.na(Growth), !is.na(Inflation))

fit <- aus_economy %>%
  model(VAR(vars(Growth, Inflation) ~ AR(1)))

# Response of both variables to a unit shock in GDP growth
fit %>%
  IRF(h = 10, impulse = "Growth")

# Orthogonalised response to a shock in inflation
fit %>%
  IRF(h = 10, impulse = "Inflation", orthogonal = TRUE)
```

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_dable	<i>Is the object a dable</i>
----------	------------------------------

Description

Is the object a dable

Usage

is_dable(x)

Arguments

x An object.

is_fable	<i>Is the object a fable</i>
----------	------------------------------

Description

Is the object a fable

Usage

is_fable(x)

Arguments

x An object.

is_mable	<i>Is the object a mable</i>
----------	------------------------------

Description

Is the object a mable

Usage

is_mable(x)

Arguments

x An object.

<code>is_model</code>	<i>Is the object a model</i>
-----------------------	------------------------------

Description

Is the object a model

Usage

```
is_model(x)
```

Arguments

<code>x</code>	An object.
----------------	------------

MAAPE	<i>Mean Arctangent Absolute Percentage Error</i>
-------	--

Description

Mean Arctangent Absolute Percentage Error

Usage

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

Arguments

<code>.resid</code>	A vector of residuals from either the training (model accuracy) or test (forecast accuracy) data.
<code>.actual</code>	A vector of responses matching the fitted values (for forecast accuracy, <code>new_data</code> must be provided).
<code>na.rm</code>	Remove the missing values before calculating the accuracy measure
<code>...</code>	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	<i>Create a new mable</i>
-------	---------------------------

Description

A mable (model table) data class (`mb1_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 (`mdl_lst`) describes how these models are related.

Usage

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

Arguments

<code>...</code>	A set of name-value pairs.
<code>key</code>	Structural variable(s) that identify each model.
<code>model</code>	Identifiers for the columns containing model(s).

mable_vars	<i>Return model column variables</i>
------------	--------------------------------------

Description

`mable_vars()` returns a character vector of the model variables in the object.

Usage

```
mable_vars(x)
```

Arguments

<code>x</code>	A dataset containing models (such as a mable).
----------------	--

MDA	<i>Directional accuracy measures</i>
-----	--------------------------------------

Description

A collection of accuracy measures based on the accuracy of the prediction’s direction (say, increasing or decreasing).

Usage

```
MDA(.resid, .actual, na.rm = TRUE, reward = 1, penalty = 0, ...)  
  
MDV(.resid, .actual, na.rm = TRUE, ...)  
  
MDPV(.resid, .actual, na.rm = TRUE, ...)  
  
directional_accuracy_measures
```

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
- reward, penalty The weights given to correct and incorrect predicted directions.
- ... Additional arguments for each measure.

Details

MDA(): Mean Directional Accuracy MDV(): Mean Directional Value MDPV(): Mean Directional Percentage Value

References

Blaskowitz and H. Herwartz (2011) "On economic evaluation of directional forecasts". *International Journal of Forecasting*, **27**(4), 1058-1065.

ME	<i>Point estimate accuracy measures</i>
----	---

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,  
  ...  
)  
  
RMSSE(  
  .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

<code>.resid</code>	A vector of residuals from either the training (model accuracy) or test (forecast accuracy) data.
<code>na.rm</code>	Remove the missing values before calculating the accuracy measure
<code>...</code>	Additional arguments for each measure.
<code>.actual</code>	A vector of responses matching the fitted values (for forecast accuracy, <code>new_data</code> must be provided).
<code>.train</code>	A vector of responses used to train the model (for forecast accuracy, the <code>orig_data</code> must be provided).
<code>demean</code>	Should the response be demeaned (MASE)
<code>.period</code>	The seasonal period of the data (defaulting to 'smallest' seasonal period). from a model, or forecasted values from the forecast.
<code>d</code>	Should the response model include a first difference?
<code>D</code>	Should the response model include a seasonal difference?
<code>na.action</code>	Function to handle missing values.

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

<code>.data</code>	A data structure suitable for the models (such as a <code>tsibble</code>)
<code>...</code>	Definitions for the models to be used. All models must share the same response variable.
<code>.safely</code>	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.

Progress

Progress on model estimation can be obtained by wrapping the code with `progressr::with_progress()`. Further customisation on how progress is reported can be controlled using the `progressr` package.

Examples

