

# Package ‘healthyR.ts’

August 22, 2023

**Title** The Time Series Modeling Companion to 'healthyR'

**Version** 0.2.10

**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 ( $\geq 0.1.2$ ), 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 ( $\geq 3.3$ )

**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** 2023-08-22 07:30:02 UTC

**R topics documented:**

calibrate_and_plot . . . . .	3
ci_hi . . . . .	5
ci_lo . . . . .	6
color_blind . . . . .	7
internal_ts_backward_event_tbl . . . . .	7
internal_ts_both_event_tbl . . . . .	8
internal_ts_forward_event_tbl . . . . .	9
model_extraction_helper . . . . .	10
step_ts_acceleration . . . . .	11
step_ts_velocity . . . . .	13
tidy_fft . . . . .	15
ts_acceleration_augment . . . . .	17
ts_acceleration_vec . . . . .	18
ts_arma_simulator . . . . .	19
ts_auto_arma . . . . .	21
ts_auto_arma_xgboost . . . . .	23
ts_auto_croston . . . . .	25
ts_auto_exp_smoothing . . . . .	27
ts_auto_glmnet . . . . .	30
ts_auto_lm . . . . .	32
ts_auto_mars . . . . .	34
ts_auto_nnetar . . . . .	36
ts_auto_prophet_boost . . . . .	38
ts_auto_prophet_reg . . . . .	40
ts_auto_recipe . . . . .	42
ts_auto_smooth_es . . . . .	44
ts_auto_svm_poly . . . . .	47
ts_auto_svm_rbf . . . . .	49
ts_auto_theta . . . . .	51
ts_auto_xgboost . . . . .	53
ts_brownian_motion . . . . .	55
ts_brownian_motion_augment . . . . .	57
ts_brownian_motion_plot . . . . .	58
ts_calendar_heatmap_plot . . . . .	59
ts_compare_data . . . . .	61
ts_event_analysis_plot . . . . .	62
ts_extract_auto_fitted_workflow . . . . .	64
ts_feature_cluster . . . . .	65
ts_feature_cluster_plot . . . . .	67
ts_forecast_simulator . . . . .	69
ts_geometric_brownian_motion . . . . .	71
ts_geometric_brownian_motion_augment . . . . .	72
ts_get_date_columns . . . . .	74
ts_info_tbl . . . . .	75
ts_is_date_class . . . . .	76
ts_lag_correlation . . . . .	77

ts_ma_plot . . . . .	79
ts_model_auto_tune . . . . .	80
ts_model_compare . . . . .	84
ts_model_rank_tbl . . . . .	86
ts_model_spec_tune_template . . . . .	88
ts_qc_run_chart . . . . .	90
ts_qq_plot . . . . .	91
ts_random_walk . . . . .	93
ts_random_walk_ggplot_layers . . . . .	95
ts_scale_color_colorblind . . . . .	96
ts_scale_fill_colorblind . . . . .	96
ts_scedacity_scatter_plot . . . . .	97
ts_sma_plot . . . . .	99
ts_splits_plot . . . . .	100
ts_time_event_analysis_tbl . . . . .	101
ts_to_tbl . . . . .	103
ts_velocity_augment . . . . .	104
ts_velocity_vec . . . . .	105
ts_vva_plot . . . . .	106
ts_wfs_arima_boost . . . . .	107
ts_wfs_auto_arima . . . . .	110
ts_wfs_ets_reg . . . . .	111
ts_wfs_lin_reg . . . . .	114
ts_wfs_mars . . . . .	115
ts_wfs_nnetar_reg . . . . .	117
ts_wfs_prophet_reg . . . . .	119
ts_wfs_svm_poly . . . . .	122
ts_wfs_svm_rbf . . . . .	124
ts_wfs_xgboost . . . . .	126

**Index****129**


---

calibrate_and_plot	<i>Helper function - Calibrate and Plot</i>
--------------------	---

---

**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: [internal\\_ts\\_backward\\_event\\_tbl\(\)](#), [internal\\_ts\\_both\\_event\\_tbl\(\)](#), [internal\\_ts\\_forward\\_event\\_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\(\)](#)

### 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\(\)](#)

**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\(\)](#)

**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: [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\(\)](#)

---

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: [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\(\)](#)

---

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**

<code>.data</code>	The date.frame/tibble that holds the data.
<code>.horizon</code>	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: [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\(\)](#)

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: [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\(\)](#)

