

Package: tscv (via r-universe)

May 13, 2026

Title Functions and Utilities for Tidy Time Series Forecasting and Time Series Cross-Validation

Version 1.0.0

Description Provides functions and tools for tidy time series analysis and forecasting as well as time series cross-validation. This is mainly a set of wrapper and helper functions as well as some extensions for the packages 'tsibble', 'fable', and 'fabletools'.

License GPL-3

URL <https://github.com/ahaeusser/tscv>,
<https://ahaeusser.github.io/tscv/>

BugReports <https://github.com/ahaeusser/tscv/issues>

Depends R (>= 4.1.0)

Imports rlang, dplyr, ggplot2, grDevices, tidyr, tidyselect, tibble, slider, lubridate, purrr, tsibble, forecast, fabletools, stats, qqplotr, scales, distributional, tidytext

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, tidyverse, fable, feasts, testthat (>= 3.0.0), covr

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Alexander Häußler [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0000-5419-8479>)

Maintainer Alexander Häußler <alexander-haeusser@gmx.de>

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

Date/Publication 2026-05-13 10:48:35 UTC

RemoteUrl <https://github.com/cran/tscv>

RemoteRef HEAD

RemoteSha b1feceb0c2e30549071daff775b66b33cdd4980e

Contents

acf_vec	3
check_data	4
DSHW	6
elec_load	7
elec_price	7
estimate_acf	8
estimate_kurtosis	9
estimate_mode	10
estimate_pacf	11
estimate_skewness	12
fitted.DSHW	13
fitted.MEDIAN	14
fitted.SMEAN	15
fitted.SMEDIAN	16
fitted.SNAIVE2	17
fitted.TBATS	18
forecast.DSHW	19
forecast.MEDIAN	20
forecast.SMEAN	21
forecast.SMEDIAN	22
forecast.SNAIVE2	23
forecast.TBATS	24
interpolate_missing	25
M4_monthly_data	26
M4_quarterly_data	27
mae_vec	27
make_accuracy	28
make_errors	31
make_future	33
make_split	35
make_tsibble	37
mape_vec	38
me_vec	39
MEDIAN	40
model_sum.DSHW	41
model_sum.MEDIAN	42
model_sum.SMEAN	43
model_sum.SMEDIAN	44
model_sum.SNAIVE2	45
model_sum.TBATS	46
mpe_vec	47

mse_vec	48
pacf_vec	49
plot_bar	50
plot_density	52
plot_histogram	55
plot_line	57
plot_point	60
plot_qq	62
residuals.DSHW	65
residuals.MEDIAN	66
residuals.SMEAN	67
residuals.SMEDIAN	68
residuals.SNAIVE2	69
residuals.TBATS	70
rmse_vec	71
scale_color_tscv	72
scale_fill_tscv	73
slice_test	74
slice_train	76
smape_vec	77
SMEAN	78
SMEDIAN	79
smooth_outlier	80
SNAIVE2	81
split_index	82
summarise_data	84
summarise_split	85
summarise_stats	87
TBATS	89
theme_tscv	90
tscv_cols	91
tscv_pal	93
Index	95

 acf_vec

Estimate autocorrelations of a numeric vector

Description

Estimate the sample autocorrelation function of a numeric vector.

Usage

```
acf_vec(x, lag_max = 24, ...)
```

Arguments

x	Numeric vector.
lag_max	Integer. Maximum lag for which the autocorrelation is estimated.
...	Further arguments passed to <code>stats::acf()</code> .

Details

`acf_vec()` is a small wrapper around `stats::acf()`. It returns the sample autocorrelations as a numeric vector and removes lag 0 from the output, because lag 0 is always equal to 1 and is usually not needed for diagnostics.

Value

A numeric vector containing the sample autocorrelations for lags 1 to `lag_max`.

See Also

Other data analysis: [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)

x <- M4_monthly_data |>
  filter(series == first(series)) |>
  pull(value)

acf_vec(
  x = x,
  lag_max = 12
)
```

check_data

Check and prepare tsibble data

Description

Check that the input data is a valid regular and ordered tsibble, fill implicit gaps if requested, and convert wide data to long format.

Usage

```
check_data(data, fill_missing = TRUE)
```

Arguments

<code>data</code>	A tsibble in long or wide format.
<code>fill_missing</code>	Logical value. If TRUE, implicit missing values are turned into explicit missing values with <code>fill_gaps()</code> .

Details

`check_data()` is a data preparation helper for time series workflows. It performs three tasks:

- checks that data is a tsibble;
- checks that the time index is regular and ordered by key and index;
- optionally turns implicit missing values into explicit missing values using `fill_gaps()`.

If the input data has no key variables, it is treated as wide data and is converted to long format. The resulting output contains the original index column, a `variable` column containing the former column names, and a `value` column containing the corresponding observations.

Existing explicit missing values are not changed.

Value

A tsibble prepared for downstream use. Wide data is returned in long format with one measurement variable named `value`.

See Also

Other data preparation: [interpolate_missing\(\)](#), [smooth_outlier\(\)](#)

Examples

```
library(dplyr)
library(tsibble)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395")) |>
  as_tsibble(
    index = index,
    key = series
  )

check_data(data)

wide_data <- data |>
  as_tibble() |>
  select(index, series, value) |>
  tidyr::pivot_wider(
    names_from = series,
    values_from = value
  ) |>
  as_tsibble(index = index)

check_data(wide_data)
```

DSHW

Double Seasonal Holt-Winters model

Description

Specify a Double Seasonal Holt-Winters model for use with `fabletools::model()`.

Usage

```
DSHW(formula, ...)
```

Arguments

`formula` A model formula specifying the response variable, for example `value`.
`...` Further arguments passed to `forecast::dshw()`, including `periods`.

Details

`DSHW()` is a model specification wrapper around `forecast::dshw()` for the `fable`, `tsibble`, and `fabletools` ecosystem.

The model is useful for time series with two important seasonal patterns, such as hourly data with daily and weekly seasonality.

The seasonal periods must be supplied through the `periods` argument.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other DSHW: [fitted.DSHW\(\)](#), [forecast.DSHW\(\)](#), [model_sum.DSHW\(\)](#), [residuals.DSHW\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_load |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 28) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("DSHW" = DSHW(value, periods = c(24, 168)))

model_frame
```

elec_load	<i>Hourly electricity load (actual values and forecasts)</i>
-----------	--

Description

Hourly tibble with actual electricity loads and load forecasts from the ENTSO-E Transparency Platform. The data set contains time series data from 2019-01-01 00:00:00 to 2019-12-31 23:00:00 for 8 bidding zones within Europe (DE, DK1, ES, FI, FR, NL, NO1, SE1). The original data are on a quarter-hourly basis (15-minutes interval), but aggregated to hourly data.

Usage

```
data(elec_load)
```

Format

A time series object of class tibble with 140.160 rows and 5 columns:

- time: Date and time index
- item: Time series name
- unit: Measured unit
- bidding_zone: Bidding zone
- value: Measurement variable

Source

[ENTSO-E Transparency Platform](#)

Examples

```
data(elec_load)
```

elec_price	<i>Hourly day-ahead electricity spot prices</i>
------------	---

Description

Hourly tibble with day-ahead electricity spot prices from the ENTSO-E Transparency Platform. The data set contains time series data from 2019-01-01 00:00:00 to 2020-12-31 23:00:00 for 8 bidding zones within Europe (DE, DK1, ES, FI, FR, NL, NO1, SE1).

Usage

```
data(elec_price)
```

Format

A time series object of class tibble with 140.352 rows and 5 columns:

- time: Date and time index
- item: Time series name
- unit: Measured unit
- bidding_zone: Bidding zone
- value: Measurement variable

Source

[ENTSO-E Transparency Platform](#)

Examples

```
data(elec_price)
```

estimate_acf	<i>Estimate autocorrelations by time series</i>
--------------	---

Description

Estimate the sample autocorrelation function for one or more time series in a tibble.

Usage

```
estimate_acf(.data, context, lag_max = 24, level = 0.9, ...)
```

Arguments

.data	A tibble containing the time series data.
context	A named list with the identifiers for series_id, value_id, and index_id.
lag_max	Integer. Maximum lag for which the autocorrelation is estimated.
level	Numeric value. Confidence level used to calculate the approximate significance bound.
...	Further arguments passed to stats::acf().

Details

estimate_acf() groups the input data by the series identifier supplied in context and estimates the sample autocorrelation function for each time series separately.

The output contains one row per series and lag. The column bound contains an approximate significance threshold based on the selected confidence level. The logical column sign indicates whether the absolute autocorrelation is larger than this threshold.

Value

A tibble with the series identifier and the columns type, lag, value, bound, and sign.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

estimate_acf(
  .data = data,
  context = context,
  lag_max = 12
)
```

estimate_kurtosis	<i>Estimate kurtosis</i>
-------------------	--------------------------

Description

Estimate the kurtosis of a numeric distribution.

Usage

```
estimate_kurtosis(x, na_rm = TRUE)
```

Arguments

x	Numeric vector.
na_rm	Logical value. If TRUE, missing values are removed before estimation.

Details

The function computes the moment-based kurtosis

$$\frac{n \sum_i (x_i - \bar{x})^4}{(\sum_i (x_i - \bar{x})^2)^2}$$

Missing values are removed by default.

This returns the usual kurtosis, not excess kurtosis. A normal distribution has kurtosis close to 3.

Value

A numeric value giving the estimated kurtosis.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
x <- c(1, 2, 3, 4, 5, NA)

estimate_kurtosis(x)
estimate_kurtosis(x, na_rm = TRUE)

set.seed(123)
y <- rnorm(100)
estimate_kurtosis(y)
```

estimate_mode

Estimate the mode of a distribution

Description

Estimate the mode of a numeric distribution using kernel density estimation.

Usage

```
estimate_mode(x, na_rm = TRUE, ...)
```

Arguments

x	Numeric vector.
na_rm	Logical value. If TRUE, missing values are removed before estimation.
...	Further arguments passed to <code>stats::density()</code> .

Details

The function computes a kernel density estimate with `stats::density()` and returns the value of `x` at which the estimated density is largest.

Missing values are removed by default. Additional arguments are passed to `stats::density()`, for example `bw`, `kernel`, or `n`.

Value

A numeric value giving the estimated mode of the distribution.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
x <- c(1, 1, 2, 2, 2, 3, 4, NA)

estimate_mode(x)
estimate_mode(x, na_rm = TRUE)
estimate_mode(x, bw = "nrd0")

set.seed(123)
y <- rnorm(100, mean = 5)
estimate_mode(y)
```

 estimate_pacf

Estimate partial autocorrelations by time series

Description

Estimate the sample partial autocorrelation function for one or more time series in a tibble.

Usage

```
estimate_pacf(.data, context, lag_max = 24, level = 0.9, ...)
```

Arguments

<code>.data</code>	A tibble containing the time series data.
<code>context</code>	A named list with the identifiers for <code>series_id</code> , <code>value_id</code> , and <code>index_id</code> .
<code>lag_max</code>	Integer. Maximum lag for which the partial autocorrelation is estimated.
<code>level</code>	Numeric value. Confidence level used to calculate the approximate significance bound.
<code>...</code>	Further arguments passed to <code>stats::pacf()</code> .

Details

`estimate_pacf()` groups the input data by the series identifier supplied in `context` and estimates the sample partial autocorrelation function for each time series separately.

The output contains one row per series and lag. The column `bound` contains an approximate significance threshold based on the selected confidence level. The logical column `sign` indicates whether the absolute partial autocorrelation is larger than this threshold.

Value

A tibble with the series identifier and the columns `type`, `lag`, `value`, `bound`, and `sign`.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

estimate_pacf(
  .data = data,
  context = context,
  lag_max = 12
)
```

<code>estimate_skewness</code>	<i>Estimate skewness</i>
--------------------------------	--------------------------

Description

Estimate the skewness of a numeric distribution.

Usage

```
estimate_skewness(x, na_rm = TRUE)
```

Arguments

x	Numeric vector.
na_rm	Logical value. If TRUE, missing values are removed before estimation.

