Package ‘fable’

June 16, 2020

Title  Forecasting Models for Tidy Time Series
Version 0.2.1
Description Provides a collection of commonly used univariate and multivariate
time series forecasting models including automatically selected exponential
smoothing (ETS) and autoregressive integrated moving average (ARIMA) models.
These models work within the 'fable' framework provided by the 'fabletools'
package, which provides the tools to evaluate, visualise, and combine models
in a workflow consistent with the tidyverse.
License GPL-3
BugReports https://github.com/tidyverts/fable/issues
Depends R (>= 3.4.0), fabletools (>= 0.2.0)
Imports Rcpp (>= 0.11.0), rlang (>= 0.4.6), stats, dplyr (>= 1.0.0),
tibble (>= 0.9.0), tibble, tidyr, utils, distributional
Suggests covr, feasts, forecast, knitr, nnet, rmarkdown, spelling,
testthat, tsibbledata (>= 0.2.0)
LinkingTo Rcpp (>= 0.11.0)
VignetteBuilder knitr
ByteCompile true
Encoding UTF-8
Language en-GB
LazyData true
RoxygenNote 7.1.0.9000
NeedsCompilation yes
Author Mitchell O'Hara-Wild [aut, cre],
Rob Hyndman [aut],
Earo Wang [aut],
Gabriel Caceres [ctb] (NNETAR implementation)
Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>
Repository CRAN
Date/Publication 2020-06-16 16:30:24 UTC
R topics documented:

AR ................................................................. 3
ARIMA .......................................................... 5
common_xregs ................................................. 7
components.ETS .............................................. 8
CROSTON ....................................................... 9
ETS .............................................................. 10
fitted.AR ........................................................ 13
fitted.ARIMA ................................................... 14
fitted.croston .................................................. 14
fitted.ETS ....................................................... 15
fitted.fable_theta ............................................. 16
fitted.model_mean ............................................. 16
fitted.NNETAR ................................................ 17
fitted.RW ....................................................... 18
fitted.TSLM ..................................................... 18
fitted.VAR ..................................................... 19
forecast.AR ..................................................... 20
forecast.ARIMA ............................................... 21
forecast.croston ............................................. 22
forecast.ETS ................................................... 22
forecast.fable_theta ........................................ 23
forecast.model_mean ....................................... 24
forecast.NNETAR ............................................ 25
forecast.RW ................................................... 26
forecast.TSLM ................................................ 28
forecast.VAR ................................................ 29
generate.ARIMA .............................................. 30
generate.ETS ................................................ 30
generate.model_mean ....................................... 31
generate.NNETAR .......................................... 32
generate.RW ................................................ 33
generate.TSLM .............................................. 34
glance.AR ..................................................... 34
glance.ARIMA ............................................... 35
glance.ETS ................................................... 36
glance.fable_theta .......................................... 37
glance.model_mean ......................................... 37
glance.NNETAR ............................................ 38
glance.RW ................................................... 39
glance.TSLM ................................................ 39
glance.VAR .................................................. 40
interpolate.ARIMA .......................................... 41
interpolate.model_mean ...................................... 42
interpolate.TSLM ............................................ 42
MEAN .......................................................... 43
NNETAR ........................................................ 44
**AR**

Estimate a AR model

**Description**

Searches through the vector of lag orders to find the best AR model which has lowest AIC, AICc or BIC value. It is implemented using OLS, and behaves comparably to `stats::ar.ols()`.

**Usage**

```r
AR(formula, ic = c("aicc", "aic", "bic"), ...)
```

**Arguments**

- `formula`: Model specification (see "Specials" section).
- `ic`: The information criterion used in selecting the model.
- `...`: Further arguments for `arima`
Details

Exogenous regressors and `common_xregs` can be specified in the model formula.

Value

A model specification.

Specials

**pdq**: The order special is used to specify the lag order for the auto-regression.

```r
order(p = 0:15, fixed = list())
```

- **p**: The order of the auto-regressive (AR) terms. If multiple values are provided, the one which minimises `ic` will be chosen.
- **fixed**: A named list of fixed parameters for coefficients. The names identify the coefficient, beginning with `ar`, and then followed by the lag order. For example,

```r
fixed = list(ar1 = 0.3, ar3 = 0)
```

**xreg**: Exogenous regressors can be included in an ARIMA model without explicitly using the `xreg()` special. Common exogenous regressor specials as specified in `common_xregs` can also be used. These regressors are handled using `stats::model.frame()`, and so interactions and other functionality behaves similarly to `stats::lm()`. The inclusion of a constant in the model follows the similar rules to `stats::lm()`, where including 1 will add a constant and 0 or -1 will remove the constant. If left out, the inclusion of a constant will be determined by minimising `ic`.

```r
xreg(..., fixed = list())
```

- **...**: Bare expressions for the exogenous regressors (such as `log(x)`)
- **fixed**: A named list of fixed parameters for coefficients. The names identify the coefficient, and should match the name of the regressor. For example,

```r
fixed = list(constant = 20)
```

See Also

*Forecasting: Principles and Practices, Vector autoregressions (section 11.2)*

Examples

```r
luteinizing_hormones <- as_tsibble(lh)
fit <- luteinizing_hormones %>%
  model(AR(value ~ order(3)))
report(fit)

fit %>%
  forecast() %>%
  autoplot(luteinizing_hormones)
```
ARIMA

Estimate an ARIMA model

Description

Searches through the model space specified in the specials to identify the best ARIMA model which has lowest AIC, AICc or BIC value. It is implemented using stats::arima() and allows ARIMA models to be used in the fable framework.

Usage

ARIMA(
  formula,
  ic = c("aicc", "aic", "bic"),
  stepwise = TRUE,
  greedy = TRUE,
  approximation = NULL,
  order_constraint = p + q + P + Q <= 6 & (!constant | d + D < 2),
  unitroot_spec = unitroot_options(),
  ...
)

Arguments

formula Model specification (see "Specials" section).
ic The information criterion used in selecting the model.
stepwise Should stepwise be used?
greedy Should the stepwise search move to the next best option immediately?
approximation Should CSS (conditional sum of squares) be used during model selection? The default (NULL) will use the approximation if there are more than 150 observations or if the seasonal period is greater than 12.
order_constraint A logical predicate on the orders of p, d, q, P, D, Q and constant to consider in the search. See "Specials" for the meaning of these terms.
unitroot_spec A specification of unit root tests to use in the selection of d and D. See unitroot_options() for more details.
...

Value

A model specification.
Parameterisation

The fable ARIMA() function uses an alternate parameterisation of constants to \texttt{stats::arima()} and \texttt{forecast::Arima()}. While the parameterisations are equivalent, the coefficients for the constant/mean will differ.

In fable, the parameterisation used is:

\[(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t\]

In stats and forecast, an ARIMA model is parameterised as:

\[(1 - \phi_1 B - \cdots - \phi_p B^p)(y_t' - \mu) = (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t\]

where \(\mu\) is the mean of \((1 - B)^d y_t\) and \(c = \mu(1 - \phi_1 - \cdots - \phi_p)\).

Specials

The \textit{specials} define the space over which ARIMA will search for the model that best fits the data. If the RHS of \texttt{formula} is left blank, the default search space is given by \texttt{pdq()} + \texttt{PDQ()}: that is, a model with candidate seasonal and nonseasonal terms, but no exogenous regressors. Note that a seasonal model requires at least 2 full seasons’ worth of data; if this is not available, ARIMA will revert to a nonseasonal model with a warning.

To specify a model fully (avoid automatic selection), the intercept and \texttt{pdq()}/\texttt{PDQ()} values must be specified: for example \texttt{formula = response ~ 1 + pdq(1,1,1) + PDQ(1,0,0)}.

