Package ‘healthyR.ts’

November 15, 2023

Title The Time Series Modeling Companion to 'healthyR'

Version 0.3.0

Description Hospital time series data analysis workflow tools, modeling, and automations. This library provides many useful tools to review common administrative time series hospital data. Some of these include average length of stay, and readmission rates. The aim is to provide a simple and consistent verb framework that takes the guesswork out of everything.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

URL https://github.com/spsanderson/healthyR.ts

BugReports https://github.com/spsanderson/healthyR.ts/issues

Imports magrittr, rlang (>= 0.1.2), tibble, timetk, tidyr, dplyr, purrr, ggplot2, lubridate, plotly, recipes, modeltime, cowplot, graphics, forcats, stringi, parsnip, workflowsets, hardhat

Suggests knitr, rmarkdown, roxygen2, scales, rsample, healthyR.ai, stringr, forecast, tidymodels, glue, xts, zoo, TSA, tune, dials, workflows, tidyselect

VignetteBuilder knitr

Depends R (>= 3.3)

NeedsCompilation no

Author Steven Sanderson [aut, cre, cph] (<https://orcid.org/0009-0006-7661-8247>)

Maintainer Steven Sanderson <spsanderson@gmail.com>

Repository CRAN

Date/Publication 2023-11-15 06:00:05 UTC

R topics documented:

`auto_stationarize` *Automatically Stationarize Time Series Data*

Description

This function attempts to make a non-stationary time series stationary. This function attempts to make a given time series stationary by applying transformations such as differencing or logarithmic transformation. If the time series is already stationary, it returns the original time series.

Usage

```
auto_stationarize(.time_series)
```

Arguments

`.time_series`	A time series object to be made stationary.

Details

If the input time series is non-stationary (determined by the Augmented Dickey-Fuller test), this function will try to make it stationary by applying a series of transformations:

1. It checks if the time series is already stationary using the Augmented Dickey-Fuller test.
2. If not stationary, it attempts a logarithmic transformation.
3. If the logarithmic transformation doesn't work, it applies differencing.

Value

If the time series is already stationary, it returns the original time series. If a transformation is applied to make it stationary, it returns a list with two elements:

- stationary_ts: The stationary time series.
- ndiffs: The order of differencing applied to make it stationary.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
# Example 1: Using the AirPassengers dataset
auto_stationarize(AirPassengers)

# Example 2: Using the BJsales dataset
auto_stationarize(BJsales)
```

`calibrate_and_plot` *Helper function - Calibrate and Plot*

Description

This function is a helper function. It will take in a set of workflows and then perform the `modeltime::modeltime_calibrate` and `modeltime::plot_modeltime_forecast()`.

Usage

```
calibrate_and_plot(
  ...,
  .type = "testing",
  .splits_obj,
  .data,
  .print_info = TRUE,
  .interactive = FALSE
)
```

Arguments

`...`	The workflow(s) you want to add to the function.
`.type`	Either the training(splits) or testing(splits) data.
`.splits_obj`	The splits object.
`.data`	The full data set.
`.print_info`	The default is TRUE and will print out the calibration accuracy tibble and the resulting plotly plot.
`.interactive`	The defaults is FALSE. This controls if a forecast plot is interactive or not via plotly.

Details

This function expects to take in workflows fitted with training data.

Value

The original time series, the simulated values and a some plots

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
## Not run:
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(parsnip))
suppressPackageStartupMessages(library(workflows))

data <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- timetk::time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_obj <- recipe(value ~ ., data = training(splits))

model_spec <- linear_reg(
   mode = "regression"
   , penalty = 0.1
   , mixture = 0.5
) %>%
   set_engine("lm")

wflw <- workflow() %>%
   add_recipe(rec_obj) %>%
   add_model(model_spec) %>%
   fit(training(splits))

output <- calibrate_and_plot(
  wflw
  , .type = "training"
  , .splits_obj = splits
```

```
  , .data = data
  , .print_info = FALSE
  , .interactive = FALSE
 )

## End(Not run)
```

---

ci_hi *Confidence Interval Generic*

Description

Gets the upper 97.5% quantile of a numeric vector.

Usage

```
ci_hi(.x, .na_rm = FALSE)
```

Arguments

.x	A vector of numeric values
.na_rm	A Boolean, defaults to FALSE. Passed to the quantile function.

Details

Gets the upper 97.5% quantile of a numeric vector.

Value

A numeric value.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Statistic: ci_lo(), ts_adf_test()

Examples

```
x <- mtcars$mpg
ci_hi(x)
```

ci_lo *Confidence Interval Generic*

Description

Gets the lower 2.5% quantile of a numeric vector.

Usage

```
ci_lo(.x, .na_rm = FALSE)
```

Arguments

`.x`	A vector of numeric values
`.na_rm`	A Boolean, defaults to FALSE. Passed to the quantile function.

Details

Gets the lower 2.5% quantile of a numeric vector.

Value

A numeric value.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Statistic: `ci_hi()`, `ts_adf_test()`

Examples

```
x <- mtcars$mpg
ci_lo(x)
```

color_blind — *Provide Colorblind Compliant Colors*

Description

8 Hex RGB color definitions suitable for charts for colorblind people.

Usage

```
color_blind()
```

Details

This function is used in others in order to help render plots for those that are color blind.

Value

A vector of 8 Hex RGB definitions.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
color_blind()
```

internal_ts_backward_event_tbl — *Event Analysis*

Description

This is a function that sits inside of the `ts_time_event_analysis_tbl()`. It is only meant to be used there. This is an internal function.

Usage

```
internal_ts_backward_event_tbl(.data, .horizon)
```

Arguments

`.data`	The date.frame/tibble that holds the data.
`.horizon`	How far do you want to look back or ahead.

Details

This is a helper function for `ts_time_event_analysis_tbl()` only.

Value

A tibble.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

---

internal_ts_both_event_tbl

Event Analysis

Description

This is a function that sits inside of the `ts_time_event_analysis_tbl()`. It is only meant to be used there. This is an internal function.

Usage

```
internal_ts_both_event_tbl(.data, .horizon)
```

Arguments

`.data`	The date.frame/tibble that holds the data.
`.horizon`	How far do you want to look back or ahead.

Details

This is a helper function for `ts_time_event_analysis_tbl()` only.

Value

A tibble.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

---

internal_ts_forward_event_tbl

Event Analysis

Description

This is a function that sits inside of the `ts_time_event_analysis_tbl()`. It is only meant to be used there. This is an internal function.

Usage

```
internal_ts_forward_event_tbl(.data, .horizon)
```

Arguments

`.data`	The date.frame/tibble that holds the data.
`.horizon`	How far do you want to look back or ahead.

Details

This is a helper function for `ts_time_event_analysis_tbl()` only.

Value

A tibble.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

`model_extraction_helper` *Model Method Extraction Helper*

Description

This takes in a model fit and returns the method of the fit object.

Usage

```
model_extraction_helper(.fit_object)
```

Arguments

`.fit_object`	A time-series fitted model

Details

Currently supports forecasting model of one of the following from the `forecast` package:

- Arima
- auto.arima
- ets
- nnetar
- `workflow` fitted models.

Value

A model description

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
# NOT RUN
## Not run:
suppressPackageStartupMessages(library(forecast))

# Create a model
fit_arima  <- auto.arima(AirPassengers)

model_extraction_helper(fit_arima)

## End(Not run)
```

---

step_ts_acceleration *Recipes Time Series Acceleration Generator*

Description

step_ts_acceleration creates a a *specification* of a recipe step that will convert numeric data into from a time series into its acceleration.

Usage

```
step_ts_acceleration(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("ts_acceleration")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric

role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variables that will be used as inputs. This field is a placeholder and will be populated once `recipes::prep()` is used.
skip	A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

Numeric Variables Unlike other steps, `step_ts_acceleration` does *not* remove the original numeric variables. `recipes::step_rm()` can be used for this purpose.

Value

For `step_ts_acceleration`, an updated version of recipe with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

- `recipes::recipe()`
- `recipes::prep()`
- `recipes::bake()`

See Also

Other Recipes: `step_ts_velocity()`

Examples

```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out    = 10
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
```

```
  step_ts_acceleration(b)

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% prep() %>% juice()
```

---

step_ts_velocity *Recipes Time Series velocity Generator*

Description

step_ts_velocity creates a a *specification* of a recipe step that will convert numeric data into from a time series into its velocity.

Usage

```
step_ts_velocity(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("ts_velocity")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
columns	A character string of variables that will be used as inputs. This field is a placeholder and will be populated once recipes::prep() is used.

skip — A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id — A character string that is unique to this step to identify it.

Details

Numeric Variables Unlike other steps, `step_ts_velocity` does *not* remove the original numeric variables. `recipes::step_rm()` can be used for this purpose.

Value

For `step_ts_velocity`, an updated version of recipe with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

- `recipes::recipe()`
- `recipes::prep()`
- `recipes::bake()`

See Also

Other Recipes: `step_ts_acceleration()`

Examples

```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out    = 10
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
  step_ts_velocity(b)

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)
```

```
# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% prep() %>% juice()
```

---

tidy_fft *Tidy Style FFT*

Description

Perform an fft using `stats::fft()` and return a tidier style output list with plots.

Usage

```
tidy_fft(
  .data,
  .date_col,
  .value_col,
  .frequency = 12L,
  .harmonics = 1L,
  .upsampling = 10L
)
```

Arguments

`.data`	The data.frame/tibble you will pass for analysis.
`.date_col`	The column that holds the date.
`.value_col`	The column that holds the data to be analyzed.
`.frequency`	The frequency of the data, 12 = monthly for example.
`.harmonics`	How many harmonic waves do you want to produce.
`.upsampling`	The up sampling of the time series.

Details

This function will perform a few different things, but primarily it will compute the Fast Discrete Fourier Transform (FFT) using `stats::fft()`. The formula is given as:

$$y[h] = sum_{k=1}^{n} z[k] * exp(-2 * pi * 1i * (k - 1) * (h - 1)/n)$$

There are many items returned inside of a list invisibly. There are four primary categories of data returned in the list. Below are the primary categories and the items inside of them.

data:

1. data
2. error_data

3. input_vector
4. maximum_harmonic_tbl
5. differenced_value_tbl
6. dff_tbl
7. ts_obj

plots:

1. harmonic_plot
2. diff_plot
3. max_har_plot
4. harmonic_plotly
5. max_har_plotly

parameters:

1. harmonics
2. upsampling
3. start_date
4. end_date
5. freq

model:

1. m
2. harmonic_obj
3. harmonic_model
4. model_summary

Value

A list object returned invisibly.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: `ts_brownian_motion_augment()`, `ts_brownian_motion()`, `ts_geometric_brownian_motion_au` `ts_geometric_brownian_motion()`, `ts_random_walk()`

Examples

```
suppressPackageStartupMessages(library(dplyr))

data_tbl <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

a <- tidy_fft(
  .data = data_tbl,
  .value_col = value,
  .date_col = date_col,
  .harmonics = 3,
  .frequency = 12
)

a$plots$max_har_plot
a$plots$harmonic_plot
```

---

ts_acceleration_augment

Augment Function Acceleration

Description

Takes a numeric vector and will return the acceleration of that vector.

Usage

```
ts_acceleration_augment(.data, .value, .names = "auto")
```

Arguments

.data	The data being passed that will be augmented by the function.
.value	This is passed rlang::enquo() to capture the vectors you want to augment.
.names	The default is "auto"

Details

Takes a numeric vector and will return the acceleration of that vector. The acceleration of a time series is computed by taking the second difference, so

$$(x_t - x_t1) - (x_t - x_t1)_t1$$

This function is intended to be used on its own in order to add columns to a tibble.

Value

A augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: `ts_growth_rate_augment()`, `ts_velocity_augment()`

Examples

```
suppressPackageStartupMessages(library(dplyr))

len_out    = 10
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

ts_acceleration_augment(data_tbl, b)
```

`ts_acceleration_vec` *Vector Function Time Series Acceleration*

Description

Takes a numeric vector and will return the acceleration of that vector.

Usage

```
ts_acceleration_vec(.x)
```

Arguments

`.x`	A numeric vector

Details

Takes a numeric vector and will return the acceleration of that vector. The acceleration of a time series is computed by taking the second difference, so

$$(x_t - x_t1) - (x_t - x_t1)_t1$$

This function can be used on it's own. It is also the basis for the function `ts_acceleration_augment()`.

Value

A numeric vector

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Vector Function: ts_growth_rate_vec(), ts_velocity_vec()

Examples

```
suppressPackageStartupMessages(library(dplyr))

len_out    = 25
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

vec_1 <- ts_acceleration_vec(data_tbl$b)

plot(data_tbl$b)
lines(data_tbl$b)
lines(vec_1, col = "blue")
```

---

ts_adf_test *Augmented Dickey-Fuller Test for Time Series Stationarity*

Description

This function performs the Augmented Dickey-Fuller test to assess the stationarity of a time series. The Augmented Dickey-Fuller (ADF) test is used to determine if a given time series is stationary. This function takes a numeric vector as input, and you can optionally specify the lag order with the .k parameter. If .k is not provided, it is calculated based on the number of observations using a formula. The test statistic and p-value are returned.

Usage

```
ts_adf_test(.x, .k = NULL)
```

Arguments

.x	A numeric vector representing the time series to be tested for stationarity.
.k	An optional parameter specifying the number of lags to use in the ADF test (default is calculated).