```
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)
progressr::with_progress(
  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

<code>model_rhs</code>	<i>Extract the right hand side of a model</i>
------------------------	---

Description

Extract the right hand side of a model

Usage

```
model_rhs(model)
```

Arguments

<code>model</code>	A formula
--------------------	-----------

<code>model_sum</code>	<i>Provide a succinct summary of a model</i>
------------------------	--

Description

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

Usage

```
model_sum(x)
```

Arguments

<code>x</code>	The model to summarise
----------------	------------------------

<code>new_model_class</code>	<i>Create a new class of models</i>
------------------------------	-------------------------------------

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, formula, ..., .env = caller_env(n = 2))
```

Arguments

model	The name of the model
train	A function that trains the model to a dataset. <code>.data</code> is a tsibble containing the data's index and response variables only. <code>formula</code> is the user's provided formula. <code>specials</code> is the evaluated specials used in the formula.
specials	Special functions produced using <code>new_specials()</code>
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. <code>...</code> is the arguments passed to <code>new_model_definition</code> , allowing you to perform different checks or training procedures according to different user inputs.
...	Further arguments to <code>R6::R6Class()</code> . This can be useful to set up additional elements used in the other functions. For example, to use <code>common_xregs</code> , an <code>origin</code> element in the model is used to store the origin for the <code>trend()</code> special. To use this special, you must add an <code>origin</code> element to the object (say with <code>origin = NULL</code>).
.env	The environment from which functions should inherit from.
.inherit	A model class to inherit from.
.class	A model class (typically created with <code>new_model_class()</code>).
formula	The user's model formula.

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	<i>Create evaluation environment for specials</i>
--------------	---

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	<i>Create a new modelling transformation</i>
--------------------	--

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
```

outliers	<i>Identify outliers</i>
----------	--------------------------

Description

Return a table of outlying observations using a fitted model.

Usage

```
outliers(object, ...)

## S3 method for class 'mbl_df'
outliers(object, ...)

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

Arguments

object	An object which can identify outliers.
...	Arguments for further methods.

Details

`outliers()` is a generic for identifying unusual observations in the data used to estimate a model. For a mable, each model's method is used to identify which observations are outliers, and the corresponding rows of the model's training data are returned as a tsibble (with a `.model` column identifying the model that detected them).

This functionality is experimental, and currently very few models provide an `outliers()` method (one example is the X-13ARIMA-SEATS decomposition from the `feasts` package). Model packages can support outlier detection by implementing an `outliers()` method for their model class, which should return the positions (or a logical vector) of the outlying observations.

A default model-based approach to outlier detection is planned, which will identify observations that are unusual relative to the model's one-step ahead fitted values (for example, residuals outside

of an interquartile range based threshold). Outlier detection is intended to be one part of a broader workflow for cleaning time series, where outliers are identified with `outliers()`, replaced with missing values, and then filled in using `interpolate()`. A higher level function combining these steps may be added once suitable defaults are better understood.

percentile_score	<i>Distribution accuracy measures</i>
------------------	---------------------------------------

Description

These accuracy measures can be used to evaluate how accurately a forecast distribution predicts a given actual value.

Usage

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

quantile_score(
  .dist,
  .actual,
  probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
  na.rm = TRUE,
  ...
)

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

distribution_accuracy_measures
```

Arguments

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

Quantile/percentile score (pinball loss)

A quantile (or percentile) score evaluates how accurately a set of quantiles (or percentiles) from the distribution match the given actual value. This score uses a pinball loss function, and can be calculated via the average of the score function given below:

The score function $s_p(q_p, y)$ is given by $(1 - p)(q_p - y)$ if $y < q_p$, and $p(y - q_p)$ if $y \geq q_p$. Where p is the quantile probability, $q_p = F^{-1}(p)$ is the quantile with probability p , and y is the actual value.

The resulting accuracy measure will average this score over all predicted points at all desired quantiles (defined via the `probs` argument).