\textbf{pdq:} The \texttt{pdq} special is used to specify non-seasonal components of the model.

\begin{verbatim}
pdq(p = 0:5, d = 0:2, q = 0:5, 
   p_init = 2, q_init = 2, fixed = list())
\end{verbatim}

\textbf{PDQ:} The \texttt{PDQ} special is used to specify seasonal components of the model. To force a nonseasonal fit, specify \texttt{PDQ(0,0,0)} in the RHS of the model formula. Note that simply omitting \texttt{PDQ} from the formula will \textit{not} result in a nonseasonal fit.

\begin{verbatim}
PDQ(P = 0:2, D = 0:1, Q = 0:2, period = NULL,
   P_init = 1, Q_init = 1, fixed = list())
\end{verbatim}
Q_init: If stepwise = TRUE, Q_init provides the initial value for Q for the stepwise search procedure.

fixed: A named list of fixed parameters for coefficients. The names identify the coefficient, beginning with either sar or sma, and then followed by the lag order. For example, fixed = list(sar1 = 0.1).

xreg: Exogenous regressors can be included in an ARIMA model without explicitly using the xreg() special. Common exogenous regressor specials as specified in common_xregs can also be used. These regressors are handled using stats::model.frame(), and so interactions and other functionality behaves similarly to stats::lm().

The inclusion of a constant in the model follows the similar rules to stats::lm(), where including 1 will add a constant and 0 or -1 will remove the constant. If left out, the inclusion of a constant will be determined by minimising ic.

xreg(..., fixed = list())

... Bare expressions for the exogenous regressors (such as log(x))

fixed: A named list of fixed parameters for coefficients. The names identify the coefficient, and should match the name of the regressor. For example, fixed = list(constant = 20).

See Also

Forecasting: Principles and Practices, ARIMA models (chapter 9) Forecasting: Principles and Practices, Dynamic regression models (chapter 10)

Examples

# Manual ARIMA specification
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ 0 + pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  report()

# Automatic ARIMA specification
library(tsibble)
library(dplyr)
tsibblesdata::global_economy %>%
  filter(Country == "Australia") %>%
  model(ARIMA(log(GDP) ~ Population))
Specials

**trend**: The trend special includes common linear trend regressors in the model. It also supports piecewise linear trend via the `knots` argument.

```r
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 to act as the starting time for the trend.

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

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

- `period`: The periodic nature of the seasonality. This can be either a number indicating the number of observations in each seasonal period, or text to indicate the duration of the seasonal window (for example, annual seasonality would be “1 year”).

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

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

- `period`: The periodic nature of the seasonality. This can be either a number indicating the number of observations in each seasonal period, or text to indicate the duration of the seasonal window (for example, annual seasonality would be “1 year”).
- `K`: The maximum order of the fourier terms.
- `origin`: An optional time value to act as the starting time for the fourier series.

---

`components.ETS`  
*Extract estimated states from an ETS model.*

**Description**

Extract estimated states from an ETS model.

**Usage**

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

**Arguments**

- `object`: An estimated model.
- `...`: Unused.

**Value**

A `fabletools::dable()` containing estimated states.
Examples

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
  components()
```

Description

Based on Croston’s (1972) method for intermittent demand forecasting, also described in Shenstone and Hyndman (2005). Croston’s method involves using simple exponential smoothing (SES) on the non-zero elements of the time series and a separate application of SES to the times between non-zero elements of the time series.

Usage

```r
CROSTON(
  formula,
  opt_crit = c("mse", "mae"),
  type = c("croston", "sba", "sbj"),
  ...
)
```

Arguments

- **formula**: Model specification (see "Specials" section).
- **opt_crit**: The optimisation criterion used to optimise the parameters.
- **type**: Which variant of Croston’s method to use. Defaults to "croston" for Croston’s method, but can also be set to "sba" for the Syntetos-Boylan approximation, and "sbj" for the Shale-Boylan-Johnston method.
- **...**: Not used.

Details

Note that forecast distributions are not computed as Croston’s method has no underlying stochastic model. In a later update, we plan to support distributions via the equivalent stochastic models that underly Croston’s method (Shenstone and Hyndman, 2005)

There are two variant methods available which apply multiplicative correction factors to the forecasts that result from the original Croston’s method. For the Syntetos-Boylan approximation (type = "sba"), this factor is $1 - \alpha/2$, and for the Shale-Boylan-Johnston method (type = "sbj"), this factor is $1 - \alpha/(2 - \alpha)$, where $\alpha$ is the smoothing parameter for the interval SES application.

Value

A model specification.
Specials

**demand**: The demand special specifies parameters for the demand SES application.

demand(initial = NULL, param = NULL, param_range = c(0, 1))

*initial* The initial value for the demand application of SES.

*param* The smoothing parameter for the demand application of SES.

*param_range* If *param = NULL*, the range of values over which to search for the smoothing parameter.

**interval**: The interval special specifies parameters for the interval SES application.

interval(initial = NULL, param = NULL, param_range = c(0, 1))

*initial* The initial value for the interval application of SES.

*param* The smoothing parameter for the interval application of SES.

*param_range* If *param = NULL*, the range of values over which to search for the smoothing parameter.

References


Examples

```r
library(tsibble)
sim_poisson <- tsibble(
  time = yearmonth("2012 Dec") + seq_len(24),
  count = rpois(24, lambda = 0.3),
  index = time
)

sim_poisson %>%
  autoplot(count)

sim_poisson %>%
  model(CROSTON(count)) %>%
  forecast(h = "2 years") %>%
  autoplot(sim_poisson)
```

ETS Exponential smoothing state space model
**ETS**

**Description**

Returns ETS model specified by the formula.

**Usage**

```r
ETS(
  formula,
  opt_crit = c("lik", "amse", "mse", "sigma", "mae"),
  nmse = 3,
  bounds = c("both", "usual", "admissible"),
  ic = c("aicc", "aic", "bic"),
  restrict = TRUE,
  ...
)
```

**Arguments**

- **formula** Model specification (see "Specials" section).
- **opt_crit** The optimization criterion. Defaults to the log-likelihood "lik", but can also be set to "mse" (Mean Square Error), "amse" (Average MSE over first nmse forecast horizons), "sigma" (Standard deviation of residuals), or "mae" (Mean Absolute Error).
- **nmse** If opt_crit == "amse", nmse provides the number of steps for average multi-step MSE (1<=nmse<=30).
- **bounds** Type of parameter space to impose: "usual" indicates all parameters must lie between specified lower and upper bounds; "admissible" indicates parameters must lie in the admissible space; "both" (default) takes the intersection of these regions.
- **ic** The information criterion used in selecting the model.
- **restrict** If TRUE (default), the models with infinite variance will not be allowed. These restricted model components are AMM, AAM, AMA, and MMA.
- **...** Other arguments

**Details**

Based on the classification of methods as described in Hyndman et al (2008).

The methodology is fully automatic. The model is chosen automatically if not specified. This methodology performed extremely well on the M3-competition data. (See Hyndman, et al, 2002, below.)

**Value**

A model specification.
Specials

The *specials* define the methods and parameters for the components (error, trend, and seasonality) of an ETS model. If more than one method is specified, ETS will consider all combinations of the specified models and select the model which best fits the data (minimising $\text{ic}$). The method argument for each specials have reasonable defaults, so if a component is not specified an appropriate method will be chosen automatically.

There are a couple of limitations to note about ETS models:

- It does not support exogenous regressors.
- It does not support missing values. You can complete missing values in the data with imputed values (e.g. with `tidyr::fill()`). or by fitting a different model type and then calling `fabletools::interpolate()` before fitting the model.

