

Package ‘trendseries’

October 1, 2026

Type Package

Title Extract Trends from Time Series

Version 1.7.0

Description Provides a unified interface to extract trends, cycles, and seasonal components from monthly and quarterly time series using established filters and smoothers from econometrics and signal extraction, with frequency-aware defaults for common economic frequencies. Rolling and year-to-date aggregations are also available, including the compounded accumulation of rates of change.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

URL <https://github.com/viniciusoike/trendseries>,
<https://viniciusoike.github.io/trendseries/>

BugReports <https://github.com/viniciusoike/trendseries/issues>

Imports cli, dlm, hpfiler, lubridate, mFilter, RcppRoll, rlang,
stats, tibble, tsbox, vctrs

Depends R (>= 4.1.0)

RoxygenNote 7.3.3

Suggests dplyr, ekioplot, ggplot2 (>= 4.0.0), knitr, rmarkdown,
scales, seasonal, testthat (>= 3.0.0), tidyr, tsibble

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Vinicius Oike [aut, cre, cph] (ORCID:
<<https://orcid.org/0009-0005-8015-9189>>)

Maintainer Vinicius Oike <viniciusoike@gmail.com>

Repository CRAN

Date/Publication 2026-10-01 12:00:02 UTC

Contents

augment_rolling	2
augment_trends	5
coffee_arabica	8
coffee_robusta	9
converters	10
decompose_series	11
deseason_series	15
detrend_series	18
df_to_ts	21
electric	22
electricity	22
extract_trends	23
gdp_construction	28
ibcbr	29
index_series	29
metadata_series	31
oil_derivatives	32
retail_autofuel	32
retail_volume	33
roll_series	34
transit_london_avgs	37
transit_london_monthly	38
ts_to_df	39
vehicles	39

Index	41
--------------	-----------

augment_rolling	<i>Add rolling aggregation columns to a data frame</i>
-----------------	--

Description

Pipe-friendly companion to [augment_trends\(\)](#) for rolling and year-to-date aggregations: 12-month accumulated totals, compounded rates of change, rolling volatility, and so on. Columns are prefixed `roll_` rather than `trend_`, because these are aggregations of the series and not estimates of its trend.

Usage

```
augment_rolling(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  stats = "sum",
  window = NULL,
  frequency = NULL,
```

```

    align = "right",
    percent = FALSE,
    na_rm = FALSE,
    suffix = NULL,
    .quiet = FALSE
  )

```

Arguments

<code>data</code>	A <code>data.frame</code> , <code>tibble</code> , or <code>data.table</code> containing the time series data.
<code>date_col</code>	Name of the date column. Defaults to "date". Must be of class <code>Date</code> .
<code>value_col</code>	Name of the value column(s). Defaults to "value". Must be numeric. A character vector of length > 1 is accepted; aggregations are computed for each column and named <code>roll_{stat}_{window}_{col}</code> .
<code>group_cols</code>	Optional grouping variables for multiple time series. Can be a character vector of column names.
<code>stats</code>	Character vector of rolling statistics. Options: "sum" (rolling total of flows), "chain" (compound accumulation of rates, $\text{prod}(1 + r) - 1$), "change" (change of a level over window periods, $x[t] / x[t - \text{window}] - 1$), "mean", "sd", "min", "max". Default is "sum".
<code>window</code>	Window length in periods, or the lag for "change". If <code>NULL</code> , defaults to the detected frequency (12 for monthly, 4 for quarterly). A numeric vector adds one column per window value. Alternatively, the string "ytd" computes an expanding year-to-date accumulation that resets each January (or Q1), and "all" an expanding accumulation from the first observation. Numeric and character windows cannot be mixed in one call, and "change" needs a numeric window.
<code>frequency</code>	The frequency of the series. Supports values from 1 (annual) to 365 (daily). Auto-detected if not specified.
<code>align</code>	Alignment of the window relative to the output position: "right" (default), "center", or "left". Ignored by "change" and by the expanding windows "ytd" and "all". An even window has no exact centre; see roll_series() for how each statistic handles that.
<code>percent</code>	Only used by <code>stats = "chain"</code> and <code>stats = "change"</code> . For "chain", if <code>FALSE</code> (default), rates are assumed to be decimals (0.005 for 0.5%). If <code>TRUE</code> , rates are assumed to be percentages (0.5 for 0.5%) and the result is returned in percent. For "change", <code>TRUE</code> returns the change in percent instead of as a decimal.
<code>na_rm</code>	If <code>TRUE</code> , missing values are ignored within each window. The default <code>FALSE</code> propagates NA, so an incomplete window yields NA. A window holding no observed values yields NA either way, as does a window holding one value for "sd". For even centered means, observed weights are renormalized under <code>na_rm = TRUE</code> ; boundary padding is kept. "change" ignores it: a missing value at either end yields NA.
<code>suffix</code>	Optional suffix appended to the generated column names.
<code>.quiet</code>	If <code>TRUE</code> , suppress informational messages.

Details

Use "sum" for flows measured in levels and "chain" for series that are already rates of change. Summing monthly inflation rates approximates the 12-month accumulation but is not equal to it; "chain" compounds them correctly. "change" turns a level into its rate of change, matching each date with the one window periods earlier rather than the row window positions above. See [roll_series\(\)](#) for the underlying computation.

"mean" overlaps with the simple moving average available through `augment_trends(methods = "ma")`. The two differ in defaults rather than in substance: rolling aggregations default to right alignment, while the moving average trend defaults to centred alignment. Given the same window and alignment they agree, including the 2xN correction for even centred windows.

Rows whose value is NA are kept in place, so window positions stay aligned with the calendar; `na_rm` then decides whether such a window yields NA or is computed from the observations that are present. Unlike [augment_trends\(\)](#), which rejects gaps inside the observed range, a rolling window has well-defined local behaviour for a gap, so these functions accept one. A period that is absent from the data altogether cannot be positioned, so it raises an error rather than shifting later observations — add the missing rows with an NA value first.

Value

A tibble with the original data plus rolling columns named `roll_{stat}_{window}` (e.g. `roll_sum_12`, `roll_chain_ytd`, `roll_change_12`), with `_{suffix}` appended when `suffix` is supplied. Rows come back in the order they were supplied in.

See Also

[roll_series\(\)](#) for the time series interface, [augment_trends\(\)](#) for trend estimation.

Examples

```
# 12-month accumulated vehicle production
vehicles |> augment_rolling(value_col = "production", window = 12)

# Several windows at once
vehicles |>
  tail(60) |>
  augment_rolling(value_col = "production", window = c(3, 6, 12))

# Rolling mean and volatility side by side
ibcbr |>
  augment_rolling(value_col = "index", stats = c("mean", "sd"), window = 12)

# Year-to-date accumulation, resetting each January
vehicles |> augment_rolling(value_col = "production", window = "ytd")

# 12-month change of an index, in percent
ibcbr |>
  augment_rolling(value_col = "index", stats = "change", percent = TRUE)

# Grouped series
retail_volume |>
```

```
augment_rolling(group_cols = "name_series", window = 12)
```

`augment_trends`*Add trend columns to data frame*

Description

Pipe-friendly function that adds trend columns to a tibble or data.frame. Designed for exploratory analysis of monthly and quarterly economic time series. Supports multiple trend extraction methods and handles grouped data.

