Package ‘fable’

May 16, 2021

Title Forecasting Models for Tidy Time Series

Version 0.3.1

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

URL https://fable.tidyverts.org, https://github.com/tidyverts/fable

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

Depends R (>= 3.4.0), fabletools (>= 0.3.0)

Imports Rcpp (>= 0.11.0), rlang (>= 0.4.6), stats, dplyr (>= 1.0.0), tsibble (>= 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

RoxygenNote 7.1.1

NeedsCompilation yes

Author Mitchell O'Hara-Wild [aut, cre],
Rob Hyndman [aut],
Earo Wang [aut],
Gabriel Caceres [ctb] (NNETAR implementation),
Tim-Gunnar Hensel [ctb],
Timothy Hyndman [ctb]

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

Repository CRAN

Date/Publication 2021-05-16 14:20:07 UTC

R topics documented:

AR 4
ARIMA 5
breusch_godfrey 8
common_xregs 8
components.ETS 9
CROSTON 10
ETS 11
fitted.AR 14
fitted.ARIMA 15
fitted.croston 15
fitted.ETS 16
fitted.fable_theta 17
fitted.model_mean 17
fitted.NNETAR 18
fitted.RW 19
fitted.TSLM 19
fitted.VAR 20
forecast.AR 21
forecast.ARIMA 22
forecast.croston 23
forecast.ETS 23
forecast.fable_theta 24
forecast.model_mean 25
forecast.NNETAR 26
forecast.RW 27
forecast.TSLM 29
forecast.VAR 30
generate.AR 31
generate.ARIMA 31
generate.ETS 32
generate.model_mean 33
generate.NNETAR 34
generate.RW 34
generate.TSLM 35
glance.AR 36
glance.ARIMA 37
glance.ETS 37
glance.fable_theta 38
glance.model_mean 39
glance.NNETAR 39
glance.RW 40
glance.TSLM 41

glance.VAR 41
interpolate.ARIMA 42
interpolate.model_mean 43
interpolate.TSLM 44
MEAN 44
NNETAR 45
refit.AR 47
refit.ARIMA 48
refit.ETS 49
refit.model_mean 50
refit.NNETAR 50
refit.RW 51
refit.TSLM 52
residuals.AR 53
residuals.ARIMA 53
residuals.croston 54
residuals.ETS 55
residuals.fable_theta 55
residuals.model_mean 56
residuals.NNETAR 57
residuals.RW 57
residuals.TSLM 58
residuals.VAR 59
RW 59
THETA 61
tidy.AR 62
tidy.ARIMA 63
tidy.croston 63
tidy.ETS 64
tidy.fable_theta 65
tidy.model_mean 65
tidy.NNETAR 66
tidy.RW 67
tidy.TSLM 67
tidy.VAR 68
TSLM 69
unitroot_options 70
VAR 70

Index **73**

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

```
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.

```
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 cho
`fixed`	A named list of fixed parameters for coefficients. The names identify the coefficient, beginning with `ar`, and then foll

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 th

See Also

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

Examples

```
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, with the 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"),
  selection_metric = function(x) x[[ic]],
  stepwise = TRUE,
  greedy = TRUE,
  approximation = NULL,
  order_constraint = p + q + P + Q <= 6 & (constant + d + D <= 2),
  unitroot_spec = unitroot_options(),
  trace = FALSE,
  ...
)
```

Arguments

`formula` Model specification (see "Specials" section).

`ic` The information criterion used in selecting the model.

`selection_metric`
A function used to compute a metric from an `Arima` object which is minimised to select the best model.

`stepwise` Should stepwise be used? (Stepwise can be much faster) `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.

`trace` If `TRUE`, the selection_metric of estimated models in the selection procedure will be outputted to the console.

`...` Further arguments for `stats::arima()`

Value

A model specification.