Value

A list containing the results of the Augmented Dickey-Fuller test:

- test_stat: The test statistic from the ADF test.
- p_value: The p-value of the test.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Statistic: ci_hi(), ci_lo()

Examples

```
# Example 1: Using the AirPassengers dataset
ts_adf_test(AirPassengers)

# Example 2: Using a custom time series vector
custom_ts <- rnorm(100, 0, 1)
ts_adf_test(custom_ts)
```

---

ts_arima_simulator *Simulate ARIMA Model*

Description

Returns a list output of any n simulations of a user specified ARIMA model. The function returns a list object with two sections:

- data
- plots

The data section of the output contains the following:

- simulation_time_series object (ts format)
- simulation_time_series_output (mts format)
- simulations_tbl (simulation_time_series_object in a tibble)
- simulations_median_value_tbl (contains the stats::median() value of the simulated data)

The plots section of the output contains the following:

- static_plot The ggplot2 plot
- plotly_plot The plotly plot

Usage

```
ts_arima_simulator(
  .n = 100,
  .num_sims = 25,
  .order_p = 0,
  .order_d = 0,
  .order_q = 0,
  .ma = c(),
  .ar = c(),
  .sim_color = "steelblue",
  .alpha = 0.05,
  .size = 1,
  ...
)
```

Arguments

.n	The number of points to be simulated.
.num_sims	The number of different simulations to be run.
.order_p	The p value, the order of the AR term.
.order_d	The d value, the number of differencing to make the series stationary
.order_q	The q value, the order of the MA term.
.ma	You can list the MA terms respectively if desired.
.ar	You can list the AR terms respectively if desired.
.sim_color	The color of the lines for the simulated series.
.alpha	The alpha component of the ggplot2 and plotly lines.
.size	The size of the median line for the ggplot2
...	Any other additional arguments for stats::arima.sim

Details

This function takes in a user specified arima model. The specification is passed to `stats::arima.sim()`

Value

A list object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

Other Simulator: ts_forecast_simulator()

Examples

```
output <- ts_arima_simulator()
output$plots$static_plot
```

---

ts_auto_arima *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_arima(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_arima",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

.data	The data being passed to the function. The time-series object.
.date_col	The column that holds the datetime.
.value_col	The column that has the value
.formula	The formula that is passed to the recipe like `value ~ .`
.rsamp_obj	The rsample splits object
.prefix	Default is `ts_arima`
.tune	Defaults to TRUE, this creates a tuning grid and tuned model.
.grid_size	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
.num_cores	How many cores do you want to use. Default is 1
.cv_assess	How many observations for assess. See `timetk::time_series_cv()`
.cv_skip	How many observations to skip. See `timetk::time_series_cv()`
.cv_slice_limit	How many slices to return. See `timetk::time_series_cv()`
.best_metric	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
.bootstrap_final	Not yet implemented.

Details

This uses the `modeltime::arima_reg()` with the engine set to `arima`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/arima_reg.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_lm()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)
```

```
data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_aa <- ts_auto_arima(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .cv_slice_limit = 2,
  .tune = FALSE
)

ts_aa$recipe_info
```

`ts_auto_arima_xgboost` *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_arima_xgboost(
  .data,
  .date_col,
  .value_col,
  .formula,
```

```
  .rsamp_obj,
  .prefix = "ts_arima_boost",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_arima_boost`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `modeltime::arima_boost()` with the `engine` set to `xgboost`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/arima_boost.html

Other Boiler_Plate: ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_theta(), ts_auto_xgboost()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_auto_arima_xgboost <- ts_auto_arima_xgboost(
  .data = data,
  .num_cores = 1,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .cv_slice_limit = 2,
  .tune = FALSE
)

ts_auto_arima_xgboost$recipe_info
```

---

ts_auto_croston *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe

- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_croston(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_croston",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_exp_smooth`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `forecast::croston()` for the `parsnip` engine. This model does not use exogenous regressors, so only a univariate model of: value ~ date will be used from the `.date_col` and `.value_col` that you provide.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/exp_smoothing.html#engine-details

https://pkg.robjhyndman.com/forecast/reference/croston.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_lm()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Other exp_smoothing: `ts_auto_exp_smoothing()`, `ts_auto_smooth_es()`, `ts_auto_theta()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_exp <- ts_auto_croston(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
```

```
  .tune = FALSE
)

ts_exp$recipe_info
```

`ts_auto_exp_smoothing` *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_exp_smoothing(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_exp_smooth",
  .tune = TRUE,
  .grid_size = 20,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`

.rsamp_obj	The rsample splits object
.prefix	Default is ts_exp_smooth
.tune	Defaults to TRUE, this creates a tuning grid and tuned model.
.grid_size	If .tune is TRUE then the .grid_size is the size of the tuning grid.
.num_cores	How many cores do you want to use. Default is 1
.cv_assess	How many observations for assess. See timetk::time_series_cv()
.cv_skip	How many observations to skip. See timetk::time_series_cv()
.cv_slice_limit	How many slices to return. See timetk::time_series_cv()
.best_metric	Default is "rmse". See modeltime::default_forecast_accuracy_metric_set()
.bootstrap_final	Not yet implemented.

Details

This uses modeltime::exp_smoothing() under the hood with the engine set to ets

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/exp_smoothing.html#engine-details

https://pkg.robjhyndman.com/forecast/reference/ets.html

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_theta(), ts_auto_xgboost()

Other exp_smoothing: ts_auto_croston(), ts_auto_smooth_es(), ts_auto_theta()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
```

```
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_exp <- ts_auto_exp_smoothing(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 20,
  .tune = FALSE
)

ts_exp$recipe_info
```

ts_auto_glmnet *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_glmnet(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_glmnet",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
```

```
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_glmnet`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses `parsnip::linear_reg()` and sets the engine to `glmnet`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/linear_reg.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_lm()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_glmnet <- ts_auto_glmnet(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .tune = FALSE
)

ts_glmnet$recipe_info
```

ts_auto_lm *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- calibration tibble and plot

Usage

```
ts_auto_lm(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_lm",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_lm`
`.bootstrap_final`	Not yet implemented.

Details

This uses `parsnip::linear_reg()` and sets the engine to `lm`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/linear_reg.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)
```

```
data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_lm <- ts_auto_lm(
  .data = data,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
)

ts_lm$recipe_info
```

`ts_auto_mars` *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_mars(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_mars",
  .tune = TRUE,
  .grid_size = 10,
```

```
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_mars`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `parsnip::mars()` function with the engine set to `earth`.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/mars.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_lm()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_auto_mars <- ts_auto_mars(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 20,
  .tune = FALSE
)

ts_auto_mars$recipe_info
```

`ts_auto_nnetar` *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_nnetar(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_nnetar",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_nnetar`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `modeltime::nnetar_reg()` function with the `engine` set to `nnetar`.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/nnetar_reg.html

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_theta(), ts_auto_xgboost()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_nnetar <- ts_auto_nnetar(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .tune = FALSE
)

ts_nnetar$recipe_info
```

---

ts_auto_prophet_boost *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_prophet_boost(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_prophet_boost",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_prophet_boost`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `modeltime::prophet_boost()` function with the `engine` set to `prophet_xgboost`.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/prophet_boost.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_lm()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_reg()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Other prophet: `ts_auto_prophet_reg()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_prophet_boost <- ts_auto_prophet_boost(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .tune = FALSE
)

ts_prophet_boost$recipe_info
```

ts_auto_prophet_reg *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_prophet_reg(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_prophet_reg",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_prophet`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.

.num_cores
: How many cores do you want to use. Default is 1

.cv_assess
: How many observations for assess. See timetk::time_series_cv()

.cv_skip
: How many observations to skip. See timetk::time_series_cv()

.cv_slice_limit
: How many slices to return. See timetk::time_series_cv()

.best_metric
: Default is "rmse". See modeltime::default_forecast_accuracy_metric_set()

.bootstrap_final
: Not yet implemented.

Details

This uses the `modeltime::prophet_reg()` function with the `engine` set to `prophet`.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/prophet_reg.html

Other Boiler_Plate: `ts_auto_arima_xgboost()`, `ts_auto_arima()`, `ts_auto_croston()`, `ts_auto_exp_smoothing()`, `ts_auto_glmnet()`, `ts_auto_lm()`, `ts_auto_mars()`, `ts_auto_nnetar()`, `ts_auto_prophet_boost()`, `ts_auto_smooth_es()`, `ts_auto_svm_poly()`, `ts_auto_svm_rbf()`, `ts_auto_theta()`, `ts_auto_xgboost()`

Other prophet: `ts_auto_prophet_boost()`

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_prophet_reg <- ts_auto_prophet_reg(
```

```
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .tune = FALSE
)

ts_prophet_reg$recipe_info
```

ts_auto_recipe *Build a Time Series Recipe*

Description

Automatically builds generic time series recipe objects from a given tibble.

Usage

```
ts_auto_recipe(
  .data,
  .date_col,
  .pred_col,
  .step_ts_sig = TRUE,
  .step_ts_rm_misc = TRUE,
  .step_ts_dummy = TRUE,
  .step_ts_fourier = TRUE,
  .step_ts_fourier_period = 365/12,
  .K = 1,
  .step_ts_yeo = TRUE,
  .step_ts_nzv = TRUE
)
```

Arguments

`.data`	The data that is going to be modeled. You must supply a tibble.
`.date_col`	The column that holds the date for the time series.
`.pred_col`	The column that is to be predicted.
`.step_ts_sig`	A Boolean indicating should the `timetk::step_timeseries_signature()` be added, default is TRUE.
`.step_ts_rm_misc`	A Boolean indicating should the following items be removed from the time series signature, default is TRUE.

- iso$
- xts$
- hour
- min
- sec
- am.pm

`.step_ts_dummy` A Boolean indicating if all_nominal_predictors() should be dummied and with one hot encoding.

`.step_ts_fourier` A Boolean indicating if `timetk::step_fourier()` should be added to the recipe.

`.step_ts_fourier_period` A number such as 365/12, 365/4 or 365 indicting the period of the fourier term. The numeric period for the oscillation frequency.

`.K` The number of orders to include for each sine/cosine fourier series. More orders increase the number of fourier terms and therefore the variance of the fitted model at the expense of bias. See details for examples of K specification.

`.step_ts_yeo` A Boolean indicating if the `recipes::step_YeoJohnson()` should be added to the recipe.

`.step_ts_nzv` A Boolean indicating if the `recipes::step_nzv()` should be run on all predictors.

Details

This will build out a couple of generic recipe objects and return those items in a list.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- initial_time_split(
 data_tbl
 , prop = 0.8
)

ts_auto_recipe(
    .data = data_tbl
    , .date_col = date_col
    , .pred_col = value
)
```

```
ts_auto_recipe(
  .data = training(splits)
  , .date_col = date_col
  , .pred_col = value
)
```

ts_auto_smooth_es *Boilerplate Workflow*

Description

This is a boilerplate function to automatically create the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_smooth_es(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_smooth_es",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

.data	The data being passed to the function. The time-series object.
.date_col	The column that holds the datetime.
.value_col	The column that has the value
.formula	The formula that is passed to the recipe like `value ~ .`

.rsamp_obj	The rsample splits object
.prefix	Default is ts_smooth_es
.tune	Defaults to TRUE, this creates a tuning grid and tuned model.
.grid_size	If .tune is TRUE then the .grid_size is the size of the tuning grid.
.num_cores	How many cores do you want to use. Default is 1
.cv_assess	How many observations for assess. See timetk::time_series_cv()
.cv_skip	How many observations to skip. See timetk::time_series_cv()
.cv_slice_limit	How many slices to return. See timetk::time_series_cv()
.best_metric	Default is "rmse". See modeltime::default_forecast_accuracy_metric_set()
.bootstrap_final	Not yet implemented.

Details

This uses modeltime::exp_smoothing() and sets the parsnip::engine to smooth_es.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/exp_smoothing.html#ref-examples

https://github.com/config-i1/smooth

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_theta(), ts_auto_xgboost()

Other exp_smoothing: ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_theta()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
```

```
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_smooth_es <- ts_auto_smooth_es(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 3,
  .tune = FALSE
)

ts_smooth_es$recipe_info
```

`ts_auto_svm_poly` *Boilerplate Workflow*

Description

This is a boilerplate function to automatically create the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_svm_poly(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_svm_poly",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
```

```
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

.data	The data being passed to the function. The time-series object.
.date_col	The column that holds the datetime.
.value_col	The column that has the value
.formula	The formula that is passed to the recipe like value ~ .
.rsamp_obj	The rsample splits object
.prefix	Default is ts_smooth_es
.tune	Defaults to TRUE, this creates a tuning grid and tuned model.
.grid_size	If .tune is TRUE then the .grid_size is the size of the tuning grid.
.num_cores	How many cores do you want to use. Default is 1
.cv_assess	How many observations for assess. See timetk::time_series_cv()
.cv_skip	How many observations to skip. See timetk::time_series_cv()
.cv_slice_limit	How many slices to return. See timetk::time_series_cv()
.best_metric	Default is "rmse". See modeltime::default_forecast_accuracy_metric_set()
.bootstrap_final	Not yet implemented.

Details

This uses parsnip::svm_poly() and sets the parsnip::engine to kernlab.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/svm_poly.html

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_rbf(), ts_auto_theta(), ts_auto_xgboost()

Other SVM: ts_auto_svm_rbf()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_auto_poly <- ts_auto_svm_poly(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 3,
  .tune = FALSE
)

ts_auto_poly$recipe_info
```

`ts_auto_svm_rbf` *Boilerplate Workflow*

Description

This is a boilerplate function to automatically create the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_svm_rbf(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_svm_rbf",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

.data	The data being passed to the function. The time-series object.
.date_col	The column that holds the datetime.
.value_col	The column that has the value
.formula	The formula that is passed to the recipe like `value ~ .`
.rsamp_obj	The rsample splits object
.prefix	Default is `ts_smooth_es`
.tune	Defaults to TRUE, this creates a tuning grid and tuned model.
.grid_size	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
.num_cores	How many cores do you want to use. Default is 1
.cv_assess	How many observations for assess. See `timetk::time_series_cv()`
.cv_skip	How many observations to skip. See `timetk::time_series_cv()`
.cv_slice_limit	How many slices to return. See `timetk::time_series_cv()`
.best_metric	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
.bootstrap_final	Not yet implemented.

Details

This uses `parsnip::svm_rb()` and sets the `parsnip::engine` to `kernlab`.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/svm_rbf.html

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_theta(), ts_auto_xgboost()

Other SVM: ts_auto_svm_poly()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_auto_rbf <- ts_auto_svm_rbf(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 3,
  .tune = FALSE
)

ts_auto_rbf$recipe_info
```

ts_auto_theta *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- calibration tibble and plot

Usage

```
ts_auto_theta(
  .data,
  .date_col,
  .value_col,
  .rsamp_obj,
  .prefix = "ts_theta",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.rsamp_obj`	The splits object
`.prefix`	Default is `ts_theta`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `forecast::thetaf()` for the `parsnip` engine. This model does not use exogenous regressors, so only a univariate model of: value ~ date will be used from the `.date_col` and `.value_col` that you provide.