Usage

```
augment_trends(  
  data,  
  date_col = "date",  
  value_col = "value",  
  group_cols = NULL,  
  group_vars = NULL,  
  methods = "stl",  
  frequency = NULL,  
  suffix = NULL,  
  window = NULL,  
  smoothing = NULL,  
  band = NULL,  
  align = NULL,  
  params = list(),  
  .quiet = FALSE  
)
```

Arguments

<code>data</code>	A data.frame, tibble, data.table, or tsibble containing the time series data. Tsibble support requires the optional tsibble package.
<code>date_col</code>	Name of the date column. Defaults to "date". Must be of class Date for data frames. For tsibbles, defaults to the index and must name that index when supplied.
<code>value_col</code>	Name of the value column(s). Defaults to "value". Must be numeric. A character vector of length > 1 is accepted; trends are extracted for each column and named trend_{method}_{col} (e.g. trend_stl_consumption).
<code>group_cols</code>	Optional grouping variables for multiple time series. Can be a character vector of column names. For tsibbles, defaults to the key and must match it when supplied.
<code>group_vars</code>	Deprecated. Use group_cols instead.

methods	Character vector of trend methods. Options: "hp", "bk", "cf", "ma", "stl", "loess", "spline", "poly", "bn", "ucm", "hamilton", "spencer", "henderson", "ewma", "wma", "triangular", "kernel", "kalman", "median", "gaussian". Default is "stl".
frequency	The frequency of the series. Supports values from 1 (annual) to 365 (daily). Auto-detected for data frames; a tsibble's yearmonth or yearquarter index supplies 12 or 4. A Date index uses the usual detection. Other tsibble index classes are not supported.
suffix	Optional suffix for trend column names. If NULL, uses method names.
window	Unified window/period parameter for moving average methods (ma, wma, triangular, stl, ewma, median, gaussian, henderson). Must be positive. If NULL, uses frequency-appropriate defaults. For EWMA, the window is converted to the smoothing factor via $\alpha = 2 / (\text{window} + 1)$. Cannot be used simultaneously with smoothing for EWMA method. For ma, median, and henderson methods, a numeric vector is accepted (e.g., c(9, 13, 23)), which adds one column per window value named trend_henderson_9, trend_henderson_13, etc. Other methods ignore extra values (with a warning).
smoothing	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: use large values (1600+) or small values (0-1) that get converted. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with window for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: a finite, positive ratio of measurement to process noise (higher = more smoothing). An explicit noise variance in params determines the other variance from this ratio. If both variances are supplied, they take precedence over smoothing. Without a ratio, unspecified measurement and process variances default to 0.1 and 0.01 times the series variance. For others: typically 0-1 range.
band	Unified band parameter for bandpass filters (bk, cf). Provide as c(low, high): the shortest and longest cycle to remove, in periods of the series (months for monthly data). Both values must be positive. Defaults to cycles of 1.5 to 8 years: c(6, 32) for quarterly data, c(18, 96) for monthly, and c(2, 8) for annual.
align	Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: "center" (default, uses surrounding values), "right" (causal, uses past values only), "left" (anti-causal, uses future values only). Note: triangular only supports "center" and "right". If NULL, uses "center" as default.
params	Optional list of method-specific parameters for fine control.
.quiet	If TRUE, suppress informational messages.

Details

This function is designed for monthly (frequency = 12) and quarterly (frequency = 4) economic data, and the defaults for each method follow the conventions for those frequencies.

For grouped data, the function applies trend extraction to each group separately, maintaining the original data structure while adding trend columns. For tsibbles, only Date, yearmonth, and

yearquarter indices are supported; the existing missing-period rules apply after conversion to calendar dates.

Value

A tibble with original data plus trend columns named `trend_{method}` or `trend_{method}_{suffix}` if suffix is provided. Rows come back in the order they were supplied in. A tsibble input returns a tsibble with its index class and key preserved.

Examples

```
# Simple STL decomposition on quarterly GDP construction data
gdp_construction |> augment_trends(value_col = "index")
```

```
# Multiple smoothing methods with unified parameter
gdp_construction |>
  augment_trends(
    value_col = "index",
    methods = c("hp", "loess", "ewma"),
    smoothing = 0.3
  )
```

```
# Moving averages with unified window on monthly data
vehicles |>
  tail(60) |>
  augment_trends(
    value_col = "production",
    methods = c("ma", "wma", "triangular"),
    window = 8
  )
```

```
# Economic indicators with different methods
ibcbr |>
  tail(48) |>
  augment_trends(
    value_col = "index",
    methods = c("median", "kalman", "kernel"),
    window = 9,
    smoothing = 0.15
  )
```

```
# Moving average with right alignment (causal filter)
vehicles |>
  tail(60) |>
  augment_trends(
    value_col = "production",
    methods = "ma",
    window = 12,
    align = "right"
  )
```

```
# Advanced: fine-tune specific methods
```

```

electric |>
  tail(72) |>
  augment_trends(
    value_col = "consumption",
    methods = "median",
    window = 7
  )

# Multiple MA windows in a single call (adds trend_ma_3, trend_ma_6, trend_ma_12)
vehicles |>
  tail(60) |>
  augment_trends(
    value_col = "production",
    methods = "ma",
    window = c(3, 6, 12)
  )

# Preserve a tsibble's index and key (if tsibble is installed)
if (requireNamespace("tsibble", quietly = TRUE)) {
  quarterly <- gdp_construction
  quarterly$date <- tsibble::yearquarter(quarterly$date)
  quarterly <- tsibble::as_tsibble(quarterly, index = date)
  augment_trends(quarterly, value_col = "index", methods = "hp")
}

```

coffee_arabica

Daily Arabica Coffee Price Indicator

Description

Daily observations of the CEPEA/ESALQ Arabica coffee price indicator, accompanied by an inflation-adjusted dollar series and a derived moving-average column.

spot_rs and spot_us are nominal at-sight prices in Brazilian reais and U.S. dollars per 60-kg bag. CEPEA converts the dollar value using the commercial selling exchange rate quoted at 16:30. usd_2022 is calculated in this package from spot_us using U.S. inflation data and priceR.

CEPEA defines the indicator as type 6 Arabica coffee, hard cup or better, well prepared, with up to 86 defects in a 300-g sample. It represents wholesale lot transactions delivered to the city of São Paulo, with freight from the producing region included. The reference regions are Cerrado and southern Minas Gerais, Mogiana and Garça in São Paulo, and northwestern Paraná. Regional weights are based on production volumes reported by IBGE and are updated on the first business day of a new crop. CEPEA excludes observations outside two standard deviations from the sample mean and includes taxes in the reported values. For term transactions, CEPEA discounts prices at CDI from 2021-03-01; before that it used NPR. The survey consults cooperatives, brokers, roasters, and exporters.

The indicator is daily and its historical series begins in September 1996. The bundled observations run from 1996-09-02 through 2025-04-17.

Usage

```
coffee_arabica
```

Format

A tibble with daily observations:

date Observation date (Date).

spot_rs CEPEA at-sight price in Brazilian reais per 60-kg bag.

spot_us CEPEA at-sight price in U.S. dollars per 60-kg bag.

usd_2022 spot_us adjusted to 2022 U.S. dollars using priceR.

trend_ma 22-observation, right-aligned moving average of usd_2022. The first 21 observations have no full window and are NA.

Source

Original data: Centro de Estudos Avançados em Economia Aplicada (CEPEA), Escola Superior de Agricultura Luiz de Queiroz (ESALQ), Universidade de São Paulo (USP), [CEPEA/ESALQ Arabica price series](#). See the [CEPEA Arabica methodology](#). CEPEA identifies its coffee data as available under the [CC BY-NC 4.0 license](#). This package distributes an adapted version by adding usd_2022 and trend_ma.

See Also

[coffee_robusta](#)

coffee_robusta

Daily Robusta Coffee Price Indicator

Description

Daily observations of the CEPEA/ESALQ Robusta coffee price indicator, accompanied by an inflation-adjusted dollar series and a derived moving-average column.

spot_rs and spot_us are nominal at-sight prices in Brazilian reais and U.S. dollars per 60-kg bag. CEPEA converts the dollar value using the commercial selling exchange rate quoted at 16:30. usd_2022 is calculated in this package from spot_us using U.S. inflation data and priceR.

CEPEA defines the indicator as type 6 Robusta coffee, screen 13 or above, with 86 defects. It represents wholesale lot transactions to be collected in the producing region, so the indicator does not include freight. The reference regions are Colatina and São Gabriel da Palha in Espírito Santo, and the final indicator is their arithmetic average. CEPEA includes taxes in the reported values and excludes observations outside two standard deviations from the sample mean. For term transactions, CEPEA discounts prices at CDI from 2021-03-01; before that it used NPR. The survey consults cooperatives, brokers, roasters, and exporters.

The indicator is daily and its historical series begins in November 2001. The bundled observations run from 2001-11-08 through 2025-04-17.

Usage

coffee_robusta

Format

A tibble with daily observations:

date Observation date (Date).

spot_rs CEPEA at-sight price in Brazilian reais per 60-kg bag.

spot_us CEPEA at-sight price in U.S. dollars per 60-kg bag.

usd_2022 spot_us adjusted to 2022 U.S. dollars using priceR.

trend_ma 22-observation, right-aligned moving average of usd_2022. The first 21 observations have no full window and are NA.