Parameterisation

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

In `fable`, the parameterisation used is:

$$(1-\phi_1B-\cdots-\phi_pB^p)(1-B)^dy_t = c+(1+\theta_1B+\cdots+\theta_qB^q)\varepsilon_t$$

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

$$(1-\phi_1B-\cdots-\phi_pB^p)(y_t'-\mu) = (1+\theta_1B+\cdots+\theta_qB^q)\varepsilon_t$$

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

Specials

The *specials* define the space over which `ARIMA` will search for the model that best fits the data. If the RHS of `formula` is left blank, the default search space is given by `pdq() + 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 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 `pdq()/PDQ()` values must be specified. For example, `formula = response ~ 1 + pdq(1,1,1) + PDQ(1,0,0)`.

pdq: The `pdq` special is used to specify non-seasonal components of the model.

```
pdq(p = 0:5, d = 0:2, q = 0:5,
    p_init = 2, q_init = 2, fixed = list())
```

p The order of the non-seasonal auto-regressive (AR) terms. If multiple values are provided, the one which minimises

d The order of integration for non-seasonal differencing. If multiple values are provided, one of the values will be sele

q The order of the non-seasonal moving average (MA) terms. If multiple values are provided, the one which minimise

p_init If stepwise = TRUE, p_init provides the initial value for p for the stepwise search procedure.

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 ar or ma,

PDQ: The PDQ special is used to specify seasonal components of the model. To force a non-seasonal fit, specify PDQ(0,0,0) in the RHS of the model formula. Note that simply omitting PDQ from the formula will *not* result in a non-seasonal fit.

```
PDQ(P = 0:2, D = 0:1, Q = 0:2, period = NULL,
    P_init = 1, Q_init = 1, fixed = list())
```

P The order of the seasonal auto-regressive (SAR) terms. If multiple values are provided, the one which minimises ic

D The order of integration for seasonal differencing. If multiple values are provided, one of the values will be selected

Q The order of the seasonal moving average (SMA) terms. If multiple values are provided, the one which minimises i

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

P_init If stepwise = TRUE, P_init provides the initial value for P for the stepwise search procedure.

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 sr

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 th

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)
tsibbledata::global_economy %>%
  filter(Country == "Australia") %>%
  model(ARIMA(log(GDP) ~ Population))
```

breusch_godfrey *Breusch-Godfrey Test*

Description

Breusch-Godfrey test for higher-order serial correlation.

Usage

```
breusch_godfrey(x, ...)

## S3 method for class 'TSLM'
breusch_godfrey(x, order = 1, type = c("Chisq", "F"), ...)
```

Arguments

`x`	A model object to be tested.
`...`	Further arguments for methods.
`order`	The maximum order of serial correlation to test for.
`type`	The type of test statistic to use.

See Also

`lmtest::bgtest()`

common_xregs *Common exogenous regressors*

Description

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

Usage

```
common_xregs
```

Specials

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

```
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.

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

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

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

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

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

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

Arguments

`object` An estimated model.

`...` Unused.

Value

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

Examples

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

CROSTON *Croston's method*

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

```
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 α 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

Croston, J. (1972) "Forecasting and stock control for intermittent demands", *Operational Research Quarterly*, **23**(3), 289-303.

Shenstone, L., and Hyndman, R.J. (2005) "Stochastic models underlying Croston's method for intermittent demand forecasting". *Journal of Forecasting*, **24**, 389-402.

Kourentzes, N. (2014) "On intermittent demand model optimisation and selection". *International Journal of Production Economics*, **156**, 180-190. doi: 10.1016/j.ijpe.2014.06.007.

Examples

```
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*

Description

Returns ETS model specified by the formula.

Usage

```
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 multistep 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 `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.

```
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

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

```
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",

alpha The value of the smoothing parameter for the level. If alpha = 0, the level will not change over time. Convers

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. Convers

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

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").

```
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") or multiplicative ("M"). All specified methoc

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

gamma The value of the smoothing parameter for the seasonal pattern. If gamma = 0, the seasonal pattern will not char

gamma_range If gamma=NULL, gamma_range provides bounds for the optimised value of gamma.