**error**: The error special is used to specify the form of the error term.

```r
error(method = c("A", "M"))
```

- **method**: The form of the error term: either additive ("A") or multiplicative ("M"). If the error is multiplicative, the data must be non-negative.

**trend**: The trend special is used to specify the form of the trend term and associated parameters.

```r
trend(method = c("N", "A", "Ad"),
       alpha = NULL, alpha_range = c(1e-04, 0.9999),
       beta = NULL, beta_range = c(1e-04, 0.9999),
       phi = NULL, phi_range = c(0.8, 0.98))
```

- **method**: The form of the trend term: either none ("N"), additive ("A"), multiplicative ("M") or damped variants ("Ad", "Md"). All specified methods are tested on the data, and the one that gives the best fit (lowest $\text{ic}$) will be kept.
- **alpha**: The value of the smoothing parameter for the level. If $\alpha = 0$, the level will not change over time. Conversely, if $\alpha = 1$ the level will update similarly to a random walk process.
- **alpha_range**: If $\alpha=NULL$, alpha_range provides bounds for the optimised value of alpha.
- **beta**: The value of the smoothing parameter for the slope. If $\beta = 0$, the slope will not change over time. Conversely, if $\beta = 1$ the slope will have no memory of past slopes.
- **beta_range**: If $\beta=NULL$, beta_range provides bounds for the optimised value of beta.
- **phi**: The value of the dampening parameter for the slope. If $\phi = 0$, the slope will be dampened immediately (no slope). Conversely, if $\phi = 1$ the slope will not be dampened.
- **phi_range**: If $\phi=NULL$, phi_range provides bounds for the optimised value of phi.

**season**: The season special is used to specify the form of the seasonal term and associated parameters. To specify a nonseasonal model you would include `season(method = "N")`.

```r
season(method = c("N", "A", "M"), period = NULL,
       gamma = NULL, gamma_range = c(1e-04, 0.9999))
```

- **method**: The form of the seasonal term: either none ("N"), additive ("A"), multiplicative ("M"). All specified methods are tested on the data, and the one that gives the best fit (lowest $\text{ic}$) will be kept.
- **period**: The periodic nature of the seasonality. This can be either a number indicating the number of observations in each seasonal period, or text to indicate the duration of the seasonal window (for example, annual seasonality would be "1 year").
- **gamma**: The value of the smoothing parameter for the seasonal pattern. If $\gamma = 0$, the seasonal pattern will not change over time. Conversely, if $\gamma = 1$ the seasonality will have no memory of past seasonal periods.
- **gamma_range**: If $\gamma=NULL$, gamma_range provides bounds for the optimised value of gamma.

**References**


See Also

*Forecasting: Principles and Practices, Exponential smoothing (chapter 8)*

Examples

```r
as_tsibble(lh) %>%
model(AR(value ~ order(3))) %>%
fitted()
```

---

**fitted.AR**

*Extract fitted values from a fable model*

**Description**

Extracts the fitted values.

**Usage**

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

**Arguments**

- `object` The time series model used to produce the forecasts
- `...` Additional arguments for forecast model methods.

**Value**

A vector of fitted values.

**Examples**

```r
as_tsibble(lh) %>%
model(AR(value ~ order(3))) %>%
fitted()
```
fitted.croston  
*Extract fitted values from a fable model*

**Description**

Extracts the fitted values.

**Usage**

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

**Arguments**

- `object`: The time series model used to produce the forecasts
- `...`: Additional arguments for forecast model methods.

**Value**

A vector of fitted values.

**Examples**

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  fitted()
```

---

fitted.ARIMA  
*Extract fitted values from a fable model*

**Description**

Extracts the fitted values.

**Usage**

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

**Arguments**

- `object`: The time series model used to produce the forecasts
- `...`: Additional arguments for forecast model methods.

**Value**

A vector of fitted values.

**Examples**

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  fitted()
```
fitted.ETS

Value

A vector of fitted values.

Examples

```r
library(tsibble)
sim_poisson <- tsibble(
  time = yearmonth("2012 Dec") + seq_len(24),
  count = rpois(24, lambda = 0.3),
  index = time
)
sim_poisson %>%
  model(CROSTON(count)) %>%
tidy()
```

### Description

Extracts the fitted values.

### Usage

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

### Arguments

- **object**: The time series model used to produce the forecasts
- **...**: Additional arguments for forecast model methods.

### Value

A vector of fitted values.

### Examples

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
fitted()
```
fitted.fable_theta  Extract fitted values from a fable model

Description
Extracts the fitted values.

Usage
## S3 method for class 'fable_theta'
fitted(object, ...)

Arguments

object  The time series model used to produce the forecasts
...
  Additional arguments for forecast model methods.

Value
A vector of fitted values.

Examples
library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
  fitted()

fitted.model_mean  Extract fitted values from a fable model

Description
Extracts the fitted values.

Usage
## S3 method for class 'model_mean'
fitted(object, ...)

Arguments

object  The time series model used to produce the forecasts
...
  Additional arguments for forecast model methods.
Value

A vector of fitted values.

Examples

```r
library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>
  fitted()
```

---

### fitted.NNETAR

*Extract fitted values from a fable model*

**Description**

Extracts the fitted values.

**Usage**

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

**Arguments**

- `object` The time series model used to produce the forecasts
- `...` Additional arguments for forecast model methods.

**Value**

A vector of fitted values.

**Examples**

```r
as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15))) %>%
  fitted()
```
fitted.RW

Extract fitted values from a fable model

Description

Extracts the fitted values.

Usage

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

Arguments

object The time series model used to produce the forecasts
...
    Additional arguments for forecast model methods.

Value

A vector of fitted values.

Examples

as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  fitted()

library(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
  fitted()

fitted.TSLM

Extract fitted values from a fable model

Description

Extracts the fitted values.

Usage

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

Arguments

object The time series model used to produce the forecasts
...

Additional arguments for forecast model methods.

Value

A vector of fitted values.

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
fitted()
```

---

### Description

Extracts the fitted values.

### Usage

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

### Arguments

- `object` The time series model used to produce the forecasts
- `...` Additional arguments for forecast model methods.

### Value

A vector of fitted values.

### Examples

```r
lung_deaths <- cbind(mdeaths, fdeaths) %>%
as_tsibble(pivot_longer = FALSE)
lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3))) %>%
fitted()
```
forecast.AR

Forecast a model from the fable package

Description

Produces forecasts from a trained model.

Usage

```r
## S3 method for class 'AR'
forecast(
  object,
  new_data = NULL,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

Arguments

- `object`: The time series model used to produce the forecasts
- `new_data`: A `tsibble` containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `times`: The number of sample paths to use in estimating the forecast distribution when `bootstrap = TRUE`.
- `...`: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
as_tsibble(lh) %>%
  model(AR(value ~ order(3))) %>%
  forecast()
```
forecast.ARIMA

Forecast a model from the fable package

Description

Produces forecasts from a trained model.

Usage

```r
## S3 method for class 'ARIMA'
forecast(
  object,
  new_data = NULL,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

Arguments

- `object`: The time series model used to produce the forecasts.
- `new_data`: A tsibble containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If TRUE, then forecast distributions are computed using simulation with resampled errors.
- `times`: The number of sample paths to use in estimating the forecast distribution when `bootstrap = TRUE`.
- `...`: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  forecast()
```
forecast.croston  
Forecast a model from the fable package

Description

Produces forecasts from a trained model.