Source

Original data: Centro de Estudos Avançados em Economia Aplicada (CEPEA), Escola Superior de Agricultura Luiz de Queiroz (ESALQ), Universidade de São Paulo (USP), [CEPEA/ESALQ Robusta price series](#). See the [CEPEA Robusta methodology](#). CEPEA identifies its coffee data as available under the [CC BY-NC 4.0 license](#). This package distributes an adapted version by adding usd_2022 and trend_ma.

See Also

[coffee_arabica](#)

converters

Data Format Conversion Utilities

Description

Functions for converting between different time series formats, frequency detection, and data frame manipulation for the trendseries package. These functions handle the interface between tibble/data.frame workflows and time series objects.

decompose_series	<i>Decompose time series into trend, seasonal, and remainder components</i>
------------------	---

Description

Pipe-friendly function that decomposes a time series into its trend, seasonal, and remainder components, adding them as columns to the input data frame.

Usage

```
decompose_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "stl",
  trend = "linear",
  transform = "none",
  frequency = NULL,
  seasadj = FALSE,
  params = list(),
  .quiet = FALSE
)
```

Arguments

data	A <code>data.frame</code> , <code>tibble</code> , or <code>data.table</code> containing the time series data.
date_col	Name of the date column. Defaults to "date". Must be of class <code>Date</code> .
value_col	Name of the value column. Defaults to "value". Must be numeric.
group_cols	Optional grouping variables for multiple time series. A character vector of column names. When provided, decomposition is applied independently to each group.
methods	Decomposition method(s). One or more of "stl", "regression", "classic", "bsm", or "seats". Default is "stl". When several methods are supplied (e.g. <code>c("stl", "classic")</code>), each one contributes its own <code>trend_*</code> , <code>seasonal_*</code> , and <code>remainder_*</code> columns so decompositions can be compared side by side. • "stl": Seasonal-Trend decomposition via Loess (<code>stats::stl()</code>). • "regression": joint OLS trend + seasonal-dummy model. • "classic": classical decomposition via moving averages (<code>stats::decompose()</code>). • "bsm": Basic Structural (state-space) Model via the Kalman smoother (<code>stats::StructTS()</code>); supports frequencies up to 12. • "seats": X-13ARIMA-SEATS decomposition (requires the seasonal package; see Details).

trend	For methods = "regression" only: the polynomial form of the trend component. One of "linear", "quadratic", or "cubic". Ignored by the other methods. Default is "linear".
transform	Transformation applied to the series before decomposition. One of "none" (default, additive decomposition) or "log". With "log", the series is log-transformed, decomposed additively, and the components are exponentiated back, yielding a <i>multiplicative</i> decomposition.
frequency	The frequency of the series. Must be greater than 1; "bsm" supports at most 12. Will be auto-detected if not specified.
seasadj	If TRUE, also add a seasadj_{method} column holding the seasonally adjusted series (the series with the seasonal component removed: trend + remainder for additive decompositions, trend * remainder for multiplicative ones). Default FALSE.
params	Optional list of method-specific parameters for fine control. Every parameter has a default, so this argument is only needed for non-standard use cases. For STL (methods = "stl"): • s.window or stl_s_window: seasonal smoothing window. Either "periodic" (default, assumes constant seasonal pattern) or a positive odd integer (larger values allow more slowly evolving seasonality). • t.window or stl_t_window: trend smoothing window (odd integer, or NULL to let stats::stl() choose automatically — recommended default). • robust or stl_robust: logical. If TRUE, uses robust fitting to reduce the influence of outliers. Default FALSE. For regression (methods = "regression"): • poly_raw: logical. If FALSE (default), uses orthogonal polynomials (numerically stable, recommended). If TRUE, uses raw polynomials (more interpretable coefficients, less stable for degree >= 2). classic , bsm , and seats take no params. For multiplicative seasonality with any method, use transform = "log".
.quiet	If TRUE, suppress informational messages.

Details

All methods require seasonal data (frequency > 1). For non-seasonal (annual) series, use [augment_trends\(\)](#) to extract a trend component only.

STL Decomposition:

Uses stats::stl() (Seasonal-Trend decomposition via Loess). The seasonal component is estimated with a loess smoother, the trend with an adaptive moving average, and the remainder is the residual. The defaults (s.window = "periodic", robust = FALSE) assume a stable seasonal pattern.

Regression Decomposition:

Fits a joint OLS model:

$$y_t = f(t) + s(t) + \epsilon_t$$

where $f(t)$ is a polynomial in time and $s(t)$ is captured by period dummy variables (month or quarter indicators). The components are isolated via stats::predict(type = "terms"):

- **Trend:** constant + polynomial terms (captures the long-run level and direction).
- **Seasonal:** period dummy terms, centred to mean zero over the sample.
- **Remainder:** residuals from the full model.

By default, orthogonal polynomials (`poly_raw = FALSE`) are used for numerical stability, which matters most for `trend = "cubic"`.

Classical Decomposition:

Uses `stats::decompose()`. The trend is a centred moving average of order equal to the frequency; the seasonal component is the average detrended value for each period; the remainder is the residual. Simple and fast, but the other methods handle evolving seasonality and the endpoints better.

Basic Structural Model (BSM):

Uses `stats::StructTS(type = "BSM")`, a state-space model with stochastic level, slope, and seasonal components estimated by maximum likelihood and extracted with the Kalman smoother (`stats::tsSmooth()`). Unlike the moving-average methods it produces trend and seasonal estimates for every observation, including the endpoints, and lets both components evolve over time. Fitting relies on numerical optimisation and can occasionally fail to converge on short or irregular series.

X-13ARIMA-SEATS (SEATS):

Uses the seasonal package, which wraps the U.S. Census Bureau's X-13ARIMA-SEATS program. `seas()` is run with its automatic defaults (model selection, log/level transformation, outlier detection, and calendar adjustment), and the SEATS trend-cycle (`s12`) and seasonally adjusted series (`s11`) are mapped to an additive trend/seasonal/remainder, whichever transformation X-13 picked internally. Because X-13 picks that transformation itself, `seats` is best used with the default `transform = "none"`; an outer log transform is redundant.

Multiplicative Seasonality:

When the seasonal amplitude grows with the level of the series (a multiplicative pattern, common in economic data), set `transform = "log"`. The series is log-transformed, decomposed additively, and the components are exponentiated back. Every method takes this same path, which requires strictly positive values.

Value

A tibble with the original columns plus, for each requested method, three new columns (and a fourth when `seasadj = TRUE`):

- `trend_{method}`: the estimated trend component.
- `seasonal_{method}`: the estimated seasonal component.
- `remainder_{method}`: what remains after removing trend and seasonal.
- `seasadj_{method}`: the seasonally adjusted series (only if `seasadj = TRUE`).

With `transform = "none"` the components should add back up to the series (`value = trend + seasonal + remainder`); with `transform = "log"` they should multiply back to it (`value = trend * seasonal * remainder`). For "classic" the trend (and hence remainder) is NA for the first and last frequency / 2 observations (the centred moving average has no boundary support).

Output rows come back in the order they were supplied in.

Examples

```

# STL decomposition (default settings work well for most economic series)
gdp_construction |>
  decompose_series(value_col = "index")

# STL with robust fitting (useful when the series has outliers)
gdp_construction |>
  decompose_series(
    value_col = "index",
    params = list(robust = TRUE)
  )

# STL with evolving seasonality (s.window controls how fast it can change)
gdp_construction |>
  decompose_series(
    value_col = "index",
    params = list(s.window = 13)
  )

# Regression with cubic trend
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "regression",
    trend = "cubic"
  )

# Classical decomposition via moving averages (boundary trend is NA)
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "classic"
  )

# Basic Structural Model (state-space, components for every observation)
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "bsm"
  )

# X-13ARIMA-SEATS (requires the 'seasonal' package)
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    decompose_series(
      value_col = "index",
      methods = "seats"
    )
}

# Multiplicative decomposition via log transform (works for any method)
oil_derivatives |>

```

```

decompose_series(
  value_col = "production",
  transform = "log"
)

# Several methods at once for side-by-side comparison
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = c("stl", "classic")
  )

# Also return the seasonally adjusted series
gdp_construction |>
  decompose_series(
    value_col = "index",
    seasadj = TRUE
  )

# Grouped decomposition: one decomposition per electricity sector
electricity |>
  decompose_series(
    group_cols = "name_series"
  )

```

deseason_series	<i>Seasonally adjust (deseason) a time series</i>
-----------------	---

Description

Pipe-friendly convenience wrapper around [decompose_series\(\)](#) focused on a single task: removing the seasonal component from a time series. It adds a `seasadj_{method}` column holding the seasonally adjusted (deseasoned) series and, optionally, the underlying trend, seasonal, and remainder components.