The percentile score is uses the same method with `probs` set to all percentiles `probs = seq(0.01, 0.99, 0.01)`.

Continuous ranked probability score (CRPS)

The continuous ranked probability score (CRPS) is the continuous analogue of the pinball loss quantile score defined above. Its value is twice the integral of the quantile score over all possible quantiles:

$$CRPS(F, y) = 2 \int_0^1 s_p(q_p, y) dp$$

It can be computed directly from the distribution via:

$$CRPS(F, y) = \int_{-\infty}^{\infty} (F(x) - 1_{y \leq x})^2 dx$$

For some forecast distribution F and actual value y .

Calculating the CRPS accuracy measure is computationally difficult for many distributions, however it can be computed quickly and exactly for Normal and empirical (sample) distributions. For other distributions the CRPS is approximated using the quantile score of many quantiles (using the number of quantiles specified in the `n_quantiles` argument).

reconcile

Forecast reconciliation

Description

[Deprecated]

Usage

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

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

Arguments

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

Details

`reconcile()` is deprecated in favour of `dplyr::mutate()` with the `reconcile_*`() functions (`reconcile_bu()`, `reconcile_td()`, `reconcile_mo()` and `reconcile_mint()`).

See Also

[aggregate_key\(\)](#)

Forecasting: Principles and Practice - Forecasting hierarchical and grouped time series

Examples

```
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())) %>%
  dplyr::mutate(lm = reconcile_mint(lm)) %>%
  forecast()
```

reconcile_bu	<i>Bottom up forecast reconciliation</i>
--------------	--

Description

[Experimental]

Usage

```
reconcile_bu(models)
```

```
bottom_up(models)
```

Arguments

`models` A column of models in a mable.

Details

Reconciles a hierarchy using the bottom up reconciliation method. The response variable of the hierarchy must be aggregated using sums. The forecasted time points must match for all series in the hierarchy.

`reconcile_bu()` is a short alias for `bottom_up()`.

See Also

`reconcile_td()`, `reconcile_mo()`, `reconcile_mint()`, `aggregate_key()`

Forecasting: Principles and Practice - Bottom-up approach

reconcile_mint

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. The forecasted time points must match for all series in the hierarchy (caution: this is not yet tested for beyond the series length).

Usage

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

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, which determines how the base forecast error covariance matrix is estimated.
sparse	If TRUE, the reconciliation will be computed using sparse matrix algebra? By default, sparse matrices will be used if the Matrix package is installed.

Details

The "wls_var", "mint_cov" and "mint_shrink" methods require a positive definite covariance matrix, which is not the case if two series in the hierarchy are identical (e.g. an aggregate with only one child). Use "wls_struct" or "ols" in these situations.

`reconcile_mint()` is a short alias for `min_trace()`.

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_bu\(\)](#), [reconcile_td\(\)](#), [reconcile_mo\(\)](#), [aggregate_key\(\)](#)

Forecasting: Principles and Practice - Forecast reconciliation

reconcile_mo	<i>Middle out forecast reconciliation</i>
--------------	---

Description

[Experimental]

Usage

```
reconcile_mo(models, split = 1)
```

```
middle_out(models, split = 1)
```

Arguments

models	A column of models in a mable.
split	The middle level of the hierarchy from which the bottom-up and top-down approaches are used above and below respectively.

Details

Reconciles a hierarchy using the middle out reconciliation method. The response variable of the hierarchy must be aggregated using sums. The forecasted time points must match for all series in the hierarchy.

`reconcile_mo()` is a short alias for `middle_out()`.

See Also

[reconcile_bu\(\)](#), [reconcile_td\(\)](#), [reconcile_mint\(\)](#), [aggregate_key\(\)](#) *Forecasting: Principles and Practice* - Middle-out approach

reconcile_td	<i>Top down forecast reconciliation</i>
--------------	---

Description**[Experimental]****Usage**

```
reconcile_td(
  models,
  method = c("forecast_proportions", "average_proportions", "proportion_averages")
)

top_down(
  models,
  method = c("forecast_proportions", "average_proportions", "proportion_averages")
)
```

Arguments

models	A column of models in a mable.
method	The method used to compute the disaggregation proportions.