Usage

## S3 method for class 'croston'
forecast(object, new_data, specials = NULL, ...)

Arguments

- **object**: The time series model used to produce the forecasts
- **new_data**: A `tsibble` containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **...**: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
library(tsibble)
sim_poisson <- tsibble(
  time = yearmonth("2012 Dec") + seq_len(24),
  count = rpois(24, lambda = 0.3),
  index = time
)
sim_poisson %>%
  model(CROSTON(count)) %>%
  forecast()
```

forecast.ETS  
Forecast a model from the fable package

Description

Produces forecasts from a trained model.
## S3 method for class 'ETS'

```r
forecast(
  object,
  new_data = NULL,
  specials = NULL,
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

### Arguments

- **object**: The time series model used to produce the forecasts.
- **new_data**: A `tsibble` containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **simulate**: If `TRUE`, prediction intervals are produced by simulation rather than using analytic formulae.
- **bootstrap**: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- **times**: The number of sample paths to use in estimating the forecast distribution if simulated intervals are used.
- **...**: Additional arguments for forecast model methods.

### Value

A list of forecasts.

### Examples

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
  forecast()
```

---

**Forecast a model from the fable package**

**Description**

Produces forecasts from a trained model.
Usage

```r
## S3 method for class 'fable_theta'
forecast(
  object,
  new_data,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

Arguments

- `object`: The time series model used to produce the forecasts.
- `new_data`: A `tsibble` containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `times`: The number of sample paths to use in estimating the forecast distribution when `bootstrap = TRUE`.
- `...`: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  forecast()
```

**Description**

Produces forecasts from a trained model.
Usage

## S3 method for class 'model_mean'
forecast(
  object,
  new_data,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
...
)

Arguments

object  The time series model used to produce the forecasts
new_data A tsibble containing future information used to forecast.
specials (passed by fabletools::forecast.mdl_df()).
bootstrap If TRUE, then forecast distributions are computed using simulation with resampled errors.
times The number of sample paths to use in estimating the forecast distribution when bootstrap = TRUE.
... Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
  forecast()

---

**forecast.NNETAR**

Forecast a model from the fable package

Description

Produces forecasts from a trained model.
Usage

```r
## S3 method for class 'NNETAR'
forecast(
  object,
  new_data,
  specials = NULL,
  simulate = TRUE,
  bootstrap = FALSE,
  times = 1000,
  ...
)
```

Arguments

- `object`: The time series model used to produce the forecasts.
- `new_data`: A `tsibble` containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `simulate`: If `TRUE`, prediction intervals are produced by simulation rather than using analytic formulae.
- `bootstrap`: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `times`: The number of sample paths to use in estimating the forecast distribution if simulated intervals are used.
- `...`: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15))) %>%
  forecast(times = 10)
```
Usage

## S3 method for class 'RW'
forecast(
  object,
  new_data,
  specials = NULL,
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)

Arguments

object          The time series model used to produce the forecasts
new_data        A tsibble containing future information used to forecast.
specials        (passed by fabletools::forecast.mdl_df()).
simulate         If TRUE, prediction intervals are produced by simulation rather than using analytic formulae.
bootstrap        If TRUE, then forecast distributions are computed using simulation with resampled errors.
times            The number of sample paths to use in estimating the forecast distribution when bootstrap = TRUE.
...              Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  forecast()

library(tsibbledata)
Aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
  forecast()
forecast.TSLM

Forecast a model from the fable package

Description

Produces forecasts from a trained model.

Usage

```r
## S3 method for class 'TSLM'
forecast(
  object,
  new_data,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

Arguments

- `object`: The time series model used to produce the forecasts
- `new_data`: A tsibble containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If TRUE, then forecast distributions are computed using simulation with resampled errors.
- `times`: The number of sample paths to use in estimating the forecast distribution when bootstrap = TRUE.
- `...`: Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
  forecast()
```
**Description**

Produces forecasts from a trained model.

**Usage**

```r
## S3 method for class 'VAR'
forecast(
  object,
  new_data = NULL,
  specials = NULL,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

**Arguments**

- **object**  
The time series model used to produce the forecasts
- **new_data**  
A `tsibble` containing future information used to forecast.
- **specials**  
(passed by `fabletools::forecast.mdl_df()`).
- **bootstrap**  
If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- **times**  
The number of sample paths to use in estimating the forecast distribution when `bootstrap = TRUE`.
- **...**  
Additional arguments for forecast model methods.

**Value**

A list of forecasts.

**Examples**

```r
lung_deaths <- cbind(mdeaths, fdeaths) %>%
as_tsibble(pivot_longer = FALSE)

lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3))) %>%
  forecast()
```
generate.ETS  

---

**generate.ETS**  

*Generate new data from a fable model*

---

**Description**

Simulates future paths from a dataset using a fitted model. 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.

**Usage**

```r
## S3 method for class 'ARIMA'
generate(x, new_data, specials, bootstrap = FALSE, ...)
```

**Arguments**

- `x`  
  A fitted model.
- `new_data`  
  A `tsibble` containing future information used to forecast.
- `specials`  
  (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`  
  If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `...`  
  Additional arguments for forecast model methods.

**See Also**

`fabletools::generate.mdl_df`

**Examples**

```r
gable_fit <- as_tsibble(USAccDeaths) %>%
  model(model = ARIMA(value ~ 0 + pdq(0,1,1) + PDQ(0,1,1)))
gable_fit %>%% generate(times = 10)
```

---

**generate.ETS**  

*Generate new data from a fable model*

---

**Description**

Simulates future paths from a dataset using a fitted model. 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.
Usage

```r
## S3 method for class 'ETS'
generate(x, new_data, specials, bootstrap = FALSE, ...)
```

Arguments

- `x`: A fitted model.
- `new_data`: A `tsibble` containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `...`: Additional arguments for forecast model methods.

See Also

- `fabletools::generate.mdl_df`

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(ETS(log(value) ~ season("A"))) %>%
  generate(times = 100)
```

---

**generate.model_mean**  
Generate new data from a fable model

Description

Simulates future paths from a dataset using a fitted model. 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.

Usage

```r
## S3 method for class 'model_mean'
generate(x, new_data, bootstrap = FALSE, ...)
```

Arguments

- `x`: A fitted model.
- `new_data`: A `tsibble` containing future information used to forecast.
- `bootstrap`: If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
- `...`: Additional arguments for forecast model methods.
See Also

fabletools::generate.mdl_df

Examples

library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
generate()

---

generate.NNETAR  
Generate new data from a fable model

Description

Simulates future paths from a dataset using a fitted model. 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.

Usage

## S3 method for class 'NNETAR'
generate(x, new_data, specials = NULL, bootstrap = FALSE, ...)

Arguments

x  
A fitted model.

new_data  
A tsibble containing future information used to forecast.

specials  
(passed by fabletools::forecast.mdl_df()).

bootstrap  
If TRUE, then forecast distributions are computed using simulation with resampled errors.

...  
Additional arguments for forecast model methods.

See Also

fabletools::generate.mdl_df

Examples

as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15))) %>%
generate()
generate.RW

Generate new data from a fable model

Description

Simulates future paths from a dataset using a fitted model. 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.

Usage

## S3 method for class 'RW'
generate(x, new_data, bootstrap = FALSE, ...)

Arguments

- `x` A fitted model.
- `new_data` A tsibble containing future information used to forecast.
- `bootstrap` If TRUE, then forecast distributions are computed using simulation with resampled errors.
- `...` Additional arguments for forecast model methods.

See Also

fabletools::generate.mdl_df

Examples

```r
as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
genenerate()

tibrary(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
genenerate()
```
**generate.TSLM**  
*Generate new data from a fable model*

**Description**
Simulates future paths from a dataset using a fitted model. 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.