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://business-science.github.io/modeltime/reference/exp_smoothing.html#engine-details

https://pkg.robjhyndman.com/forecast/reference/thetaf.html

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_xgboost()

Other exp_smoothing: ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_smooth_es()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_theta <- ts_auto_theta(
  .data = data,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits
)

ts_theta$recipe_info
```

---

ts_auto_xgboost *Boilerplate Workflow*

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification

- workflow
- tuned model (grid ect)
- calibration tibble and plot

Usage

```
ts_auto_xgboost(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_xgboost",
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .cv_assess = 12,
  .cv_skip = 3,
  .cv_slice_limit = 6,
  .best_metric = "rmse",
  .bootstrap_final = FALSE
)
```

Arguments

`.data`	The data being passed to the function. The time-series object.
`.date_col`	The column that holds the datetime.
`.value_col`	The column that has the value
`.formula`	The formula that is passed to the recipe like `value ~ .`
`.rsamp_obj`	The rsample splits object
`.prefix`	Default is `ts_xgboost`
`.tune`	Defaults to TRUE, this creates a tuning grid and tuned model.
`.grid_size`	If `.tune` is TRUE then the `.grid_size` is the size of the tuning grid.
`.num_cores`	How many cores do you want to use. Default is 1
`.cv_assess`	How many observations for assess. See `timetk::time_series_cv()`
`.cv_skip`	How many observations to skip. See `timetk::time_series_cv()`
`.cv_slice_limit`	How many slices to return. See `timetk::time_series_cv()`
`.best_metric`	Default is "rmse". See `modeltime::default_forecast_accuracy_metric_set()`
`.bootstrap_final`	Not yet implemented.

Details

This uses the `parsnip::boost_tree()` with the engine set to `xgboost`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Boiler_Plate: ts_auto_arima_xgboost(), ts_auto_arima(), ts_auto_croston(), ts_auto_exp_smoothing(), ts_auto_glmnet(), ts_auto_lm(), ts_auto_mars(), ts_auto_nnetar(), ts_auto_prophet_boost(), ts_auto_prophet_reg(), ts_auto_smooth_es(), ts_auto_svm_poly(), ts_auto_svm_rbf(), ts_auto_theta()

Examples

```
library(dplyr)
library(timetk)
library(modeltime)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_xgboost <- ts_auto_xgboost(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .tune = FALSE
)

ts_xgboost$recipe_info
```

ts_brownian_motion *Brownian Motion*

Description

Create a Brownian Motion Tibble

Usage

```
ts_brownian_motion(
  .time = 100,
  .num_sims = 10,
  .delta_time = 1,
  .initial_value = 0,
  .return_tibble = TRUE
)
```

Arguments

.time	Total time of the simulation.
.num_sims	Total number of simulations.
.delta_time	Time step size.
.initial_value	Integer representing the initial value.
.return_tibble	The default is TRUE. If set to FALSE then an object of class matrix will be returned.

Details

Brownian Motion, also known as the Wiener process, is a continuous-time random process that describes the random movement of particles suspended in a fluid. It is named after the physicist Robert Brown, who first described the phenomenon in 1827.

The equation for Brownian Motion can be represented as:

```
W(t) = W(0) + sqrt(t) * Z
```

Where W(t) is the Brownian motion at time t, W(0) is the initial value of the Brownian motion, sqrt(t) is the square root of time, and Z is a standard normal random variable.

Brownian Motion has numerous applications, including modeling stock prices in financial markets, modeling particle movement in fluids, and modeling random walk processes in general. It is a useful tool in probability theory and statistical analysis.

Value

A tibble/matrix

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: tidy_fft(), ts_brownian_motion_augment(), ts_geometric_brownian_motion_augment(), ts_geometric_brownian_motion(), ts_random_walk()

Examples

```
ts_brownian_motion()
```

---

ts_brownian_motion_augment *Brownian Motion*

Description

Create a Brownian Motion Tibble

Usage

```
ts_brownian_motion_augment(
  .data,
  .date_col,
  .value_col,
  .time = 100,
  .num_sims = 10,
  .delta_time = NULL
)
```

Arguments

.data	The data.frame/tibble being augmented.
.date_col	The column that holds the date.
.value_col	The value that is going to get augmented. The last value of this column becomes the initial value internally.
.time	How many time steps ahead.
.num_sims	How many simulations should be run.
.delta_time	Time step size.

Details

Brownian Motion, also known as the Wiener process, is a continuous-time random process that describes the random movement of particles suspended in a fluid. It is named after the physicist Robert Brown, who first described the phenomenon in 1827.

The equation for Brownian Motion can be represented as:

```
W(t) = W(0) + sqrt(t) * Z
```

Where W(t) is the Brownian motion at time t, W(0) is the initial value of the Brownian motion, sqrt(t) is the square root of time, and Z is a standard normal random variable.

Brownian Motion has numerous applications, including modeling stock prices in financial markets, modeling particle movement in fluids, and modeling random walk processes in general. It is a useful tool in probability theory and statistical analysis.

Value

A tibble/matrix

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: tidy_fft(), ts_brownian_motion(), ts_geometric_brownian_motion_augment(), ts_geometric_brownian_motion(), ts_random_walk()

Examples

```
rn <- rnorm(31)
df <- data.frame(
 date_col = seq.Date(from = as.Date("2022-01-01"),
                     to = as.Date("2022-01-31"),
                     by = "day"),
 value = rn
)

ts_brownian_motion_augment(
  .data = df,
  .date_col = date_col,
  .value_col = value
)
```

ts_brownian_motion_plot *Auto-Plot a Geometric/Brownian Motion Augment*

Description

Plot an augmented Geometric/Brownian Motion.

Usage

```
ts_brownian_motion_plot(.data, .date_col, .value_col, .interactive = FALSE)
```

Arguments

`.data`	The data you are going to pass to the function to augment.
`.date_col`	The column that holds the date
`.value_col`	The column that holds the value
`.interactive`	The default is FALSE, TRUE will produce an interactive plotly plot.

Details

This function will take output from either the `ts_brownian_motion_augment()` or the `ts_geometric_brownian_motion_a`
function and plot them. The legend is set to "none" if the simulation count is higher than 9.

Value

A ggplot2 object or an interactive `plotly` plot

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Plot: `ts_event_analysis_plot()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`

Examples

```
library(dplyr)

df <- ts_to_tbl(AirPassengers) %>% select(-index)

augmented_data <- df %>%
  ts_brownian_motion_augment(
    .date_col = date_col,
    .value_col = value,
    .time = 144
  )
```

```
augmented_data %>%
  ts_brownian_motion_plot(.date_col = date_col, .value_col = value)
```

---

ts_calendar_heatmap_plot

Time Series Calendar Heatmap

Description

Takes in data that has been aggregated to the day level and makes a calendar heatmap.

Usage

```
ts_calendar_heatmap_plot(
  .data,
  .date_col,
  .value_col,
  .low = "red",
  .high = "green",
  .plt_title = "",
  .interactive = TRUE
)
```

Arguments

.data	The time-series data with a date column and value column.
.date_col	The column that has the datetime values
.value_col	The column that has the values
.low	The color for the low value, must be quoted like "red". The default is "red"
.high	The color for the high value, must be quoted like "green". The default is "green"
.plt_title	The title of the plot
.interactive	Default is TRUE to get an interactive plot using `plotly::ggplotly()`. It can be set to FALSE to get a ggplot plot.

Details

The data provided must have been aggregated to the day level, if not funky output could result and it is possible nothing will be output but errors. There must be a date column and a value column, those are the only items required for this function to work.

This function is intentionally inflexible, it complains more and does less in order to force the user to supply a clean data-set.

Value

A ggplot2 plot or if interactive a plotly plot

Author(s)

Steven P. Sanderson II, MPH

Examples

```
data_tbl <- data.frame(
  date_col = seq.Date(
    from = as.Date("2020-01-01"),
    to   = as.Date("2022-06-01"),
    length.out = 365*2 + 180
    ),
  value = rnorm(365*2+180, mean = 100)
)

ts_calendar_heatmap_plot(
  .data          = data_tbl
  , .date_col    = date_col
  , .value_col   = value
  , .interactive = FALSE
)
```

`ts_compare_data` *Compare data over time periods*

Description

Given a tibble/data.frame, you can get date from two different but comparative date ranges. Lets say you want to compare visits in one year to visits from 2 years before without also seeing the previous 1 year. You can do that with this function.

Usage

```
ts_compare_data(.data, .date_col, .start_date, .end_date, .periods_back)
```

Arguments

`.data`	The date.frame/tibble that holds the data
`.date_col`	The column with the date value
`.start_date`	The start of the period you want to analyze
`.end_date`	The end of the period you want to analyze
`.periods_back`	How long ago do you want to compare data too. Time units are collapsed using `lubridate::floor_date()`. The value can be:

- second
- minute
- hour
- day
- week
- month
- bimonth
- quarter
- season
- halfyear
- year

Arbitrary unique English abbreviations as in the lubridate::period() constructor are allowed.

Details

- Uses the timetk::filter_by_time() function in order to filter the date column.
- Uses the timetk::subtract_time() function to subtract time from the start date.

Value

A tibble.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Time_Filtering: ts_time_event_analysis_tbl()

Examples

```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(timetk))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

ts_compare_data(
  .data           = data_tbl
  , .date_col     = date_col
  , .start_date   = "1955-01-01"
  , .end_date     = "1955-12-31"
  , .periods_back = "2 years"
  ) %>%
  summarise_by_time(
    .date_var = date_col
    , .by     = "year"
```

```
    , visits  = sum(value)
  )
```

---

ts_event_analysis_plot *Time Series Event Analysis Plot*

Description

Plot out the data from the `ts_time_event_analysis_tbl()` function.

Usage

```
ts_event_analysis_plot(
  .data,
  .plot_type = "mean",
  .plot_ci = TRUE,
  .interactive = FALSE
)
```

Arguments

`.data`	The data that comes from the `ts_time_event_analysis_tbl()`
`.plot_type`	The default is "mean" which will show the mean event change of the output from the analysis tibble. The possible values for this are: mean, median, and individual.
`.plot_ci`	The default is TRUE. This will only work if you choose one of the aggregate plots of either "mean" or "median"
`.interactive`	The default is FALSE. TRUE will return a plotly plot.

Details

This function will take in data strictly from the `ts_time_event_analysis_tbl()` and plot out the data. You can choose what type of plot you want in the parameter of `.plot_type`. This will give you a choice of "mean", "median", and "individual".

You can also plot the upper and lower confidence intervals if you choose one of the aggregate plots ("mean"/"median").

Value

A ggplot2 object

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Plot: ts_brownian_motion_plot(), ts_qq_plot(), ts_scedacity_scatter_plot()

Examples

```
library(dplyr)
df <- ts_to_tbl(AirPassengers) %>% select(-index)

ts_time_event_analysis_tbl(
  .data = df,
  .horizon = 6,
  .date_col = date_col,
  .value_col = value,
  .direction = "both"
) %>%
  ts_event_analysis_plot()

ts_time_event_analysis_tbl(
  .data = df,
  .horizon = 6,
  .date_col = date_col,
  .value_col = value,
  .direction = "both"
) %>%
  ts_event_analysis_plot(.plot_type = "individual")
```

---

ts_extract_auto_fitted_workflow

Extract Boilerplate Items

Description

Extract the fitted workflow from a `ts_auto_` function.

Usage

```
ts_extract_auto_fitted_workflow(.input)
```

Arguments

`.input`	This is the output list object of a `ts_auto_` function.

Details

Extract the fitted workflow from a `ts_auto_` function. This will only work on those functions that are designated as *Boilerplate*.

Value

A fitted workflow object.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
## Not run:
library(dplyr)

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
  data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

ts_lm <- ts_auto_lm(
  .data = data,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
)

ts_extract_auto_fitted_workflow(ts_lm)

## End(Not run)
```

---

ts_feature_cluster *Time Series Feature Clustering*

Description

This function returns an output list of data and plots that come from using the K-Means clustering algorithm on a time series data.

Usage

```
ts_feature_cluster(
  .data,
  .date_col,
  .value_col,
  ...,
  .features = c("frequency", "entropy", "acf_features"),
  .scale = TRUE,
  .prefix = "ts_",
  .centers = 3
)
```

Arguments

`.data`	The data passed must be a `data.frame/tibble` only.
`.date_col`	The date column.
`.value_col`	The column that holds the value of the time series where you want the features and clustering performed on.
`...`	This is where you can place grouping variables that are passed off to `dplyr::group_by()`
`.features`	This is a quoted string vector using c() of features that you would like to pass. You can pass any feature you make or those from the `tsfeatures` package.
`.scale`	If TRUE, time series are scaled to mean 0 and sd 1 before features are computed
`.prefix`	A prefix to prefix the feature columns. Default: "ts_"
`.centers`	An integer of how many different centers you would like to generate. The default is 3.

Details

This function will return a list object output. The function itself requires that a time series tibble/data.frame get passed to it, along with the `.date_col`, the `.value_col` and a period of data. It uses the underlying function `timetk::tk_tsfeatures()` and takes the output of that and performs a clustering analysis using the `K-Means` algorithm.

The function has a parameter of `.features` which can take any of the features listed in the `tsfeatures` package by Rob Hyndman. You can also create custom functions in the `.GlobalEnviron` and it will take them as quoted arguments.

So you can make a function as follows

`my_mean <- function(x){return(mean(x, na.rm = TRUE))}`

You can then call this by using `.features = c("my_mean")`.

The output of this function includes the following:

Data Section

- ts_feature_tbl
- user_item_matrix_tbl
- mapped_tbl

- scree_data_tbl
- input_data_tbl (the original data)

Plots

- static_plot
- plotly_plot

Value

A list output

Author(s)

Steven P. Sanderson II, MPH

See Also

https://pkg.robjhyndman.com/tsfeatures/index.html

Other Clustering: `ts_feature_cluster_plot()`

Examples

```
library(dplyr)

data_tbl <- ts_to_tbl(AirPassengers) %>%
  mutate(group_id = rep(1:12, 12))

ts_feature_cluster(
  .data = data_tbl,
  .date_col = date_col,
  .value_col = value,
  group_id,
  .features = c("acf_features","entropy"),
  .scale = TRUE,
  .prefix = "ts_",
  .centers = 3
)
```

---

ts_feature_cluster_plot

Time Series Feature Clustering

Description

This function returns an output list of data and plots that come from using the `K-Means` clustering algorithm on a time series data.