**Examples**

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

# Create a model
fit_arma <- auto.arma(AirPassengers)

model_extraction_helper(fit_arma)

## End(Not run)
```

---

step\_ts\_acceleration *Recipes Time Series Acceleration Generator*

---

**Description**

step\_ts\_acceleration creates 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 <code>numeric</code>
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 <code>recipes::prep()</code> is used.

skip	A logical. Should the step be skipped when the recipe is baked by <code>bake.recipe()</code> ? While all operations are baked when <code>prep.recipe()</code> 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 <code>skip = TRUE</code> 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(prepare(rec_obj), data_tbl)

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

---

step\_ts\_velocity      *Recipes Time Series velocity Generator*

---

### Description

step\_ts\_velocity creates 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 <code>numeric</code>
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 <code>recipes::prep()</code> is used.
skip	A logical. Should the step be skipped when the recipe is baked by <code>bake.recipe()</code> ? While all operations are baked when <code>prep.recipe()</code> 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 <code>skip = TRUE</code> 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**

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

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

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\\_augment\(\)](#), [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

<code>.data</code>	The data being passed that will be augmented by the function.
<code>.value</code>	This is passed <code>rlang::enquo()</code> to capture the vectors you want to augment.
<code>.names</code>	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\\_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\\_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\_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_arma_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**

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

**Details**

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

**Value**

A list object.

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

<https://www.machinelearningplus.com/time-series/arma-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**

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object

<code>.prefix</code>	Default is <code>ts_arima</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <code>timetk::time_series_cv()</code>
<code>.cv_skip</code>	How many observations to skip. See <code>timetk::time_series_cv()</code>
<code>.cv_slice_limit</code>	How many slices to return. See <code>timetk::time_series_cv()</code>
<code>.best_metric</code>	Default is "rmse". See <code>modeltime::default_forecast_accuracy_metric_set()</code>
<code>.bootstrap_final</code>	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](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_auto_arma <- ts_auto_arma(
  .data = data,
  .num_cores = 2,
  .date_col = date_col,
  .value_col = value,
  .rsamp_obj = splits,
  .formula = value ~ .,
  .grid_size = 5,
  .cv_slice_limit = 2
)

ts_auto_arma$recipe_info
```

---

ts\_auto\_arma\_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_arma_xgboost(
  .data,
  .date_col,
  .value_col,
  .formula,
  .rsamp_obj,
  .prefix = "ts_arma_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**

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

**Details**

This uses the `modeltime::arma_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/arma\\_boost.html](https://business-science.github.io/modeltime/reference/arma_boost.html)

Other Boiler\_Plate: `ts_auto_arma()`, `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
)

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

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_exp_smooth</code>
<code>.tune</code>	Defaults to TRUE, this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is TRUE then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_skip</code>	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_slice_limit</code>	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
<code>.best_metric</code>	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
<code>.bootstrap_final</code>	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://business-science.github.io/modeltime/reference/exp_smoothing.html#engine-details)

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

Other Boiler\_Plate: `ts_auto_arma_xgboost()`, `ts_auto_arma()`, `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
)

ts_exp$recipe_info
```

**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**

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like <code>value ~ .</code>
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_exp_smooth</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_skip</code>	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_slice_limit</code>	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
<code>.best_metric</code>	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
<code>.bootstrap_final</code>	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://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
)

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.

<code>.grid_size</code>	If <code>.tune</code> is TRUE then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <code>timetk::time_series_cv()</code>
<code>.cv_skip</code>	How many observations to skip. See <code>timetk::time_series_cv()</code>
<code>.cv_slice_limit</code>	How many slices to return. See <code>timetk::time_series_cv()</code>
<code>.best_metric</code>	Default is "rmse". See <code>modeltime::default_forecast_accuracy_metric_set()</code>
<code>.bootstrap_final</code>	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](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
)

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](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**

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_mars</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	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_arma_xgboost()`, `ts_auto_arma()`, `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  
)  
  
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 <code>timetk::time_series_cv()</code>
.cv_skip	How many observations to skip. See <code>timetk::time_series_cv()</code>
.cv_slice_limit	How many slices to return. See <code>timetk::time_series_cv()</code>
.best_metric	Default is "rmse". See <code>modeltime::default_forecast_accuracy_metric_set()</code>
.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](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
)

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 <a href="#">timetk::time_series_cv()</a>
.cv_skip	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
.cv_slice_limit	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
.best_metric	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
.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](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
)

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**

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like <code>value ~ .</code>
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_prophet</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_skip</code>	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_slice_limit</code>	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
<code>.best_metric</code>	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
<code>.bootstrap_final</code>	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](https://business-science.github.io/modeltime/reference/prophet_reg.html)

Other Boiler\_Plate: `ts_auto_arma_xgboost()`, `ts_auto_arma()`, `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
)

ts_prophet_reg$recipe_info
```