**Usage**
```r
## S3 method for class 'TSLM'
genarate(x, new_data, specials, bootstrap = FALSE, ...)
```

**Arguments**
- `x`: A fitted model.
- `new_data`: A tsibble containing future information used to forecast.
- `specials`: (passed by `fabletools::forecast.mdl_df()`).
- `bootstrap`: If TRUE, then forecast distributions are computed using simulation with resampled errors.
- `...`: Additional arguments for forecast model methods.

**See Also**
`fabletools::generate.mdl_df`

**Examples**
```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
  generate()
```

---

**glance.AR**  
*Glance a AR*

**Description**
Construct a single row summary of the AR model.

**Usage**
```r
## S3 method for class 'AR'
glance(x, ...)
```
Arguments

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

Details

Contains the variance of residuals (\(\text{sigma2}\)), the log-likelihood (\(\text{log_lik}\)), and information criterion (\(\text{AIC}\), \(\text{AICc}\), \(\text{BIC}\)).

Value

A one row tibble summarising the model’s fit.

Examples

```
as_tsibble(lh) %>%
  model(AR(value ~ order(3))) %>%
  glance()
```

---

**Description**

Construct a single row summary of the ARIMA model.

**Usage**

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

**Arguments**

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

**Details**

Contains the variance of residuals (\(\text{sigma2}\)), the log-likelihood (\(\text{log_lik}\)), information criterion (\(\text{AIC}\), \(\text{AICc}\), \(\text{BIC}\)) and the characteristic roots (\(\text{ar_roots}\) and \(\text{ma_roots}\)).

**Value**

A one row tibble summarising the model’s fit.
Examples

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  glance()
```

---

**glance.ETS**

*Glance an ETS model*

**Description**

Construct a single row summary of the ETS model.

**Usage**

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

**Arguments**

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

**Details**

Contains the variance of residuals (`sigma2`), the log-likelihood (`log_lik`), and information criterion (`AIC`, `AICc`, `BIC`).

**Value**

A one row tibble summarising the model’s fit.

**Examples**

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
  glance()
```
Description

Construct a single row summary of the average method model.

Usage

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

Arguments

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

Details

Contains the variance of residuals ($\sigma^2$).

Value

A one row tibble summarising the model’s fit.

Description

Construct a single row summary of the average method model.

Usage

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

Arguments

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

Details

Contains the variance of residuals ($\sigma^2$).
Value

A one row tibble summarising the model’s fit.

Examples

```r
library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
  glance()
```

---

**glance.NNETAR**

Glance a NNETAR model

Description

Construct a single row summary of the NNETAR model. Contains the variance of residuals ($\sigma^2$).

Usage

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

Arguments

- `x` model or other R object to convert to single-row data frame
- `...` other arguments passed to methods

Value

A one row tibble summarising the model’s fit.

Examples

```r
as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15))) %>%
  glance()
```
glance.RW

Glance a lag walk model

Description

Construct a single row summary of the lag walk model. Contains the variance of residuals ($\sigma^2$).

Usage

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

Arguments

- `x`: model or other R object to convert to single-row data frame
- `...`: other arguments passed to methods

Value

A one row tibble summarising the model’s fit.

Examples

```r
as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  glance()

library(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
  glance()
```

glance.TSLM

Glance a TSLM

Description

Construct a single row summary of the TSLM model.

Usage

```r
## S3 method for class 'TSLM'
glance(x, ...)
```
Arguments

x  model or other R object to convert to single-row data frame
...
other arguments passed to methods

Details

Contains the R squared \(r_{\text{ squared}}\), variance of residuals \(\sigma^2\), the log-likelihood \(\log_{\text{ lik}}\), and information criterion \(\text{AIC, AICc, BIC}\).

Value

A one row tibble summarising the model’s fit.

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
  glance()
```

---

**glance.VAR**  
*Glance a VAR*

Description

Construct a single row summary of the VAR model.

Usage

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

Arguments

x  model or other R object to convert to single-row data frame
...
other arguments passed to methods

Details

Contains the variance of residuals \(\sigma^2\), the log-likelihood \(\log_{\text{ lik}}\), and information criterion \(\text{AIC, AICc, BIC}\).

Value

A one row tibble summarising the model’s fit.
Examples

```
lung_deaths <- cbind(mdeaths, fdeaths) %>%
  as_tsibble(pivot_longer = FALSE)

lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3))) %>%
  glance()
```

---

**interpolate.ARIMA**

**Interpolate missing values from a fable model**

**Description**

Applies a model specific estimation technique to predict the values of missing values in a `tsibble`, and replace them.

**Usage**

```r
## S3 method for class 'ARIMA'
interpolate(object, new_data, specials, ...)
```

**Arguments**

- `object` The time series model used to produce the forecasts
- `new_data` A `tsibble` containing future information used to forecast.
- `specials` (passed by `fabletools::forecast.mdl_df()`).
- `...` Additional arguments for forecast model methods.

**Value**

A `tibble` of the same dimension of `new_data` with missing values interpolated.

**Examples**

```
library(tsibbledata)

olympic_running %>%
  model(arima = ARIMA(Time ~ trend())) %>%
  interpolate(olympic_running)
```
Interpolate missing values from a fable model

Description

Applies a model specific estimation technique to predict the values of missing values in a tsibble, and replace them.

Usage

## S3 method for class 'model_mean'
interpolate(object, new_data, specials, ...)

Arguments

- **object**: The time series model used to produce the forecasts
- **new_data**: A tsibble containing future information used to forecast.
- **specials**: (passed by fabletools::forecast.mdl_df()).
- **...**: Additional arguments for forecast model methods.

Value

A tibble of the same dimension of new_data with missing values interpolated.

Examples

```r
library(tsibbledata)
olympic_running %>%
  model(mean = MEAN(Time)) %>%
  interpolate(olympic_running)
```

Interpolate missing values from a fable model

Description

Applies a model specific estimation technique to predict the values of missing values in a tsibble, and replace them.

Usage

## S3 method for class 'TSLM'
interpolate(object, new_data, specials, ...)

```r
library(tsibbledata)
olympic_running %>%
  model(mean = MEAN(Time)) %>%
  interpolate(olympic_running)
```
Arguments

object  The time series model used to produce the forecasts
new_data  A tsibble containing future information used to forecast.
specials  (passed by fabletools::forecast.mdl_df()).
...  Additional arguments for forecast model methods.

Value

A tsibble of the same dimension of new_data with missing values interpolated.

Examples

```r
library(tsibbledata)

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

Description

MEAN() returns an iid model applied to the formula’s response variable.

Usage

MEAN(formula, ...)

Arguments

formula  Model specification.
...  Not used.

Details

The model does not use any specials, and so everything on the formula’s right-hand-side will be ignored.

Value

A model specification.

Specials

`window`: The window special is used to specify a rolling window for the mean.

`window(size = NULL)`
size The size (number of observations) for the rolling window. If NULL (default), a rolling window will not be used.

See Also

Forecasting: Principles and Practices, Some simple forecasting methods (section 3.2)

Examples

library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand))

NNETAR Neural Network Time Series Forecasts

Description

Feed-forward neural networks with a single hidden layer and lagged inputs for forecasting univariate time series.

Usage

NNETAR(formula, n_nodes = NULL, n_networks = 20, scale_inputs = TRUE, ...)

Arguments

formula Model specification (see "Specials" section).
n_nodes Number of nodes in the hidden layer. Default is half of the number of input nodes (including external regressors, if given) plus 1.
n_networks Number of networks to fit with different random starting weights. These are then averaged when producing forecasts.
scale_inputs If TRUE, inputs are scaled by subtracting the column means and dividing by their respective standard deviations. Scaling is applied after transformations.
... Other arguments passed to \link[nnet]{nnet}.