Usage

```
ts_feature_cluster_plot(
  .data,
  .date_col,
  .value_col,
  ...,
  .center = 3,
  .facet_ncol = 3,
  .smooth = FALSE
)
```

Arguments

`.data`	The data passed must be the output of the `ts_feature_cluster()` function.
`.date_col`	The date column.
`.value_col`	The column that holds the value of the time series that the featurs were built from.
`...`	This is where you can place grouping variables that are passed off to `dplyr::group_by()`
`.center`	An integer of the chosen amount of centers from the `ts_feature_cluster()` function.
`.facet_ncol`	This is passed to the `timetk::plot_time_series()` function.
`.smooth`	This is passed to the `timetk::plot_time_series()` function and is set to a default of FALSE.

Details

This function will return a list object output. The function itself requires that the `ts_feature_cluster()` be passed to it as it will look for a specific attribute internally.

The output of this function includes the following:

Data Section

- original_data
- kmm_data_tbl
- user_item_tbl
- cluster_tbl

Plots

- static_plot
- plotly_plot

K-Means Object

- k-means object

Value

A list output

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Clustering: ts_feature_cluster()

Examples

```
library(dplyr)

data_tbl <- ts_to_tbl(AirPassengers) %>%
  mutate(group_id = rep(1:12, 12))

output <- ts_feature_cluster(
  .data = data_tbl,
  .date_col = date_col,
  .value_col = value,
  group_id,
  .features = c("acf_features","entropy"),
  .scale = TRUE,
  .prefix = "ts_",
  .centers = 3
)

ts_feature_cluster_plot(
  .data = output,
  .date_col = date_col,
  .value_col = value,
  .center = 2,
  group_id
)
```

---

ts_forecast_simulator *Time-series Forecasting Simulator*

Description

Creating different forecast paths for forecast objects (when applicable), by utilizing the underlying model distribution with the simulate function.

Usage

```
ts_forecast_simulator(
  .model,
  .data,
  .ext_reg = NULL,
  .frequency = NULL,
  .bootstrap = TRUE,
  .horizon = 4,
  .iterations = 25,
  .sim_color = "steelblue",
  .alpha = 0.05
)
```

Arguments

`.model`	A forecasting model of one of the following from the `forecast` package: • `Arima` • `auto.arima` • `ets` • `nnetar` • `Arima()` with xreg
`.data`	The data that is used for the `.model` parameter. This is used with `timetk::tk_index()`
`.ext_reg`	A `tibble` or `matrix` of future xregs that should be the same length as the horizon you want to forecast.
`.frequency`	This is for the conversion of an internal table and should match the time frequency of the data.
`.bootstrap`	A boolean value of TRUE/FALSE. From `forecast::simulate.Arima()` Do simulation using resampled errors rather than normally distributed errors.
`.horizon`	An integer defining the forecast horizon.
`.iterations`	An integer, set the number of iterations of the simulation.
`.sim_color`	Set the color of the simulation paths lines.
`.alpha`	Set the opacity level of the simulation path lines.

Details

This function expects to take in a model of either `Arima`, `auto.arima`, `ets` or `nnetar` from the `forecast` package. You can supply a forecasting horizon, iterations and a few other items. You may also specify an Arima() model using xregs.

Value

The original time series, the simulated values and a some plots

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Simulator: ts_arima_simulator()

Examples

```
suppressPackageStartupMessages(library(forecast))
suppressPackageStartupMessages(library(dplyr))

# Create a model
fit <- auto.arima(AirPassengers)
data_tbl <- ts_to_tbl(AirPassengers)

# Simulate 50 possible forecast paths, with .horizon of 12 months
output <- ts_forecast_simulator(
  .model        = fit
  , .horizon    = 12
  , .iterations = 50
  , .data       = data_tbl
)

output$ggplot
```

---

ts_geometric_brownian_motion

Geometric Brownian Motion

Description

Create a Geometric Brownian Motion.

Usage

```
ts_geometric_brownian_motion(
  .num_sims = 100,
  .time = 25,
  .mean = 0,
  .sigma = 0.1,
  .initial_value = 100,
  .delta_time = 1/365,
  .return_tibble = TRUE
)
```

Arguments

.num_sims	Total number of simulations.
.time	Total time of the simulation.
.mean	Expected return

`.sigma`	Volatility
`.initial_value`	Integer representing the initial value.
`.delta_time`	Time step size.
`.return_tibble`	The default is TRUE. If set to FALSE then an object of class matrix will be returned.

Details

Geometric Brownian Motion (GBM) is a statistical method for modeling the evolution of a given financial asset over time. It is a type of stochastic process, which means that it is a system that undergoes random changes over time.

GBM is widely used in the field of finance to model the behavior of stock prices, foreign exchange rates, and other financial assets. It is based on the assumption that the asset's price follows a random walk, meaning that it is influenced by a number of unpredictable factors such as market trends, news events, and investor sentiment.

The equation for GBM is:

```
dS/S = mdt + sdW
```

where S is the price of the asset, t is time, m is the expected return on the asset, s is the volatility of the asset, and dW is a small random change in the asset's price.

GBM can be used to estimate the likelihood of different outcomes for a given asset, and it is often used in conjunction with other statistical methods to make more accurate predictions about the future performance of an asset.

This function provides the ability of simulating and estimating the parameters of a GBM process. It can be used to analyze the behavior of financial assets and to make informed investment decisions.

Value

A tibble/matrix

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: `tidy_fft()`, `ts_brownian_motion_augment()`, `ts_brownian_motion()`, `ts_geometric_brownian_motion_augment()`, `ts_random_walk()`

Examples

```
ts_geometric_brownian_motion()
```

`ts_geometric_brownian_motion_augment`
Geometric Brownian Motion

Description

Create a Geometric Brownian Motion.

Usage

```
ts_geometric_brownian_motion_augment(
  .data,
  .date_col,
  .value_col,
  .num_sims = 10,
  .time = 25,
  .mean = 0,
  .sigma = 0.1,
  .delta_time = 1/365
)
```

Arguments

`.data`	The data you are going to pass to the function to augment.
`.date_col`	The column that holds the date
`.value_col`	The column that holds the value
`.num_sims`	Total number of simulations.
`.time`	Total time of the simulation.
`.mean`	Expected return
`.sigma`	Volatility
`.delta_time`	Time step size.

Details

Geometric Brownian Motion (GBM) is a statistical method for modeling the evolution of a given financial asset over time. It is a type of stochastic process, which means that it is a system that undergoes random changes over time.

GBM is widely used in the field of finance to model the behavior of stock prices, foreign exchange rates, and other financial assets. It is based on the assumption that the asset's price follows a random walk, meaning that it is influenced by a number of unpredictable factors such as market trends, news events, and investor sentiment.

The equation for GBM is:

```
dS/S = mdt + sdW
```

where S is the price of the asset, t is time, m is the expected return on the asset, s is the volatility of the asset, and dW is a small random change in the asset's price.

GBM can be used to estimate the likelihood of different outcomes for a given asset, and it is often used in conjunction with other statistical methods to make more accurate predictions about the future performance of an asset.

This function provides the ability of simulating and estimating the parameters of a GBM process. It can be used to analyze the behavior of financial assets and to make informed investment decisions.

Value

A tibble/matrix

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: `tidy_fft()`, `ts_brownian_motion_augment()`, `ts_brownian_motion()`, `ts_geometric_brownian_motion()`, `ts_random_walk()`

Examples

```
rn <- rnorm(31)
df <- data.frame(
 date_col = seq.Date(from = as.Date("2022-01-01"),
                     to = as.Date("2022-01-31"),
                     by = "day"),
 value = rn
)

ts_geometric_brownian_motion_augment(
  .data = df,
  .date_col = date_col,
  .value_col = value
)
```

`ts_get_date_columns` *Get date or datetime variables (column names)*

Description

Get date or datetime variables (column names)

Usage

```
ts_get_date_columns(.data)
```

Arguments

`.data`	An object of class `data.frame`

Details

`ts_get_date_columns` returns the column names of date or datetime variables in a data frame.

Value

A vector containing the column names that are of date/date-like classes.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
ts_to_tbl(AirPassengers) %>%
  ts_get_date_columns()
```

`ts_growth_rate_augment` *Augment Data with Time Series Growth Rates*

Description

This function is used to augment a data frame or tibble with time series growth rates of selected columns. You can provide a data frame or tibble as the first argument, the column(s) for which you want to calculate the growth rates using the `.value` parameter, and optionally specify custom names for the new columns using the `.names` parameter.

Usage

```
ts_growth_rate_augment(.data, .value, .names = "auto")
```

Arguments

`.data`	A data frame or tibble containing the data to be augmented.
`.value`	A quosure specifying the column(s) for which you want to calculate growth rates.
`.names`	Optional. A character vector specifying the names of the new columns to be created. Use "auto" for automatic naming.

Value

A tibble that includes the original data and additional columns representing the growth rates of the selected columns. The column names are either automatically generated or as specified in the `.names` parameter.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: `ts_acceleration_augment()`, `ts_velocity_augment()`

Examples

```
data <- data.frame(
  Year = 1:5,
  Income = c(100, 120, 150, 180, 200),
  Expenses = c(50, 60, 75, 90, 100)
)
ts_growth_rate_augment(data, .value = c(Income, Expenses))
```

`ts_growth_rate_vec` *Vector Function Time Series Growth Rate*

Description

This function computes the growth rate of a numeric vector, typically representing a time series, with optional transformations like scaling, power, and lag differences.

Usage

```
ts_growth_rate_vec(.x, .scale = 100, .power = 1, .log_diff = FALSE, .lags = 1)
```

Arguments

`.x`	A numeric vector
`.scale`	A numeric value that is used to scale the output
`.power`	A numeric value that is used to raise the output to a power
`.log_diff`	A logical value that determines whether the output is a log difference
`.lags`	An integer that determines the number of lags to use

Details

The function calculates growth rates for a time series, allowing for scaling, exponentiation, and lag differences. It can be useful for financial data analysis, among other applications.

The growth rate is computed as follows:

- If lags is positive and log_diff is FALSE: growth_rate = (((x / lag(x, lags))^power) - 1) * scale
- If lags is positive and log_diff is TRUE: growth_rate = log(x / lag(x, lags)) * scale
- If lags is negative and log_diff is FALSE: growth_rate = (((x / lead(x, -lags))^power) - 1) * scale
- If lags is negative and log_diff is TRUE: growth_rate = log(x / lead(x, -lags)) * scale

Value

A list object of workflows.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Vector Function: `ts_acceleration_vec()`, `ts_velocity_vec()`

Examples

```
# Calculate the growth rate of a time series without any transformations.
ts_growth_rate_vec(c(100, 110, 120, 130))

# Calculate the growth rate with scaling and a power transformation.
ts_growth_rate_vec(c(100, 110, 120, 130), .scale = 10, .power = 2)

# Calculate the log differences of a time series with lags.
ts_growth_rate_vec(c(100, 110, 120, 130), .log_diff = TRUE, .lags = -1)

# Plot
plot.ts(AirPassengers)
plot.ts(ts_growth_rate_vec(AirPassengers))
```

`ts_info_tbl` *Get Time Series Information*

Description

This function will take in a data set and return to you a tibble of useful information.

Usage

```
ts_info_tbl(.data, .date_col)
```

Arguments

`.data`	The data you are passing to the function
`.date_col`	This is only needed if you are passing a tibble.

Details

This function can accept objects of the following classes:

- ts
- xts
- mts
- zoo
- tibble/data.frame

The function will return the following pieces of information in a tibble:

- name
- class
- frequency
- start
- end
- var
- length

Value

A tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
ts_info_tbl(AirPassengers)
ts_info_tbl(BJsales)
```

---

ts_is_date_class *Check if an object is a date class*

Description

Check if an object is a date class

Usage

```
ts_is_date_class(.x)
```

Arguments

.x	A vector to check

Value

Logical (TRUE/FALSE)

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
seq.Date(from = as.Date("2022-01-01"), by = "day", length.out = 10) %>%
ts_is_date_class()

letters %>% ts_is_date_class()
```

`ts_lag_correlation` *Time Series Lag Correlation Analysis*

Description

This function outputs a list object of both data and plots.

The data output are the following:

- lag_list
- lag_tbl
- correlation_lag_matrix
- correlation_lag_tbl

The plots output are the following:

- lag_plot
- plotly_lag_plot
- correlation_heatmap
- plotly_heatmap

Usage

```
ts_lag_correlation(
  .data,
  .date_col,
  .value_col,
  .lags = 1,
  .heatmap_color_low = "white",
  .heatmap_color_hi = "steelblue"
)
```

Arguments

`.data`	A tibble of time series data
`.date_col`	A date column
`.value_col`	The value column being analyzed
`.lags`	This is a vector of integer lags, ie 1 or c(1,6,12)

.heatmap_color_low
: What color should the low values of the heatmap of the correlation matrix be, the default is 'white'

.heatmap_color_hi
: What color should the low values of the heatmap of the correlation matrix be, the default is 'steelblue'

Details

This function takes in a time series data in the form of a tibble and outputs a list object of data and plots. This function will take in an argument of '.lags' and get those lags in your data, outputting a correlation matrix, heatmap and lag plot among other things of the input data.

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
library(dplyr)

df <- ts_to_tbl(AirPassengers) %>% select(-index)
lags <- c(1,3,6,12)

output <- ts_lag_correlation(
  .data = df,
  .date_col = date_col,
  .value_col = value,
  .lags = lags
)

output$data$correlation_lag_matrix
output$plots$lag_plot
```

ts_ma_plot *Time Series Moving Average Plot*

Description

This function will produce two plots. Both of these are moving average plots. One of the plots is from `xts::plot.xts()` and the other a `ggplot2` plot. This is done so that the user can choose which type is best for them. The plots are stacked so each graph is on top of the other.