**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 <code>timetk::step_timeseries_signature()</code> 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. <ul style="list-style-type: none"> <li>• iso\$</li> <li>• xts\$</li> <li>• hour</li> <li>• min</li> <li>• sec</li> <li>• am.pm</li> </ul> |
| .step_ts_dummy          | A Boolean indicating if <code>all_nominal_predictors()</code> should be dummied and with one hot encoding.  |
| .step_ts_fourier        | A Boolean indicating if <code>timetk::step_fourier()</code> should be added to the recipe.  |
| .step_ts_fourier_period | A number such as 365/12, 365/4 or 365 indicating 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

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like <code>value ~ .</code>
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_smooth_es</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_skip</code>	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_slice_limit</code>	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
<code>.best_metric</code>	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
<code>.bootstrap_final</code>	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://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
)

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

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_smooth_es</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	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](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  
)  
  
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**

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_smooth_es</code>
<code>.tune</code>	Defaults to TRUE, this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is TRUE then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <code>timetk::time_series_cv()</code>
<code>.cv_skip</code>	How many observations to skip. See <code>timetk::time_series_cv()</code>
<code>.cv_slice_limit</code>	How many slices to return. See <code>timetk::time_series_cv()</code>
<code>.best_metric</code>	Default is "rmse". See <code>modeltime::default_forecast_accuracy_metric_set()</code>
<code>.bootstrap_final</code>	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](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
)

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://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

<code>.data</code>	The data being passed to the function. The time-series object.
<code>.date_col</code>	The column that holds the datetime.
<code>.value_col</code>	The column that has the value
<code>.formula</code>	The formula that is passed to the recipe like value ~ .
<code>.rsamp_obj</code>	The rsample splits object
<code>.prefix</code>	Default is <code>ts_xgboost</code>
<code>.tune</code>	Defaults to <code>TRUE</code> , this creates a tuning grid and tuned model.
<code>.grid_size</code>	If <code>.tune</code> is <code>TRUE</code> then the <code>.grid_size</code> is the size of the tuning grid.
<code>.num_cores</code>	How many cores do you want to use. Default is 1
<code>.cv_assess</code>	How many observations for assess. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_skip</code>	How many observations to skip. See <a href="#">timetk::time_series_cv()</a>
<code>.cv_slice_limit</code>	How many slices to return. See <a href="#">timetk::time_series_cv()</a>
<code>.best_metric</code>	Default is "rmse". See <a href="#">modeltime::default_forecast_accuracy_metric_set()</a>
<code>.bootstrap_final</code>	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
)

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

<code>.time</code>	Total time of the simulation.
<code>.num_sims</code>	Total number of simulations.
<code>.delta_time</code>	Time step size.
<code>.initial_value</code>	Integer representing the initial value.
<code>.return_tibble</code>	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**

<code>.data</code>	The data you are going to pass to the function to augment.
<code>.date_col</code>	The column that holds the date
<code>.value_col</code>	The column that holds the value
<code>.interactive</code>	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_augment()` 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

<code>.data</code>	The time-series data with a date column and value column.
<code>.date_col</code>	The column that has the datetime values
<code>.value_col</code>	The column that has the values
<code>.low</code>	The color for the low value, must be quoted like "red". The default is "red"
<code>.high</code>	The color for the high value, must be quoted like "green". The default is "green"
<code>.plt_title</code>	The title of the plot
<code>.interactive</code>	Default is TRUE to get an interactive plot using <code>plotly::ggplotly()</code> . 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	<i>Compare data over time periods</i>
-----------------	---------------------------------------

---

### 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

<code>.data</code>	The date.frame/tibble that holds the data
<code>.date_col</code>	The column with the date value
<code>.start_date</code>	The start of the period you want to analyze
<code>.end_date</code>	The end of the period you want to analyze
<code>.periods_back</code>	How long ago do you want to compare data too. Time units are collapsed using <code>lubridate::floor_date()</code> . The value can be: <ul style="list-style-type: none"><li>• second</li><li>• minute</li><li>• hour</li><li>• day</li><li>• week</li><li>• month</li><li>• bimonth</li><li>• quarter</li><li>• season</li><li>• halfyear</li><li>• year</li></ul>