Details

A feed-forward neural network is fitted with lagged values of the response as inputs and a single hidden layer with size nodes. The inputs are for lags 1 to p, and lags m to mP where m is the seasonal period specified.

If exogenous regressors are provided, its columns are also used as inputs. Missing values are currently not supported by this model. A total of repeats networks are fitted, each with random starting weights. These are then averaged when computing forecasts. The network is trained for one-step forecasting. Multi-step forecasts are computed recursively.

For non-seasonal data, the fitted model is denoted as an NNAR(p,k) model, where k is the number of hidden nodes. This is analogous to an AR(p) model but with non-linear functions. For seasonal data, the fitted model is called an NNAR(p,P,k)[m] model, which is analogous to an ARIMA(p,0,0)(P,0,0)[m] model but with non-linear functions.
Value

A model specification.

Specials

**AR:** The AR special is used to specify auto-regressive components in each of the nodes of the neural network.

\[ \text{AR}(p = \text{NULL}, P = 1, \text{period} = \text{NULL}) \]

- \( p \): The order of the non-seasonal auto-regressive (AR) terms. If \( p = \text{NULL} \), an optimal number of lags will be selected for a linear AR(\( p \)) model via AIC. For seasonal time series, this will be computed on the seasonally adjusted data (via STL decomposition).
- \( P \): The order of the seasonal auto-regressive (SAR) terms.
- \( \text{period} \): The periodic nature of the seasonality. This can be either a number indicating the number of observations in each seasonal period, or text to indicate the duration of the seasonal window (for example, annual seasonality would be “1 year”).

**xreg:** Exogenous regressors can be included in an NNETAR model without explicitly using the \text{xreg()} special. Common exogenous regressor specials as specified in \text{common_xregs} can also be used. These regressors are handled using \text{stats::model.frame()}, and so interactions and other functionality behaves similarly to \text{stats::lm()}.

\[ \text{xreg}(...) \]

... Bare expressions for the exogenous regressors (such as \( \log(x) \))

See Also

*Forecasting: Principles and Practices, Neural network models (section 11.3)*

Examples

```r
as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15)))
```

---

### refit.AR

**Refit an AR model**

Description

Applies a fitted AR model to a new dataset.

Usage

```r
## S3 method for class 'AR'
refit(object, new_data, specials = NULL, reestimate = FALSE, ...)
```
Arguments
- **object**: The time series model used to produce the forecasts.
- **new_data**: A `tsibble` containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **reestimate**: If `TRUE`, the coefficients for the fitted model will be re-estimated to suit the new data.
  - Additional arguments for forecast model methods.

Value
A refitted model.

Examples
```r
lung_deaths_male <- as_tsibble(mdeaths)
lung_deaths_female <- as_tsibble(fdeaths)

fit <- lung_deaths_male %>%
  model(AR(value ~ 1 + order(10)))

report(fit)

fit %>%
  refit(lung_deaths_female) %>%
  report()
```

---

**refit.ARIMA**

*Refit an ARIMA model*

Description
Applies a fitted ARIMA model to a new dataset.

Usage
```r
## S3 method for class 'ARIMA'
refit(object, new_data, specials = NULL, reestimate = FALSE, ...)
```

Arguments
- **object**: The time series model used to produce the forecasts.
- **new_data**: A `tsibble` containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **reestimate**: If `TRUE`, the coefficients for the fitted model will be re-estimated to suit the new data.
  - Additional arguments for forecast model methods.
Value

A refitted model.

Examples

lung_deaths_male <- as_tsibble(mdeaths)
lung_deaths_female <- as_tsibble(fdeaths)

fit <- lung_deaths_male %>%
  model(ARIMA(value ~ 1 + pdq(2, 0, 0) + PDQ(2, 1, 0)))
report(fit)

fit %>%
  refit(lung_deaths_female) %>%
  report()

---

**refit.ETS**

*Refit an ETS model*

Description

Applies a fitted ETS model to a new dataset.

Usage

```r
## S3 method for class 'ETS'
refit(
  object, 
  new_data, 
  specials = NULL, 
  reestimate = FALSE, 
  reinitialise = TRUE, 
  ...
)
```

Arguments

- **object**: The time series model used to produce the forecasts.
- **new_data**: A `tsibble` containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **reestimate**: If `TRUE`, the coefficients for the fitted model will be re-estimated to suit the new data.
- **reinitialise**: If `TRUE`, the initial parameters will be re-estimated to suit the new data.
- **...**: Additional arguments for forecast model methods.
Examples

```r
lung_deaths_male <- as_tsibble(mdeaths)
lung_deaths_female <- as_tsibble(fdeaths)

fit <- lung_deaths_male %>%
  model(ETS(value))

report(fit)

fit %>%
  refit(lung_deaths_female, reinitialise = TRUE) %>%
  report()
```

---

refit.TSLM  
**Refit a TSLM**

Description

Applies a fitted TSLM to a new dataset.

Usage

```r
## S3 method for class 'TSLM'
refit(object, new_data, specials = NULL, reestimate = FALSE, ...)
```

Arguments

- **object**: The time series model used to produce the forecasts
- **new_data**: A tsibble containing future information used to forecast.
- **specials**: (passed by `fabletools::forecast.mdl_df()`).
- **reestimate**: If TRUE, the coefficients for the fitted model will be re-estimated to suit the new data.
- **...**: Additional arguments for forecast model methods.

Examples

```r
lung_deaths_male <- as_tsibble(mdeaths)
lung_deaths_female <- as_tsibble(fdeaths)

fit <- lung_deaths_male %>%
  model(TSLM(value ~ trend() + season()))

report(fit)

fit %>%
  refit(lung_deaths_female) %>%
  report()
```
residuals.AR

Extract residuals values from a fable model

Description

Extracts the residuals.

Usage

```r
## S3 method for class 'AR'
residuals(object, type = c("innovation", "regression"), ...)
```

Arguments

- `object` The time series model used to produce the forecasts
- `type` The type of the residuals to extract.
- `...` Additional arguments for forecast model methods.

Value

A vector of fitted residuals.

Examples

```r
as_tsibble(lh) %>%
  model(AR(value ~ order(3))) %>%
  residuals()
```

residuals.ARIMA

Extract residuals values from a fable model

Description

Extracts the residuals.

Usage

```r
## S3 method for class 'ARIMA'
residuals(object, type = c("innovation", "regression"), ...)
```

Arguments

- `object` The time series model used to produce the forecasts
- `type` The type of the residuals to extract.
- `...` Additional arguments for forecast model methods.
Value

A vector of fitted residuals.

Examples

USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
residuals()

## S3 method for class 'croston'

residuals(object, ...)

Arguments

object The time series model used to produce the forecasts
...
Additional arguments for forecast model methods.

Value

A vector of fitted residuals.

Examples

library(tsibble)
sim_poisson <- tsibble(
  time = yearmonth("2012 Dec") + seq_len(24),
  count = rpois(24, lambda = 0.3),
  index = time
)

sim_poisson %>%
  model(CROSTON(count)) %>%
residuals()
residuals.ETS  
*Extract residuals values from a fable model*

**Description**

Extracts the residuals.

**Usage**

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

**Arguments**

- `object`: The time series model used to produce the forecasts
- `...`: Additional arguments for forecast model methods.

**Value**

A vector of fitted residuals.