Usage

```
ts_ma_plot(
  .data,
  .date_col,
  .value_col,
  .ts_frequency = "monthly",
  .main_title = NULL,
  .secondary_title = NULL,
  .tertiary_title = NULL
)
```

Arguments

`.data`	The data you want to visualize. This should be pre-processed and the aggregation should match the `.frequency` argument.
`.date_col`	The data column from the `.data` argument.
`.value_col`	The value column from the `.data` argument
`.ts_frequency`	The frequency of the aggregation, quoted, ie. "monthly", anything else will default to weekly, so it is very important that the data passed to this function be in either a weekly or monthly aggregation.
`.main_title`	The title of the main plot.
`.secondary_title`	The title of the second plot.
`.tertiary_title`	The title of the third plot.

Details

This function expects to take in a data.frame/tibble. It will return a list object so it is a good idea to save the output to a variable and extract from there.

Value

A few time series data sets and two plots.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
suppressPackageStartupMessages(library(dplyr))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

output <- ts_ma_plot(
  .data = data_tbl,
  .date_col = date_col,
  .value_col = value
)

output$pgrid
output$xts_plt
output$data_summary_tbl %>% head()

output <- ts_ma_plot(
  .data = data_tbl,
  .date_col = date_col,
  .value_col = value,
  .ts_frequency = "week"
)

output$pgrid
output$xts_plt
output$data_summary_tbl %>% head()
```

---

ts_model_auto_tune *Time Series Model Tuner*

Description

This function will create a tuned model. It uses the `ts_model_spec_tune_template()` under the hood to get the generic template that is used in the grid search.

Usage

```
ts_model_auto_tune(
  .modeltime_model_id,
  .calibration_tbl,
  .splits_obj,
  .drop_training_na = TRUE,
  .date_col,
  .value_col,
```

```
  .tscv_assess = "12 months",
  .tscv_skip = "6 months",
  .slice_limit = 6,
  .facet_ncol = 2,
  .grid_size = 30,
  .num_cores = 1,
  .best_metric = "rmse"
)
```

Arguments

`.modeltime_model_id`	The .model_id from a calibrated modeltime table.
`.calibration_tbl`	A calibrated modeltime table.
`.splits_obj`	The time_series_split object.
`.drop_training_na`	A boolean that will drop NA values from the training(splits) data
`.date_col`	The column that holds the date values.
`.value_col`	The column that holds the time series values.
`.tscv_assess`	A character expression like "12 months". This gets passed to `timetk::time_series_cv()`
`.tscv_skip`	A character expression like "6 months". This gets passed to `timetk::time_series_cv()`
`.slice_limit`	An integer that gets passed to `timetk::time_series_cv()`
`.facet_ncol`	The number of faceted columns to be passed to plot_time_series_cv_plan
`.grid_size`	An integer that gets passed to the `dials::grid_latin_hypercube()` function.
`.num_cores`	The default is 1, you can set this to any integer value as long as it is equal to or less than the available cores on your machine.
`.best_metric`	The default is "rmse" and this can be set to any default dials metric. This must be passed as a character.

Details

This function can work with the following parsnip/modeltime engines:

- "auto_arima"
- "auto_arima_xgboost"
- "ets"
- "croston"
- "theta"
- "stlm_ets"
- "tbats"
- "stlm_arima"
- "nnetar"

- "prophet"
- "prophet_xgboost"
- "lm"
- "glmnet"
- "stan"
- "spark"
- "keras"
- "earth"
- "xgboost"
- "kernlab"

This function returns a list object with several items inside of it. There are three categories of items that are inside of the list.

- `data`
- `model_info`
- `plots`

The `data` section has the following items:

- `calibration_tbl` This is the calibration data passed into the function.
- `calibration_tuned_tbl` This is a calibration tibble that has used the tuned workflow.
- `tscv_data_tbl` This is the tibble of the time series cross validation.
- `tuned_results` This is a tuning results tibble with all slices from the time series cross validation.
- `best_tuned_results_tbl` This is a tibble of the parameters for the best test set with the chosen metric.
- `tscv_obj` This is the actual time series cross validation object returned from `timetk::time_series_cv()`

The `model_info` section has the following items:

- `model_spec` This is the original modeltime/parsnip model specification.
- `model_spec_engine` This is the engine used for the model specification.
- `model_spec_tuner` This is the tuning model template returned from `ts_model_spec_tune_template()`
- `plucked_model` This is the model that we have plucked from the calibration tibble for tuning.
- `wflw_tune_spec` This is a new workflow with the `model_spec_tuner` attached.
- `grid_spec` This is the grid search specification for the tuning process.
- `tuned_tscv_wflw_spec` This is the final tuned model where the workflow and model have been finalized. This would be the model that you would want to pull out if you are going to work with it further.

The `plots` section has the following items:

- `tune_results_plt` This is a static ggplot of the grid search.
- `tscv_pl` This is the time series cross validation plan plot.

Value

A list object with multiple items.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Model Tuning: ts_model_spec_tune_template()

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
## Not run:
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))

data <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
    data
    , date_col
    , assess = 12
    , skip = 3
    , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
  .data = data
  , .date_col = date_col
  , .pred_col = value
)

wfsets <- ts_wfs_mars(
  .model_type = "earth"
  , .recipe_list = rec_objs
)

wf_fits <- wfsets %>%
  modeltime_fit_workflowset(
    data = training(splits)
    , control = control_fit_workflowset(
     allow_par = TRUE
```

```
    , verbose = TRUE
   )
  )

models_tbl <- wf_fits %>%
  filter(.model != "NULL")

calibration_tbl <- models_tbl %>%
  modeltime_calibrate(new_data = testing(splits))

output <- ts_model_auto_tune(
  .modeltime_model_id = 1,
  .calibration_tbl = calibration_tbl,
  .splits_obj = splits,
  .drop_training_na = TRUE,
  .date_col = date_col,
  .value_col = value,
  .tscv_assess = "12 months",
  .tscv_skip = "3 months",
  .num_cores = parallel::detectCores() - 1
)

## End(Not run)
```

`ts_model_compare` *Compare Two Time Series Models*

Description

This function will expect to take in two models that will be used for comparison. It is useful to use this after appropriately following the modeltime workflow and getting two models to compare. This is an extension of the calibrate and plot, but it only takes two models and is most likely better suited to be used after running a model through the `ts_model_auto_tune()` function to see the difference in performance after a base model has been tuned.

Usage

```
ts_model_compare(
  .model_1,
  .model_2,
  .type = "testing",
  .splits_obj,
  .data,
  .print_info = TRUE,
  .metric = "rmse"
)
```

Arguments

.model_1	The model being compared to the base, this can also be a hyperparameter tuned model.
.model_2	The base model.
.type	The default is the testing tibble, can be set to training as well.
.splits_obj	The splits object
.data	The original data that was passed to splits
.print_info	This is a boolean, the default is TRUE
.metric	This should be one of the following character strings: • "mae" • "mape" • "mase" • "smape" • "rmse" • "rsq"

Details

This function expects to take two models. You must tell it if it will be assessing the training or testing data, where the testing data is the default. You must therefore supply the splits object to this function along with the origianl dataset. You must also tell it which default modeltime accuracy metric should be printed on the graph itself. You can also tell this function to print information to the console or not. A static `ggplot2` polot and an interactive `plotly` plot will be returned inside of the output list.

Value

The function outputs a list invisibly.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
## Not run:
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(dplyr))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
  data       = data_tbl,
  date_var   = date_col,
  assess     = "12 months",
  cumulative = TRUE
)

rec_obj <- ts_auto_recipe(
 .data     = data_tbl,
 .date_col = date_col,
 .pred_col = value
)

wfs_mars <- ts_wfs_mars(.recipe_list = rec_obj)

wf_fits <- wfs_mars %>%
  modeltime_fit_workflowset(
    data = training(splits)
    , control = control_fit_workflowset(
         allow_par = FALSE
         , verbose = TRUE
       )
 )

calibration_tbl <- wf_fits %>%
    modeltime_calibrate(new_data = testing(splits))

base_mars <- calibration_tbl %>% pluck_modeltime_model(1)
date_mars <- calibration_tbl %>% pluck_modeltime_model(2)

ts_model_compare(
 .model_1    = base_mars,
 .model_2    = date_mars,
 .type       = "testing",
 .splits_obj = splits,
 .data       = data_tbl,
 .print_info = TRUE,
 .metric     = "rmse"
 )$plots$static_plot

## End(Not run)
```

ts_model_rank_tbl *Model Rank*

Description

This takes in a calibration tibble and computes the ranks of the models inside of it.

Usage

```
ts_model_rank_tbl(.calibration_tbl)
```

Arguments

`.calibration_tbl`
A calibrated modeltime table.

Details

This takes in a calibration tibble and computes the ranks of the models inside of it. It computes for now only the default `yardstick` metrics from `modeltime` These are the following using the `dplyr` `min_rank()` function with `desc` use on `rsq`:

- "rmse"
- "mae"
- "mape"
- "smape"
- "rsq"

Value

A tibble with models ranked by metric performance order

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
# NOT RUN
## Not run:
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(workflows))
suppressPackageStartupMessages(library(parsnip))
suppressPackageStartupMessages(library(recipes))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
  data_tbl,
  date_var = date_col,
  assess = "12 months",
  cumulative = TRUE
)

rec_obj <- recipe(value ~ ., training(splits))

model_spec_arima <- arima_reg() %>%
  set_engine(engine = "auto_arima")

model_spec_mars <- mars(mode = "regression") %>%
  set_engine("earth")

wflw_fit_arima <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_arima) %>%
  fit(training(splits))

wflw_fit_mars <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_mars) %>%
  fit(training(splits))

model_tbl <- modeltime_table(wflw_fit_arima, wflw_fit_mars)

calibration_tbl <- model_tbl %>%
  modeltime_calibrate(new_data = testing(splits))

ts_model_rank_tbl(calibration_tbl)


## End(Not run)
```

---

ts_model_spec_tune_template
Time Series Model Spec Template

Description

This function will create a generic tuneable model specification, this function can be used by itself and is called internally by `ts_model_auto_tune()`.

Usage

```
ts_model_spec_tune_template(.parsnip_engine = NULL, .model_spec_class = NULL)
```

Arguments

`.parsnip_engine`
: The model engine that is used by `parsnip::set_engine()`.

`.model_spec_class`
: The model spec class that is use by `parsnip`. For example the 'kernlab' engine can use both `svm_poly` and `svm_rbf`.

Details

This function takes in a single parameter and uses that to output a generic tuneable model specification. This function can work with the following parsnip/modeltime engines:

- "auto_arima"
- "auto_arima_xgboost"
- "ets"
- "croston"
- "theta"
- "smooth_es"
- "stlm_ets"
- "tbats"
- "stlm_arima"
- "nnetar"
- "prophet"
- "prophet_xgboost"
- "lm"
- "glmnet"
- "stan"
- "spark"

- "keras"
- "earth"
- "xgboost"
- "kernlab"

Value

A tuneable parsnip model specification.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Model Tuning: ts_model_auto_tune()

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
ts_model_spec_tune_template("ets")
ts_model_spec_tune_template("prophet")
```

---

ts_qc_run_chart *Quality Control Run Chart*

Description

A control chart is a specific type of graph that shows data points between upper and lower limits over a period of time. You can use it to understand if the process is in control or not. These charts commonly have three types of lines such as upper and lower specification limits, upper and lower limits and planned value. By the help of these lines, Control Charts show the process behavior over time.

Usage

```
ts_qc_run_chart(
  .data,
  .date_col,
  .value_col,
  .interactive = FALSE,
  .median = TRUE,
  .cl = TRUE,
  .mcl = TRUE,
  .ucl = TRUE,
  .lc = FALSE,
  .lmcl = FALSE,
  .llcl = FALSE
)
```

Arguments

`.data`	The data.frame/tibble to be passed.
`.date_col`	The column holding the timestamp.
`.value_col`	The column with the values to be analyzed.
`.interactive`	Default is FALSE, TRUE for an interactive plotly plot.
`.median`	Default is TRUE. This will show the median line of the data.
`.cl`	This is the first upper control line
`.mcl`	This is the second sigma control line positive
`.ucl`	This is the third sigma control line positive
`.lc`	This is the first negative control line
`.lmcl`	This is the second sigma negative control line
`.llcl`	This si the thrid sigma negative control line

Details

- Expects a time-series tibble/data.frame
- Expects a date column and a value column

Value

A static ggplot2 graph or if .interactive is set to TRUE a plotly plot

Author(s)

Steven P. Sanderson II, MPH

Examples

```
library(dplyr)

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

data_tbl %>%
  ts_qc_run_chart(
    .date_col    = date_col
    , .value_col = value
    , .llcl      = TRUE
  )
```

---

ts_qq_plot *Time Series Model QQ Plot*

Description

This takes in a calibration tibble and will produce a QQ plot.

Usage

```
ts_qq_plot(.calibration_tbl, .model_id = NULL, .interactive = FALSE)
```

Arguments

`.calibration_tbl`	A calibrated modeltime table.
`.model_id`	The id of a particular model from a calibration tibble. If there are multiple models in the tibble and this remains **NULL** then the plot will be returned using `ggplot2::facet_grid(~ .model_id)`
`.interactive`	A boolean with a default value of FALSE. TRUE will produce an interactive `plotly` plot.

Details

This takes in a calibration tibble and will create a QQ plot. You can also pass in a `model_id` and a boolean for `interactive` which will return a `plotly::ggplotly` interactive plot.

Value

A QQ plot.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot

Other Plot: ts_brownian_motion_plot(), ts_event_analysis_plot(), ts_scedacity_scatter_plot()

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
# NOT RUN
## Not run:
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(workflows))
suppressPackageStartupMessages(library(parsnip))
suppressPackageStartupMessages(library(recipes))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
  data_tbl,
  date_var = date_col,
  assess = "12 months",
  cumulative = TRUE
)

rec_obj <- recipe(value ~ ., training(splits))

model_spec_arima <- arima_reg() %>%
  set_engine(engine = "auto_arima")

model_spec_mars <- mars(mode = "regression") %>%
  set_engine("earth")

wflw_fit_arima <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_arima) %>%
  fit(training(splits))

wflw_fit_mars <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_mars) %>%
  fit(training(splits))

model_tbl <- modeltime_table(wflw_fit_arima, wflw_fit_mars)
```

```
calibration_tbl <- model_tbl %>%
  modeltime_calibrate(new_data = testing(splits))

ts_qq_plot(calibration_tbl)


## End(Not run)
```

---

ts_random_walk *Random Walk Function*

Description

This function takes in four arguments and returns a tibble of random walks.

