Package ‘fabletools’

January 19, 2021

Title Core Tools for Packages in the 'fable' Framework

Version 0.3.0

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,
progressr,
lifecycle

Suggests covr,
crayon,
fable (>= 0.2.0),
future.apply,
knitr,
pillar (>= 1.0.1),
feasts (>= 0.1.2),
rmarkdown,
spelling,
testthat,
tsibbledata (>= 0.2.0),

lubridate,
Matrix

VignetteBuilder knitr

RdMacros lifecycle

ByteCompile true

Encoding UTF-8

Language en-GB

LazyData true

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

RoxygenNote 7.1.1

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]
- 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` *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_index *Expand a dataset to include temporal aggregates*

Description

Experimental

Usage

```
aggregate_index(.data, .window, ..., .offset = "end", .bin_size = NULL)
```

Arguments

- `.data` A tsibble.
- `.window` Temporal aggregations to include. The default (NULL) will automatically identify appropriate temporal aggregations. This can be specified in several ways (see details).
- `...` <data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result.

  The value can be:
  - A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`.
  - A vector of length `n`, e.g. `quantile()`.
  - A data frame, to add multiple columns from a single expression.
- `.offset` Offset the temporal aggregation windows to align with the start or end of the data. If FALSE, no offset will be applied (giving common breakpoints for temporal bins.)
- `.bin_size` Temporary. Define the number of observations in each temporal bucket

Details

This feature is very experimental. It currently allows for temporal aggregation of daily data as a proof of concept.

The aggregation `.window` can be specified in several ways:

- A character string, containing one of "day", "week", "month", "quarter" or "year". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s".
- A number, taken to be in days.
- A `difftime` object.

Examples

```
library(tsibble)
pedestrian %>%
  # Currently only supports daily data
  index_by(Date) %>%
  dplyr::summarise(Count = sum(Count)) %>%
  # Compute weekly aggregates
  fabletools:::aggregate_index("1 week", Count = sum(Count))
```

---

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.

...
: <data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result.

  The value can be:

  - A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`.
  - A vector of length `n`, e.g. `quantile()`.
  - A data frame, 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()`, `is_aggregated()`

Examples

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

---

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)
)
```

`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 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` *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, ...)

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

Arguments

`x`	Object to be coerced to a fable (`fbl_ts`)
`...`	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` *Coerce a dataset to a mable*

Description

Coerce a dataset to a mable

Usage

```
as_mable(x, ...)

## S3 method for class 'data.frame'
as_mable(x, key = NULL, model = NULL, ...)
```

Arguments

`x`	A dataset containing a list model column.
`...`	Additional arguments passed to other methods.
`key`	Structural variable(s) that identify each model.
`model`	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, type = NULL, ...)
```

Arguments

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

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()
}
```

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, 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.
`scale_bars`	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 `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),
  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 size = 3.
point_forecast	The point forecast measure to be displayed in the plot.

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)
```

bottom_up — *Bottom up forecast reconciliation*

Description

Experimental

Usage

```
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.

See Also

reconcile(), aggregate_key()

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

```
## Not run:
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()
}

## End(Not run)
```

`construct_fc` *Construct a new set of forecasts*

Description

Deprecated

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

This function is deprecated. `forecast()` methods for a model should return a vector of distributions using the distributional package.

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 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

`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)
}
```

`distribution_var` *Return distribution variable*

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` *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 also contain a single distribution column that uses values from the distributional package.

Usage

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

Arguments

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

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 should 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

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()`.
if(requireNamespace("feasts")) 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))))
```

`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, 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 — *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,
  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),
  ...
)
```

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).
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)`.
bias_adjust	Deprecated. Please use `point_forecast` to specify the desired point forecast method.
simulate	Should forecasts be based on simulated future paths instead of analytical results.
bootstrap	Should innovations from simulated forecasts be bootstrapped from the model's fitted residuals. This allows the forecast distribution to have a different underlying shape which could better represent the nature of your data.
times	The number of future paths for simulations if `simulate = TRUE`.