References

Hyndman, R.J., Koehler, A.B., Snyder, R.D., and Grose, S. (2002) "A state space framework for automatic forecasting using exponential smoothing methods", *International J. Forecasting*, **18**(3), 439–454.

Hyndman, R.J., Akram, Md., and Archibald, B. (2008) "The admissible parameter space for exponential smoothing models". *Annals of Statistical Mathematics*, **60**(2), 407–426.

Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) *Forecasting with exponential smoothing: the state space approach*, Springer-Verlag. http://www.exponentialsmoothing.net.

See Also

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

Examples

```
as_tsibble(USAccDeaths) %>%
  model(ETS(log(value) ~ season("A")))
```

`fitted.AR` *Extract fitted values from a fable model*

Description

Extracts the fitted values.

Usage

```
## 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

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

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

Description

Extracts the fitted values.

Usage

```
## 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

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

`fitted.croston` *Extract fitted values from a fable model*

Description

Extracts the fitted values.

Usage

```
## 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

```
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()
```

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

Description

Extracts the fitted values.

Usage

```
## 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

```
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

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

---

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

Description

Extracts the fitted values.

Usage

```
## 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

```
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

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

---

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

Description

Extracts the fitted values.

Usage

```
## 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

```
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

```
## 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

```
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

```
## 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

```
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

```
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.

Usage

```
## S3 method for class 'ETS'
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 if simulated intervals are used.
`...`	Additional arguments for forecast model methods.

Value

A list of forecasts.

Examples

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

`forecast.fable_theta` *Forecast a model from the fable package*

Description

Produces forecasts from a trained model.

Usage

```
## 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

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

forecast.model_mean *Forecast a model from the fable package*

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

```
## 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

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

`forecast.RW` *Forecast a model from the fable package*

Description

Produces forecasts from a trained model.

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

```
## 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

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

`forecast.VAR` *Forecast a model from the fable package*

Description

Produces forecasts from a trained model.

Usage

```
## 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

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

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

generate.AR *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 'AR'
generate(x, new_data = NULL, 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(lh) %>%
  model(AR(value ~ order(3))) %>%
  generate()
```

generate.ARIMA *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 '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

```
fable_fit <- as_tsibble(USAccDeaths) %>%
  model(model = ARIMA(value ~ 0 + pdq(0,1,1) + PDQ(0,1,1)))
fable_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

```
## 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

```
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

```
## 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

```
as_tsibble(Nile) %>%
  model(NAIVE(value)) %>%
  generate()

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

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

```
## S3 method for class 'TSLM'
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

```
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

```
## 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 (sigma2), the log-likelihood (log_lik), and information criterion (AIC, AICc, BIC).

Value

A one row tibble summarising the model's fit.

Examples

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

glance.ARIMA *Glance an ARIMA model*

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 (sigma2), the log-likelihood (log_lik), information criterion (AIC, AICc, BIC) and the characteristic roots (ar_roots and ma_roots).

Value

A one row tibble summarising the model's fit.

Examples

```
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

```
## 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

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

`glance.fable_theta` *Glance a theta method*

Description

Construct a single row summary of the average method model.

Usage

```
## 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 (`sigma2`).

Value

A one row tibble summarising the model's fit.

glance.model_mean *Glance a average method model*

Description

Construct a single row summary of the average method model.

Usage

```
## 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 (`sigma2`).

Value

A one row tibble summarising the model's fit.

Examples

```
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 (`sigma2`).

Usage

```
## 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

```
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 (`sigma2`).

Usage

```
## 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

```
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

```
## 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_squared`), 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

```
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

```
## 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 (`sigma2`), the log-likelihood (`log_lik`), and information criterion (`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

```
## 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.model_mean`

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

```
library(tsibbledata)

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

interpolate.TSLM *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, ...)
```

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

MEAN *Mean models*

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.