Usage

```
ts_random_walk(
  .mean = 0,
  .sd = 0.1,
  .num_walks = 100,
  .periods = 100,
  .initial_value = 1000
)
```

Arguments

.mean	The desired mean of the random walks
.sd	The standard deviation of the random walks
.num_walks	The number of random walks you want generated
.periods	The length of the random walk(s) you want generated
.initial_value	The initial value where the random walks should start

Details

Monte Carlo simulations were first formally designed in the 1940's while developing nuclear weapons, and since have been heavily used in various fields to use randomness solve problems that are potentially deterministic in nature. In finance, Monte Carlo simulations can be a useful tool to give a sense of how assets with certain characteristics might behave in the future. While there are more complex and sophisticated financial forecasting methods such as ARIMA (Auto-Regressive Integrated Moving Average) and GARCH (Generalized Auto-Regressive Conditional Heteroskedasticity) which attempt to model not only the randomness but underlying macro factors such as seasonality and volatility clustering, Monte Carlo random walks work surprisingly well in illustrating market volatility as long as the results are not taken too seriously.

Value

A tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Data Generator: tidy_fft(), ts_brownian_motion_augment(), ts_brownian_motion(), ts_geometric_brownian_motion_augment(), ts_geometric_brownian_motion()

Examples

```
ts_random_walk(
.mean = 0,
.sd = 1,
.num_walks = 25,
.periods = 180,
.initial_value = 6
)
```

---

ts_random_walk_ggplot_layers

Get Random Walk ggplot2 *layers*

Description

Get layers to add to a ggplot graph from the ts_random_walk() function.

Usage

```
ts_random_walk_ggplot_layers(.data)
```

Arguments

.data	The data passed to the function.

Details

- Set the intercept of the initial value from the random walk
- Set the max and min of the cumulative sum of the random walks

Value

A ggplot2 layers object

Author(s)

Steven P. Sanderson II, MPH

Examples

```
library(ggplot2)

df <- ts_random_walk()

df %>%
  ggplot(
    mapping = aes(
      x = x
      , y = cum_y
      , color = factor(run)
      , group = factor(run)
    )
  ) +
  geom_line(alpha = 0.8) +
  ts_random_walk_ggplot_layers(df)
```

---

ts_scale_color_colorblind

Provide Colorblind Compliant Colors

Description

8 Hex RGB color definitions suitable for charts for colorblind people.

Usage

```
ts_scale_color_colorblind(..., theme = "ts")
```

Arguments

...	Data passed in from a ggplot object
theme	Right now this is ts only. Anything else will render an error.

Details

This function is used in others in order to help render plots for those that are color blind.

Value

A gggplot layer

Author(s)

Steven P. Sanderson II, MPH

ts_scale_fill_colorblind

Provide Colorblind Compliant Colors

Description

8 Hex RGB color definitions suitable for charts for colorblind people.

Usage

```
ts_scale_fill_colorblind(..., theme = "ts")
```

Arguments

...	Data passed in from a ggplot object
theme	Right now this is ts only. Anything else will render an error.

Details

This function is used in others in order to help render plots for those that are color blind.

Value

A gggplot layer

Author(s)

Steven P. Sanderson II, MPH

ts_scedacity_scatter_plot

Time Series Model Scedacity Plot

Description

This takes in a calibration tibble and will produce a scedacity plot.

Usage

```
ts_scedacity_scatter_plot(
  .calibration_tbl,
  .model_id = NULL,
  .interactive = FALSE
)
```

Arguments

.calibration_tbl
: A calibrated modeltime table.

.model_id
: The id of a particular model from a calibration tibble. If there are multiple models in the tibble and this remains **NULL** then the plot will be returned using `ggplot2::facet_grid(~ .model_id)`

.interactive
: A boolean with a default value of FALSE. TRUE will produce an interactive `plotly` plot.

Details

This takes in a calibration tibble and will create a scedacity plot. You can also pass in a `model_id` and a boolean for `interactive` which will return a `plotly::ggplotly` interactive plot.

Value

A Scedacity plot.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://en.wikipedia.org/wiki/Homoscedasticity

Other Plot: `ts_brownian_motion_plot()`, `ts_event_analysis_plot()`, `ts_qq_plot()`

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doublediff_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
# NOT RUN
## Not run:
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(workflows))
suppressPackageStartupMessages(library(parsnip))
suppressPackageStartupMessages(library(recipes))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
```

```
  data_tbl,
  date_var = date_col,
  assess = "12 months",
  cumulative = TRUE
)

rec_obj <- recipe(value ~ ., training(splits))

model_spec_arima <- arima_reg() %>%
  set_engine(engine = "auto_arima")

model_spec_mars <- mars(mode = "regression") %>%
  set_engine("earth")

wflw_fit_arima <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_arima) %>%
  fit(training(splits))

wflw_fit_mars <- workflow() %>%
  add_recipe(rec_obj) %>%
  add_model(model_spec_mars) %>%
  fit(training(splits))

model_tbl <- modeltime_table(wflw_fit_arima, wflw_fit_mars)

calibration_tbl <- model_tbl %>%
  modeltime_calibrate(new_data = testing(splits))

ts_scedacity_scatter_plot(calibration_tbl)


## End(Not run)
```

`ts_sma_plot` *Simple Moving Average Plot*

Description

This function will take in a value column and return any number `n` moving averages.

Usage

```
ts_sma_plot(
  .data,
  .date_col,
  .value_col,
  .sma_order = 2,
  .func = mean,
```

```
  .align = "center",
  .partial = FALSE
)
```

Arguments

`.data`	The data that you are passing, must be a data.frame/tibble.
`.date_col`	The column that holds the date.
`.value_col`	The column that holds the value.
`.sma_order`	This will default to 1. This can be a vector like c(2,4,6,12)
`.func`	The unquoted function you want to pass, mean, median, etc
`.align`	This can be either "left", "center", "right"
`.partial`	This is a bool value of TRUE/FALSE, the default is TRUE

Details

This function will accept a time series object or a tibble/data.frame. This is a simple wrapper around `timetk::slidify_vec()`. It uses that function to do the underlying moving average work.

It can only handle a single moving average at a time and therefore if multiple are called for, it will loop through and append data to a tibble object.

Value

Will return a list object.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
df <- ts_to_tbl(AirPassengers)
out <- ts_sma_plot(df, date_col, value, .sma_order = c(3,6))

out$data

out$plots$static_plot
```

ts_splits_plot *Time Series Splits Plot*

Description

Sometimes we want to see the training and testing data in a plot. This is a simple wrapper around a couple of functions from the `timetk` package.

Usage

```
ts_splits_plot(.splits_obj, .date_col, .value_col)
```

Arguments

`.splits_obj`	The predefined splits object.
`.date_col`	The date column for the time series.
`.value_col`	The value column of the time series.

Details

You should already have a splits object defined. This function takes in three parameters, the splits object, a date column and the value column.

Value

A time series cv plan plot

Author(s)

Steven P. Sanderson II, MPH

See Also

- https://business-science.github.io/timetk/reference/index.html#section-cross-validation-plan-v (timetk)
- https://business-science.github.io/timetk/reference/plot_time_series_cv_plan.html(tk_time_sers_cv_plan)
- https://business-science.github.io/timetk/reference/plot_time_series_cv_plan.html(plot_time_series_cv_plan)

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))

data <- ts_to_tbl(AirPassengers) %>%
  select(-index)

splits <- time_series_split(
    data
    , date_col
    , assess = 12
    , skip = 3
    , cumulative = TRUE
)

ts_splits_plot(
    .splits_obj = splits,
    .date_col   = date_col,
    .value_col  = value
)
```

---

ts_time_event_analysis_tbl

Event Analysis

Description

Given a tibble/data.frame, you can get information on what happens before, after, or in both directions of some given event, where the event is defined by some percentage increase/decrease in values from time `t` to `t+1`

Usage

```
ts_time_event_analysis_tbl(
  .data,
  .date_col,
  .value_col,
  .percent_change = 0.05,
  .horizon = 12,
  .precision = 2,
  .direction = "forward",
  .filter_non_event_groups = TRUE
)
```

Arguments

.data The date.frame/tibble that holds the data.

.date_col The column with the date value.

.value_col The column with the value you are measuring.

.percent_change
This defaults to 0.05 which is a 5% increase in the .value_col.

.horizon How far do you want to look back or ahead.

.precision The default is 2 which means it rounds the lagged 1 value percent change to 2 decimal points. You may want more for more finely tuned results, this will result in fewer groupings.

.direction The default is forward. You can supply either forward, backwards or both.

.filter_non_event_groups
The default is TRUE, this drops groupings with no events on the rare occasion it does occur.

Details

This takes in a data.frame/tibble of a time series. It requires a date column, and a value column. You can convert a ts/xts/zoo/mts object into a tibble by using the ts_to_tbl() function.

You will provide the function with a percentage change in the form of -1 to 1 inclusive. You then provide a time horizon in which you want to see. For example you may want to see what happens to AirPassengers after a 0.1 percent increase in volume.

The next most important thing to supply is the direction. Do you want to see what typically happens after such an event, what leads up to such an event, or both.

Value

A tibble.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Time_Filtering: ts_compare_data()

Examples

```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(ggplot2))

df_tbl <- ts_to_tbl(AirPassengers) %>% select(-index)

tst <- ts_time_event_analysis_tbl(df_tbl, date_col, value, .direction = "both",
.horizon = 6)
```

```
glimpse(tst)

tst %>%
  ggplot(aes(x = x, y = mean_event_change)) +
  geom_line() +
  geom_line(aes(y = event_change_ci_high), color = "blue", linetype = "dashed") +
  geom_line(aes(y = event_change_ci_low), color = "blue", linetype = "dashed") +
  geom_vline(xintercept = 7, color = "red", linetype = "dashed") +
  theme_minimal() +
  labs(
    title = "'AirPassengers' Event Analysis at 5% Increase",
    subtitle = "Vertical Red line is normalized event epoch - Direction: Both",
    x = "",
    y = "Mean Event Change"
  )
```

`ts_to_tbl` *Coerce a time-series object to a tibble*

Description

This function takes in a time-series object and returns it in a `tibble` format.

Usage

```
ts_to_tbl(.data)
```

Arguments

`.data`	The time-series object you want transformed into a `tibble`

Details

This function makes use of `timetk::tk_tbl()` under the hood to obtain the initial `tibble` object. After the inital object is obtained a new column called `date_col` is constructed from the `index` column using `lubridate` if an index column is returned.

Value

A tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
ts_to_tbl(BJsales)
ts_to_tbl(AirPassengers)
```

---

ts_velocity_augment *Augment Function Velocity*

Description

Takes a numeric vector and will return the velocity of that vector.

Usage

```
ts_velocity_augment(.data, .value, .names = "auto")
```

Arguments

.data	The data being passed that will be augmented by the function.
.value	This is passed rlang::enquo() to capture the vectors you want to augment.
.names	The default is "auto"

Details

Takes a numeric vector and will return the velocity of that vector. The velocity of a time series is computed by taking the first difference, so

$$x_t - x_t 1$$

This function is intended to be used on its own in order to add columns to a tibble.

Value

A augmented

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: `ts_acceleration_augment()`, `ts_growth_rate_augment()`

Examples

```
suppressPackageStartupMessages(library(dplyr))

len_out    = 10
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

ts_velocity_augment(data_tbl, b)
```

---

ts_velocity_vec *Vector Function Time Series Acceleration*

Description

Takes a numeric vector and will return the velocity of that vector.

Usage

```
ts_velocity_vec(.x)
```

Arguments

.x	A numeric vector

Details

Takes a numeric vector and will return the velocity of that vector. The velocity of a time series is computed by taking the first difference, so

$$x_t - x_t 1$$

This function can be used on it's own. It is also the basis for the function `ts_velocity_augment()`.

Value

A numeric vector

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Vector Function: ts_acceleration_vec(), ts_growth_rate_vec()

Examples

```
suppressPackageStartupMessages(library(dplyr))

len_out    = 25
by_unit    = "month"
start_date = as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a    = rnorm(len_out),
  b    = runif(len_out)
)

vec_1 <- ts_velocity_vec(data_tbl$b)

plot(data_tbl$b)
lines(data_tbl$b)
lines(vec_1, col = "blue")
```

---

ts_vva_plot *Time Series Value, Velocity and Acceleration Plot*

Description

This function will produce three plots faceted on a single graph. The three graphs are the following:

- Value Plot (Actual values)
- Value Velocity Plot
- Value Acceleration Plot

Usage

```
ts_vva_plot(.data, .date_col, .value_col)
```

Arguments

.data	The data you want to visualize. This should be pre-processed and the aggregation should match the .frequency argument.
.date_col	The data column from the .data argument.
.value_col	The value column from the .data argument

Details

This function expects to take in a data.frame/tibble. It will return a list object that contains the augmented data along with a static plot and an interactive plotly plot. It is important that the data be prepared and have at minimum a date column and the value column as they need to be supplied to the function. If your data is a ts, xts, zoo or mts then use `ts_to_tbl()` to convert it to a tibble.

Value

The original time series augmented with the differenced data, a static plot and a plotly plot of the ggplot object. The output is a list that gets returned invisibly.