**Examples**

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
  residuals()
```

---

residuals.fable_theta  
*Extract residuals values from a fable model*

**Description**

Extracts the residuals.

**Usage**

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

**Arguments**

- `object`: The time series model used to produce the forecasts
- `...`: Additional arguments for forecast model methods.
residuals.model_mean

Value

A vector of fitted residuals.

Examples

```r
library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
  residuals()
```

Description

Extracts the residuals.

Usage

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

Arguments

- `object` The time series model used to produce the forecasts
- `...` Additional arguments for forecast model methods.

Value

A vector of fitted residuals.

Examples

```r
library(tsibbledata)
vic_elec %>%
  model(avg = MEAN(Demand)) %>%
  residuals()
```
**residuals.NNETAR**

*Extract residuals values from a fable model*

**Description**

Extracts the residuals.

**Usage**

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

**Arguments**

- **object**
  The time series model used to produce the forecasts
- **...**
  Additional arguments for forecast model methods.

**Value**

A vector of fitted residuals.

**Examples**

```r
as_tsibble(airmiles) %>%
  model(nn = NNETAR(box_cox(value, 0.15))) %>%
  residuals()
```

**residuals.RW**

*Extract residuals values from a fable model*

**Description**

Extracts the residuals.

**Usage**

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

**Arguments**

- **object**
  The time series model used to produce the forecasts
- **...**
  Additional arguments for forecast model methods.
residuals.TSLM

Value

A vector of fitted residuals.

Examples

```r
as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  residuals()

library(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
  residuals()
```

Description

Extracts the residuals.

Usage

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

Arguments

- `object`: The time series model used to produce the forecasts.
- `...`: Additional arguments for forecast model methods.

Value

A vector of fitted residuals.

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
  residuals()
```
residuals.VAR

Extract residuals values from a fable model

Description
Extracts the residuals.

Usage
```r
## S3 method for class 'VAR'
residuals(object, ...)
```

Arguments
- `object` The time series model used to produce the forecasts
- `...` Additional arguments for forecast model methods.

Value
A vector of fitted residuals.

Examples
```r
lung_deaths <- cbind(mdeaths, fdeaths) %>%
  as_tsibble(pivot_longer = FALSE)

lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3))) %>%
  residuals()
```

RW
Random walk models

Description
RW() returns a random walk model, which is equivalent to an ARIMA(0,1,0) model with an optional drift coefficient included using drift(). naive() is simply a wrapper to rwf() for simplicity. snaive() returns forecasts and prediction intervals from an ARIMA(0,0,0)(0,1,0)m model where m is the seasonal period.

Usage
```r
RW(formula, ...)

NAIVE(formula, ...)

SNAIVE(formula, ...)
```
Arguments

- formula: Model specification (see "Specials" section).
- ...: Not used.

Details

The random walk with drift model is

\[ Y_t = c + Y_{t-1} + Z_t \]

where \( Z_t \) is a normal iid error. Forecasts are given by

\[ Y_n(h) = ch + Y_n \]

If there is no drift (as in naive), the drift parameter \( c=0 \). Forecast standard errors allow for uncertainty in estimating the drift parameter (unlike the corresponding forecasts obtained by fitting an ARIMA model directly).

The seasonal naive model is

\[ Y_t = Y_{t-m} + Z_t \]

where \( Z_t \) is a normal iid error.

Value

A model specification.

Specials

- lag: The `lag` special is used to specify the lag order for the random walk process. If left out, this special will automatically be included.
  
  `lag(lag = NULL)`

  `lag` The lag order for the random walk process. If `lag = m`, forecasts will return the observation from \( m \) time periods ago. This

- drift: The `drift` special can be used to include a drift/trend component into the model. By default, drift is not included unless `drift()` is included in the formula.
  
  `drift(drift = TRUE)`

  `drift` If `drift = TRUE`, a drift term will be included in the model.

See Also

- Forecasting: Principles and Practices, Some simple forecasting methods (section 3.2)

Examples

```r
library(tsibbledata)
```
The theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift. This is demonstrated in Hyndman and Billah (2003).

Usage

THETA(formula, ...)

Arguments

formula Model specification.
... Not used.

Details

The series is tested for seasonality using the test outlined in A&N. If deemed seasonal, the series is seasonally adjusted using a classical multiplicative decomposition before applying the theta method. The resulting forecasts are then reseasonalized.

More general theta methods are available in the forecTheta package.

Value

A model specification.

Specials

season: The season special is used to specify the parameters of the seasonal adjustment via classical decomposition.

window(period = NULL, method = c("multiplicative", "additive"))

period The periodic nature of the seasonality. This can be either a number indicating the number of observations in each season or a string indicating the duration of the seasonal window (for example, annual seasonality would be "1 year").

method The type of classical decomposition to apply. The original Theta method always used multiplicative seasonal decomposition.
**Author(s)**

Rob J Hyndman, Mitchell O’Hara-Wild

**References**


---

**tidy.AR**  
*Tidy a fable model*

**Description**

Returns the coefficients from the model in a tibble format.

**Usage**

```r
## S3 method for class 'AR'
tidy(x)
```

**Arguments**

- `x` An object to be converted into a tidy `tibble::tibble()`.

**Value**

The model’s coefficients in a tibble.

**Examples**

```r
as_tsibble(lh) %>%
  model(AR(value ~ order(3))) %>%
  tidy()
```
**Description**

Returns the coefficients from the model in a tibble format.

**Usage**

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

**Arguments**

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.

**Value**

The model’s coefficients in a tibble.

**Examples**

```r
USAccDeaths %>%
as_tsibble() %>%
model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
tidy()
```

---

**Description**

Returns the coefficients from the model in a tibble format.

**Usage**

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

**Arguments**

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.
tidy.ETS

Value

The model's coefficients in a tibble.

Examples

```r
library(tsibble)
sim_poisson <- tsibble(
  time = yearmonth("2012 Dec") + seq_len(24),
  count = rpois(24, lambda = 0.3),
  index = time
)
sim_poisson %>%
  model(CROSTON(count)) %>%
  tidy()
```

Description

Returns the coefficients from the model in a tibble format.

Usage

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

Arguments

- `x` An object to be converted into a tidy `tibble::tibble()`.
- `...` Additional arguments to tidying method.

Value

The model's coefficients in a tibble.

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(ets = ETS(log(value) ~ season("A"))) %>%
  tidy()
```
tidy.fable_theta  

Description

Returns the coefficients from the model in a tibble format.

Usage

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

Arguments

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.

Value

The model’s coefficients in a tibble.

Examples

```r
USAccDeaths %>%
  as_tsibble() %>%
  model(arima = ARIMA(log(value) ~ pdq(0, 1, 1) + PDQ(0, 1, 1))) %>%
  tidy()
```

tidy.model_mean  

Description

Returns the coefficients from the model in a tibble format.

Usage

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

Arguments

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.
tidy.NNETAR

Value

The model's coefficients in a tibble.

Examples

```r
library(tsibbledata)
vic_elec %>%
model(avg = MEAN(Demand)) %>%
tidy()
```

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

Arguments

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.

Value

The model's coefficients in a tibble.

Examples

```r
as_tsibble(airmiles) %>%
model(nn = NNETAR(box_cox(value, 0.15))) %>%
tidy()
```
tidy.RW

**Tidy a fable model**

**Description**

Returns the coefficients from the model in a tibble format.

**Usage**

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

**Arguments**

- `x`: An object to be converted into a tidy `tibble::tibble()`.
- `...`: Additional arguments to tidying method.

**Value**

The model's coefficients in a tibble.

**Examples**

```r
as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  tidy()
```

```r
library(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year"))) %>%
  tidy()
```

---

**tidy.TSLM**

**Tidy a fable model**

**Description**

Returns the coefficients from the model in a tibble format.

**Usage**