Usage

```

deseason_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "stl",
  transform = "none",
  frequency = NULL,
  components = FALSE,
  params = list(),
  .quiet = FALSE
)

```

Arguments

<code>data</code>	A <code>data.frame</code> , <code>tibble</code> , or <code>data.table</code> containing the time series data.
<code>date_col</code>	Name of the date column. Defaults to "date". Must be of class <code>Date</code> .
<code>value_col</code>	Name of the value column. Defaults to "value". Must be numeric.
<code>group_cols</code>	Optional grouping variables for multiple time series. A character vector of column names. When provided, decomposition is applied independently to each group.
<code>methods</code>	Seasonal-adjustment method(s). One or more of "stl" (default) or "seats". When both are supplied, each contributes its own <code>seasadj_{method}</code> column (and component columns when <code>components = TRUE</code>) so the adjustments can be compared side by side. "stl": Seasonal-Trend decomposition via Loess (<code>stats::stl()</code>). "seats": X-13ARIMA-SEATS decomposition (requires the <code>seasonal</code> package; see decompose_series() for details).
<code>transform</code>	Transformation applied to the series before decomposition. One of "none" (default, additive decomposition) or "log". With "log", the series is log-transformed, decomposed additively, and the components are exponentiated back, yielding a <i>multiplicative</i> decomposition.
<code>frequency</code>	The frequency of the series. Must be greater than 1; "bsm" supports at most 12. Will be auto-detected if not specified.
<code>components</code>	If <code>FALSE</code> (default), only the seasonally adjusted <code>seasadj_{method}</code> column is added. If <code>TRUE</code> , the <code>trend_{method}</code> , <code>seasonal_{method}</code> , and <code>remainder_{method}</code> columns are also added (the full decompose_series() output).
<code>params</code>	Optional list of method-specific parameters for fine control. Every parameter has a default, so this argument is only needed for non-standard use cases. For STL (<code>methods = "stl"</code>): <code>s.window</code> or <code>stl_s_window</code>: seasonal smoothing window. Either "periodic" (default, assumes constant seasonal pattern) or a positive odd integer (larger values allow more slowly evolving seasonality). <code>t.window</code> or <code>stl_t_window</code>: trend smoothing window (odd integer, or <code>NULL</code> to let <code>stats::stl()</code> choose automatically — recommended default). <code>robust</code> or <code>stl_robust</code>: logical. If <code>TRUE</code>, uses robust fitting to reduce the influence of outliers. Default <code>FALSE</code>. For regression (<code>methods = "regression"</code>): <code>poly_raw</code>: logical. If <code>FALSE</code> (default), uses orthogonal polynomials (numerically stable, recommended). If <code>TRUE</code>, uses raw polynomials (more interpretable coefficients, less stable for degree ≥ 2). classic, bsm, and seats take no <code>params</code>. For multiplicative seasonality with any method, use <code>transform = "log"</code>.
<code>.quiet</code>	If <code>TRUE</code> , suppress informational messages.

Details

`deseason_series()` is a thin wrapper: it calls `decompose_series()` with `seasadj = TRUE` and then keeps only the seasonally adjusted column unless `components = TRUE`. All seasonal-adjustment behaviour, validation, grouping, and the `transform = "log"` (multiplicative) path are inherited unchanged from `decompose_series()`. See its documentation for method internals and the meaning of the `params` argument.

For a full trend/seasonal/remainder decomposition, or for the regression, classic, and bsm methods, use `decompose_series()` directly.

Value

A tibble with the original columns plus, for each requested method, a `seasadj_{method}` column holding the seasonally adjusted series. When `components = TRUE`, the `trend_{method}`, `seasonal_{method}`, and `remainder_{method}` columns are added as well.

The seasonally adjusted series is the series with the seasonal component removed: `trend + remainder` for additive decompositions, `trend * remainder` when `transform = "log"`. Output rows come back in the order they were supplied in.

See Also

`decompose_series()` for the underlying decomposition and the full set of methods; `augment_trends()` to extract a trend component only.

Examples

```
# Seasonally adjust a quarterly series (STL, the default)
gdp_construction |>
  deseason_series(value_col = "index")

# Also keep the trend, seasonal, and remainder components
gdp_construction |>
  deseason_series(value_col = "index", components = TRUE)

# Multiplicative adjustment via log transform (seasonal swings grow with level)
gdp_construction |>
  deseason_series(value_col = "index", transform = "log")

# X-13ARIMA-SEATS adjustment (requires the 'seasonal' package)
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    deseason_series(value_col = "index", methods = "seats")
}

# Compare STL and SEATS adjustments side by side
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    deseason_series(value_col = "index", methods = c("stl", "seats"))
}

# Grouped seasonal adjustment: one adjustment per electricity sector
```

```
electricity |>
  deseason_series(group_cols = "name_series")
```

detrend_series	<i>Detrend a time series</i>
----------------	------------------------------

Description

Pipe-friendly convenience wrapper around `augment_trends()` focused on a single task: removing the trend from a time series. It adds a `detrend_{method}` column holding the detrended series (the deviation from trend, often called the *cycle* in economics) and, optionally, the underlying trend itself.

For econometric filters such as "hp" (the default), "bk", "cf", and "hamilton", the detrended series is the business-cycle component those filters were designed to isolate (e.g. the output gap).

Usage

```
detrend_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "hp",
  transform = "none",
  frequency = NULL,
  components = FALSE,
  window = NULL,
  smoothing = NULL,
  band = NULL,
  align = NULL,
  params = list(),
  .quiet = FALSE
)
```

Arguments

data	A <code>data.frame</code> , <code>tibble</code> , <code>data.table</code> , or <code>tsibble</code> containing the time series data. <code>Tsibble</code> support requires the optional tsibble package.
date_col	Name of the date column. Defaults to "date". Must be of class <code>Date</code> for data frames. For <code>tsibbles</code> , defaults to the index and must name that index when supplied.
value_col	Name of the value column(s). Defaults to "value". Must be numeric. A character vector of length > 1 is accepted; each column is detrended separately and the results are named <code>detrend_{method}_{col}</code> (e.g. <code>detrend_hp_consumption</code>).

group_cols	Optional grouping variables for multiple time series. Can be a character vector of column names. For tsibbles, defaults to the key and must match it when supplied.
methods	Character vector of trend methods used for detrending. Any method supported by augment_trends() is accepted. Default is "hp" (Hodrick-Prescott filter with frequency-appropriate smoothing). When several methods are supplied, each one contributes its own <code>detrend_{method}</code> column so the detrended series can be compared side by side.
transform	Transformation applied before detrending. One of: "none" (default): the trend is fitted to the raw series and <code>detrend = value - trend</code>, in the units of the series. "log": the trend is fitted to the log series and <code>detrend = log(value) - log(trend)</code>, the <i>log deviation from trend</i>. Multiplied by 100, this is approximately the percentage deviation from trend (the convention for output gaps). Requires strictly positive values. The <code>trend_{method}</code> columns (when <code>components = TRUE</code>) are reported back in the units of the series.
frequency	The frequency of the series. Supports values from 1 (annual) to 365 (daily). Auto-detected for data frames; a tsibble's yearmonth or yearquarter index supplies 12 or 4. A Date index uses the usual detection. Other tsibble index classes are not supported.
components	If FALSE (default), only the detrended <code>detrend_{method}</code> column is added. If TRUE, the fitted <code>trend_{method}</code> column is also kept.
window	Unified window/period parameter for moving average methods; see augment_trends() . For "ma", "median", and "henderson", a numeric vector is accepted (e.g. <code>c(6, 12)</code>), which adds one detrended column per window value (<code>detrend_ma_6</code> , <code>detrend_ma_12</code> , ...).
smoothing	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: use large values (1600+) or small values (0-1) that get converted. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with window for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: a finite, positive ratio of measurement to process noise (higher = more smoothing). An explicit noise variance in <code>params</code> determines the other variance from this ratio. If both variances are supplied, they take precedence over smoothing. Without a ratio, unspecified measurement and process variances default to 0.1 and 0.01 times the series variance. For others: typically 0-1 range.
band	Unified band parameter for bandpass filters (bk, cf). Provide as <code>c(low, high)</code> : the shortest and longest cycle to remove, in periods of the series (months for monthly data). Both values must be positive. Defaults to cycles of 1.5 to 8 years: <code>c(6, 32)</code> for quarterly data, <code>c(18, 96)</code> for monthly, and <code>c(2, 8)</code> for annual.
align	Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: "center" (default, uses surrounding values), "right" (causal, uses past values only), "left" (anti-causal, uses future values only). Note: triangular only supports "center" and "right". If NULL, uses "center" as default.