Details

The function computes the moment-based skewness

$$\frac{\frac{1}{n} \sum_i (x_i - \bar{x})^3}{\left(\frac{1}{n} \sum_i (x_i - \bar{x})^2\right)^{3/2}}$$

Missing values are removed by default. Positive values indicate a distribution with a longer or heavier right tail; negative values indicate a distribution with a longer or heavier left tail.

Value

A numeric value giving the estimated skewness.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
x <- c(1, 2, 3, 4, 10, NA)

estimate_skewness(x)
estimate_skewness(x, na_rm = TRUE)

set.seed(123)
y <- rexp(100)
estimate_skewness(y)
```

fitted.DSHW

Extract fitted values from a DSHW model

Description

Extract fitted values from a fitted DSHW model.

Usage

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

Arguments

object A fitted DSHW model object.
 ... Additional arguments. Currently not used.

Value

Fitted values.

See Also

Other DSHW: [DSHW\(\)](#), [forecast.DSHW\(\)](#), [model_sum.DSHW\(\)](#), [residuals.DSHW\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_load |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 28) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("DSHW" = DSHW(value, periods = c(24, 168)))

fitted(model_frame)
```

fitted.MEDIAN

Extract fitted values from a median model

Description

Extract fitted values from a fitted MEDIAN model.

Usage

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

Arguments

object A fitted MEDIAN model object.
 ... Additional arguments. Currently not used.

Value

Fitted values.

See Also

Other MEDIAN: [MEDIAN\(\)](#), [forecast.MEDIAN\(\)](#), [model_sum.MEDIAN\(\)](#), [residuals.MEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("MEDIAN" = MEDIAN(value ~ window()))

fitted(model_frame)
```

fitted.SMEAN

Extract fitted values from a seasonal mean model

Description

Extract fitted values from a fitted SMEAN model.

Usage

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

Arguments

object A fitted SMEAN model object.
... Additional arguments. Currently not used.

Value

Fitted values.

See Also

Other SMEAN: [SMEAN\(\)](#), [forecast.SMEAN\(\)](#), [model_sum.SMEAN\(\)](#), [residuals.SMEAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("SMEAN" = SMEAN(value ~ lag("year")))

fitted(model_frame)
```

fitted.SMEDIAN

Extract fitted values from a seasonal median model

Description

Extract fitted values from a fitted SMEDIAN model.

Usage

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

Arguments

`object` A fitted SMEDIAN model object.
`...` Additional arguments. Currently not used.

Value

Fitted values.

See Also

Other SMEDIAN: [SMEDIAN\(\)](#), [forecast.SMEDIAN\(\)](#), [model_sum.SMEDIAN\(\)](#), [residuals.SMEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 21) |>
  as_tsibble(index = time)
```

```
model_frame <- train_frame |>
  model("SMEDIAN" = SMEDIAN(value ~ lag("week")))

fitted(model_frame)
```

fitted.SNAIVE2	<i>Extract fitted values from a SNAIVE2 model</i>
----------------	---

Description

Extract fitted values from a fitted SNAIVE2 model.

Usage

```
## S3 method for class 'SNAIVE2'
fitted(object, ...)
```

Arguments

object	A fitted SNAIVE2 model object.
...	Additional arguments. Currently not used.

Value

Fitted values.

See Also

Other SNAIVE2: [SNAIVE2\(\)](#), [forecast.SNAIVE2\(\)](#), [model_sum.SNAIVE2\(\)](#), [residuals.SNAIVE2\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 21) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("SNAIVE2" = SNAIVE2(value))

fitted(model_frame)
```

`fitted.TBATS`*Extract fitted values from a TBATS model*

Description

Extract fitted values from a fitted TBATS model.

Usage

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

Arguments

<code>object</code>	A fitted TBATS model object.
<code>...</code>	Additional arguments. Currently not used.

Details

This method is used by `fitted()` when extracting fitted values from a mable containing a TBATS model.

Value

Fitted values.

See Also

Other TBATS: [TBATS\(\)](#), [forecast.TBATS\(\)](#), [model_sum.TBATS\(\)](#), [residuals.TBATS\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("TBATS" = TBATS(value, periods = c(24, 168)))  
  
fitted(model_frame)
```

forecast.DSHW	<i>Forecast a DSHW model</i>
---------------	------------------------------

Description

Forecast a fitted DSHW model.

Usage

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

Arguments

object	A fitted DSHW model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Value

A vector of forecast distributions.

See Also

Other DSHW: [DSHW\(\)](#), [fitted.DSHW\(\)](#), [model_sum.DSHW\(\)](#), [residuals.DSHW\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_load |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 28) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("DSHW" = DSHW(value, periods = c(24, 168)))  
  
forecast(model_frame, h = 24)
```

forecast.MEDIAN	<i>Forecast a median model</i>
-----------------	--------------------------------

Description

Forecast a fitted MEDIAN model.

Usage

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

Arguments

object	A fitted MEDIAN model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Value

A vector of forecast distributions.

See Also

Other MEDIAN: [MEDIAN\(\)](#), [fitted.MEDIAN\(\)](#), [model_sum.MEDIAN\(\)](#), [residuals.MEDIAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- M4_monthly_data |>  
  filter(series == first(series)) |>  
  as_tsibble(index = index)  
  
model_frame <- train_frame |>  
  model("MEDIAN" = MEDIAN(value ~ window()))  
  
forecast(model_frame, h = 12)
```

forecast.SMEAN	<i>Forecast a seasonal mean model</i>
----------------	---------------------------------------

Description

Forecast a fitted SMEAN model.

Usage

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

Arguments

object	A fitted SMEAN model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Value

A vector of forecast distributions.

See Also

Other SMEAN: [SMEAN\(\)](#), [fitted.SMEAN\(\)](#), [model_sum.SMEAN\(\)](#), [residuals.SMEAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- M4_monthly_data |>  
  filter(series == first(series)) |>  
  as_tsibble(index = index)  
  
model_frame <- train_frame |>  
  model("SMEAN" = SMEAN(value ~ lag("year")))  
  
forecast(model_frame, h = 12)
```

forecast.SMEDIAN *Forecast a seasonal median model*

Description

Forecast a fitted SMEDIAN model.

Usage

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

Arguments

object	A fitted SMEDIAN model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Value

A vector of forecast distributions.

See Also

Other SMEDIAN: [SMEDIAN\(\)](#), [fitted.SMEDIAN\(\)](#), [model_sum.SMEDIAN\(\)](#), [residuals.SMEDIAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SMEDIAN" = SMEDIAN(value ~ lag("week")))  
  
forecast(model_frame, h = 24)
```

forecast.SNAIVE2	<i>Forecast a SNAIVE2 model</i>
------------------	---------------------------------

Description

Forecast a fitted SNAIVE2 model.

Usage

```
## S3 method for class 'SNAIVE2'  
forecast(object, new_data, specials = NULL, ...)
```

Arguments

object	A fitted SNAIVE2 model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Value

A vector of forecast distributions.

See Also

Other SNAIVE2: [SNAIVE2\(\)](#), [fitted.SNAIVE2\(\)](#), [model_sum.SNAIVE2\(\)](#), [residuals.SNAIVE2\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SNAIVE2" = SNAIVE2(value))  
  
forecast(model_frame, h = 24)
```

forecast.TBATS *Forecast a TBATS model*

Description

Forecast a fitted TBATS model.

Usage

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

Arguments

object	A fitted TBATS model object.
new_data	A tsibble containing future time points.
specials	Parsed specials. Currently not used.
...	Additional arguments. Currently not used.

Details

This method is used by `forecast()` when forecasting a mable containing a TBATS model.

Value

A vector of forecast distributions.

See Also

Other TBATS: [TBATS\(\)](#), [fitted.TBATS\(\)](#), [model_sum.TBATS\(\)](#), [residuals.TBATS\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("TBATS" = TBATS(value, periods = c(24, 168)))  
  
forecast(model_frame, h = 24)
```

interpolate_missing *Interpolate missing values*

Description

Interpolate missing values in a numeric time series.

Usage

```
interpolate_missing(x, periods, ...)
```

Arguments

x	Numeric vector containing the time series observations.
periods	Numeric vector giving the seasonal periods of the time series, for example 12 for monthly data with yearly seasonality or <code>c(24, 168)</code> for hourly data with daily and weekly seasonality.
...	Further arguments passed to <code>forecast::msts()</code> or <code>forecast::na.interp()</code> .

Details

`interpolate_missing()` is a small wrapper around `forecast::na.interp()`. The input vector is first converted to an `msts` object using the seasonal periods supplied in `periods`.

For non-seasonal time series, missing values are replaced using linear interpolation. For seasonal time series, `forecast::na.interp()` uses an STL-based approach: the series is decomposed, the seasonally adjusted series is interpolated, and the seasonal component is added back.

The function returns a plain numeric vector with the same length as the input.

Value

A numeric vector with missing values interpolated.

See Also

Other data preparation: [check_data\(\)](#), [smooth_outlier\(\)](#)

Examples

```
library(dplyr)

x <- M4_monthly_data |>
  filter(series == first(series)) |>
  pull(value)

x_missing <- x
x_missing[c(10, 20, 30)] <- NA
```

```
x_interpolated <- interpolate_missing(  
  x = x_missing,  
  periods = 12  
)  
  
anyNA(x_missing)  
anyNA(x_interpolated)  
  
hourly <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 14) |>  
  pull(value)  
  
hourly_missing <- hourly  
hourly_missing[c(24, 48, 72)] <- NA  
  
interpolate_missing(  
  x = hourly_missing,  
  periods = c(24, 168)  
)
```

M4_monthly_data

Monthly time series data from the M4 Competition

Description

The data set contains 30 selected time series on a monthly basis from the M4 Competition.

Usage

```
data(M4_monthly_data)
```

Format

A time series object of class `tibble` with 7881 rows and 4 columns:

- `index`: Date and time index
- `series`: Time series ID from M4 forecasting competition
- `category`: Category from M4 forecasting competition
- `value`: Measurement variable

Source

[M4 Competition](#)

Examples

```
data(M4_monthly_data)
```

M4_quarterly_data	<i>Quarterly time series data from the M4 Competition</i>
-------------------	---

Description

The data set contains 30 selected time series on a quarterly basis from the M4 Competition.

Usage

```
data(M4_quarterly_data)
```

Format

A time series object of class tibble with 2818 rows and 4 columns:

- index: Date and time index
- series: Time series ID from M4 forecasting competition
- category: Category from M4 forecasting competition
- value: Measurement variable

Source

[M4 Competition](#)

Examples

```
data(M4_quarterly_data)
```

mae_vec	<i>Calculate the mean absolute error</i>
---------	--

Description

Calculate the mean absolute error of a numeric vector.

Usage

```
mae_vec(truth, estimate, na_rm = TRUE)
```

Arguments

truth	Numeric vector containing the actual values.
estimate	Numeric vector containing the forecasts.
na_rm	Logical value. If TRUE, missing values are removed.

Details

`mae_vec()` computes the average absolute forecast error `abs(truth - estimate)`. The metric is reported in the same units as the original data.

Value

A numeric value.

See Also

Other accuracy functions: [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 30)
estimate <- c(8, 22, 25)

mae_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
mae_vec(truth_na, estimate_na)
```

<code>make_accuracy</code>	<i>Estimate point forecast accuracy</i>
----------------------------	---

Description

Estimate accuracy metrics for point forecasts generated from rolling-origin time series cross-validation.

Usage

```
make_accuracy(
  future_frame,
  main_frame,
  context,
  dimension = "split",
  benchmark = NULL
)
```

Arguments

<code>future_frame</code>	A tibble containing point forecasts. It must contain the columns specified by <code>context</code> , as well as <code>model</code> , <code>split</code> , <code>horizon</code> , and <code>point</code> .
<code>main_frame</code>	A tibble containing the observed values. It must contain the series identifier, time index, and value column specified by <code>context</code> .

context	A named list with the identifiers for <code>series_id</code> , <code>value_id</code> , and <code>index_id</code> .
dimension	Character value. Determines the dimension over which accuracy is summarized. Common choices are "split" and "horizon".
benchmark	Optional character value giving the model name used as the benchmark for the relative mean absolute error rMAE.