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

<code>.data</code>	The data that comes from the <code>ts_time_event_analysis_tbl()</code>
<code>.plot_type</code>	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.
<code>.plot_ci</code>	The default is TRUE. This will only work if you choose one of the aggregate plots of either "mean" or "median"
<code>.interactive</code>	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

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

So you can make a function as follows

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

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 features 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

<code>.model</code>	A forecasting model of one of the following from the forecast package: <ul style="list-style-type: none"><li>• <a href="#">Arima</a></li><li>• <a href="#">auto.arima</a></li><li>• <a href="#">ets</a></li><li>• <a href="#">nnetar</a></li><li>• <code>Arima()</code> with <code>xreg</code></li></ul>
<code>.data</code>	The data that is used for the <code>.model</code> parameter. This is used with <code>timetk::tk_index()</code>
<code>.ext_reg</code>	A tibble or matrix of future xregs that should be the same length as the horizon you want to forecast.
<code>.frequency</code>	This is for the conversion of an internal table and should match the time frequency of the data.

<code>.bootstrap</code>	A boolean value of TRUE/FALSE. From <code>forecast::simulate.Arima()</code> Do simulation using resampled errors rather than normally distributed errors.
<code>.horizon</code>	An integer defining the forecast horizon.
<code>.iterations</code>	An integer, set the number of iterations of the simulation.
<code>.sim_color</code>	Set the color of the simulation paths lines.
<code>.alpha</code>	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

<code>.num_sims</code>	Total number of simulations.
<code>.time</code>	Total time of the simulation.
<code>.mean</code>	Expected return
<code>.sigma</code>	Volatility
<code>.initial_value</code>	Integer representing the initial value.
<code>.delta_time</code>	Time step size.
<code>.return_tibble</code>	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:

$$dS/S = \mu dt + \sigma dW$$

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:

$$dS/S = \mu dt + \sigma dW$$

where  $S$  is the price of the asset,  $t$  is time,  $\mu$  is the expected return on the asset,  $\sigma$  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: [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\(\)](#)

**Examples**

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

---

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: [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\(\)](#)

**Examples**

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

---

ts_is_date_class	<i>Check if an object is a date class</i>
------------------	---

---

**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: [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\(\)](#)

## 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**

<code>.data</code>	A tibble of time series data
<code>.date_col</code>	A date column
<code>.value_col</code>	The value column being analyzed
<code>.lags</code>	This is a vector of integer lags, ie 1 or <code>c(1,6,12)</code>
<code>.heatmap_color_low</code>	What color should the low values of the heatmap of the correlation matrix be, the default is 'white'
<code>.heatmap_color_hi</code>	What color should the high 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: [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\(\)](#)

**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

<code>.data</code>	The data you want to visualize. This should be pre-processed and the aggregation should match the <code>.frequency</code> argument.
<code>.date_col</code>	The data column from the <code>.data</code> argument.
<code>.value_col</code>	The value column from the <code>.data</code> argument
<code>.ts_frequency</code>	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.
<code>.main_title</code>	The title of the main plot.
<code>.secondary_title</code>	The title of the second plot.
<code>.tertiary_title</code>	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

<code>.modeltime_model_id</code>	The <code>.model_id</code> from a calibrated <code>modeltime</code> table.
<code>.calibration_tbl</code>	A calibrated <code>modeltime</code> table.
<code>.splits_obj</code>	The <code>time_series_split</code> object.
<code>.drop_training_na</code>	A boolean that will drop NA values from the training(splits) data
<code>.date_col</code>	The column that holds the date values.
<code>.value_col</code>	The column that holds the time series values.
<code>.tscv_assess</code>	A character expression like "12 months". This gets passed to <code>timetk::time_series_cv()</code>
<code>.tscv_skip</code>	A character expression like "6 months". This gets passed to <code>timetk::time_series_cv()</code>
<code>.slice_limit</code>	An integer that gets passed to <code>timetk::time_series_cv()</code>
<code>.facet_ncol</code>	The number of faceted columns to be passed to <code>plot_time_series_cv_plan</code>
<code>.grid_size</code>	An integer that gets passed to the <code>dials::grid_latin_hypercube()</code> function.
<code>.num_cores</code>	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.
<code>.best_metric</code>	The default is "rmse" and this can be set to any default <code>dials</code> 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: [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\(\)](#)

