Package ‘fable’

September 23, 2019

Title Forecasting Models for Tidy Time Series

Version 0.1.0

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

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

Depends R (>= 3.4.0), fabletools

Imports Rcpp (>= 0.11.0), rlang (>= 0.2.0), stats, dplyr (>= 0.8.0), tsibble (>= 0.8.0), tidyr

Suggests covr, feasts, forecast, knitr, nnet, rmarkdown, spelling, testthat, tsibbledata

LinkingTo Rcpp (>= 0.11.0)

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 6.1.1

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 2019-09-23 15:10:06 UTC

R topics documented:

`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, 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` and `Q` to consider in the search.
`unitroot_spec`	A specification of unit root tests to use in the selection of `d` and `D`. See `unitroot_options()` for more details.
`...`	Further arguments for `stats::arima()`

Value

A model specification.

Specials

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)
```

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.

PDQ: The PDQ special is used to specify seasonal components of the model.

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

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.

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, 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) ~ pdq(0,1,1) + PDQ(0,1,1)))

# Automatic ARIMA specification
```

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

`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()
```

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 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.
`...`	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

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").

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 s
`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.

```
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").

`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.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.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.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(log(mdeaths), fdeaths) ~ AR(3))) %>%
  fitted()
```

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 boostrap = 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.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.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 `boostrap = 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,
  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 `boostrap = 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 boostrap = 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 `boostrap = 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(log(mdeaths), fdeaths) ~ AR(3))) %>%
  forecast()
```

---

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.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.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(log(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.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

This model does not support usage of any specials. It only computes the mean!

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 `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.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.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.ARIMA *Extract residuals values 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 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()
```

residuals.ETS *Extract residuals values 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.model_mean` *Extract residuals values 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 values 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 values 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 values 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 values 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(log(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")))
```

`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.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.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(log(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

Options for the unit root tests for order of integration

Usage

```
unitroot_options(ndiffs_alpha = 0.05, nsdiffs_alpha = 0.05,
  ndiffs_pvalue = ~feasts::unitroot_kpss(.)["kpss_pvalue"],
  nsdiffs_pvalue = ~feasts::feat_stl(., .period)[2] < 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 `.` 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(log(mdeaths), fdeaths) ~ AR(3)))

report(fit)

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

Index