<code>params</code>	Optional list of method-specific parameters for fine control.
<code>.quiet</code>	If TRUE, suppress informational messages.

Details

`detrend_series()` is a thin wrapper: it calls `augment_trends()` with the requested methods and subtracts each fitted trend from the series (on the log scale when `transform = "log"`). All trend-fitting behaviour, validation, grouping, and the unified parameters (`window`, `smoothing`, `band`, `align`, `params`) are inherited unchanged from `augment_trends()`. See its documentation for method internals and parameter details. Tibble input supports the same `Date`, `yearmonth`, and `yearquarter` indices as `augment_trends()`.

Detrending does **not** remove seasonality: the detrended series of a raw seasonal series still contains the seasonal swings, and seasonality can leak into the cycle estimated by filters such as HP. For seasonal data, seasonally adjust first and detrend the adjusted series (see Examples), or use `decompose_series()` for a full trend/seasonal/remainder split.

Value

A tibble with the original columns plus, for each requested method, a `detrend_{method}` column holding the detrended series. When `components = TRUE`, the `trend_{method}` column is kept as well.

Each detrended column mirrors the name of the trend column it derives from: `window` vectors yield `detrend_ma_6`, `detrend_ma_12`, and a trend column renamed to avoid a naming conflict yields a matching detrended name.

With `transform = "none"` the trend and the detrended series should add back up to the original ($\text{value} = \text{trend} + \text{detrend}$); with `transform = "log"` the relation is $\text{value} = \text{trend} * \exp(\text{detrend})$. Methods with boundary effects (e.g. `"bk"`, `"hamilton"`) produce NA trend values at the affected observations, and the detrended series is NA there too.

Output rows come back in the order they were supplied in. A tibble input returns a tibble with its index class and key preserved.

See Also

`augment_trends()` for the underlying trend extraction and the full set of methods; `deseason_series()` to remove seasonality; `decompose_series()` for a full decomposition.

Examples

```
# HP-filter detrending (the default): adds a detrend_hp column
gdp_construction |>
  detrend_series(value_col = "index")

# Log deviation from trend (x 100 ~ percentage gap, the output-gap convention)
gdp_construction |>
  detrend_series(value_col = "index", transform = "log")

# Keep the fitted trend alongside the detrended series
gdp_construction |>
  detrend_series(value_col = "index", components = TRUE)
```

```
# Compare detrending methods side by side
gdp_construction |>
  detrend_series(value_col = "index", methods = c("hp", "stl", "loess"))

# Seasonal data: deseason first, then detrend the adjusted series
gdp_construction |>
  deseason_series(value_col = "index") |>
  detrend_series(value_col = "seasadj_stl")

# Grouped detrending: one trend per electricity sector
electricity |>
  detrend_series(group_cols = "name_series")
```

df_to_ts

*Convert a data.frame into a time series (ts)***Description**

Converts a series, stored in a `data.frame` or `tibble`, into a `ts` object.

A `ts` positions observations by period rather than by date, so the input must cover a complete run of periods. Rows are sorted before conversion, and a missing or duplicated period is rejected rather than silently shifting every later observation onto the wrong date. Missing *values* are kept in place, which leaves the series correctly dated.

Usage

```
df_to_ts(x, date_col = "date", value_col = "value", frequency = 12)
```

Arguments

<code>x</code>	A <code>data.frame</code> , <code>tibble</code> or <code>data.table</code> .
<code>date_col</code>	Name of the date column. Defaults to 'date'. Must be of class <code>Date</code> .
<code>value_col</code>	Name of the value column. Defaults to 'value'. Must be numeric.
<code>frequency</code>	The frequency of the series. Can be a shortened string (e.g. "M" for monthly) or a number (e.g. 12).

Value

A `ts` object

Examples

```
ibc <- df_to_ts(ibcbr, value_col = "index", frequency = "M")
class(ibc)
plot(ibc)
```

electric

Residential Electricity Consumption

Description

Monthly residential electricity consumption in Brazil, measured in GWh.

Usage

electric

Format

A tibble with monthly observations:

date Date of the first day of the month

consumption Electricity consumption in GWh

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date.

Source

Centrais Elétricas Brasileiras S.A. (Eletrobras), disseminated by the Banco Central do Brasil through the Sistema Gerenciador de Séries Temporais (SGS), series 1403. See the [official SGS metadata](#) and [series page](#).

electricity

Electricity Consumption by Sector

Description

Monthly electricity consumption in Brazil by sector (GWh), in long format. This dataset combines the residential, commercial, and industrial SGS series by row-binding them; it does not calculate a total or otherwise transform their values.

Usage

electricity

Format

A tibble with monthly observations:

date Date of the first day of the month

name_series Sector identifier: "electric_residential", "electric_commercial", or "electric_industrial"

value Electricity consumption in GWh

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date. The residential series is also available in wide format as [electric](#).

Source

Centrais Elétricas Brasileiras S.A. (Eletrobras), disseminated by the Banco Central do Brasil through the Sistema Gerenciador de Séries Temporais (SGS). The component series are 1402 (commercial), 1403 (residential), and 1404 (industrial). See the official [SGS metadata for series 1402, 1403, and 1404](#).

See Also

[electric](#) for the residential-only wide-format series.

extract_trends

Extract trends from time series objects

Description

Extract trend components from time series objects using various econometric methods. Designed for monthly and quarterly economic data analysis. Returns trend components as time series objects or a list of time series.

Usage

```
extract_trends(  
  ts_data,  
  methods = "stl",  
  window = NULL,  
  smoothing = NULL,  
  band = NULL,  
  align = NULL,  
  params = list(),  
  .quiet = FALSE  
)
```

Arguments

ts_data	A time series object (ts, xts, or zoo) or any object convertible via tsbox. Missing values inside the observed span are rejected: methods disagree on what to do with them, and several fail silently. Impute them first. Leading and trailing missing values are excluded from estimation and returned as NA.
methods	Character vector of trend methods. Options: "hp", "bk", "cf", "ma", "stl", "loess", "spline", "poly", "bn", "ucm", "hamilton", "spencer", "henderson", "ewma", "wma", "triangular", "kernel", "kalman", "median", "gaussian". Default is "stl".
window	Unified window/period parameter for moving average methods (ma, wma, triangular, stl, ewma, median, gaussian, henderson). Must be positive. If NULL, uses frequency-appropriate defaults. For EWMA, the window is converted to the smoothing factor via $\alpha = 2 / (\text{window} + 1)$. Cannot be used simultaneously with smoothing for EWMA method. For ma, median, and henderson methods, a numeric vector is accepted (e.g., c(9, 13, 23)), which runs the method once per window value and returns a named list with keys like henderson_9, henderson_13, henderson_23. Other methods ignore extra values (with a warning).
smoothing	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: a value above 1 is lambda itself; a value of 1 or less is a fraction of the default lambda for the frequency, $1600 * (\text{frequency} / 4)^4$. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with window for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: a finite, positive ratio of measurement to process noise (higher = more smoothing). An explicit noise variance in params determines the other variance from this ratio. If both variances are supplied, they take precedence over smoothing. Without a ratio, unspecified measurement and process variances default to 0.1 and 0.01 times the series variance. For others: typically 0-1 range.
band	Unified band parameter for bandpass filters (bk, cf). Provide as c(low, high): the shortest and longest cycle to remove, in periods of the series (months for monthly data). Both values must be positive. Defaults to cycles of 1.5 to 8 years: c(6, 32) for quarterly data, c(18, 96) for monthly, and c(2, 8) for annual.
align	Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: "center" (default, uses surrounding values), "right" (causal, uses past values only), "left" (anti-causal, uses future values only). Note: triangular only supports "center" and "right". If NULL, uses "center" as default.
params	Optional list of method-specific parameters for fine control: • HP Filter: hp_onesided (logical, default FALSE) - Use one-sided (real-time) filter instead of two-sided • STL: stl_s_window or s.window (numeric/"periodic", default "periodic") - Seasonal window, stl_t_window or t.window (numeric/NULL, default NULL) - Trend window, stl_robust or robust (logical, default FALSE)

- Use robust fitting. Note: Both dot notation (s.window) and underscore notation (stl_s_window) are accepted.
- **Spline:** spline_cv (logical/NULL) - Cross-validation method: NULL (none), TRUE (leave-one-out), FALSE (GCV)
- **Polynomial:** poly_degree (integer, default 1), poly_raw (logical, default FALSE for orthogonal polynomials)
- **UCM:** ucm_type (character) - Model type: "level", "trend", or "BSM". Defaults to "BSM" for frequencies 2 to 12 and "level" otherwise. Explicit "BSM" requests require frequency at most 12.
- **Others:** bn_ar_order, hamilton_h, hamilton_p, kernel_type, kalman_measurement_noise, kalman_process_noise, median_endrule, gaussian_sigma, wma_weights.
- **Note:** Alignment parameters (ma_align, wma_align, triangular_align, gaussian_align) can still be passed via params but it's recommended to use the unified align parameter instead.