Details

`make_accuracy()` compares point forecasts in `future_frame` with the observed values in `main_frame`. The two data sets are joined using the series identifier and time index defined in `context`.

Accuracy can be summarized along different cross-validation dimensions:

- `dimension = "split"` summarizes accuracy separately for each test split.
- `dimension = "horizon"` summarizes accuracy separately for each forecast horizon.

The following point forecast accuracy metrics are returned:

- ME: mean error.
- MAE: mean absolute error.
- MSE: mean squared error.
- RMSE: root mean squared error.
- MAPE: mean absolute percentage error.
- sMAPE: symmetric mean absolute percentage error.
- MPE: mean percentage error.

If `benchmark` is supplied, the function also computes the relative mean absolute error rMAE. The rMAE is calculated as the model's MAE divided by the MAE of the benchmark model for the same series and selected dimension.

Value

A tibble containing the forecast accuracy metrics. The output contains the series identifier, model name, selected dimension, dimension value `n`, metric name, and metric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)
library(fable)

context <- list(
  series_id = "series",
```

```
value_id = "value",
index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

train_frame <- slice_train(
  main_frame = main_frame,
  split_frame = split_frame,
  context = context
) |>
  as_tsibble(
    index = index,
    key = c(series, split)
  )

model_frame <- train_frame |>
  model(
    "SNAIVE" = SNAIVE(value ~ lag("year"))
  )

fable_frame <- model_frame |>
  forecast(h = 18)

future_frame <- make_future(
  fable = fable_frame,
  context = context
)

accuracy_horizon <- make_accuracy(
  future_frame = future_frame,
  main_frame = main_frame,
  context = context,
  dimension = "horizon"
)

accuracy_horizon

accuracy_split <- make_accuracy(
  future_frame = future_frame,
```

```

    main_frame = main_frame,
    context = context,
    dimension = "split"
  )

  accuracy_split

```

 make_errors

Calculate forecast errors and percentage errors

Description

Calculate forecast errors and percentage forecast errors for point forecasts.

Usage

```
make_errors(future_frame, main_frame, context)
```

Arguments

future_frame	A tibble containing the forecasts. It must contain the columns specified by context, as well as model, split, horizon, and point.
main_frame	A tibble containing the observed values. It must contain the series identifier, time index, and value column specified by context.
context	A named list with the identifiers for series_id, value_id, and index_id.

Details

make_errors() compares point forecasts in future_frame with the observed values in main_frame. The two data sets are joined by the series identifier and time index specified in context.

The forecast error is calculated as $\text{error} = \text{actual} - \text{point}$. The percentage forecast error is calculated as $\text{pct_error} = (\text{actual} - \text{point} / \text{point}) * 100$.

Positive errors indicate that the forecast is below the observed value. Negative errors indicate that the forecast is above the observed value.

The returned data contains:

- series_id: Unique identifier for the time series as specified in context.
- model: Forecasting model name.
- split: Train-test split identifier.
- horizon: Forecast horizon.
- error: Forecast error.
- pct_error: Percentage forecast error.

Value

A tibble containing forecast errors and percentage forecast errors.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)
library(fable)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

train_frame <- slice_train(
  main_frame = main_frame,
  split_frame = split_frame,
  context = context
) |>
  as_tsibble(
    index = index,
    key = c(series, split)
  )

model_frame <- train_frame |>
  model(
    "SNAIVE" = SNAIVE(value ~ lag("year"))
  )

fable_frame <- model_frame |>
  forecast(h = 18)

future_frame <- make_future(
  fable = fable_frame,
```

```

    context = context
  )

  error_frame <- make_errors(
    future_frame = future_frame,
    main_frame = main_frame,
    context = context
  )

  error_frame

```

 make_future

Convert forecasts to a future frame

Description

Convert forecasts from a fable object to a standardized forecast table.

Usage

```
make_future(fable, context)
```

Arguments

fable	A fable created with <code>forecast()</code> .
context	A named list with the identifiers for <code>series_id</code> , <code>value_id</code> , and <code>index_id</code> .

Details

`make_future()` converts the output of `forecast()` into a tibble with a consistent structure for downstream evaluation, plotting, and accuracy calculation.

The returned `future_frame` contains one row per forecasted observation, time series, split, and model. It includes the following columns:

- the time index column specified by `context$index_id`;
- the series identifier column specified by `context$series_id`;
- `model`: the forecasting model name;
- `split`: the train-test split identifier;
- `horizon`: the forecast horizon within each series, split, and model;
- `point`: the point forecast, taken from the `.mean` column of the fable.

This format is used by functions such as `make_accuracy()` and `make_errors()`.

Value

A tibble containing forecasts in standardized `future_frame` format.

See Also

Other time series cross-validation: [make_split\(\)](#), [make_tsibble\(\)](#), [slice_test\(\)](#), [slice_train\(\)](#), [split_index\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fable)
library(fabletools)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

train_frame <- slice_train(
  main_frame = main_frame,
  split_frame = split_frame,
  context = context
) |>
  as_tsibble(
    index = index,
    key = c(series, split)
  )

model_frame <- train_frame |>
  model(
    "SNAIVE" = SNAIVE(value ~ lag("year"))
  )

fable_frame <- model_frame |>
  forecast(h = 18)

future_frame <- make_future(
  fable = fable_frame,
```

```

    context = context
  )

  future_frame

```

make_split

Create train-test splits for time series cross-validation

Description

Create a split frame with train and test indices for one or more time series.

Usage

```

make_split(
  main_frame,
  context,
  type,
  value,
  n_ahead,
  n_skip = 0,
  n_lag = 0,
  mode = "slide",
  exceed = TRUE
)

```

Arguments

main_frame	A tibble containing the time series data.
context	A named list with the identifiers for series_id, value_id, and index_id.
type	Character value. The type of initial split. Possible values are "first", "last", and "prob".
value	Numeric value specifying the initial split.
n_ahead	Integer. The forecast horizon, i.e. the number of observations in each test window.
n_skip	Integer. The number of observations to skip between split origins. The default is 0.
n_lag	Integer. The number of lagged observations to include before the test window. This is useful if lagged predictors are required when constructing test features. The default is 0.
mode	Character value. Either "slide" for a fixed-window approach or "stretch" for an expanding-window approach.
exceed	Logical value. If TRUE, out-of-sample splits exceeding the original sample size are created.

Details

`make_split()` creates rolling-origin train-test splits for time series cross-validation. The output is used by functions such as `slice_train()` and `slice_test()` to extract the corresponding training and testing samples from `main_frame`.

The function supports two training-window modes:

- `mode = "slide"` creates a fixed-window approach. The training window has constant length and moves forward over time.
- `mode = "stretch"` creates an expanding-window approach. The training window starts at the first observation and grows over time.

The initial training window is controlled by `type` and `value`:

- `type = "first"` uses the first value observations as the initial training window.
- `type = "last"` keeps the last value observations for testing and derives the initial training window from the remaining sample.
- `type = "prob"` uses `floor(value * n_total)` observations as the initial training window.

The argument `n_skip` controls how far the rolling origin moves between consecutive splits. For non-overlapping test windows, use `n_skip = n_ahead - 1`.

Value

A tibble containing the split plan. The output has one row per time series and split, with list-columns `train` and `test` containing integer row positions.

See Also

Other time series cross-validation: [make_future\(\)](#), [make_tsibble\(\)](#), [slice_test\(\)](#), [slice_train\(\)](#), [split_index\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series == "M23100")

# Fixed-window split plan
fixed_split <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
```

```
n_ahead = 18,  
n_skip = 17,  
n_lag = 0,  
mode = "slide",  
exceed = FALSE  
)  
  
fixed_split  
  
# Expanding-window split plan  
expanding_split <- make_split(  
  main_frame = main_frame,  
  context = context,  
  type = "first",  
  value = 120,  
  n_ahead = 18,  
  n_skip = 17,  
  n_lag = 0,  
  mode = "stretch",  
  exceed = FALSE  
)  
  
expanding_split
```

make_tsibble	<i>Convert data to a tsibble</i>
--------------	----------------------------------

Description

Convert a tibble containing time series data to a tsibble.

Usage

```
make_tsibble(main_frame, context)
```

Arguments

main_frame	A tibble containing the time series data.
context	A named list with the identifiers for series_id, value_id, and index_id.

Details

make_tsibble() is a small helper for time series cross-validation workflows. It uses the time index and series identifier supplied in context to create a regular tsibble.

The input data must contain the columns specified by context\$index_id and context\$series_id. The column specified by context\$index_id is used as the time index, and the column specified by context\$series_id is used as the key.

Value

A tsibble with the same columns as `main_frame`, using the index and key defined in context.

See Also

Other time series cross-validation: [make_future\(\)](#), [make_split\(\)](#), [slice_test\(\)](#), [slice_train\(\)](#), [split_index\(\)](#)

Examples

```
library(dplyr)
library(tsibble)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

tsibble_frame <- make_tsibble(
  main_frame = main_frame,
  context = context
)

tsibble_frame
```

`mape_vec`*Calculate the mean absolute percentage error*

Description

Calculate the mean absolute percentage error of a numeric vector.

Usage

```
mape_vec(truth, estimate, na_rm = TRUE)
```

Arguments

<code>truth</code>	Numeric vector containing the actual values.
<code>estimate</code>	Numeric vector containing the forecasts.
<code>na_rm</code>	Logical value. If TRUE, missing values are removed.

Details

mape_vec() computes the average absolute percentage forecast error: $\text{abs}((\text{truth} - \text{estimate}) / \text{truth}) * 100$.

This metric is undefined when truth is zero and may return Inf or NaN in such cases.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 40)
estimate <- c(8, 22, 30)

mape_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
mape_vec(truth_na, estimate_na)
```

me_vec

Calculate the mean error

Description

Calculate the mean error of a numeric vector.

Usage

```
me_vec(truth, estimate, na_rm = TRUE)
```

Arguments

truth	Numeric vector containing the actual values.
estimate	Numeric vector containing the forecasts.
na_rm	Logical value. If TRUE, missing values are removed.

Details

me_vec() computes the average signed forecast error $\text{truth} - \text{estimate}$. Positive values indicate that forecasts are, on average, below the observed values. Negative values indicate that forecasts are, on average, above the observed values.

The metric is reported in the same units as the original data.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 30)
estimate <- c(8, 22, 25)

me_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
me_vec(truth_na, estimate_na)
```

MEDIAN

Median model

Description

Specify a median benchmark model for use with `fabletools::model()`.

Usage

```
MEDIAN(formula, ...)
```

Arguments

<code>formula</code>	A model formula specifying the response and optional <code>window()</code> special, for example <code>value ~ window()</code> .
<code>...</code>	Further arguments.

Details

`MEDIAN()` forecasts future values using the median of the observed response. The `window()` special controls whether the median is estimated using all observations, a fixed trailing window, or a rolling window.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other MEDIAN: [fitted.MEDIAN\(\)](#), [forecast.MEDIAN\(\)](#), [model_sum.MEDIAN\(\)](#), [residuals.MEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("MEDIAN" = MEDIAN(value ~ window()))

model_frame
```

model_sum.DSHW	<i>Summarize a DSHW model</i>
----------------	-------------------------------

Description

Return a short model label for a fitted DSHW model.

Usage

```
## S3 method for class 'DSHW'
model_sum(x)
```

Arguments

x A fitted DSHW model object.

Value

A character string.

See Also

Other DSHW: [DSHW\(\)](#), [fitted.DSHW\(\)](#), [forecast.DSHW\(\)](#), [residuals.DSHW\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_load |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 28) |>
  as_tsibble(index = time)
```

```
model_frame <- train_frame |>
  model("DSHW" = DSHW(value, periods = c(24, 168)))

model_frame
```

model_sum.MEDIAN	<i>Summarize a median model</i>
------------------	---------------------------------

Description

Return a short model label for a fitted MEDIAN model.