```
AR(p = NULL, P = 1, period = NULL)
```

`p` The order of the non-seasonal auto-regressive (AR) terms. If `p = NULL`, an optimal number of lags will be selected fo

`P` The order of the seasonal auto-regressive (SAR) terms.

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

xreg: Exogenous regressors can be included in an NNETAR 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

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

Examples

```
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

```
## 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

```
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

```
## 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

```
## 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

```
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.model_mean *Refit a MEAN model*

Description

Applies a fitted average method model to a new dataset.

Usage

```
## S3 method for class 'model_mean'
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 mean for the fitted model will be re-estimated to suit the new data.
...	Additional arguments for forecast model methods.

Examples

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

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

report(fit)

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

refit.NNETAR *Refit a NNETAR model*

Description

Applies a fitted NNETAR model to a new dataset.

Usage

```
## S3 method for class 'NNETAR'
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 networks will be initialized with random starting weights to suit the new data. If `FALSE`, for every network the best individual set of weights found in the pre-estimation process is used as the starting weight vector.

`...` 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(NNETAR(value))

report(fit)

fit %>%
  refit(new_data = lung_deaths_female, reestimate = FALSE) %>%
  report()
```

`refit.RW` *Refit a lag walk model*

Description

Applies a fitted random walk model to a new dataset.

Usage

```
## S3 method for class 'RW'
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 lag walk model will be re-estimated to suit the new data.

`...` Additional arguments for forecast model methods.

Details

The models NAIVE and SNAIVE have no specific model parameters. Using refit for one of these models will provide the same estimation results as one would use fabletools::model(NAIVE(...)) (or fabletools::model(SNAIVE(...)).

Examples

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

fit <- lung_deaths_male %>%
  model(RW(value ~ drift()))

report(fit)

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

---

refit.TSLM *Refit a* TSLM

Description

Applies a fitted TSLM to a new dataset.

Usage

```
## 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

```
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 from a fable model*

Description

Extracts the residuals.

Usage

```
## 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 residuals to extract.
`...`	Additional arguments for forecast model methods.

Value

A vector of fitted residuals.

Examples

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

`residuals.ARIMA` *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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 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()
```

`residuals.croston` *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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 from a fable model*

Description

Extracts the residuals.

Usage

```
## 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

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

residuals.fable_theta *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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.

Value

A vector of fitted residuals.

Examples

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

---

residuals.model_mean *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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

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

residuals.NNETAR *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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

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

residuals.RW *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## S3 method for class 'RW'
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

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

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

`residuals.TSLM` *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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

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

residuals.VAR *Extract residuals from a fable model*

Description

Extracts the residuals.

Usage

```
## 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

```
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

```
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. Thi

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

```
library(tsibbledata)
```

```
aus_production %>%
  model(rw = RW(Beer ~ drift()))

as_tsibble(Nile) %>%
  model(NAIVE(value))
library(tsibbledata)
aus_production %>%
  model(snaive = SNAIVE(Beer ~ lag("year")))
```

THETA *Theta method*

Description

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.

```
season(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 se

`method` The type of classical decomposition to apply. The original Theta method always used multiplicative seasonal decom

Author(s)

Rob J Hyndman, Mitchell O'Hara-Wild

References

Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. *International Journal of Forecasting* **16**, 521-530.

Hyndman, R.J., and Billah, B. (2003) Unmasking the Theta method. *International J. Forecasting*, **19**, 287-290.

Examples

```
# Theta method with transform
deaths <- as_tsibble(USAccDeaths)
deaths %>%
  model(theta = THETA(log(value))) %>%
  forecast(h = "4 years") %>%
  autoplot(deaths)

# Compare seasonal specifications
library(tsibbledata)
library(dplyr)
aus_retail %>%
  filter(Industry == "Clothing retailing") %>%
  model(theta_multiplicative = THETA(Turnover ~ season(method = "multiplicative")),
        theta_additive = THETA(Turnover ~ season(method = "additive"))) %>%
  accuracy()
```

tidy.AR *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## 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

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

tidy.ARIMA *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## 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

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

tidy.croston *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## S3 method for class 'croston'
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

```
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()
```

`tidy.ETS` *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## 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

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

`tidy.fable_theta` *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## 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

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

`tidy.model_mean` *Tidy a fable model*

Description

Returns the coefficients from the model in a `tibble` format.