.quiet If TRUE, suppress informational messages.

Details

This function focuses on monthly (frequency = 12) and quarterly (frequency = 4) economic data. It uses established econometric methods with appropriate defaults:

- **HP Filter:** lambda = 1600 (quarterly), 129600 (monthly), 6.25 (annual), following Ravn and Uhlig (2002). Supports both two-sided and one-sided (real-time) variants
- **Baxter-King:** Bandpass filter for business cycles (1.5 to 8 years by default)
- **Christiano-Fitzgerald:** Asymmetric bandpass filter
- **Moving Average:** Centered, frequency-appropriate windows
- **STL:** Seasonal-trend decomposition
- **Loess:** Local polynomial regression
- **Spline:** Smoothing splines
- **Polynomial:** Linear/polynomial trends
- **Beveridge-Nelson:** Permanent/transitory decomposition
- **UCM:** Unobserved Components Model (basic structural model up to monthly data, local level otherwise)
- **Hamilton:** Regression-based alternative to HP filter
- **Advanced MA:** EWMA with various implementations
- **Kernel Smoother:** Non-parametric regression with various kernel functions
- **Kalman Smoother:** Adaptive filtering for noisy time series
- **Median Filter:** Robust filtering using running medians to remove outliers
- **Gaussian Filter:** Weighted average with Gaussian (normal) density weights

Parameter Usage Notes:

- **HP Filter:** Use hp_onesided=TRUE for real-time analysis or when future data should not influence current estimates. One-sided filter is appropriate for nowcasting, policy analysis, and avoiding look-ahead bias. Default two-sided filter is optimal for historical analysis.

- **EWMA:** Use either window (converted to $\alpha = 2 / (\text{window} + 1)$) OR smoothing (alpha parameter), not both
- **Kalman:** Use smoothing parameter or params list for fine control of noise parameters
- **Spline:** Use spline_cv to control cross-validation (NULL=none, TRUE=LOO-CV, FALSE=GCV)
- **Polynomial:** Use poly_raw=FALSE for orthogonal polynomials (more stable for degree > 2) or poly_raw=TRUE for raw polynomials. Warning issued for degree > 3 (overfitting risk).
- **UCM:** Choose model type - "level" (simplest), "trend" (time-varying slope), or "BSM" (with seasonal component, requires seasonal data). Variances are estimated by maximum likelihood, so smoothing does not apply. The trend is the smoothed level. On seasonal data, "level" and "trend" can absorb the seasonality into the level and return the series itself, which is why the default is "BSM" up to monthly data. "BSM" carries one state per season and becomes very slow on weekly or daily data.

Value

If single method, returns a ts object. If multiple methods, returns a named list of ts objects.

Examples

```
# Single method
hp_trend <- extract_trends(AirPassengers, methods = "hp")

# Multiple methods with unified smoothing
smooth_trends <- extract_trends(
  AirPassengers,
  methods = c("hp", "loess", "ewma"),
  smoothing = 0.3
)

# EWMA with window (alpha derived from window size)
ewma_window <- extract_trends(AirPassengers, methods = "ewma", window = 12)

# EWMA with alpha (traditional formula)
ewma_alpha <- extract_trends(AirPassengers, methods = "ewma", smoothing = 0.2)

# Moving averages with unified window
ma_trends <- extract_trends(
  AirPassengers,
  methods = c("ma", "wma", "triangular"),
  window = 8
)

# Bandpass filters with unified band
bp_trends <- extract_trends(
  AirPassengers,
  methods = c("bk", "cf"),
  band = c(18, 96)
)

# Moving average with right alignment (causal filter)
```

```
ma_causal <- extract_trends(
  AirPassengers,
  methods = "ma",
  window = 12,
  align = "right"
)

# Signal processing methods with specific parameters
finance_trends <- extract_trends(
  AirPassengers,
  methods = c("kalman", "gaussian"),
  window = 9, # For Gaussian filter
  params = list(kalman_measurement_noise = 0.1) # Kalman-specific parameter
)

# Spline with cross-validation options
spline_trends <- extract_trends(
  AirPassengers,
  methods = "spline",
  params = list(spline_cv = FALSE) # Use GCV instead of default
)

# Polynomial with orthogonal vs raw polynomials
poly_trends <- extract_trends(
  AirPassengers,
  methods = "poly",
  params = list(poly_degree = 2, poly_raw = FALSE) # Orthogonal (default)
)

# UCM with different model types
ucm_trends <- extract_trends(
  AirPassengers,
  methods = "ucm",
  params = list(ucm_type = "BSM") # Basic Structural Model with seasonality
)

# HP Filter: One-sided (real-time) vs Two-sided (historical)
hp_realtime <- extract_trends(
  AirPassengers,
  methods = "hp",
  params = list(hp_onesided = TRUE) # For nowcasting and real-time analysis
)

# STL with custom parameters via params (both notations work)
stl_custom1 <- extract_trends(
  AirPassengers,
  methods = "stl",
  params = list(s.window = 21, robust = TRUE) # Dot notation
)

stl_custom2 <- extract_trends(
  AirPassengers,
  methods = "stl",
```

```

  params = list(stl_s_window = 21, stl_robust = TRUE) # Underscore notation
)

# Advanced: fine-tune specific methods
custom_trends <- extract_trends(
  AirPassengers,
  methods = c("median", "kalman"),
  window = 7,
  params = list(median_endrule = "constant")
)

```

gdp_construction	<i>Quarterly GDP — Building Industry</i>
------------------	--

Description

Quarterly index of observed gross domestic product (GDP) for Brazil's building industry. The BCB SGS identifies the series as "Building industry SCN-2010" and reports its unit as an index.

Usage

```
gdp_construction
```

Format

A tibble with quarterly observations:

date Date of the first day of the quarter

index Observed building-industry GDP index

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date.

Source

Instituto Brasileiro de Geografia e Estatística (IBGE), disseminated by the Banco Central do Brasil through the Sistema Gerenciador de Séries Temporais (SGS), series 22087. See the [official SGS metadata](#) and [series page](#).

ibcbr	<i>Central Bank Economic Activity Index (IBC-Br)</i>
-------	--

Description

Monthly Central Bank Economic Activity Index (IBC-Br), a timely indicator of national economic activity. It is built from proxies for agriculture, industry, and services, aggregated using weights derived from the supply and use tables of the Brazilian National Accounts.

Usage

ibcbr

Format

A tibble with monthly observations:

date Date of the first day of the month

index IBC-Br index value

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date.

Source

Banco Central do Brasil — Departamento Econômico, disseminated through the Sistema Gerenciador de Séries Temporais (SGS), series 24363. See the [official SGS metadata](#) and [series page](#).

index_series	<i>Index one or more series</i>
--------------	---------------------------------

Description

Rescale numeric series relative to their earliest observed value or to the arithmetic mean over a selected base period.