Usage

```
## S3 method for class 'MEDIAN'
model_sum(x)
```

Arguments

x A fitted MEDIAN model object.

Value

A character string.

See Also

Other MEDIAN: [MEDIAN\(\)](#), [fitted.MEDIAN\(\)](#), [forecast.MEDIAN\(\)](#), [residuals.MEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("MEDIAN" = MEDIAN(value ~ window()))

model_frame
```

model_sum.SMEAN	<i>Summarize a seasonal mean model</i>
-----------------	--

Description

Return a short model label for a fitted SMEAN model.

Usage

```
## S3 method for class 'SMEAN'  
model_sum(x)
```

Arguments

x A fitted SMEAN model object.

Value

A character string.

See Also

Other SMEAN: [SMEAN\(\)](#), [fitted.SMEAN\(\)](#), [forecast.SMEAN\(\)](#), [residuals.SMEAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- M4_monthly_data |>  
  filter(series == first(series)) |>  
  as_tsibble(index = index)  
  
model_frame <- train_frame |>  
  model("SMEAN" = SMEAN(value ~ lag("year")))  
  
model_frame
```

model_sum.SMEDIAN	<i>Summarize a seasonal median model</i>
-------------------	--

Description

Return a short model label for a fitted SMEDIAN model.

Usage

```
## S3 method for class 'SMEDIAN'  
model_sum(x)
```

Arguments

x A fitted SMEDIAN model object.

Value

A character string.

See Also

Other SMEDIAN: [SMEDIAN\(\)](#), [fitted.SMEDIAN\(\)](#), [forecast.SMEDIAN\(\)](#), [residuals.SMEDIAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SMEDIAN" = SMEDIAN(value ~ lag("week")))  
  
model_frame
```

model_sum.SNAIVE2	<i>Summarize a SNAIVE2 model</i>
-------------------	----------------------------------

Description

Return a short model label for a fitted SNAIVE2 model.

Usage

```
## S3 method for class 'SNAIVE2'  
model_sum(x)
```

Arguments

x A fitted SNAIVE2 model object.

Value

A character string.

See Also

Other SNAIVE2: [SNAIVE2\(\)](#), [fitted.SNAIVE2\(\)](#), [forecast.SNAIVE2\(\)](#), [residuals.SNAIVE2\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SNAIVE2" = SNAIVE2(value))  
  
model_frame
```

model_sum.TBATS	<i>Summarize a TBATS model</i>
-----------------	--------------------------------

Description

Return a short model label for a fitted TBATS model.

Usage

```
## S3 method for class 'TBATS'  
model_sum(x)
```

Arguments

x A fitted TBATS model object.

Value

A character string.

See Also

Other TBATS: [TBATS\(\)](#), [fitted.TBATS\(\)](#), [forecast.TBATS\(\)](#), [residuals.TBATS\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("TBATS" = TBATS(value, periods = c(24, 168)))  
  
model_frame
```

mpe_vec	<i>Calculate the mean percentage error</i>
---------	--

Description

Calculate the mean percentage error of a numeric vector.

Usage

```
mpe_vec(truth, estimate, na_rm = TRUE)
```

Arguments

truth	Numeric vector containing the actual values.
estimate	Numeric vector containing the forecasts.
na_rm	Logical value. If TRUE, missing values are removed.

Details

`mpe_vec()` computes the average signed percentage forecast error: $((\text{truth} - \text{estimate}) / \text{truth}) * 100$. Positive values indicate that forecasts are, on average, below the observed values in percentage terms. Negative values indicate that forecasts are, on average, above the observed values.

This metric is undefined when truth is zero and may return Inf, -Inf, or NaN in such cases.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 40)
estimate <- c(8, 22, 30)

mpe_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
mpe_vec(truth_na, estimate_na)
```

`mse_vec`*Calculate the mean squared error*

Description

Calculate the mean squared error of a numeric vector.

Usage

```
mse_vec(truth, estimate, na_rm = TRUE)
```

Arguments

<code>truth</code>	Numeric vector containing the actual values.
<code>estimate</code>	Numeric vector containing the forecasts.
<code>na_rm</code>	Logical value. If TRUE, missing values are removed.

Details

`mse_vec()` computes the average squared forecast error $(\text{truth} - \text{estimate})^2$. The metric is reported in squared units of the original data.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [rmse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 30)
estimate <- c(8, 22, 25)

mse_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
mse_vec(truth_na, estimate_na)
```

`pacf_vec`*Estimate partial autocorrelations of a numeric vector*

Description

Estimate the sample partial autocorrelation function of a numeric vector.

Usage

```
pacf_vec(x, lag_max = 24, ...)
```

Arguments

<code>x</code>	Numeric vector.
<code>lag_max</code>	Integer. Maximum lag for which the partial autocorrelation is estimated.
<code>...</code>	Further arguments passed to <code>stats::pacf()</code> .

Details

`pacf_vec()` is a small wrapper around `stats::pacf()`. It returns the sample partial autocorrelations as a numeric vector for lags 1 to `lag_max`.

Value

A numeric vector containing the sample partial autocorrelations for lags 1 to `lag_max`.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)

x <- M4_monthly_data |>
  filter(series == first(series)) |>
  pull(value)

pacf_vec(
  x = x,
  lag_max = 12
)
```

`plot_bar`*Plot data as a bar chart*

Description

Create a bar chart for grouped numeric values.

Usage

```
plot_bar(  
  data,  
  x,  
  y,  
  position = "dodge",  
  facet_var = NULL,  
  facet_scale = "free",  
  facet_nrow = NULL,  
  facet_ncol = NULL,  
  color = NULL,  
  flip = FALSE,  
  reorder = FALSE,  
  title = NULL,  
  subtitle = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  caption = NULL,  
  bar_size = 0.75,  
  bar_color = "grey35",  
  bar_alpha = 1,  
  theme_set = theme_tscv(),  
  theme_config = list(),  
  ...  
)
```

Arguments

<code>data</code>	A data.frame, tibble, or tsibble in long format.
<code>x</code>	Unquoted column in data used on the x-axis.
<code>y</code>	Unquoted column in data containing numeric values shown on the y-axis.
<code>position</code>	Character value defining the bar position. Common values are "dodge" and "stack".
<code>facet_var</code>	Optional unquoted column in data used for faceting.
<code>facet_scale</code>	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
<code>facet_nrow</code>	Optional integer. Number of rows in the facet layout.

facet_ncol	Optional integer. Number of columns in the facet layout.
color	Optional unquoted column in data used to map bar fill color.
flip	Logical value. If TRUE, the plot is flipped using <code>ggplot2::coord_flip()</code> .
reorder	Logical value. If TRUE, add <code>tidytext::scale_x_reordered()</code> for reordered facet labels.
title	Character value. Plot title.
subtitle	Character value. Plot subtitle.
xlab	Character value. Label for the x-axis.
ylab	Character value. Label for the y-axis.
caption	Character value. Plot caption.
bar_size	Numeric value defining the bar border line width.
bar_color	Character value defining the bar fill color. Ignored when <code>color</code> is supplied.
bar_alpha	Numeric value between 0 and 1 defining bar transparency.
theme_set	A complete <code>ggplot2</code> theme.
theme_config	A named list with additional arguments passed to <code>ggplot2::theme()</code> .
...	Currently not used.

Details

`plot_bar()` is a convenience wrapper around `ggplot2::geom_bar()` with `stat = "identity"`. It is intended for data that already contains summarized values, for example accuracy metrics, counts, or grouped summary statistics.

The arguments `x`, `y`, `facet_var`, and `color` are passed as unquoted column names.

If `color` is supplied, bar fill colors are mapped to that variable and `bar_color` is ignored. If `color` is not supplied, all bars are drawn using `bar_color`.

The argument `position` controls how bars are displayed when `color` is supplied. Common values are `"dodge"` and `"stack"`.

If `flip = TRUE`, the x-axis and y-axis are swapped using `ggplot2::coord_flip()`.

If `reorder = TRUE`, `tidytext::scale_x_reordered()` is added. This is useful when the x-axis has been reordered within facets with `tidytext::reorder_within()`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

stats <- summarise_stats(
  .data = data,
  context = context
)

plot_bar(
  data = stats,
  x = series,
  y = mean,
  title = "Average Value by Series",
  xlab = "Series",
  ylab = "Mean"
)

acf_data <- estimate_acf(
  .data = data,
  context = context,
  lag_max = 12
)

plot_bar(
  data = acf_data,
  x = lag,
  y = value,
  facet_var = series,
  title = "Autocorrelation by Series",
  subtitle = "Sample autocorrelation up to lag 12",
  xlab = "Lag",
  ylab = "ACF"
)
```

plot_density

Plot a kernel density estimate

Description

Create a density plot for one or more numeric variables using kernel density estimation.

Usage

```

plot_density(
  data,
  x,
  facet_var = NULL,
  facet_scale = "free",
  facet_nrow = NULL,
  facet_ncol = NULL,
  color = NULL,
  fill = NULL,
  title = NULL,
  subtitle = NULL,
  xlab = NULL,
  ylab = NULL,
  caption = NULL,
  line_width = 0.1,
  line_type = "solid",
  line_color = "grey35",
  fill_color = "grey35",
  fill_alpha = 0.5,
  theme_set = theme_tscv(),
  theme_config = list(),
  ...
)

```

Arguments

<code>data</code>	A data.frame, tibble, or tsibble in long format.
<code>x</code>	Unquoted column in data containing numeric values.
<code>facet_var</code>	Optional unquoted column in data used for faceting.
<code>facet_scale</code>	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
<code>facet_nrow</code>	Optional integer. Number of rows in the facet layout.
<code>facet_ncol</code>	Optional integer. Number of columns in the facet layout.
<code>color</code>	Optional unquoted column in data used to map density line and fill color.
<code>fill</code>	Optional unquoted column in data used to map density fill color. Currently not used directly; use color for grouped density plots.
<code>title</code>	Character value. Plot title.
<code>subtitle</code>	Character value. Plot subtitle.
<code>xlab</code>	Character value. Label for the x-axis.
<code>ylab</code>	Character value. Label for the y-axis.
<code>caption</code>	Character value. Plot caption.
<code>line_width</code>	Numeric value defining the density line width.
<code>line_type</code>	Character or numeric value defining the density line type.

line_color	Character value defining the density line color. Ignored when color is supplied.
fill_color	Character value defining the fill color under the density curve. Ignored when color is supplied.
fill_alpha	Numeric value between 0 and 1 defining fill transparency.
theme_set	A complete ggplot2 theme.
theme_config	A named list with additional arguments passed to <code>ggplot2::theme()</code> .
...	Further arguments passed to <code>ggplot2::geom_density()</code> .

Details

`plot_density()` is a convenience wrapper around `ggplot2::geom_density()`. It is useful for comparing the distribution of values across one or more time series, models, groups, or residual sets.

The arguments `x`, `facet_var`, `color`, and `fill` are passed as unquoted column names.

If `color` is supplied, both line color and fill color are mapped to that variable. In this case, `line_color` and `fill_color` are ignored. If `color` is not supplied, all density curves use `line_color` and `fill_color`.

Missing values are removed before plotting.

Additional arguments can be passed to `ggplot2::geom_density()` through `...`, for example `adjust`, `bw`, or `kernel`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

plot_density(
  data = data,
  x = value,
  facet_var = series,
  title = "Distribution of M4 Monthly Values",
  subtitle = "Kernel density estimates by series",
  xlab = "Value",
  ylab = "Density"
```

```
)  
  
plot_density(  
  data = data,  
  x = value,  
  color = series,  
  title = "Distribution of M4 Monthly Values",  
  subtitle = "Kernel density estimates by series",  
  xlab = "Value",  
  ylab = "Density",  
  adjust = 1.2  
)
```

plot_histogram	<i>Plot data as a histogram</i>
----------------	---------------------------------

Description

Create a histogram for one or more numeric variables.