Author(s)

Steven P. Sanderson II, MPH

Examples

```
suppressPackageStartupMessages(library(dplyr))

data_tbl <- ts_to_tbl(AirPassengers) %>%
  select(-index)

ts_vva_plot(data_tbl, date_col, value)$plots$static_plot
```

---

ts_wfs_arima_boost   *Auto Arima XGBoost Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_arima_boost(
  .model_type = "all_engines",
  .recipe_list,
  .trees = 10,
  .min_node = 2,
  .tree_depth = 6,
  .learn_rate = 0.015,
  .stop_iter = NULL,
  .seasonal_period = 0,
  .non_seasonal_ar = 0,
  .non_seasonal_differences = 0,
  .non_seasonal_ma = 0,
  .seasonal_ar = 0,
  .seasonal_differences = 0,
```

```
    .seasonal_ma = 0
)
```

Arguments

`.model_type` This is where you will set your engine. It uses `modeltime::arima_boost()` under the hood and can take one of the following:

- "arima_xgboost"
- "auto_arima_xgboost
- "all_engines" - This will make a model spec for all available engines.

`.recipe_list` You must supply a list of recipes. list(rec_1, rec_2, ...)

`.trees` An integer for the number of trees contained in the ensemble.

`.min_node` An integer for the minimum number of data points in a node that is required for the node to be split further.

`.tree_depth` An integer for the maximum depth of the tree (i.e. number of splits) (specific engines only).

`.learn_rate` A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (specific engines only).

`.stop_iter` The number of iterations without improvement before stopping (xgboost only).

`.seasonal_period` Set to 0,

`.non_seasonal_ar` Set to 0,

`.non_seasonal_differences` Set to 0,

`.non_seasonal_ma` Set to 0,

`.seasonal_ar` Set to 0,

`.seasonal_differences` Set to 0,

`.seasonal_ma` Set to 0,

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This uses the option `set_engine("auto_arima_xgboost")` or `set_engine("arima_xgboost")`

`modeltime::arima_boost()` arima_boost() is a way to generate a specification of a time series model that uses boosting to improve modeling errors (residuals) on Exogenous Regressors. It works with both "automated" ARIMA (auto.arima) and standard ARIMA (arima). The main algorithms are:

- Auto ARIMA + XGBoost Errors (engine = auto_arima_xgboost, default)
- ARIMA + XGBoost Errors (engine = arima_xgboost)

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://business-science.github.io/modeltime/reference/arima_boost.html

Other Auto Workflowsets: ts_wfs_auto_arima(), ts_wfs_ets_reg(), ts_wfs_lin_reg(), ts_wfs_mars(), ts_wfs_nnetar_reg(), ts_wfs_prophet_reg(), ts_wfs_svm_poly(), ts_wfs_svm_rbf(), ts_wfs_xgboost()

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_arima_boost("all_engines", rec_objs)
wf_sets
```

---

ts_wfs_auto_arima *Auto Arima (Forecast auto_arima) Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_auto_arima(.model_type = "auto_arima", .recipe_list)
```

Arguments

`.model_type` This is where you will set your engine. It uses `modeltime::arima_reg()` under the hood and can take one of the following:

- "auto_arima"

`.recipe_list` You must supply a list of recipes. list(rec_1, rec_2, ...)

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This only uses the option `set_engine("auto_arima")` and therefore the .model_type is not needed. The parameter is kept because it is possible in the future that this could change, and it keeps with the framework of how other functions are written.

`modeltime::arima_reg()` arima_reg() is a way to generate a specification of an ARIMA model before fitting and allows the model to be created using different packages. Currently the only package is `forecast`.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://business-science.github.io/modeltime/reference/arima_reg.html

Other Auto Workflowsets: `ts_wfs_arima_boost()`, `ts_wfs_ets_reg()`, `ts_wfs_lin_reg()`, `ts_wfs_mars()`, `ts_wfs_nnetar_reg()`, `ts_wfs_prophet_reg()`, `ts_wfs_svm_poly()`, `ts_wfs_svm_rbf()`, `ts_wfs_xgboost()`

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)
```

```
splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_auto_arima("auto_arima", rec_objs)
wf_sets
```

`ts_wfs_ets_reg` *Auto ETS Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_ets_reg(
  .model_type = "all_engines",
  .recipe_list,
  .seasonal_period = "auto",
  .error = "auto",
  .trend = "auto",
  .season = "auto",
  .damping = "auto",
  .smooth_level = 0.1,
  .smooth_trend = 0.1,
  .smooth_seasonal = 0.1
)
```

Arguments

`.model_type` This is where you will set your engine. It uses `modeltime::exp_smoothing()` under the hood and can take one of the following:

- "ets"
- "croston"
- "theta"

- "smooth_es"
- "all_engines" - This will make a model spec for all available engines.

`.recipe_list` You must supply a list of recipes. list(rec_1, rec_2, ...)

`.seasonal_period`
: A seasonal frequency. Uses "auto" by default. A character phrase of "auto" or time-based phrase of "2 weeks" can be used if a date or date-time variable is provided. See Fit Details below.

`.error` The form of the error term: "auto", "additive", or "multiplicative". If the error is multiplicative, the data must be non-negative.

`.trend` The form of the trend term: "auto", "additive", "multiplicative" or0 "none".

`.season` The form of the seasonal term: "auto", "additive", "multiplicative" or "none".

`.damping` Apply damping to a trend: "auto", "damped", or "none".

`.smooth_level` This is often called the "alpha" parameter used as the base level smoothing factor for exponential smoothing models.

`.smooth_trend` This is often called the "beta" parameter used as the trend smoothing factor for exponential smoothing models.

`.smooth_seasonal`
: This is often called the "gamma" parameter used as the seasonal smoothing factor for exponential smoothing models.

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This uses the following engines:

`modeltime::exp_smoothing()` exp_smoothing() is a way to generate a specification of an Exponential Smoothing model before fitting and allows the model to be created using different packages. Currently the only package is forecast. Several algorithms are implemented:

- "ets"
- "croston"
- "theta"
- "smooth_es

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://business-science.github.io/modeltime/reference/exp_smoothing.html

Other Auto Workflowsets: ts_wfs_arima_boost(), ts_wfs_auto_arima(), ts_wfs_lin_reg(), ts_wfs_mars(), ts_wfs_nnetar_reg(), ts_wfs_prophet_reg(), ts_wfs_svm_poly(), ts_wfs_svm_rbf(), ts_wfs_xgboost()

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_ets_reg("all_engines", rec_objs)
wf_sets
```

---

ts_wfs_lin_reg *Auto Linear Regression Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_lin_reg(.model_type, .recipe_list, .penalty = 1, .mixture = 0.5)
```

Arguments

.model_type — This is where you will set your engine. It uses `parsnip::linear_reg()` under the hood and can take one of the following:

- "lm"
- "glmnet"
- "all_engines" - This will make a model spec for all available engines.

Not yet implemented are:

- "stan"
- "spark"
- "keras"

.recipe_list — You must supply a list of recipes. list(rec_1, rec_2, ...)

.penalty — The penalty parameter of the glmnet. The default is 1

.mixture — The mixture parameter of the glmnet. The default is 0.5

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the `glmnet` model specification, but if you choose you can set them yourself if you have a good understanding of what they should be.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/(workflowsets)

Other Auto Workflowsets: `ts_wfs_arima_boost()`, `ts_wfs_auto_arima()`, `ts_wfs_ets_reg()`, `ts_wfs_mars()`, `ts_wfs_nnetar_reg()`, `ts_wfs_prophet_reg()`, `ts_wfs_svm_poly()`, `ts_wfs_svm_rbf()`, `ts_wfs_xgboost()`

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)
```

```
splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_lin_reg("all_engines", rec_objs)
wf_sets
```

---

ts_wfs_mars *Auto MARS (Earth) Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_mars(
  .model_type = "earth",
  .recipe_list,
  .num_terms = 200,
  .prod_degree = 1,
  .prune_method = "backward"
)
```

Arguments

.model_type	This is where you will set your engine. It uses parsnip::mars() under the hood and can take one of the following: • "earth"
.recipe_list	You must supply a list of recipes. list(rec_1, rec_2, ...)
.num_terms	The number of features that will be retained in the final model, including the intercept.
.prod_degree	The highest possible interaction degree.
.prune_method	The pruning method. This is a character, the default is "backward". You can choose from one of the following: • "backward"

- "none"
- "exhaustive"
- "forward"
- "seqrep"
- "cv"

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This only uses the option `set_engine("earth")` and therefore the .model_type is not needed. The parameter is kept because it is possible in the future that this could change, and it keeps with the framework of how other functions are written.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://parsnip.tidymodels.org/reference/mars.html

Other Auto Workflowsets: `ts_wfs_arima_boost()`, `ts_wfs_auto_arima()`, `ts_wfs_ets_reg()`, `ts_wfs_lin_reg()`, `ts_wfs_nnetar_reg()`, `ts_wfs_prophet_reg()`, `ts_wfs_svm_poly()`, `ts_wfs_svm_rbf()`, `ts_wfs_xgboost()`

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
```

```
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_mars("earth", rec_objs)
wf_sets
```

---

ts_wfs_nnetar_reg *Auto NNETAR Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_nnetar_reg(
  .model_type = "nnetar",
  .recipe_list,
  .non_seasonal_ar = 0,
  .seasonal_ar = 0,
  .hidden_units = 5,
  .num_networks = 10,
  .penalty = 0.1,
  .epochs = 10
)
```

Arguments

.model_type	This is where you will set your engine. It uses modeltime::nnetar_reg() under the hood and can take one of the following: • "nnetar"
.recipe_list	You must supply a list of recipes. list(rec_1, rec_2, ...)
.non_seasonal_ar	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
.seasonal_ar	The order of the seasonal auto-regressive (SAR) terms. Often denoted "P" in PDQ-notation.
.hidden_units	An integer for the number of units in the hidden model.
.num_networks	Number of networks to fit with different random starting weights. These are then averaged when producing forecasts.
.penalty	A non-negative numeric value for the amount of weight decay.
.epochs	An integer for the number of training iterations.

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This uses the following engines:

`modeltime::nnetar_reg()` nnetar_reg() is a way to generate a specification of an NNETAR model before fitting and allows the model to be created using different packages. Currently the only package is forecast.

- "nnetar"

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://business-science.github.io/modeltime/reference/nnetar_reg.html

Other Auto Workflowsets: `ts_wfs_arima_boost()`, `ts_wfs_auto_arima()`, `ts_wfs_ets_reg()`, `ts_wfs_lin_reg()`, `ts_wfs_mars()`, `ts_wfs_prophet_reg()`, `ts_wfs_svm_poly()`, `ts_wfs_svm_rbf()`, `ts_wfs_xgboost()`

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
```

```
  , .pred_col = value
)

wf_sets <- ts_wfs_nnetar_reg("nnetar", rec_objs)
wf_sets
```

ts_wfs_prophet_reg *Auto PROPHET Regression Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_prophet_reg(
  .model_type = "all_engines",
  .recipe_list,
  .growth = NULL,
  .changepoint_num = 25,
  .changepoint_range = 0.8,
  .seasonality_yearly = "auto",
  .seasonality_weekly = "auto",
  .seasonality_daily = "auto",
  .season = "additive",
  .prior_scale_changepoints = 25,
  .prior_scale_seasonality = 1,
  .prior_scale_holidays = 1,
  .logistic_cap = NULL,
  .logistic_floor = NULL,
  .trees = 50,
  .min_n = 10,
  .tree_depth = 5,
  .learn_rate = 0.01,
  .loss_reduction = NULL,
  .stop_iter = NULL
)
```

Arguments

.model_type This is where you will set your engine. It uses modeltime::prophet_reg() under the hood and can take one of the following:

- "prophet" Or modeltime::prophet_boost() under the hood and can take one of the following:
- "prophet_xgboost" You can also choose:
- "all_engines" - This will make a model spec for all available engines.

.recipe_list
: You must supply a list of recipes. list(rec_1, rec_2, ...)

.growth
: String 'linear' or 'logistic' to specify a linear or logistic trend.

.changepoint_num
: Number of potential changepoints to include for modeling trend.

.changepoint_range
: Adjusts the flexibility of the trend component by limiting to a percentage of data before the end of the time series. 0.80 means that a changepoint cannot exist after the first 80% of the data.

.seasonality_yearly
: One of "auto", TRUE or FALSE. Set to FALSE for `prophet_xgboost`. Toggles on/off a seasonal component that models year-over-year seasonality.

.seasonality_weekly
: One of "auto", TRUE or FALSE. Toggles on/off a seasonal component that models week-over-week seasonality. Set to FALSE for `prophet_xgboost`

.seasonality_daily
: One of "auto", TRUE or FALSE. Toggles on/off a seasonal componet that models day-over-day seasonality. Set to FALSE for `prophet_xgboost`

.season
: 'additive' (default) or 'multiplicative'.

.prior_scale_changepoints
: Parameter modulating the flexibility of the automatic changepoint selection. Large values will allow many changepoints, small values will allow few changepoints.

.prior_scale_seasonality
: Parameter modulating the strength of the seasonality model. Larger values allow the model to fit larger seasonal fluctuations, smaller values dampen the seasonality.

.prior_scale_holidays
: Parameter modulating the strength of the holiday components model, unless overridden in the holidays input.

.logistic_cap
: When growth is logistic, the upper-bound for "saturation".

.logistic_floor
: When growth is logistic, the lower-bound for "saturation"

.trees
: An integer for the number of trees contained in the ensemble.

.min_n
: An integer for the minimum number of data points in a node that is required for the node to be split further.

.tree_depth
: An integer for the maximum depth of the tree (i.e. number of splits) (specific engines only).

.learn_rate
: A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (specific engines only).

.loss_reduction
: A number for the reduction in the loss function required to split further (specific engines only).

.stop_iter
: The number of iterations without improvement before stopping (xgboost only).

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the `prophet` and `prophet_xgboost` model specification, but if you choose you can set them yourself if you have a good understanding of what they should be.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/(workflowsets)

https://business-science.github.io/modeltime/reference/prophet_reg.html

https://business-science.github.io/modeltime/reference/prophet_boost.html

Other Auto Workflowsets: `ts_wfs_arima_boost()`, `ts_wfs_auto_arima()`, `ts_wfs_ets_reg()`, `ts_wfs_lin_reg()`, `ts_wfs_mars()`, `ts_wfs_nnetar_reg()`, `ts_wfs_svm_poly()`, `ts_wfs_svm_rbf()`, `ts_wfs_xgboost()`

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_prophet_reg("all_engines", rec_objs)
wf_sets
```