Usage

```
index_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  base_period = NULL,
  base_value = 100,
  na_rm = FALSE,
  suffix = NULL,
  .quiet = FALSE
)
```

Arguments

<code>data</code>	A non-empty <code>data.frame</code> , <code>tibble</code> , or <code>data.table</code> .
<code>date_col</code>	Name of the Date column. Defaults to "date".
<code>value_col</code>	Non-empty character vector naming numeric value columns.
<code>group_cols</code>	Optional character vector naming grouping columns. Each group receives its own reference value.
<code>base_period</code>	NULL to use the earliest non-missing observation; one or two four-digit integer years; one or two Date values; or the name of a Date column holding one base date per group. Two values define an inclusive range and may be supplied in either order. A single date selects the calendar period containing it at the detected frequency.
<code>base_value</code>	Finite positive number assigned to the reference. Defaults to 100.
<code>na_rm</code>	Whether to remove missing values when averaging an explicit base period. With the default FALSE, a missing base observation raises an error instead of blanking the entire indexed series. This argument has no effect when <code>base_period = NULL</code> .
<code>suffix</code>	Optional non-missing character suffix for generated names.
<code>.quiet</code>	If TRUE, suppress frequency-detection messages. Warnings about incomplete base periods are never suppressed.

Details

When `base_period` is supplied, dates are matched at the detected calendar frequency of each group. Thus, for monthly data, `as.Date("2019-01-01")` also matches an observation dated at month end. Weekly and daily series use exact interval containment. A partly observed base interval produces a warning. With `base_period = NULL`, a warning identifies any series whose first dated value is missing and whose reference moves to a later date.

Value

A tibble containing the original columns, in their original order, followed by `index_{value_col}` columns (and `_{suffix}` when supplied).

See Also

[augment_trends\(\)](#) for trend estimation and [augment_rolling\(\)](#) for rolling and year-to-date aggregations.

Examples

```
vehicles |>
  index_series(value_col = "production")

retail_volume |>
  index_series(group_cols = "name_series", base_period = 2019)
```

metadata_series	<i>Series Metadata</i>
-----------------	------------------------

Description

Metadata for all economic series included in the package.

Usage

```
metadata_series
```

Format

A tibble with metadata:

series_name Short series identifier
description Full series description
frequency Data frequency (D = daily, M = monthly, Q = quarterly)
source Data source
date_col Name of the date column in the dataset
value_col Name of the main value column(s) in the dataset
group_cols Grouping column(s) for long-format datasets, or NA
date_min First observation date
date_max Last observation date

Source

Various. BCB-SGS metadata and source institutions are documented on the individual dataset pages; see the [BCB SGS series search](#).

oil_derivatives	<i>Petroleum Derivatives Production</i>
-----------------	---

Description

Monthly total production of petroleum derivatives in Brazil, measured in thousand barrels per day.

Usage

oil_derivatives

Format

A tibble with monthly observations:

date Date of the first day of the month

production Petroleum derivatives production in thousand barrels per day

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date.

Source

Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP), disseminated by the Banco Central do Brasil through the Sistema Gerenciador de Séries Temporais (SGS), series 1391. See the [official SGS metadata](#) and [series page](#).

retail_autofuel	<i>Great Britain Retail Sales Volume Index: Automotive Fuel</i>
-----------------	---

Description

Non-seasonally adjusted chained volume measure of retail sales for automotive fuel in Great Britain. This is the ONS "Automotive Fuel, All Businesses" series from Table 3M of the Retail Sales Index reference tables.

Usage

retail_autofuel

Format

A tibble with monthly observations:

date First day of the reference month (Date).

value Non-seasonally adjusted chained volume index of sales per week (2023 = 100).

name Series name, always Retail Sales - Automotive Fuel.

frequency Frequency code, always "M" (monthly).

source Short source label, always "ONS".

Details

Values are index numbers of sales per week, with 2023 = 100. ONS uses a four-week, four-week, five-week reporting cycle; date records the first day of the corresponding reference month and should not be interpreted as a calendar-month total. The Retail Sales Index covers Great Britain, not the whole UK: Northern Ireland, the Isle of Man, and the Channel Islands are excluded.

The bundled snapshot covers January 1996 through January 2026.

Source

Office for National Statistics (ONS), [Retail Sales Index](#) reference tables, Table 3M, "Chained volume of retail sales, non-seasonally adjusted". See the [Retail Sales Index \(RSI\) Quality and Methodology Information report](#). Contains public sector information licensed under the [Open Government Licence v3.0](#), except where otherwise stated. This package distributes a processed subset of the ONS table; see `data-raw/uk_data.R` for the transformation code.

See Also

[retail_volume](#)

retail_volume

Great Britain Retail Sales Volume Indices

Description

Selected non-seasonally adjusted chained volume measures of retail sales in Great Britain. The data are sourced from Table 3M of the Office for National Statistics (ONS) Retail Sales Index reference tables and retain the "All Businesses" series.

Usage

retail_volume

Format

- A tibble with monthly observations:
- date** First day of the reference month (Date).
- name_series** ONS series name in lower-case, hyphenated form.
- value** Non-seasonally adjusted chained volume index of sales per week (2023 = 100).

Details

Values are index numbers of sales per week, with 2023 = 100. ONS uses a four-week, four-week, five-week reporting cycle; date records the first day of the corresponding reference month and should not be interpreted as a calendar-month total. The Retail Sales Index covers Great Britain, not the whole UK: Northern Ireland, the Isle of Man, and the Channel Islands are excluded.

The bundled snapshot covers January 1988 through January 2026. The name_series column contains the following selected series: household-goods-stores, computers-and-telecomms-equipment, electrical-household-appliances, pharmaceutical-medical-cosmetic-and-toilet-goods, books-newspapers-and-periodicals, alcoholic-drinks-other-beverages-and-tobacco, clothing, all-retailing-including-automotive-fuel, and all-retailing-excluding-automotive-fuel. Missing values in value represent empty cells in the ONS source table.

Source

Office for National Statistics (ONS), [Retail Sales Index](#) reference tables, Table 3M, "Chained volume of retail sales, non-seasonally adjusted". See the [Retail Sales Index \(RSI\) Quality and Methodology Information report](#). Contains public sector information licensed under the [Open Government Licence v3.0](#), except where otherwise stated. This package distributes a processed subset of the ONS table; see data-raw/uk_data.R for the transformation code.

See Also

[retail_autofuel](#)

roll_series	<i>Rolling aggregations for time series objects</i>
-------------	---

Description

Compute rolling and year-to-date aggregations of a time series. Unlike [extract_trends\(\)](#), which estimates a trend in the units of the series, these are *aggregations*: a 12-month rolling sum is a 12-month total, not a level estimate. The two families are kept separate for that reason, so rolling results are not accepted by [detrend_series\(\)](#).

Usage

```
roll_series(
  ts_data,
  stats = "sum",
  window = NULL,
  align = "right",
  percent = FALSE,
  na_rm = FALSE,
  .quiet = FALSE
)
```

Arguments

ts_data	A time series object (ts, xts, or zoo) or any object convertible via tsbox.
stats	Character vector of rolling statistics. Options: "sum" (rolling total of flows), "chain" (compound accumulation of rates, $\text{prod}(1 + r) - 1$), "change" (change of a level over window periods, $x[t] / x[t - \text{window}] - 1$), "mean", "sd", "min", "max". Default is "sum".
window	Window length in periods, or the lag for "change". If NULL, defaults to the series frequency (12 for monthly, 4 for quarterly). A numeric vector runs the statistic once per window value. Alternatively, the string "ytd" computes an expanding year-to-date accumulation that resets each January (or Q1), and "all" an expanding accumulation from the first observation. Numeric and character windows cannot be mixed in one call, and "change" needs a numeric window.
align	Alignment of the window relative to the output position: "right" (default, causal — uses the current and preceding observations), "center", or "left". Right alignment is the convention for accumulated economic indicators. Ignored by "change" and by the expanding windows "ytd" and "all". An even window has no exact centre; see Details for how each statistic handles that.
percent	Only used by stats = "chain" and stats = "change". For "chain", if FALSE (default), rates are assumed to be decimals (0.005 for 0.5%). If TRUE, rates are assumed to be percentages (0.5 for 0.5%) and the result is returned in percent. For "change", TRUE returns the change in percent instead of as a decimal.
na_rm	If TRUE, missing values are ignored within each window. The default FALSE propagates NA, so an incomplete window yields NA. A window holding no observed values yields NA either way, as does a window holding one value for "sd". For even centered means, observed weights are renormalized under na_rm = TRUE; boundary padding is kept. "change" ignores it: a missing value at either end yields NA.
.quiet	If TRUE, suppress informational messages.