Usage

```
plot_histogram(  
  data,  
  x,  
  facet_var = NULL,  
  facet_scale = "free",  
  facet_nrow = NULL,  
  facet_ncol = NULL,  
  color = NULL,  
  fill = NULL,  
  title = NULL,  
  subtitle = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  caption = NULL,  
  line_color = "grey35",  
  line_width = 0.5,  
  fill_color = "grey35",  
  fill_alpha = 1,  
  theme_set = theme_tscv(),  
  theme_config = list(),  
  ...  
)
```

Arguments

data A data.frame, tibble, or tsibble in long format.

<code>x</code>	Unquoted column in data containing numeric values.
<code>facet_var</code>	Optional unquoted column in data used for faceting.
<code>facet_scale</code>	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
<code>facet_nrow</code>	Optional integer. Number of rows in the facet layout.
<code>facet_ncol</code>	Optional integer. Number of columns in the facet layout.
<code>color</code>	Optional unquoted column in data used to map histogram outline and fill color.
<code>fill</code>	Optional unquoted column in data used to map histogram fill color. Currently not used directly; use <code>color</code> for grouped histograms.
<code>title</code>	Character value. Plot title.
<code>subtitle</code>	Character value. Plot subtitle.
<code>xlab</code>	Character value. Label for the x-axis.
<code>ylab</code>	Character value. Label for the y-axis.
<code>caption</code>	Character value. Plot caption.
<code>line_color</code>	Character value defining the histogram bar outline color. Ignored when <code>color</code> is supplied.
<code>line_width</code>	Numeric value defining the histogram bar outline width.
<code>fill_color</code>	Character value defining the histogram bar fill color. Ignored when <code>color</code> is supplied.
<code>fill_alpha</code>	Numeric value between 0 and 1 defining bar transparency.
<code>theme_set</code>	A complete <code>ggplot2</code> theme.
<code>theme_config</code>	A named list with additional arguments passed to <code>ggplot2::theme()</code> .
<code>...</code>	Further arguments passed to <code>ggplot2::geom_histogram()</code> .

Details

`plot_histogram()` is a convenience wrapper around `ggplot2::geom_histogram()`. It is useful for visualizing the distribution of values across one or more time series, models, groups, or residual sets.

The arguments `x`, `facet_var`, `color`, and `fill` are passed as unquoted column names.

If `color` is supplied, both bar outline color and fill color are mapped to that variable. In this case, `line_color` and `fill_color` are ignored. If `color` is not supplied, all histogram bars use `line_color` and `fill_color`.

Missing values are removed before plotting.

Additional arguments can be passed to `ggplot2::geom_histogram()` through `...`, for example `bins`, `binwidth`, or `boundary`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

plot_histogram(
  data = data,
  x = value,
  facet_var = series,
  title = "Distribution of M4 Monthly Values",
  subtitle = "Histograms by series",
  xlab = "Value",
  ylab = "Count",
  bins = 20
)

plot_histogram(
  data = data,
  x = value,
  color = series,
  title = "Distribution of M4 Monthly Values",
  subtitle = "Grouped histograms by series",
  xlab = "Value",
  ylab = "Count",
  bins = 20,
  position = "identity",
  fill_alpha = 0.4
)
```

plot_line

Plot data as a line chart

Description

Create a line chart for one or more time series or grouped numeric variables.

Usage

```
plot_line(
  data,
  x,
  y,
  facet_var = NULL,
```

```

facet_scale = "free",
facet_nrow = NULL,
facet_ncol = NULL,
color = NULL,
title = NULL,
subtitle = NULL,
xlab = NULL,
ylab = NULL,
caption = NULL,
line_size = 0.75,
line_type = "solid",
line_color = "grey35",
line_alpha = 1,
theme_set = theme_tscv(),
theme_config = list(),
...
)

```

Arguments

data	A data.frame, tibble, or tsibble in long format.
x	Unquoted column in data used on the x-axis.
y	Unquoted column in data containing numeric values shown on the y-axis.
facet_var	Optional unquoted column in data used for faceting.
facet_scale	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
facet_nrow	Optional integer. Number of rows in the facet layout.
facet_ncol	Optional integer. Number of columns in the facet layout.
color	Optional unquoted column in data used to map line color.
title	Character value. Plot title.
subtitle	Character value. Plot subtitle.
xlab	Character value. Label for the x-axis.
ylab	Character value. Label for the y-axis.
caption	Character value. Plot caption.
line_size	Numeric value defining the line width.
line_type	Character or numeric value defining the line type.
line_color	Character value defining the line color. Ignored when color is supplied.
line_alpha	Numeric value between 0 and 1 defining line transparency.
theme_set	A complete ggplot2 theme.
theme_config	A named list with additional arguments passed to ggplot2::theme().
...	Currently not used.

Details

`plot_line()` is a convenience wrapper around `ggplot2::geom_line()` for plotting data in long format. It supports optional grouping by color, optional faceting, and the default `tscv` theme.

The arguments `x`, `y`, `facet_var`, and `color` are passed as unquoted column names.

If `color` is supplied, line colors are mapped to that variable and `line_color` is ignored. If `color` is not supplied, all lines are drawn using `line_color`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

plot_line(
  data = data,
  x = index,
  y = value,
  facet_var = series,
  title = "M4 Monthly Time Series",
  subtitle = "Selected monthly series from the M4 forecasting competition",
  xlab = "Time",
  ylab = "Value",
  caption = "Data: M4 Forecasting Competition"
)

plot_line(
  data = data,
  x = index,
  y = value,
  color = series,
  title = "M4 Monthly Time Series",
  xlab = "Time",
  ylab = "Value",
  line_size = 1.5
)
```

`plot_point`*Plot data as a scatterplot*

Description

Create a scatterplot for two variables.

Usage

```
plot_point(  
  data,  
  x,  
  y,  
  facet_var = NULL,  
  facet_scale = "free",  
  facet_nrow = NULL,  
  facet_ncol = NULL,  
  color = NULL,  
  title = NULL,  
  subtitle = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  caption = NULL,  
  point_size = 1.5,  
  point_type = 16,  
  point_color = "grey35",  
  point_alpha = 1,  
  theme_set = theme_tscv(),  
  theme_config = list(),  
  ...  
)
```

Arguments

<code>data</code>	A data frame, tibble, or tsibble in long format.
<code>x</code>	Unquoted column in data used on the x-axis.
<code>y</code>	Unquoted column in data containing numeric values shown on the y-axis.
<code>facet_var</code>	Optional unquoted column in data used for faceting.
<code>facet_scale</code>	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
<code>facet_nrow</code>	Optional integer. Number of rows in the facet layout.
<code>facet_ncol</code>	Optional integer. Number of columns in the facet layout.
<code>color</code>	Optional unquoted column in data used to map point color.
<code>title</code>	Character value. Plot title.

subtitle	Character value. Plot subtitle.
xlab	Character value. Label for the x-axis.
ylab	Character value. Label for the y-axis.
caption	Character value. Plot caption.
point_size	Numeric value defining the point size.
point_type	Numeric or character value defining the point shape.
point_color	Character value defining the point color. Ignored when color is supplied.
point_alpha	Numeric value between 0 and 1 defining point transparency.
theme_set	A complete ggplot2 theme.
theme_config	A named list with additional arguments passed to <code>ggplot2::theme()</code> .
...	Currently not used.

Details

`plot_point()` is a convenience wrapper around `ggplot2::geom_point()`. It is useful for plotting relationships between two variables, for example observed values over time, forecast errors by horizon, or one numeric diagnostic against another.

The arguments `x`, `y`, `facet_var`, and `color` are passed as unquoted column names.

If `color` is supplied, point colors are mapped to that variable and `point_color` is ignored. If `color` is not supplied, all points are drawn using `point_color`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series == "M23100")

plot_point(
  data = data,
  x = index,
  y = value,
  title = "M4 Monthly Time Series",
  subtitle = "Series M23100",
  xlab = "Time",
```

```
  ylab = "Value"
)

acf_data <- estimate_acf(
  .data = M4_monthly_data |>
    filter(series %in% c("M23100", "M14395")),
  context = list(
    series_id = "series",
    value_id = "value",
    index_id = "index"
  ),
  lag_max = 12
)

plot_point(
  data = acf_data,
  x = lag,
  y = value,
  color = series,
  title = "Autocorrelation by Series",
  subtitle = "Sample autocorrelation up to lag 12",
  xlab = "Lag",
  ylab = "ACF",
  point_size = 4
)
```

plot_qq

Create a quantile-quantile plot

Description

Create a quantile-quantile plot for one or more numeric variables.

Usage

```
plot_qq(
  data,
  x,
  facet_var = NULL,
  facet_scale = "free",
  facet_nrow = NULL,
  facet_ncol = NULL,
  color = NULL,
  title = NULL,
  subtitle = NULL,
  xlab = NULL,
  ylab = NULL,
  caption = NULL,
  point_size = 2,
```

```

    point_shape = 16,
    point_color = "grey35",
    point_fill = "grey35",
    point_alpha = 0.25,
    line_width = 0.25,
    line_type = "solid",
    line_color = "grey35",
    line_alpha = 1,
    band_color = "grey35",
    band_alpha = 0.25,
    theme_set = theme_tscv(),
    theme_config = list(),
    ...
  )

```

Arguments

<code>data</code>	A <code>data.frame</code> , <code>tibble</code> , or <code>tsibble</code> in long format.
<code>x</code>	Unquoted column in <code>data</code> containing numeric values.
<code>facet_var</code>	Optional unquoted column in <code>data</code> used for faceting.
<code>facet_scale</code>	Character value defining facet axis scaling. Common values are "free", "fixed", "free_x", and "free_y".
<code>facet_nrow</code>	Optional integer. Number of rows in the facet layout.
<code>facet_ncol</code>	Optional integer. Number of columns in the facet layout.
<code>color</code>	Optional unquoted column in <code>data</code> used to map point, line, and confidence-band colors.
<code>title</code>	Character value. Plot title.
<code>subtitle</code>	Character value. Plot subtitle.
<code>xlab</code>	Character value. Label for the x-axis.
<code>ylab</code>	Character value. Label for the y-axis.
<code>caption</code>	Character value. Plot caption.
<code>point_size</code>	Numeric value defining the point size.
<code>point_shape</code>	Numeric or character value defining the point shape.
<code>point_color</code>	Character value defining the point outline color. Ignored when <code>color</code> is supplied.
<code>point_fill</code>	Character value defining the point fill color. Ignored when <code>color</code> is supplied.
<code>point_alpha</code>	Numeric value between 0 and 1 defining point transparency.
<code>line_width</code>	Numeric value defining the qq-line width.
<code>line_type</code>	Character or numeric value defining the qq-line type.
<code>line_color</code>	Character value defining the qq-line color. Ignored when <code>color</code> is supplied.
<code>line_alpha</code>	Numeric value between 0 and 1 defining line transparency.
<code>band_color</code>	Character value defining the confidence-band fill color. Ignored when <code>color</code> is supplied.

<code>band_alpha</code>	Numeric value between 0 and 1 defining confidence-band transparency.
<code>theme_set</code>	A complete ggplot2 theme.
<code>theme_config</code>	A named list with additional arguments passed to <code>ggplot2::theme()</code> .
<code>...</code>	Further arguments passed to <code>qqplotr::stat_qq_point()</code> , <code>qqplotr::stat_qq_line()</code> , and <code>qqplotr::stat_qq_band()</code> .

Details

`plot_qq()` is a convenience wrapper around the `qqplotr` functions `stat_qq_point()`, `stat_qq_line()`, and `stat_qq_band()`. It is useful for checking whether values, residuals, or forecast errors approximately follow a theoretical distribution.

By default, the function creates a normal quantile-quantile plot with pointwise confidence bands.

The arguments `x`, `facet_var`, and `color` are passed as unquoted column names.

If `color` is supplied, point colors, line colors, and confidence-band fills are mapped to that variable. In this case, `point_color`, `point_fill`, `line_color`, and `band_color` are ignored. If `color` is not supplied, the fixed styling arguments are used.

Additional arguments can be passed to the underlying `qqplotr` statistics through `...`, for example distributional arguments supported by `qqplotr`.

Additional theme settings can be supplied through `theme_config`. This should be a named list of arguments passed to `ggplot2::theme()`.

Value

An object of class `ggplot`.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

plot_qq(
  data = data,
  x = value,
  facet_var = series,
  title = "QQ Plot of M4 Monthly Values",
  subtitle = "Normal quantile-quantile plots by series",
  xlab = "Theoretical quantiles",
  ylab = "Sample quantiles"
)

stats <- data |>
```

```

group_by(series) |>
mutate(value_centered = value - mean(value, na.rm = TRUE)) |>
ungroup()

plot_qq(
  data = stats,
  x = value_centered,
  color = series,
  title = "QQ Plot of Centered M4 Monthly Values",
  subtitle = "Normal quantile-quantile plots by series",
  xlab = "Theoretical quantiles",
  ylab = "Sample quantiles"
)

```

residuals.DSHW	<i>Extract residuals from a DSHW model</i>
----------------	--

Description

Extract residuals from a fitted DSHW model.