Details

Reconciles a hierarchy using the top down reconciliation method. The response variable of the hierarchy must be aggregated using sums. The forecasted time points must match for all series in the hierarchy.

reconcile_td() is a short alias for top_down().

See Also

[reconcile_bu\(\)](#), [reconcile_mo\(\)](#), [reconcile_mint\(\)](#), [aggregate_key\(\)](#)

Forecasting: Principles and Practice - Top-down approaches

refit.mbl_df	<i>Refit a mable to a new dataset</i>
--------------	---------------------------------------

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 'mbl_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

```
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	<i>Register a feature function</i>
------------------	------------------------------------

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.mbl_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 'mbl_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	<i>Extract the response variable from a model</i>
----------	---

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

response_vars	<i>Return response variables</i>
---------------	----------------------------------

Description

response_vars() returns a character vector of the response variables in the object.

Usage

```
response_vars(x)
```

Arguments

x	A dataset containing a response variable (such as a mable, fable, or dable).
---	--

scenarios	<i>A set of future scenarios for forecasting</i>
-----------	--

Description

A set of future scenarios for forecasting

Usage

```
scenarios(..., names_to = ".scenario")
```

Arguments

...	Input data for each scenario
names_to	The column name used to identify each scenario

simulate_iid	<i>Simulate forecast sample paths</i>
--------------	---------------------------------------

Description**[Experimental]**

Wraps a fitted model (or a model column from a mable) so that `forecast()` produces its distribution from simulated sample paths (via `generate()`) instead of closed-form/analytical results. Innovations are drawn independently at each time point, parametrically from the model's assumed error distribution, identically to calling `generate()` on the model directly. This is primarily useful for models (or transformations) whose forecast distribution has no closed form, and for producing forecasts that are directly comparable to `bootstrap_iid()/bootstrap_block()` (which only differ in how innovations are drawn).

Usage

```
simulate_iid(model, times = 5000)
```

Arguments

model	A fitted model, typically a model column from a mable modified with <code>dplyr::mutate()</code> (e.g. <code>mutate(ets = simulate_iid(ets))</code>). Alternatively, an unfitted model specification can be wrapped directly inside <code>model()</code> (e.g. <code>model(ets = simulate_iid(ETS(Trips)))</code>), in which case each series is fitted and simulated independently, identically to wrapping each series' <code>mdl_ts</code> individually.
times	The number of simulated sample paths to generate.

See Also

`bootstrap_iid()`, `bootstrap_block()`, `forecast()`, `generate()`

Examples

```
library(fable)
library(dplyr)
aus_production %>%
  model(ets = ETS(Beer)) %>%
  mutate(ets = simulate_iid(ets, times = 1000)) %>%
  forecast()

# Equivalently, applied to the model specification before fitting:
aus_production %>%
  model(ets = simulate_iid(ETS(Beer), times = 1000)) %>%
  forecast()
```

skill_score	<i>Forecast skill score measure</i>
-------------	-------------------------------------

Description

This function converts other error metrics such as MSE into a skill score. The reference or benchmark forecasting method is the Naive method for non-seasonal data, and the seasonal naive method for seasonal data. When used within [accuracy.fbl_df](#), it is important that the data contains both the training and test data, as the training data is used to compute the benchmark forecasts.

Usage

```
skill_score(measure)
```

Arguments

measure The accuracy measure to use in computing the skill score.

Examples

```
skill_score(MSE)

library(fable)
library(tsibble)

lung_deaths <- as_tsibble(cbind(mdeaths, fdeaths))
lung_deaths %>%
  dplyr::filter(index < yearmonth("1979 Jan")) %>%
  model(
    ets = ETS(value ~ error("M") + trend("A") + season("A")),
    lm = TSLM(value ~ trend() + season())
  ) %>%
  forecast(h = "1 year") %>%
  accuracy(lung_deaths, measures = list(skill = skill_score(MSE)))
```

special_xreg	<i>Helper special for producing a model matrix of exogenous regressors</i>
--------------	--

Description

Helper special for producing a model matrix of exogenous regressors

Usage