Usage

```
## 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.

Value

The model's coefficients in a tibble.

Examples

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

---

tidy.NNETAR *Tidy a fable model*

Description

Returns the coefficients from the model in a tibble format.

Usage

```
## 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

```
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

```
## 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

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

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

```
## S3 method for class 'TSLM'
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

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

---

tidy.VAR *Tidy a fable model*

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 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

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

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

```
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) that provides a p-value for the unit root test. 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 . or .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

```
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

```
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

∗ **datasets**
common_xregs, 8

AR, 4
ARIMA, 5

breusch_godfrey, 8

common_xregs, *4*, *7*, 8, *46*, *69*, *71*
components.ETS, 9
CROSTON, 10

ETS, 11

fabletools::dable(), *9*
fabletools::forecast.mdl_df(), *21–32*, *34*, *35*, *42–44*, *47–52*
fabletools::generate.mdl_df, *31–36*
fabletools::interpolate(), *13*
feasts::feat_stl(), *70*
feasts::unitroot_kpss(), *70*
fitted.AR, 14
fitted.ARIMA, 15
fitted.croston, 15
fitted.ETS, 16
fitted.fable_theta, 17
fitted.model_mean, 17
fitted.NNETAR, 18
fitted.RW, 19
fitted.TSLM, 19
fitted.VAR, 20
forecast.AR, 21
forecast.ARIMA, 22
forecast.croston, 23
forecast.ETS, 23
forecast.fable_theta, 24
forecast.model_mean, 25
forecast.NNETAR, 26
forecast.RW, 27
forecast.TSLM, 29
forecast.VAR, 30
forecast::Arima(), *6*

generate.AR, 31
generate.ARIMA, 31
generate.ETS, 32
generate.model_mean, 33
generate.NNETAR, 34
generate.RW, 34
generate.TSLM, 35
glance.AR, 36
glance.ARIMA, 37
glance.ETS, 37
glance.fable_theta, 38
glance.model_mean, 39
glance.NNETAR, 39
glance.RW, 40
glance.TSLM, 41
glance.VAR, 41

interpolate.ARIMA, 42
interpolate.model_mean, 43
interpolate.TSLM, 44

lmtest::bgtest(), *8*

MEAN, 44

NAIVE *(*RW*)*, 59
NNETAR, 45

refit.AR, 47
refit.ARIMA, 48
refit.ETS, 49
refit.model_mean, 50
refit.NNETAR, 50
refit.RW, 51
refit.TSLM, 52
report.AR *(*AR*)*, 4
report.ARIMA *(*ARIMA*)*, 5
report.ETS *(*ETS*)*, 11
report.model_mean *(*MEAN*)*, 44

report.NNETAR *(*NNETAR*)*, 45
report.RW *(*RW*)*, 59
report.TSLM *(*TSLM*)*, 69
report.VAR *(*VAR*)*, 70
residuals.AR, 53
residuals.ARIMA, 53
residuals.croston, 54
residuals.ETS, 55
residuals.fable_theta, 55
residuals.model_mean, 56
residuals.NNETAR, 57
residuals.RW, 57
residuals.TSLM, 58
residuals.VAR, 59
RW, 59

SNAIVE *(*RW*)*, 59
stats::ar.ols(), *4*
stats::arima(), *5, 6*
stats::lm(), *4*, *7*, *46*, *69*, *71*
stats::model.frame(), *4*, *7*, *46*, *69*, *71*
stats::model.matrix(), *69*

THETA, 61
tibble::tibble(), *62–68*
tidy.AR, 62
tidy.ARIMA, 63
tidy.croston, 63
tidy.ETS, 64
tidy.fable_theta, 65
tidy.model_mean, 65
tidy.NNETAR, 66
tidy.RW, 67
tidy.TSLM, 67
tidy.VAR, 68
tidyr::fill(), *13*
TSLM, 69

unitroot_options, 70
unitroot_options(), *6*

VAR, 70