ts_wfs_svm_poly *Auto SVM Poly (Kernlab) Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_svm_poly(
  .model_type = "kernlab",
  .recipe_list,
  .cost = 1,
  .degree = 1,
  .scale_factor = 1,
  .margin = 0.1
)
```

Arguments

`.model_type`	This is where you will set your engine. It uses `parsnip::svm_poly()` under the hood and can take one of the following: • "kernlab"
`.recipe_list`	You must supply a list of recipes. list(rec_1, rec_2, ...)
`.cost`	A positive number for the cose of predicting a sample within or on the wrong side of the margin.
`.degree`	A positive number for polynomial degree.
`.scale_factor`	A positive number for the polynomial scaling factor.
`.margin`	A positive number for the epsilon in the SVM insensitive loss function (regression only.)

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This only uses the option `set_engine("kernlab")` and therefore the .model_type is not needed. The parameter is kept because it is possible in the future that this could change, and it keeps with the framework of how other functions are written.

`parsnip::svm_poly()` svm_poly() defines a support vector machine model. For classification, the model tries to maximize the width of the margin between classes. For regression, the model optimizes a robust loss function that is only affected by very large model residuals.

This SVM model uses a nonlinear function, specifically a polynomial function, to create the decision boundary or regression line.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://parsnip.tidymodels.org/reference/svm_poly.html

Other Auto Workflowsets: ts_wfs_arima_boost(), ts_wfs_auto_arima(), ts_wfs_ets_reg(), ts_wfs_lin_reg(), ts_wfs_mars(), ts_wfs_nnetar_reg(), ts_wfs_prophet_reg(), ts_wfs_svm_rbf(), ts_wfs_xgboost()

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_svm_poly("kernlab", rec_objs)
wf_sets
```

ts_wfs_svm_rbf — *Auto SVM RBF (Kernlab) Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_svm_rbf(
  .model_type = "kernlab",
  .recipe_list,
  .cost = 1,
  .rbf_sigma = 0.01,
  .margin = 0.1
)
```

Arguments

`.model_type`	This is where you will set your engine. It uses `parsnip::svm_rbf()` under the hood and can take one of the following: • "kernlab"
`.recipe_list`	You must supply a list of recipes. list(rec_1, rec_2, ...)
`.cost`	A positive number for the cost of predicting a sample within or on the wrong side of the margin.
`.rbf_sigma`	A positive number for the radial basis function.
`.margin`	A positive number for the epsilon in the SVM insensitive loss function (regression only).

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This only uses the option `set_engine("kernlab")` and therefore the .model_type is not needed. The parameter is kept because it is possible in the future that this could change, and it keeps with the framework of how other functions are written.

`parsnip::svm_rbf()` svm_rbf() defines a support vector machine model. For classification, the model tries to maximize the width of the margin between classes. For regression, the model optimizes a robust loss function that is only affected by very large model residuals.

This SVM model uses a nonlinear function, specifically a polynomial function, to create the decision boundary or regression line.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://parsnip.tidymodels.org/reference/svm_rbf.html

Other Auto Workflowsets: ts_wfs_arima_boost(), ts_wfs_auto_arima(), ts_wfs_ets_reg(), ts_wfs_lin_reg(), ts_wfs_mars(), ts_wfs_nnetar_reg(), ts_wfs_prophet_reg(), ts_wfs_svm_poly(), ts_wfs_xgboost()

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_svm_rbf("kernlab", rec_objs)
wf_sets
```

ts_wfs_xgboost *Auto XGBoost (XGBoost) Workflowset Function*

Description

This function is used to quickly create a workflowsets object.

Usage

```
ts_wfs_xgboost(
  .model_type = "xgboost",
  .recipe_list,
  .trees = 15L,
  .min_n = 1L,
  .tree_depth = 6L,
  .learn_rate = 0.3,
  .loss_reduction = 0,
  .sample_size = 1,
  .stop_iter = Inf
)
```

Arguments

`.model_type` This is where you will set your engine. It uses parsnip::boost_tree under the hood and can take one of the following:

- "xgboost"

`.recipe_list` You must supply a list of recipes. list(rec_1, rec_2, ...)

`.trees` The number of trees (type: integer, default: 15L)

`.min_n` Minimal Node Size (type: integer, default: 1L)

`.tree_depth` Tree Depth (type: integer, default: 6L)

`.learn_rate` Learning Rate (type: double, default: 0.3)

`.loss_reduction` Minimum Loss Reduction (type: double, default: 0.0)

`.sample_size` Proportion Observations Sampled (type: double, default: 1.0)

`.stop_iter` The number of ierations Before Stopping (type: integer, default: Inf)

Details

This function expects to take in the recipes that you want to use in the modeling process. This is an automated workflow process. There are sensible defaults set for the model specification, but if you choose you can set them yourself if you have a good understanding of what they should be. The mode is set to "regression".

This only uses the option `set_engine("xgboost")` and therefore the .model_type is not needed. The parameter is kept because it is possible in the future that this could change, and it keeps with the framework of how other functions are written.

parsnip::boost_tree() xgboost::xgb.train() creates a series of decision trees forming an ensemble. Each tree depends on the results of previous trees. All trees in the ensemble are combined to produce a final prediction.

Value

Returns a workflowsets object.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://workflowsets.tidymodels.org/

https://parsnip.tidymodels.org/reference/details_boost_tree_xgboost.html

https://arxiv.org/abs/1603.02754

Other Auto Workflowsets: ts_wfs_arima_boost(), ts_wfs_auto_arima(), ts_wfs_ets_reg(), ts_wfs_lin_reg(), ts_wfs_mars(), ts_wfs_nnetar_reg(), ts_wfs_prophet_reg(), ts_wfs_svm_poly(), ts_wfs_svm_rbf()

Examples

```
suppressPackageStartupMessages(library(modeltime))
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(rsample))

data <- AirPassengers %>%
  ts_to_tbl() %>%
  select(-index)

splits <- time_series_split(
   data
  , date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)

rec_objs <- ts_auto_recipe(
 .data = training(splits)
 , .date_col = date_col
 , .pred_col = value
)

wf_sets <- ts_wfs_xgboost("xgboost", rec_objs)
wf_sets
```

util_difflog_ts *Differencing with Log Transformation to Make Time Series Stationary*

Description

This function attempts to make a non-stationary time series stationary by applying differencing with a logarithmic transformation. It iteratively increases the differencing order until stationarity is achieved or informs the user if the transformation is not possible.

Usage

```
util_difflog_ts(.time_series)
```

Arguments

`.time_series` A time series object to be made stationary.

Details

The function calculates the frequency of the input time series using the `stats::frequency` function and checks if the minimum value of the time series is greater than 0. It then applies differencing with a logarithmic transformation incrementally until the Augmented Dickey-Fuller test indicates stationarity (p-value < 0.05) or until the differencing order reaches the frequency of the data.

If differencing with a logarithmic transformation successfully makes the time series stationary, it returns the stationary time series and related information as a list with the following elements:

- stationary_ts: The stationary time series after the transformation.
- ndiffs: The order of differencing applied to make it stationary.
- adf_stats: Augmented Dickey-Fuller test statistics on the stationary time series.
- trans_type: Transformation type, which is "diff_log" in this case.
- ret: TRUE to indicate a successful transformation.

If the data either had a minimum value less than or equal to 0 or requires more differencing than its frequency allows, it informs the user and suggests trying double differencing with a logarithmic transformation.

Value

If the time series is already stationary or the differencing with a logarithmic transformation is successful,

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
# Example 1: Using a time series dataset
util_difflog_ts(AirPassengers)

# Example 2: Using a different time series dataset
util_difflog_ts(BJsales)$ret
```

---

util_doubledifflog_ts *Double Differencing with Log Transformation to Make Time Series Stationary*

Description

This function attempts to make a non-stationary time series stationary by applying double differencing with a logarithmic transformation. It iteratively increases the differencing order until stationarity is achieved or informs the user if the transformation is not possible.

Usage

```
util_doubledifflog_ts(.time_series)
```

Arguments

.time_series	A time series object to be made stationary.

Details

The function calculates the frequency of the input time series using the `stats::frequency` function and checks if the minimum value of the time series is greater than 0. It then applies double differencing with a logarithmic transformation incrementally until the Augmented Dickey-Fuller test indicates stationarity (p-value < 0.05) or until the differencing order reaches the frequency of the data.

If double differencing with a logarithmic transformation successfully makes the time series stationary, it returns the stationary time series and related information as a list with the following elements:

- stationary_ts: The stationary time series after the transformation.
- ndiffs: The order of differencing applied to make it stationary.

- adf_stats: Augmented Dickey-Fuller test statistics on the stationary time series.
- trans_type: Transformation type, which is "double_diff_log" in this case.
- ret: TRUE to indicate a successful transformation.

If the data either had a minimum value less than or equal to 0 or requires more differencing than its frequency allows, it informs the user that the data could not be stationarized.

Value

If the time series is already stationary or the double differencing with a logarithmic transformation is successful, it returns a list as described in the details section. If the transformation is not possible, it informs the user and returns a list with ret set to FALSE, indicating that the data could not be stationarized.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_log_ts(), util_singlediff_ts()

Examples

```
# Example 1: Using a time series dataset
util_doubledifflog_ts(AirPassengers)

# Example 2: Using a different time series dataset
util_doubledifflog_ts(BJsales)$ret
```

---

util_doublediff_ts *Double Differencing to Make Time Series Stationary*

Description

This function attempts to make a non-stationary time series stationary by applying double differencing. It iteratively increases the differencing order until stationarity is achieved.

Usage

```
util_doublediff_ts(.time_series)
```

Arguments

`.time_series` A time series object to be made stationary.

Details

The function calculates the frequency of the input time series using the `stats::frequency` function. It then applies double differencing incrementally until the Augmented Dickey-Fuller test indicates stationarity (p-value < 0.05) or until the differencing order reaches the frequency of the data.

If double differencing successfully makes the time series stationary, it returns the stationary time series and related information as a list with the following elements:

- stationary_ts: The stationary time series after double differencing.
- ndiffs: The order of differencing applied to make it stationary.
- adf_stats: Augmented Dickey-Fuller test statistics on the stationary time series.
- trans_type: Transformation type, which is "double_diff" in this case.
- ret: TRUE to indicate a successful transformation.

If the data requires more double differencing than its frequency allows, it informs the user and suggests trying differencing with the natural logarithm instead.

Value

If the time series is already stationary or the double differencing is successful, it returns a list as described in the details section. If additional differencing is required, it informs the user and returns a list with ret set to FALSE, suggesting trying differencing with the natural logarithm.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: `auto_stationarize()`, `calibrate_and_plot()`, `internal_ts_backward_event_tbl()`, `internal_ts_both_event_tbl()`, `internal_ts_forward_event_tbl()`, `model_extraction_helper()`, `ts_get_date_columns()`, `ts_info_tbl()`, `ts_is_date_class()`, `ts_lag_correlation()`, `ts_model_auto_tune()`, `ts_model_compare()`, `ts_model_rank_tbl()`, `ts_model_spec_tune_template()`, `ts_qq_plot()`, `ts_scedacity_scatter_plot()`, `ts_to_tbl()`, `util_difflog_ts()`, `util_doubledifflog_ts()`, `util_log_ts()`, `util_singlediff_ts()`

Examples

```
# Example 1: Using a time series dataset
util_doublediff_ts(AirPassengers)

# Example 2: Using a different time series dataset
util_doublediff_ts(BJsales)$ret
```

util_log_ts — *Logarithmic Transformation to Make Time Series Stationary*

Description

This function attempts to make a non-stationary time series stationary by applying a logarithmic transformation. If successful, it returns the stationary time series. If the transformation fails, it informs the user.

Usage

```
util_log_ts(.time_series)
```

Arguments

`.time_series`	A time series object to be made stationary.

Details

This function checks if the minimum value of the input time series is greater than or equal to zero. If yes, it performs the Augmented Dickey-Fuller test on the logarithm of the time series. If the p-value of the test is less than 0.05, it concludes that the logarithmic transformation made the time series stationary and returns the result as a list with the following elements:

- stationary_ts: The stationary time series after the logarithmic transformation.
- ndiffs: Not applicable in this case, marked as NA.
- adf_stats: Augmented Dickey-Fuller test statistics on the stationary time series.
- trans_type: Transformation type, which is "log" in this case.
- ret: TRUE to indicate a successful transformation.

If the minimum value of the time series is less than or equal to 0 or if the logarithmic transformation doesn't make the time series stationary, it informs the user and returns a list with ret set to FALSE.

Value

If the time series is already stationary or the logarithmic transformation is successful, it returns a list as described in the details section. If the transformation fails, it returns a list with ret set to FALSE.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_singlediff_ts()

Examples

```
# Example 1: Using a time series dataset
util_log_ts(AirPassengers)

# Example 2: Using a different time series dataset
util_log_ts(BJsales.lead)$ret
```

---

util_singlediff_ts *Single Differencing to Make Time Series Stationary*

Description

This function attempts to make a non-stationary time series stationary by applying single differencing. It iteratively increases the differencing order until stationarity is achieved.

Usage

```
util_singlediff_ts(.time_series)
```

Arguments

.time_series	A time series object to be made stationary.

Details

The function calculates the frequency of the input time series using the `stats::frequency` function. It then applies single differencing incrementally until the Augmented Dickey-Fuller test indicates stationarity (p-value < 0.05) or until the differencing order reaches the frequency of the data.

If single differencing successfully makes the time series stationary, it returns the stationary time series and related information as a list with the following elements:

- stationary_ts: The stationary time series after differencing.
- ndiffs: The order of differencing applied to make it stationary.
- adf_stats: Augmented Dickey-Fuller test statistics on the stationary time series.
- trans_type: Transformation type, which is "diff" in this case.

- ret: TRUE to indicate a successful transformation.

If the data requires more single differencing than its frequency allows, it informs the user and returns a list with ret set to FALSE, indicating that double differencing may be needed.

Value

If the time series is already stationary or the single differencing is successful, it returns a list as described in the details section. If additional differencing is required, it informs the user and returns a list with ret set to FALSE.

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Utility: auto_stationarize(), calibrate_and_plot(), internal_ts_backward_event_tbl(), internal_ts_both_event_tbl(), internal_ts_forward_event_tbl(), model_extraction_helper(), ts_get_date_columns(), ts_info_tbl(), ts_is_date_class(), ts_lag_correlation(), ts_model_auto_tune(), ts_model_compare(), ts_model_rank_tbl(), ts_model_spec_tune_template(), ts_qq_plot(), ts_scedacity_scatter_plot(), ts_to_tbl(), util_difflog_ts(), util_doublediff_ts(), util_doubledifflog_ts(), util_log_ts()

Examples

```
# Example 1: Using a time series dataset
util_singlediff_ts(AirPassengers)

# Example 2: Using a different time series dataset
util_singlediff_ts(BJsales)$ret
```

Index