```
special_xreg(...)
```

Arguments

... Arguments for `fable_xreg_matrix` (see Details)

Details

Currently the `fable_xreg_matrix` helper supports a single argument named `default_intercept`. If this argument is `TRUE` (passed via ... above), then the intercept will be returned in the matrix if not specified (much like the behaviour of `lm()`). If `FALSE`, then the intercept will only be included if explicitly requested via `1` in the formula.

stream	<i>Extend a fitted model with new data</i>
--------	--

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 'mbl_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.mbl_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 'mbl_df'
tidy(x, ...)

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

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

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

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

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

Arguments

<code>x, object</code>	A mable.
<code>...</code>	Arguments for model methods.

Examples

```
library(fable)
library(tsibbledata)

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

winkler_score	<i>Interval estimate accuracy measures</i>
---------------	--

Description

`coverage()` computes the proportion of actual values that fall within the central `level%` interval of the distribution (bounds inclusive). It is not included in `interval_accuracy_measures` by default, but can be added via `measures = list(coverage = coverage)` in [accuracy\(\)](#).

Usage

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

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

```
scaled_pinball_loss(
  .dist,
  .actual,
  .train,
  level = 95,
  na.rm = TRUE,
  demean = FALSE,
  .period,
  d = .period == 1,
  D = .period > 1,
  ...
)
```

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

```
interval_accuracy_measures
```

Arguments

<code>.dist</code>	The distribution of fitted values from the model, or forecasted values from the forecast.
<code>.actual</code>	A vector of responses matching the fitted values (for forecast accuracy, <code>new_data</code> must be provided).
<code>level</code>	The level of the forecast interval.
<code>na.rm</code>	Remove the missing values before calculating the accuracy measure
<code>...</code>	Additional arguments for each measure.
<code>.train</code>	A vector of responses used to train the model (for forecast accuracy, the <code>orig_data</code> must be provided).
<code>demean</code>	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?

Index

* package

fabletools-package, 3

accuracy(), 61
accuracy.fbl_df, 58
accuracy.fbl_df (accuracy.mbl_df), 4
accuracy.mbl_df, 4
accuracy.mdl_lst (accuracy.mbl_df), 4
accuracy.mdl_ts (accuracy.mbl_df), 4
ACF1 (ME), 41
agg_vec, 6
aggregate_key, 6, 36
aggregate_key(), 6, 13, 16, 17, 50–53
as_dable, 7
as_fable, 8
as_mable, 9
augment.mbl_df, 9
augment.mdl_lst (augment.mbl_df), 9
augment.mdl_ts (augment.mbl_df), 9
autolayer.fbl_ts (autoplot.fbl_ts), 11
autolayer.tbl_ts (autoplot.tbl_ts), 12
autoplot(), 22
autoplot.dcmp_ts, 10
autoplot.fbl_ts, 11
autoplot.tbl_ts, 12

base::formals(), 27
base::mean(), 27
bootstrap_block (bootstrap_iid), 13
bootstrap_block(), 30, 32, 57
bootstrap_iid, 13
bootstrap_iid(), 13, 23, 30, 32, 57
bottom_up (reconcile_bu), 50
box_cox, 14

coef.mbl_df (tidy.mbl_df), 60
coef.mdl_lst (tidy.mbl_df), 60
coef.mdl_ts (tidy.mbl_df), 60
coherent_cmat, 15
coherent_cmat(), 17
coherent_smat, 16
coherent_smat(), 15, 16
combination_ensemble, 17
combination_ensemble(), 19
combination_model, 18
combination_weighted, 19
combination_weighted(), 18
common_periods, 19
common_xregs, 20
components(), 24
components.mbl_df, 21
components.mdl_ts (components.mbl_df), 21
conformal_scp, 22
coverage (winkler_score), 61
CRPS (percentile_score), 48

dable, 23
decomposition_model, 24
directional_accuracy_measures (MDA), 40
distribution_accuracy_measures, 5
distribution_accuracy_measures
 (percentile_score), 48
distribution_var, 25
dplyr::mutate(), 13, 50, 57

estimate, 25

fable, 26
fable::VAR(), 35
fabletools (fabletools-package), 3
fabletools-package, 3
feature_set, 26
feature_set(), 27, 54
features, 27
Features by package, 26, 27
Features by tag, 26, 27
features_all (features), 27
features_at (features), 27
features_if (features), 27