Usage

```

## S3 method for class 'DSHW'
residuals(object, ...)

```

Arguments

object	A fitted DSHW model object.
...	Additional arguments. Currently not used.

Value

Residuals.

See Also

Other DSHW: [DSHW\(\)](#), [fitted.DSHW\(\)](#), [forecast.DSHW\(\)](#), [model_sum.DSHW\(\)](#)

Examples

```

library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_load |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 28) |>
  as_tsibble(index = time)

```

```
model_frame <- train_frame |>
  model("DSHW" = DSHW(value, periods = c(24, 168)))

residuals(model_frame)
```

residuals.MEDIAN *Extract residuals from a median model*

Description

Extract residuals from a fitted MEDIAN model.

Usage

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

Arguments

object A fitted MEDIAN model object.
... Additional arguments. Currently not used.

Value

Residuals.

See Also

Other MEDIAN: [MEDIAN\(\)](#), [fitted.MEDIAN\(\)](#), [forecast.MEDIAN\(\)](#), [model_sum.MEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("MEDIAN" = MEDIAN(value ~ window()))

residuals(model_frame)
```

residuals.SMEAN	<i>Extract residuals from a seasonal mean model</i>
-----------------	---

Description

Extract residuals from a fitted SMEAN model.

Usage

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

Arguments

object	A fitted SMEAN model object.
...	Additional arguments. Currently not used.

Value

Residuals.

See Also

Other SMEAN: [SMEAN\(\)](#), [fitted.SMEAN\(\)](#), [forecast.SMEAN\(\)](#), [model_sum.SMEAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- M4_monthly_data |>  
  filter(series == first(series)) |>  
  as_tsibble(index = index)  
  
model_frame <- train_frame |>  
  model("SMEAN" = SMEAN(value ~ lag("year")))  
  
residuals(model_frame)
```

residuals.SMEDIAN *Extract residuals from a seasonal median model*

Description

Extract residuals from a fitted SMEDIAN model.

Usage

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

Arguments

object A fitted SMEDIAN model object.
... Additional arguments. Currently not used.

Value

Residuals.

See Also

Other SMEDIAN: [SMEDIAN\(\)](#), [fitted.SMEDIAN\(\)](#), [forecast.SMEDIAN\(\)](#), [model_sum.SMEDIAN\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SMEDIAN" = SMEDIAN(value ~ lag("week")))  
  
residuals(model_frame)
```

residuals.SNAIVE2 *Extract residuals from a SNAIVE2 model*

Description

Extract residuals from a fitted SNAIVE2 model.

Usage

```
## S3 method for class 'SNAIVE2'  
residuals(object, ...)
```

Arguments

object A fitted SNAIVE2 model object.
... Additional arguments. Currently not used.

Value

Residuals.

See Also

Other SNAIVE2: [SNAIVE2\(\)](#), [fitted.SNAIVE2\(\)](#), [forecast.SNAIVE2\(\)](#), [model_sum.SNAIVE2\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("SNAIVE2" = SNAIVE2(value))  
  
residuals(model_frame)
```

residuals.TBATS *Extract residuals from a TBATS model*

Description

Extract residuals from a fitted TBATS model.

Usage

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

Arguments

object A fitted TBATS model object.
... Additional arguments. Currently not used.

Details

This method is used by `residuals()` when extracting residuals from a mable containing a TBATS model.

Value

Residuals.

See Also

Other TBATS: [TBATS\(\)](#), [fitted.TBATS\(\)](#), [forecast.TBATS\(\)](#), [model_sum.TBATS\(\)](#)

Examples

```
library(dplyr)  
library(tsibble)  
library(fabletools)  
  
train_frame <- elec_price |>  
  filter(bidding_zone == "DE") |>  
  slice_head(n = 24 * 21) |>  
  as_tsibble(index = time)  
  
model_frame <- train_frame |>  
  model("TBATS" = TBATS(value, periods = c(24, 168)))  
  
residuals(model_frame)
```

rmse_vec	<i>Calculate the root mean squared error</i>
----------	--

Description

Calculate the root mean squared error of a numeric vector.

Usage

```
rmse_vec(truth, estimate, na_rm = TRUE)
```

Arguments

truth	Numeric vector containing the actual values.
estimate	Numeric vector containing the forecasts.
na_rm	Logical value. If TRUE, missing values are removed.

Details

`rmse_vec()` computes the square root of the mean squared forecast error. The metric is reported in the same units as the original data.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [smape_vec\(\)](#)

Examples

```
truth <- c(10, 20, 30)
estimate <- c(8, 22, 25)

rmse_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
rmse_vec(truth_na, estimate_na)
```

scale_color_tscv *Create a tscv color scale*

Description

Create a ggplot2 color scale based on a predefined tscv palette.

Usage

```
scale_color_tscv(palette = "main", discrete = TRUE, reverse = FALSE, ...)
```

Arguments

palette	Character value. Name of the palette.
discrete	Logical value. If TRUE, create a discrete color scale. If FALSE, create a continuous color scale.
reverse	Logical value. If TRUE, the palette is reversed.
...	Additional arguments passed to <code>ggplot2::discrete_scale()</code> or <code>ggplot2::scale_color_gradientn()</code>

Details

`scale_color_tscv()` creates either a discrete or continuous color scale for the color aesthetic.

For discrete variables, the function uses `ggplot2::discrete_scale()`. For continuous variables, it uses `ggplot2::scale_color_gradientn()`.

Available palettes are "main", "cool", "hot", "mixed", and "grey".

Value

A ggplot2 scale object.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

plot_line(
  data = data,
  x = index,
  y = value,
  color = series,
```

```

  title = "M4 Monthly Time Series",
  subtitle = "Selected monthly series",
  xlab = "Time",
  ylab = "Value"
) +
  scale_color_tscv(palette = "main")

```

scale_fill_tscv *Create a tscv fill scale*

Description

Create a ggplot2 fill scale based on a predefined tscv palette.

Usage

```
scale_fill_tscv(palette = "main", discrete = TRUE, reverse = FALSE, ...)
```

Arguments

palette	Character value. Name of the palette.
discrete	Logical value. If TRUE, create a discrete fill scale. If FALSE, create a continuous fill scale.
reverse	Logical value. If TRUE, the palette is reversed.
...	Additional arguments passed to <code>ggplot2::discrete_scale()</code> or <code>ggplot2::scale_fill_gradientn()</code>

Details

`scale_fill_tscv()` creates either a discrete or continuous fill scale for the fill aesthetic.

For discrete variables, the function uses `ggplot2::discrete_scale()`. For continuous variables, it uses `ggplot2::scale_fill_gradientn()`.

Available palettes are "main", "cool", "hot", "mixed", and "grey".

Value

A ggplot2 scale object.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

stats <- summarise_stats(
  .data = data,
  context = context
)

plot_bar(
  data = stats,
  x = series,
  y = mean,
  color = series,
  title = "Average Value by Series",
  xlab = "Series",
  ylab = "Mean"
) +
  scale_fill_tscv(palette = "main")
```

slice_test

Slice test data from a split frame

Description

Extract test observations from a complete time series data set according to a split plan created by `make_split()`.

Usage

```
slice_test(main_frame, split_frame, context)
```

Arguments

`main_frame` A tibble containing the complete time series data.

`split_frame` A tibble containing train and test indices, usually created by `make_split()`.

`context` A named list with the identifiers for `series_id`, `value_id`, and `index_id`.

Details

`slice_test()` uses the row positions stored in the `test` list-column of `split_frame` to extract the corresponding observations from `main_frame`. The function is designed for rolling-origin time series cross-validation workflows.

The returned data has the same columns as `main_frame`, plus a `split` column identifying the train-test split. If `main_frame` contains multiple time series, slicing is performed separately for each series using the series identifier supplied in context.

When `make_split()` was called with `n_lag > 0`, the test data may include lagged observations before the forecast horizon.

Value

A tibble containing the sliced test data. It contains the same columns as `main_frame`, plus a `split` column.

See Also

Other time series cross-validation: [make_future\(\)](#), [make_split\(\)](#), [make_tsibble\(\)](#), [slice_train\(\)](#), [split_index\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series == "M23100")

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

test_frame <- slice_test(
  main_frame = main_frame,
  split_frame = split_frame,
  context = context
)
```

test_frame

slice_train *Slice training data from a split frame*

Description

Extract training observations from a complete time series data set according to a split plan created by `make_split()`.

Usage

```
slice_train(main_frame, split_frame, context)
```

Arguments

`main_frame` A tibble containing the complete time series data.

`split_frame` A tibble containing train and test indices, usually created by `make_split()`.

`context` A named list with the identifiers for `series_id`, `value_id`, and `index_id`.

Details

`slice_train()` uses the row positions stored in the `train` list-column of `split_frame` to extract the corresponding observations from `main_frame`. The function is designed for rolling-origin time series cross-validation workflows.

The returned data has the same columns as `main_frame`, plus a `split` column identifying the train-test split. If `main_frame` contains multiple time series, slicing is performed separately for each series using the series identifier supplied in `context`.

Value

A tibble containing the sliced training data. It contains the same columns as `main_frame`, plus a `split` column.

See Also

Other time series cross-validation: [make_future\(\)](#), [make_split\(\)](#), [make_tsibble\(\)](#), [slice_test\(\)](#), [split_index\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

main_frame <- M4_monthly_data |>
  filter(series == "M23100")

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

train_frame <- slice_train(
  main_frame = main_frame,
  split_frame = split_frame,
  context = context
)

train_frame
```

smape_vec*Calculate the symmetric mean absolute percentage error*

Description

Calculate the symmetric mean absolute percentage error of a numeric vector.

Usage

```
smape_vec(truth, estimate, na_rm = TRUE)
```

Arguments

truth	Numeric vector containing the actual values.
estimate	Numeric vector containing the forecasts.
na_rm	Logical value. If TRUE, missing values are removed.

Details

`smape_vec()` computes the symmetric mean absolute percentage error: $\text{abs}(\text{estimate} - \text{truth}) / ((\text{abs}(\text{truth}) + \text{abs}(\text{estimate})) / 2) * 100$.

This metric is undefined when both `truth` and `estimate` are zero and may return NaN in such cases.

Value

A numeric value.

See Also

Other accuracy functions: [mae_vec\(\)](#), [make_accuracy\(\)](#), [make_errors\(\)](#), [mape_vec\(\)](#), [me_vec\(\)](#), [mpe_vec\(\)](#), [mse_vec\(\)](#), [rmse_vec\(\)](#)

Examples

```
truth <- c(10, 20, 40)
estimate <- c(8, 22, 30)

smape_vec(truth, estimate)

truth_na <- c(10, 20, NA)
estimate_na <- c(8, 22, 25)
smape_vec(truth_na, estimate_na)
```

SMEAN

Seasonal mean model

Description

Specify a seasonal mean benchmark model for use with `fabletools::model()`.