Details

stats = "sum" and stats = "chain" answer the same question for different kinds of series. For a flow measured in levels (units sold, jobs created), the 12-month accumulation is the sum. For

a series that is already a rate of change (monthly inflation, monthly returns), summing is only an approximation; the correct accumulation compounds the rates:

$$(1 + r_1)(1 + r_2) \cdots (1 + r_k) - 1$$

stats = "change" goes the other way, from a level (an index, a price, real income) to its rate of change over window periods. Chaining the one-period changes over k periods gives back the k-period change. The lag counts periods on the calendar grid for monthly, quarterly and annual series, and observations for daily and weekly series.

Note that a rolling sum is proportional to the simple moving average available through `extract_trends()`: `roll_series(x, "sum", window = k)` equals `k` times `extract_trends(x, "ma", window = k, align = "right")`. The rolling version is the one to reach for when the accumulated quantity is itself the number of interest. The two part company for an even window under `align = "center"`, where the moving average is weighted and the sum is not.

An even window centred on an observation has one more period on one side than the other. "mean" resolves this the way the ma trend method does, with the 2xN filter that puts half weight on the two endpoints, so `roll_series(x, "mean", window = k, align = "center")` matches `extract_trends(x, "ma", window = k, align = "center")`. The other statistics have no such correction and use a window with one extra period after the anchor.

Value

If a single statistic and a single window are requested, a ts object. Otherwise a named list of ts objects with names of the form {stat}_{window} (e.g. sum_12, chain_ytd, change_12).

See Also

`augment_rolling()` for the data frame interface, `extract_trends()` for trend estimation.

Examples

```
# 12-month rolling sum of vehicle production
prod_ts <- df_to_ts(vehicles, value_col = "production", frequency = 12)
roll_series(prod_ts, "sum", window = 12)

# Accumulated growth over 12 months, from monthly rates in percent
ibc_ts <- df_to_ts(ibcbr, value_col = "index", frequency = 12)
rates <- roll_series(ibc_ts, "change", window = 1, percent = TRUE)
roll_series(rates, "chain", window = 12, percent = TRUE)

# Year-to-date accumulation, resetting each January
roll_series(rates, "chain", window = "ytd", percent = TRUE)

# Cumulative growth since the start of the series
roll_series(rates, "chain", window = "all", percent = TRUE)

# 12-month change of the index, in percent
roll_series(ibc_ts, "change", window = 12, percent = TRUE)

# Several statistics and windows at once
```

```
roll_series(prod_ts, stats = c("sum", "sd"), window = c(3, 12))
```

transit_london_avgs	<i>TfL Network Demand: Average Daily Journeys</i>
---------------------	---

Description

Monthly averages of TfL's reported daily journey counts for London's bus and Tube networks, split by mode and UK business-day status. The average is calculated over the daily observations in each month and group; it is not a monthly total.

Usage

```
transit_london_avgs
```

Format

A tibble with monthly observations:

date_month First day of the calendar month (Date).

transit_mode Transit mode: "bus" or "tube".

is_business_day 1 for a UK business day and 0 for a weekend or UK holiday.

avg_daily_journeys Arithmetic mean of the reported daily journey counts for the month, mode, and business-day group.

Details

The business-day flag uses the QuantLib/UnitedKingdom calendar from RQuantLib: 1 identifies a business day and 0 identifies a Saturday, Sunday, or UK holiday. The underlying counts are derived from TfL's ticketing system and should not be interpreted as an absolute measure of passenger numbers or journeys made. They are approximate, rounded to the nearest thousand, and exclude passengers who did not tap in or out.

The version bundled with trendseries covers daily source records from 2019-01-01 through 2025-12-27. December 2025 is therefore a partial month. TfL may revise historical rows when its source files are refreshed.

Source

Transport for London (TfL), [Network demand data](#), and the [Network Demand Dashboard](#). The package distributes an aggregated and reshaped version of TfL's daily Journeys files and classifies dates with the UK business calendar. TfL's [Transport Data Service terms](#) require the attribution: **Powered by TfL Open Data**. This package is not affiliated with or endorsed by TfL.

See Also

[transit_london_monthly](#)

`transit_london_monthly`*TfL Network Demand: Monthly Bus and Tube Journeys*

Description

Monthly totals of TfL's reported journey counts for London's bus and Tube networks. The source files contain daily counts for the two modes; this dataset sums those daily values by calendar month.

Usage

`transit_london_monthly`

Format

A tibble with monthly observations:

date_month First day of the calendar month (Date).

transit_mode Transit mode: "bus" or "tube".

journey_monthly Sum of the reported daily journey counts for the month.

Details

These are recorded journey counts derived from TfL's ticketing system, not an absolute measure of passenger numbers or journeys made. TfL notes that the figures cover activity recorded through Oyster cards, contactless payment cards, and paper tickets, and do not include passengers who did not tap in or out. The source figures are approximate and rounded to the nearest thousand. This dataset is not the separate station footfall dataset.

TfL publishes separate files for 2019–2022 and a consolidated file for subsequent observations. The version bundled with `trendseries` covers daily source records from 2019-01-01 through 2025-12-27. Because the final source file ends on 27 December 2025, the December 2025 monthly totals are partial. TfL may revise historical rows when its source files are refreshed.

Source

Transport for London (TfL), [Network demand data](#), and the [Network Demand Dashboard](#). The package distributes an aggregated and reshaped version of TfL's daily Journeys files. TfL's [Transport Data Service terms](#) require the attribution: **Powered by TfL Open Data**. This package is not affiliated with or endorsed by TfL.

See Also

[transit_london_avgs](#)

ts_to_df	<i>Convert time series to tibble</i>
----------	--------------------------------------

Description

Converts a ts object into a tibble with two columns: date and value. Optionally, the columns can be renamed using date_col and value_col.

Usage

```
ts_to_df(x, date_col = NULL, value_col = NULL)
```

Arguments

x	A time series as a ts object
date_col	Name for the date column. Defaults to "date".
value_col	Name for the value column. Defaults to "value".

Value

a tibble

Examples

```
# example code
ts_to_df(AirPassengers)

# Using a custom name for the value column
ts_to_df(AirPassengers, value_col = "passengers")
```

vehicles	<i>Total Vehicle Sales</i>
----------	----------------------------

Description

Monthly total vehicle sales in Brazil, measured in units. The BCB SGS title for series 1378 is "Vehicle sales (total)".

Usage

```
vehicles
```

Format

A tibble with monthly observations:

date Date of the first day of the month

production Total vehicle sales in units. The column name is retained for compatibility with existing package examples.

Details

The package snapshot was downloaded through 2025-12-31. BCB may revise historical observations and extend the series after that date.

Source

Associação Nacional dos Fabricantes de Veículos Automotores (Anfavea), disseminated by the Banco Central do Brasil through the Sistema Gerenciador de Séries Temporais (SGS), series 1378. See the [official SGS metadata](#) and [series page](#).

Index

* datasets

- coffee_arabica, [8](#)
- coffee_robusta, [9](#)
- electric, [22](#)
- electricity, [22](#)
- gdp_construction, [28](#)
- ibcbr, [29](#)
- metadata_series, [31](#)
- oil_derivatives, [32](#)
- retail_autofuel, [32](#)
- retail_volume, [33](#)
- transit_london_avgs, [37](#)
- transit_london_monthly, [38](#)
- vehicles, [39](#)

augment_rolling, [2](#)

augment_rolling(), [31](#), [36](#)

augment_trends, [5](#)

augment_trends(), [2](#), [4](#), [12](#), [17–20](#), [31](#)

coffee_arabica, [8](#), [10](#)

coffee_robusta, [9](#), [9](#)

converters, [10](#)

decompose_series, [11](#)

decompose_series(), [15–17](#), [20](#)

deseason_series, [15](#)

deseason_series(), [20](#)

detrend_series, [18](#)

detrend_series(), [34](#)

df_to_ts, [21](#)

electric, [22](#), [23](#)

electricity, [22](#)

extract_trends, [23](#)

extract_trends(), [34](#), [36](#)

gdp_construction, [28](#)

ibcbr, [29](#)

index_series, [29](#)

metadata_series, [31](#)

oil_derivatives, [32](#)

retail_autofuel, [32](#), [34](#)

retail_volume, [33](#), [33](#)

roll_series, [34](#)

roll_series(), [3](#), [4](#)

transit_london_avgs, [37](#), [38](#)

transit_london_monthly, [37](#), [38](#)

ts_to_df, [39](#)

vehicles, [39](#)