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

```
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,
  bootstrap = FALSE,
  bootstrap_block_size = 1,
  ...
)
```

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.
`bootstrap`	If TRUE, then forecast distributions are computed using simulation with resampled errors.
`bootstrap_block_size`	The bootstrap block size specifies the number of contiguous residuals to be taken in each bootstrap sample.

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)
}
```

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()
}
```

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_dable` *Is the object a dable*

Description

Is the object a dable

Usage

```
is_dable(x)
```

Arguments

`x`	An object.

`is_fable` *Is the object a fable*

Description

Is the object a fable

Usage

```
is_fable(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, model = NULL)
```

Arguments

`...`	<dynamic-dots> A set of name-value pairs. These arguments are processed with `rlang::quos()` and support unquote via `!!` and unquote-splice via `!!!`. Use := to create columns that start with a dot. Arguments are evaluated sequentially. You can refer to previously created elements directly or using the .data pronoun. An existing `.data` pronoun, provided e.g. inside `dplyr::mutate()`, is not available.
`key`	Structural variable(s) that identify each model.
`model`	Identifiers for the columns containing model(s).

mable_vars — *Return model column variables*

Description

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

Usage

```
mable_vars(x)
```

Arguments

`x`	A dataset containing models (such as a mable).

`MDA` *Directional accuracy measures*

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.

Format

An object of class `list` of length 3.

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 *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,
  ...
)

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

`.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 8.

`middle_out` *Middle out forecast reconciliation*

Description

Experimental

Usage

```
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.

See Also

`reconcile()`, `aggregate_key()` *Forecasting: Principles and Practice* - Middle-out approach `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. The forecasted time points must match for all series in the hierarchy (caution: this is not yet tested for beyond the series length).

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. All models must share the same response variable.
`.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.

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

```
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)
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

`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_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, formula, ..., .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()`).
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` *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, ...)

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

.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.
probs	A vector of probabilities at which the metric is evaluated.
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

response_vars — *Return response variables*

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 — *A set of future scenarios for forecasting*

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

skill_score *Forecast skill score measure*

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_ts, 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)

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

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()
}
```

top_down — *Top down forecast reconciliation*

Description

Experimental

Usage

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

Arguments

models	A column of models in a mable.
method	The reconciliation method to use.

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.

See Also

reconcile(), aggregate_key()

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

`unpack_hilo` *Unpack a hilo column*

Description

Allows a hilo column to be unpacked into its component columns: "lower", "upper", and "level".

Usage

```
unpack_hilo(data, cols, names_sep = "_", names_repair = "check_unique")
```

Arguments

`data` A data frame.

`cols` Name of hilo columns to unpack.

`names_sep` If `NULL`, the default, the names will be left as is. In `pack()`, inner names will come from the former outer names; in `unpack()`, the new outer names will come from the inner names.

If a string, the inner and outer names will be used together. In `pack()`, the names of the new outer columns will be formed by pasting together the outer and the inner column names, separated by `names_sep`. In `unpack()`, the new inner names will have the outer names (+ `names_sep`) automatically stripped. This makes `names_sep` roughly symmetric between packing and unpacking.

`names_repair` Used to check that output data frame has valid names. Must be one of the following options:

- "minimal": no name repair or checks, beyond basic existence,
- "unique": make sure names are unique and not empty,
- "check_unique": (the default), no name repair, but check they are unique,
- "universal": make the names unique and syntactic
- a function: apply custom name repair.
- tidyr_legacy: use the name repair from tidyr 0.8.
- a formula: a purrr-style anonymous function (see `rlang::as_function()`)

See `vctrs::vec_as_names()` for more details on these terms and the strategies used to enforce them.

See Also

`tidyr::unpack()`

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, ...)

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,
  ...
)

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.
`.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?

Format

An object of class `list` of length 1.

Index