- fitted.mbl_df, 28
- fitted.mdl_ts(fitted.mbl_df), 28
- forecast(), 13, 14, 23, 57
- forecast.mbl_df, 29
- forecast.mdl_ts(forecast.mbl_df), 29
- forecast.mdl_ts_sim(forecast.mbl_df), 29
- fourier(common_xregs), 20
- future::plan(), 43
- generate(), 13, 14, 57
- generate.mbl_df, 31
- generate.mdl_bootstrap_block
(generate.mbl_df), 31
- generate.mdl_bootstrap_iid
(generate.mbl_df), 31
- generate.mdl_ts(generate.mbl_df), 31
- get_frequencies(common_periods), 19
- ggplot2::geom_line(), 10, 12
- ggplot2::vars(), 12
- glance.mbl_df, 33
- glance.mdl_lst(glance.mbl_df), 33
- glance.mdl_ts(glance.mbl_df), 33
- hfitted(fitted.mbl_df), 28
- hilo(), 30
- hypothesize.mbl_df, 34
- hypothesize.mdl_ts
(hypothesize.mbl_df), 34
- interpolate(), 48
- interpolate.mbl_df, 34
- interpolate.mdl_ts
(interpolate.mbl_df), 34
- interval_accuracy_measures, 5
- interval_accuracy_measures
(winkler_score), 61
- inv_box_cox(box_cox), 14
- invert_transformation
(new_transformation), 46
- IRF, 35
- is_aggregated, 36
- is_aggregated(), 7
- is_dable, 37
- is_fable, 37
- is_mable, 37
- is_model, 38
- MAAPE, 38
- mable, 39
- mable_vars, 39
- MAE (ME), 41
- MAPE (ME), 41
- MASE (ME), 41
- MDA, 40
- MDPV (MDA), 40
- MDV (MDA), 40
- ME, 41
- middle_out(reconcile_mo), 52
- min_trace(reconcile_mint), 51
- min_trace(), 17
- model, 42
- model(), 13, 57
- model_lhs, 43
- model_rhs, 44
- model_sum, 44
- MPE (ME), 41
- MSE (ME), 41
- new_model_class, 44
- new_model_class(), 45
- new_model_definition(new_model_class), 44
- new_specials, 46
- new_specials(), 45
- new_transformation, 46
- NULL model, 42
- outliers, 47
- percentile_score, 48
- pinball_loss(winkler_score), 61
- point_accuracy_measures, 5
- point_accuracy_measures (ME), 41
- quantile_score(percentile_score), 48
- R6::R6Class(), 45
- reconcile, 49
- reconcile_bu, 50
- reconcile_bu(), 7, 50, 52, 53
- reconcile_mint, 51
- reconcile_mint(), 7, 13, 50–53
- reconcile_mo, 52
- reconcile_mo(), 7, 50–53
- reconcile_td, 53
- reconcile_td(), 7, 50–52
- refit(), 23

refit.mbl_df, [53](#)
refit.mdl_ts(refit.mbl_df), [53](#)
register_feature, [54](#)
register_feature(), [26](#)
report, [55](#)
report(), [30](#)
residuals.mbl_df, [55](#)
residuals.mdl_ts(residuals.mbl_df), [55](#)
response, [56](#)
response_vars, [56](#)
RMSE (ME), [41](#)
RMSSE (ME), [41](#)

scaled_pinball_loss(winkler_score), [61](#)
scenarios, [57](#)
season(common_xregs), [20](#)
set.seed(), [32](#)
simulate_iid, [57](#)
simulate_iid(), [14](#), [23](#), [30](#)
skill_score, [58](#)
special_xreg, [59](#)
stats::simulate(), [31](#)
stats::var(), [27](#)
stream, [59](#)

tidy.mbl_df, [60](#)
tidy.mdl_lst(tidy.mbl_df), [60](#)
tidy.mdl_ts(tidy.mbl_df), [60](#)
top_down(reconcile_td), [53](#)
trend(common_xregs), [20](#)
tsibble::tsibble(), [23](#), [26](#)

unpack_hilo(), [30](#)

winkler_score, [61](#)