Usage

```
SMEAN(formula, ...)
```

Arguments

<code>formula</code>	A model formula specifying the response and <code>lag()</code> special, for example <code>value ~ lag("year")</code> .
<code>...</code>	Further arguments.

Details

`SMEAN()` forecasts each future observation using the historical mean of the matching seasonal position. Use the `lag()` special to define the seasonal period, for example `lag("year")` for monthly data.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other SMEAN: `fitted.SMEAN()`, `forecast.SMEAN()`, `model_sum.SMEAN()`, `residuals.SMEAN()`

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- M4_monthly_data |>
  filter(series == first(series)) |>
  as_tsibble(index = index)

model_frame <- train_frame |>
  model("SMEAN" = SMEAN(value ~ lag("year")))

model_frame
```

SMEDIAN

Seasonal median model

Description

Specify a seasonal median benchmark model for use with `fabletools::model()`.

Usage

```
SMEDIAN(formula, ...)
```

Arguments

<code>formula</code>	A model formula specifying the response and <code>lag()</code> special, for example <code>value ~ lag("week")</code> .
<code>...</code>	Further arguments.

Details

`SMEDIAN()` forecasts each future observation using the historical median of the matching seasonal position. Use the `lag()` special to define the seasonal period, for example `lag("week")` for hourly data with weekly seasonality.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other SMEDIAN: [fitted.SMEDIAN\(\)](#), [forecast.SMEDIAN\(\)](#), [model_sum.SMEDIAN\(\)](#), [residuals.SMEDIAN\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 21) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("SMEDIAN" = SMEDIAN(value ~ lag("week")))

model_frame
```

smooth_outlier

Identify and replace outliers

Description

Identify outliers in a numeric time series and replace them with smoothed values.

Usage

```
smooth_outlier(x, periods, ...)
```

Arguments

x	Numeric vector containing the time series observations.
periods	Numeric vector giving the seasonal periods of the time series, for example 12 for monthly data with yearly seasonality or <code>c(24, 168)</code> for hourly data with daily and weekly seasonality.
...	Further arguments passed to <code>forecast::msts()</code> or <code>forecast::tsoutliers()</code> .

Details

`smooth_outlier()` is a small wrapper around `forecast::tsoutliers()`. The input vector is first converted to an `msts` object using the seasonal periods supplied in `periods`.

For non-seasonal time series, `forecast::tsoutliers()` uses a `supsmu`-based approach. For seasonal time series, the series is decomposed using `STL` and outliers are identified on the remainder component. Detected outliers are replaced by the replacement values returned by `forecast::tsoutliers()`.

The function returns a plain numeric vector with the same length as the input.

Value

A numeric vector where detected outliers are replaced by smoothed values.

See Also

Other data preparation: [check_data\(\)](#), [interpolate_missing\(\)](#)

Examples

```
library(dplyr)

x <- M4_monthly_data |>
  filter(series == first(series)) |>
  pull(value)

x_outlier <- x
x_outlier[20] <- x_outlier[20] * 5

x_smoothed <- smooth_outlier(
  x = x_outlier,
  periods = 12
)

x_outlier[20]
x_smoothed[20]

hourly <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 14) |>
  pull(value)

hourly_outlier <- hourly
hourly_outlier[48] <- hourly_outlier[48] * 5

smooth_outlier(
  x = hourly_outlier,
  periods = c(24, 168)
)
```

SNAIVE2

Seasonal naive model with weekday-specific lags

Description

Specify a seasonal naive benchmark model for use with `fabletools::model()`.

Usage

```
SNAIVE2(formula, ...)
```

Arguments

formula A model formula specifying the response variable, for example value.
 ... Further arguments.

Details

SNAIVE2() is intended for hourly time series. It uses a daily lag for Tuesday to Friday observations and a weekly lag otherwise. This can be useful for electricity price or load data where weekdays have similar intraday structure and weekends require a weekly comparison.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other SNAIVE2: [fitted.SNAIVE2\(\)](#), [forecast.SNAIVE2\(\)](#), [model_sum.SNAIVE2\(\)](#), [residuals.SNAIVE2\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 21) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("SNAIVE2" = SNAIVE2(value))

model_frame
```

split_index

Create indices for train and test splits

Description

Create train and test indices for time series cross-validation.

Usage

```
split_index(
  n_total,
  n_init,
  n_ahead,
  n_skip = 0,
```

```

n_lag = 0,
mode = "slide",
exceed = FALSE
)

```

Arguments

n_total	Integer. The total number of observations in the time series.
n_init	Integer. The number of observations in the initial training window.
n_ahead	Integer. The forecast horizon, i.e. the number of observations in each test window.
n_skip	Integer. The number of observations to skip between split origins. The default is 0.
n_lag	Integer. The number of lagged observations to include before the test window. The default is 0.
mode	Character value. Either "slide" for a fixed-window approach or "stretch" for an expanding-window approach.
exceed	Logical value. If TRUE, test indices may exceed the original sample size.

Details

split_index() creates integer index vectors for rolling-origin resampling. The function can create either fixed-window or expanding-window splits:

- mode = "slide" creates a fixed training window that moves forward over time.
- mode = "stretch" creates an expanding training window that always starts at the first observation.

The first training window contains n_init observations. Each test window contains n_ahead observations. The argument n_skip controls how many observations are skipped between consecutive split origins. For example, with n_ahead = 18 and n_skip = 17, consecutive test windows are non-overlapping.

If n_lag > 0, the test indices include lagged observations before the forecast horizon. This is useful when lagged predictors are needed for constructing features during testing.

If exceed = TRUE, additional out-of-sample test indices are allowed to exceed the original sample size.

Value

A list with two elements:

- train: a list of integer vectors with training indices.
- test: a list of integer vectors with test indices.

See Also

Other time series cross-validation: [make_future\(\)](#), [make_split\(\)](#), [make_tsibble\(\)](#), [slice_test\(\)](#), [slice_train\(\)](#)

Examples

```
# Fixed-window splits
fixed_index <- split_index(
  n_total = 180,
  n_init = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "slide",
  exceed = FALSE
)

fixed_index

# Expanding-window splits
expanding_index <- split_index(
  n_total = 180,
  n_init = 120,
  n_ahead = 18,
  n_skip = 17,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
)

expanding_index
```

summarise_data

Summarise time series data

Description

Calculate basic data-quality summary statistics for one or more time series.

Usage

```
summarise_data(.data, context)
```

Arguments

`.data` A tibble in long format containing time series data.
`context` A named list with the identifiers for `series_id`, `value_id`, and `index_id`.

Details

`summarise_data()` groups the input data by the series identifier supplied in `context` and returns one row per time series.

The function reports:

- start: first time index;
- end: last time index;
- n_obs: number of observations;
- n_missing: number of missing values;
- pct_missing: percentage of missing values;
- n_zeros: number of zero values;
- pct_zeros: percentage of zero values.

Value

A tibble containing one row per time series and the calculated summary statistics.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_split\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

summarise_data(
  .data = data,
  context = context
)
```

summarise_split

Summarise train-test splits

Description

Summarise the time and row-index ranges of training and test samples.

Usage

```
summarise_split(data)
```

Arguments

data A valid tsibble in long format. It must contain the columns `split`, `sample`, and `id`.

Details

`summarise_split()` is intended for data sets that contain sliced training and test observations from a time series cross-validation workflow. The input must be a tsibble with the columns `split`, `sample`, and `id`:

- `split`: train-test split identifier;
- `sample`: sample label, usually "train" or "test";
- `id`: integer row index within the original time series.

The function returns one row per split. For each split, it reports the time range and index range of each sample.

Value

A tibble containing the summarized split ranges.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_stats\(\)](#)

Examples

```
library(dplyr)
library(tsibble)

context <- list(
  series_id = "bidding_zone",
  value_id = "value",
  index_id = "time"
)

main_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 120)

split_frame <- make_split(
  main_frame = main_frame,
  context = context,
  type = "first",
  value = 48,
  n_ahead = 24,
  n_skip = 23,
  n_lag = 0,
  mode = "stretch",
  exceed = FALSE
```

```
)  
  
train_frame <- slice_train(  
  main_frame = main_frame,  
  split_frame = split_frame,  
  context = context  
) |>  
  mutate(sample = "train")  
  
test_frame <- slice_test(  
  main_frame = main_frame,  
  split_frame = split_frame,  
  context = context  
) |>  
  mutate(sample = "test")  
  
split_data <- bind_rows(train_frame, test_frame) |>  
  group_by(bidding_zone, split, sample) |>  
  mutate(id = row_number()) |>  
  ungroup() |>  
  as_tsibble(  
    index = time,  
    key = c(bidding_zone, split, sample)  
  )  
  
summarise_split(split_data)
```

summarise_stats

Summarise distributional statistics by time series

Description

Calculate descriptive statistics for one or more time series.

Usage

```
summarise_stats(.data, context)
```

Arguments

`.data` A tibble in long format containing time series data.
`context` A named list with the identifiers for `series_id`, `value_id`, and `index_id`.

Details

`summarise_stats()` groups the input data by the series identifier supplied in `context` and returns one row per time series.

The function reports:

- mean: arithmetic mean;
- median: median;
- mode: kernel-density based mode estimate;
- sd: standard deviation;
- p0: minimum;
- p25: 25 percent quantile;
- p75: 75 percent quantile;
- p100: maximum;
- skewness: moment-based skewness;
- kurtosis: moment-based kurtosis.

Missing values are removed when calculating the statistics.

Value

A tibble containing one row per time series and the calculated descriptive statistics.

See Also

Other data analysis: [acf_vec\(\)](#), [estimate_acf\(\)](#), [estimate_kurtosis\(\)](#), [estimate_mode\(\)](#), [estimate_pacf\(\)](#), [estimate_skewness\(\)](#), [pacf_vec\(\)](#), [summarise_data\(\)](#), [summarise_split\(\)](#)

Examples

```
library(dplyr)

context <- list(
  series_id = "series",
  value_id = "value",
  index_id = "index"
)

data <- M4_monthly_data |>
  filter(series %in% c("M23100", "M14395"))

summarise_stats(
  .data = data,
  context = context
)
```

TBATS

TBATS model

Description

Specify a TBATS model for use with `fabletools::model()`.

Usage

```
TBATS(formula, ...)
```

Arguments

<code>formula</code>	A model formula specifying the response variable, for example <code>value</code> .
<code>...</code>	Further arguments passed to <code>forecast::tbats()</code> , including <code>periods</code> .

Details

`TBATS()` is a model specification wrapper around `forecast::tbats()` for the `fable`, `tsibble`, and `fabletools` ecosystem.

TBATS stands for trigonometric seasonality, Box-Cox transformation, ARMA errors, trend, and seasonal components. It can be useful for time series with multiple or complex seasonal patterns.

The seasonal periods must be supplied through the `periods` argument.

Value

A model definition that can be used inside `fabletools::model()`.

See Also

Other TBATS: [fitted.TBATS\(\)](#), [forecast.TBATS\(\)](#), [model_sum.TBATS\(\)](#), [residuals.TBATS\(\)](#)

Examples

```
library(dplyr)
library(tsibble)
library(fabletools)

train_frame <- elec_price |>
  filter(bidding_zone == "DE") |>
  slice_head(n = 24 * 21) |>
  as_tsibble(index = time)

model_frame <- train_frame |>
  model("TBATS" = TBATS(value, periods = c(24, 168)))

model_frame
```

`theme_tscv`*Custom ggplot2 theme for tscv*

Description

Create the default ggplot2 theme used by the tscv package.

Usage

```
theme_tscv(  
  base_size = 11,  
  base_family = "",  
  base_line_size = base_size/22,  
  base_rect_size = base_size/22  
)
```

Arguments

`base_size` Numeric value. Base font size.
`base_family` Character value. Base font family.
`base_line_size` Numeric value. Base line width for line elements.
`base_rect_size` Numeric value. Base line width for rectangle elements.