```r
## S3 method for class 'TSLM'
tidy(x, ...)
```
Arguments

x  An object to be converted into a tidy tibble::tibble().
...

Value

The model's coefficients in a tibble.

Examples

as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season())) %>%
tidy()

Description

Returns the coefficients from the model in a tibble format.

Usage

## S3 method for class 'VAR'
tidy(x)

Arguments

x  An object to be converted into a tidy tibble::tibble().

Value

The model's coefficients in a tibble.

Examples

lung_deaths <- cbind(mdeaths, fdeaths) %>%
as_tsibble(pivot_longer = FALSE)
lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3))) %>%
tidy()
TSLM

Fit a linear model with time series components

Description

The model formula will be handled using `stats::model.matrix()`, and so the same approach to include interactions in `stats::lm()` applies when specifying the formula. In addition to `stats::lm()`, it is possible to include `common_xregs` in the model formula, such as `trend()`, `season()`, and `fourier()`.

Usage

TSLM(formula)

Arguments

formula Model specification.

Value

A model specification.

Specials

`xreg`: Exogenous regressors can be included in an ARIMA model without explicitly using the `xreg()` special. Common exogenous regressor specials as specified in `common_xregs` can also be used. These regressors are handled using `stats::model.frame()`, and so interactions and other functionality behaves similarly to `stats::lm()`.

`xreg(...)`

... Bare expressions for the exogenous regressors (such as `log(x)`)

See Also

`stats::lm()`, `stats::model.matrix()` Forecasting: Principles and Practices, Time series regression models (chapter 6)

Examples

```r
as_tsibble(USAccDeaths) %>%
  model(lm = TSLM(log(value) ~ trend() + season()))
```

```r
library(tsibbledata)
olympic_running %>%
  model(TSLM(Time ~ trend())) %>%
  interpolate(olympic_running)
```
unitroot_options  

Options for the unit root tests for order of integration

Description

By default, a kpss test (via `feasts::unitroot_kpss()` will be performed for testing the required first order differences, and a test of the seasonal strength (via `feasts::feat_stl()` seasonal_strength) being above the 0.64 threshold is used for determining seasonal required differences.

Usage

```r
unitroot_options(
    ndiffs_alpha = 0.05,
    nsdiffs_alpha = 0.05,
    ndiffs_pvalue = ~feasts::unitroot_kpss(.)["kpss_pvalue"],
    nsdiffs_pvalue = ur_seasonal_strength(0.64)
)
```

Arguments

- `ndiffs_alpha, nsdiffs_alpha`
  The level for the test specified in the pval functions. As long as `pval < alpha`, differences will be added.
- `ndiffs_pvalue, nsdiffs_pvalue`
  A function (or lambda expression) which returns the probability of the . As long as `pval < alpha`, differences will be added.
  For the function for the seasonal p-value, the seasonal period will be provided as the `period` argument to this function. A vector of data to test is available as `.x`.

Value

A list of parameters

VAR  

Estimate a VAR model

Description

Searches through the vector of lag orders to find the best VAR model which has lowest AIC, AICc or BIC value. It is implemented using OLS per equation.

Usage

```r
VAR(formula, ic = c("aicc", "aic", "bic"), ...)
```
Arguments

formula  Model specification (see "Specials" section).
ic  The information criterion used in selecting the model.
...  Further arguments for arima

Details

Exogenous regressors and common_xregs can be specified in the model formula.

Value

A model specification.

Specials

pdq:  The AR special is used to specify the lag order for the auto-regression.

\(AR(p = 0:5)\)

p  The order of the auto-regressive (AR) terms. If multiple values are provided, the one which minimises ic will be chosen.

xreg:  Exogenous regressors can be included in an ARIMA model without explicitly using the xreg() special. Common exogenous regressor specials as specified in common_xregs can also be used. These regressors are handled using stats::model.frame(), and so interactions and other functionality behaves similarly to stats::lm().

The inclusion of a constant in the model follows the similar rules to stats::lm(), where including 1 will add a constant and 0 or -1 will remove the constant. If left out, the inclusion of a constant will be determined by minimising ic.

xreg(...)  ...  Bare expressions for the exogenous regressors (such as \(\log(x)\))

See Also

Forecasting: Principles and Practices, Vector autoregressions (section 11.2)

Examples

```r
lung_deaths <- cbind(mdeaths, fdeaths) %>%
as_tsibble(pivot_longer = FALSE)

fit <- lung_deaths %>%
  model(VAR(vars(mdeaths, fdeaths) ~ AR(3)))

report(fit)
```
fit %>%
  forecast() %>%
  autoplot(lung_deaths)
Index

*Topic datasets
  common_xregs, 7
AR, 3
ARIMA, 5
common_xregs, 4, 7, 45, 63, 67
components.ETS, 8
CROSTON, 9
ETS, 10
fabletools::dable(), 8
fabletools::forecast.mdl_df(), 20–32, 34, 41–43, 46–48
fabletools::generate.mdl_df, 30–34
fabletools::interpolate(), 12
feasts::feat_stl(), 66
feasts::unitroot_kpss(), 66
fitted.AR, 13
fitted.ARIMA, 14
fitted.croston, 14
fitted.ETS, 15
fitted.ETS_theta, 16
fitted.model_mean, 16
fitted.NNETAR, 17
fitted.RW, 18
fitted.TSLM, 18
fitted.VAR, 19
forecast.AR, 20
forecast.ARIMA, 21
forecast.croston, 22
forecast.ETS, 22
forecast.ETS_theta, 23
forecast.model_mean, 24
forecast.NNETAR, 25
forecast.RW, 26
forecast.TSLM, 28
forecast.VAR, 29
forecast::Arima(), 6
generate.ARIMA, 30
generate.ETS, 30
generate.model_mean, 31
generate.NNETAR, 32
generate.RW, 33
generate.TSLM, 34
glance.AR, 34
glance.ARIMA, 35
glance.ETS, 36
glance.fable_theta, 37
glance.model_mean, 37
glance.NNETAR, 38
glance.RW, 39
glance.TSLM, 39
glance.VAR, 40
interpolate.ARIMA, 41
interpolate.model_mean, 42
interpolate.TSLM, 42
MEAN, 43
NAIVE (RW), 55
NNETAR, 44
refit.AR, 45
refit.ARIMA, 46
refit.ETS, 47
refit.TSLM, 48
report.AR (AR), 3
report.ARIMA (ARIMA), 5
report.ETS (ETS), 10
report.model_mean (MEAN), 43
report.NNETAR (NNETAR), 44
report.RW (RW), 55
report.TSLM (TSLM), 65
report.VAR (VAR), 66
residuals.AR, 49
residuals.ARIMA, 49
residuals.RW, 50
residuals.croston, 50

69
residuals.ETS, 51
residuals.fable_theta, 51
residuals.model_mean, 52
residuals.NNETAR, 53
residuals.RW, 53
residuals.TSLM, 54
residuals.VAR, 55
RW, 55

SNAIVE (RW), 55
stats::ar.ols(), 3
stats::arima(), 5, 6
stats::lm(), 4, 7, 45, 65, 67
stats::model.frame(), 4, 7, 45, 65, 67
stats::model.matrix(), 65

THETA, 57
tibble::tibble(), 58–64
tidy.AR, 58
tidy.ARIMA, 59
tidy.croston, 59
tidy.ETS, 60
tidy.fable_theta, 61
tidy.model_mean, 61
tidy.NNETAR, 62
tidy.RW, 63
tidy.TSLM, 63
tidy.VAR, 64
tidyr::fill(), 12
TSLM, 65

unitroot_options, 66
unitroot_options(), 5

VAR, 66