Details

`theme_tscv()` returns a complete ggplot2 theme with a clean layout, subtle grid lines, bottom legend placement, and formatting for plot titles, subtitles, captions, facets, and axes.

The theme is used as the default theme in the plotting helpers of the tscv package, such as `plot_line()`, `plot_point()`, `plot_bar()`, `plot_histogram()`, `plot_density()`, and `plot_qq()`.

Since the returned object is a regular ggplot2 theme, it can also be added directly to any ggplot object with `+ theme_tscv()`.

Value

A complete ggplot2 theme.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [tscv_cols\(\)](#), [tscv_pal\(\)](#)

Examples

```

library(dplyr)
library(ggplot2)

data <- M4_monthly_data |>
  filter(series == "M23100") |>
  mutate(index = as.Date(index))

# Plot with the default tscv theme
plot_line(
  data = data,
  x = index,
  y = value,
  title = "M4 Monthly Time Series",
  subtitle = "Series M23100",
  xlab = "Time",
  ylab = "Value",
  theme_set = theme_tscv()
)

# The same plot with the default ggplot2 grey theme
plot_line(
  data = data,
  x = index,
  y = value,
  title = "M4 Monthly Time Series",
  subtitle = "Series M23100",
  xlab = "Time",
  ylab = "Value",
  theme_set = theme_grey()
)

# theme_tscv() can also be added to regular ggplot objects
ggplot(data, aes(x = index, y = value)) +
  geom_line(color = "grey35") +
  labs(
    title = "M4 Monthly Time Series",
    subtitle = "Series M23100",
    x = "Time",
    y = "Value"
  ) +
  theme_tscv()

```

tscv_cols

Extract tscv colors

Description

Extract named colors from the tscv color palette as hexadecimal color codes.

Usage

```
tscv_cols(...)
```

Arguments

... Character values giving the names of colors to extract.

Details

`tscv_cols()` returns the hexadecimal color codes used by the `tscv` package. If no color names are supplied, all available colors are returned.

Available colors are: "red", "green", "blue", "orange", "yellow", "light grey", and "dark grey".

Value

A named character vector of hexadecimal color codes.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_pal\(\)](#)

Examples

```
# Return all available tscv colors
tscv_cols()

# Return selected colors
tscv_cols("steelblue", "orange", "green")

# Use a tscv color in a plot
library(dplyr)

data <- M4_monthly_data |>
  filter(series == "M23100")

plot_line(
  data = data,
  x = index,
  y = value,
  title = "M4 Monthly Time Series",
  subtitle = "Series M23100",
  xlab = "Time",
  ylab = "Value",
  line_color = tscv_cols("steelblue")
)
```

tscv_pal	<i>Create a tscv color palette</i>
----------	------------------------------------

Description

Create a color interpolation function based on one of the predefined tscv palettes.

Usage

```
tscv_pal(palette = "main", reverse = FALSE, ...)
```

Arguments

palette	Character value. Name of the palette.
reverse	Logical value. If TRUE, the palette is reversed.
...	Additional arguments passed to <code>grDevices::colorRampPalette()</code> .

Details

`tscv_pal()` returns a palette function created with `grDevices::colorRampPalette()`. The returned function can be used to generate any number of colors from the selected palette.

Available palettes are:

- "main": blue, green, yellow.
- "cool": blue, green.
- "hot": yellow, orange, red.
- "mixed": blue, green, yellow, orange, red.
- "grey": light grey, dark grey.

Value

A palette function that takes an integer and returns hexadecimal color codes.

See Also

Other data visualization: [plot_bar\(\)](#), [plot_density\(\)](#), [plot_histogram\(\)](#), [plot_line\(\)](#), [plot_point\(\)](#), [plot_qq\(\)](#), [scale_color_tscv\(\)](#), [scale_fill_tscv\(\)](#), [theme_tscv\(\)](#), [tscv_cols\(\)](#)

Examples

```
# Create a palette function
pal <- tscv_pal("main")

# Generate five colors
pal(5)

# Reverse the palette
```

```
tscv_pal("hot", reverse = TRUE)(5)

# Use generated colors in base R
barplot(
  height = c(3, 5, 4),
  col = tscv_pal("main")(3)
)
```

Index

- * **DSHW**
 - DSHW, 6
 - fitted.DSHW, 13
 - forecast.DSHW, 19
 - model_sum.DSHW, 41
 - residuals.DSHW, 65
- * **MEDIAN**
 - fitted.MEDIAN, 14
 - forecast.MEDIAN, 20
 - MEDIAN, 40
 - model_sum.MEDIAN, 42
 - residuals.MEDIAN, 66
- * **SMEAN**
 - fitted.SMEAN, 15
 - forecast.SMEAN, 21
 - model_sum.SMEAN, 43
 - residuals.SMEAN, 67
 - SMEAN, 78
- * **SMEDIAN**
 - fitted.SMEDIAN, 16
 - forecast.SMEDIAN, 22
 - model_sum.SMEDIAN, 44
 - residuals.SMEDIAN, 68
 - SMEDIAN, 79
- * **SNAIVE2**
 - fitted.SNAIVE2, 17
 - forecast.SNAIVE2, 23
 - model_sum.SNAIVE2, 45
 - residuals.SNAIVE2, 69
 - SNAIVE2, 81
- * **TBATS**
 - fitted.TBATS, 18
 - forecast.TBATS, 24
 - model_sum.TBATS, 46
 - residuals.TBATS, 70
 - TBATS, 89
- * **accuracy functions**
 - mae_vec, 27
 - make_accuracy, 28
 - make_errors, 31
 - mape_vec, 38
 - me_vec, 39
 - mpe_vec, 47
 - mse_vec, 48
 - rmse_vec, 71
 - smape_vec, 77
- * **data analysis**
 - acf_vec, 3
 - estimate_acf, 8
 - estimate_kurtosis, 9
 - estimate_mode, 10
 - estimate_pacf, 11
 - estimate_skewness, 12
 - pacf_vec, 49
 - summarise_data, 84
 - summarise_split, 85
 - summarise_stats, 87
- * **data preparation**
 - check_data, 4
 - interpolate_missing, 25
 - smooth_outlier, 80
- * **data visualization**
 - plot_bar, 50
 - plot_density, 52
 - plot_histogram, 55
 - plot_line, 57
 - plot_point, 60
 - plot_qq, 62
 - scale_color_tscv, 72
 - scale_fill_tscv, 73
 - theme_tscv, 90
 - tscv_cols, 91
 - tscv_pal, 93
- * **datasets**
 - elec_load, 7
 - elec_price, 7
 - M4_monthly_data, 26
 - M4_quarterly_data, 27

- * **time series cross-validation**
 - make_future, 33
 - make_split, 35
 - make_tsibble, 37
 - slice_test, 74
 - slice_train, 76
 - split_index, 82
- acf_vec, 3, 9–13, 49, 85, 86, 88
- check_data, 4, 25, 81
- DSHW, 6, 14, 19, 41, 65
- elec_load, 7
- elec_price, 7
- estimate_acf, 4, 8, 10–13, 49, 85, 86, 88
- estimate_kurtosis, 4, 9, 9, 11–13, 49, 85, 86, 88
- estimate_mode, 4, 9, 10, 10, 12, 13, 49, 85, 86, 88
- estimate_pacf, 4, 9, 10, 11, 11, 13, 49, 85, 86, 88
- estimate_skewness, 4, 9–11, 12, 12, 49, 85, 86, 88
- fitted.DSHW, 6, 13, 19, 41, 65
- fitted.MEDIAN, 14, 20, 40, 42, 66
- fitted.SMEAN, 15, 21, 43, 67, 79
- fitted.SMEDIAN, 16, 22, 44, 68, 80
- fitted.SNAIVE2, 17, 23, 45, 69, 82
- fitted.TBATS, 18, 24, 46, 70, 89
- forecast.DSHW, 6, 14, 19, 41, 65
- forecast.MEDIAN, 15, 20, 40, 42, 66
- forecast.SMEAN, 15, 21, 43, 67, 79
- forecast.SMEDIAN, 16, 22, 44, 68, 80
- forecast.SNAIVE2, 17, 23, 45, 69, 82
- forecast.TBATS, 18, 24, 46, 70, 89
- interpolate_missing, 5, 25, 81
- M4_monthly_data, 26
- M4_quarterly_data, 27
- mae_vec, 27, 29, 32, 39, 40, 47, 48, 71, 78
- make_accuracy, 28, 28, 32, 39, 40, 47, 48, 71, 78
- make_errors, 28, 29, 31, 39, 40, 47, 48, 71, 78
- make_future, 33, 36, 38, 75, 76, 83
- make_split, 34, 35, 38, 75, 76, 83
- make_tsibble, 34, 36, 37, 75, 76, 83
- mape_vec, 28, 29, 32, 38, 40, 47, 48, 71, 78
- me_vec, 28, 29, 32, 39, 39, 47, 48, 71, 78
- MEDIAN, 15, 20, 40, 42, 66
- model_sum.DSHW, 6, 14, 19, 41, 65
- model_sum.MEDIAN, 15, 20, 40, 42, 66
- model_sum.SMEAN, 15, 21, 43, 67, 79
- model_sum.SMEDIAN, 16, 22, 44, 68, 80
- model_sum.SNAIVE2, 17, 23, 45, 69, 82
- model_sum.TBATS, 18, 24, 46, 70, 89
- mpe_vec, 28, 29, 32, 39, 40, 47, 48, 71, 78
- mse_vec, 28, 29, 32, 39, 40, 47, 48, 71, 78
- pacf_vec, 4, 9–13, 49, 85, 86, 88
- plot_bar, 50, 54, 57, 59, 61, 64, 72, 73, 90, 92, 93
- plot_density, 51, 52, 57, 59, 61, 64, 72, 73, 90, 92, 93
- plot_histogram, 51, 54, 55, 59, 61, 64, 72, 73, 90, 92, 93
- plot_line, 51, 54, 57, 57, 61, 64, 72, 73, 90, 92, 93
- plot_point, 51, 54, 57, 59, 60, 64, 72, 73, 90, 92, 93
- plot_qq, 51, 54, 57, 59, 61, 62, 72, 73, 90, 92, 93
- residuals.DSHW, 6, 14, 19, 41, 65
- residuals.MEDIAN, 15, 20, 40, 42, 66
- residuals.SMEAN, 15, 21, 43, 67, 79
- residuals.SMEDIAN, 16, 22, 44, 68, 80
- residuals.SNAIVE2, 17, 23, 45, 69, 82
- residuals.TBATS, 18, 24, 46, 70, 89
- rmse_vec, 28, 29, 32, 39, 40, 47, 48, 71, 78
- scale_color_tscv, 51, 54, 57, 59, 61, 64, 72, 73, 90, 92, 93
- scale_fill_tscv, 51, 54, 57, 59, 61, 64, 72, 73, 90, 92, 93
- slice_test, 34, 36, 38, 74, 76, 83
- slice_train, 34, 36, 38, 75, 76, 83
- smape_vec, 28, 29, 32, 39, 40, 47, 48, 71, 77
- SMEAN, 15, 21, 43, 67, 78
- SMEDIAN, 16, 22, 44, 68, 79
- smooth_outlier, 5, 25, 80
- SNAIVE2, 17, 23, 45, 69, 81
- split_index, 34, 36, 38, 75, 76, 82
- summarise_data, 4, 9–13, 49, 84, 86, 88
- summarise_split, 4, 9–13, 49, 85, 85, 88
- summarise_stats, 4, 9–13, 49, 85, 86, 87

TBATS, [18](#), [24](#), [46](#), [70](#), [89](#)
theme_tscv, [51](#), [54](#), [57](#), [59](#), [61](#), [64](#), [72](#), [73](#), [90](#),
[92](#), [93](#)
tscv_cols, [51](#), [54](#), [57](#), [59](#), [61](#), [64](#), [72](#), [73](#), [90](#),
[91](#), [93](#)
tscv_pal, [51](#), [54](#), [57](#), [59](#), [61](#), [64](#), [72](#), [73](#), [90](#),
[92](#), [93](#)