**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**

<code>.model_1</code>	The model being compared to the base, this can also be a hyperparameter tuned model.
<code>.model_2</code>	The base model.
<code>.type</code>	The default is the testing tibble, can be set to training as well.
<code>.splits_obj</code>	The splits object
<code>.data</code>	The original data that was passed to splits
<code>.print_info</code>	This is a boolean, the default is TRUE
<code>.metric</code>	This should be one of the following character strings: <ul style="list-style-type: none"> <li>• "mae"</li> <li>• "mape"</li> <li>• "mase"</li> <li>• "smape"</li> <li>• "rmse"</li> <li>• "rsq"</li> </ul>

**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 original 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 plot 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: [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\(\)](#)

**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)

```

**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: [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\(\)](#)

**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: [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\(\)](#)

**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**

<code>.data</code>	The data.frame/tibble to be passed.
<code>.date_col</code>	The column holding the timestamp.
<code>.value_col</code>	The column with the values to be analyzed.
<code>.interactive</code>	Default is FALSE, TRUE for an interactive plotly plot.

<code>.median</code>	Default is TRUE. This will show the median line of the data.
<code>.c1</code>	This is the first upper control line
<code>.m1</code>	This is the second sigma control line positive
<code>.u1</code>	This is the third sigma control line positive
<code>.l1</code>	This is the first negative control line
<code>.lm1</code>	This is the second sigma negative control line
<code>.ll1</code>	This is the third 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
    , .ll1 = 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**

<code>.calibration_tbl</code>	A calibrated modeltime table.
<code>.model_id</code>	The id of a particular model from a calibration tibble. If there are multiple models in the tibble and this remains <b>NULL</b> then the plot will be returned using <code>ggplot2::facet_grid(~ .model_id)</code>
<code>.interactive</code>	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%93plot>

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

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_scedacity_scatter_plot()`, `ts_to_tbl()`

**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

<code>.mean</code>	The desired mean of the random walks
<code>.sd</code>	The standard deviation of the random walks
<code>.num_walks</code>	The number of random walks you want generated
<code>.periods</code>	The length of the random walk(s) you want generated
<code>.initial_value</code>	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 ggplot 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 ggplot 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**

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

**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**

<code>.splits_obj</code>	The predefined splits object.
<code>.date_col</code>	The date column for the time series.
<code>.value_col</code>	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/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(tk_time_sers_cv_plan))
- [https://business-science.github.io/timetk/reference/plot\\_time\\_series\\_cv\\_plan.html\(plot\\_time\\_series\\_cv\\_plan\)](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

<code>.data</code>	The <code>data.frame/tibble</code> that holds the data.
<code>.date_col</code>	The column with the date value.
<code>.value_col</code>	The column with the value you are measuring.
<code>.percent_change</code>	This defaults to 0.05 which is a 5% increase in the <code>.value_col</code> .
<code>.horizon</code>	How far do you want to look back or ahead.
<code>.precision</code>	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.
<code>.direction</code>	The default is forward. You can supply either forward, backwards or both.
<code>.filter_non_event_groups</code>	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 initial 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: `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()`

**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**

<code>.data</code>	The data being passed that will be augmented by the function.
<code>.value</code>	This is passed <code>rlang::enquo()</code> to capture the vectors you want to augment.
<code>.names</code>	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 tibble

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Augment Function: [ts\\_acceleration\\_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	<i>Vector Function Time Series Acceleration</i>
-----------------	---

---

**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\(\)](#)

**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

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

<code>.model_type</code>	This is where you will set your engine. It uses <code>modeltime::arima_boost()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>"arima_xgboost"</li> <li>"auto_arima_xgboost"</li> <li>"all_engines" - This will make a model spec for all available engines.</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.trees</code>	An integer for the number of trees contained in the ensemble.
<code>.min_node</code>	An integer for the minimum number of data points in a node that is required for the node to be split further.
<code>.tree_depth</code>	An integer for the maximum depth of the tree (i.e. number of splits) (specific engines only).
<code>.learn_rate</code>	A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (specific engines only).
<code>.stop_iter</code>	The number of iterations without improvement before stopping (xgboost only).
<code>.seasonal_period</code>	Set to 0,
<code>.non_seasonal_ar</code>	Set to 0,
<code>.non_seasonal_differences</code>	Set to 0,
<code>.non_seasonal_ma</code>	Set to 0,
<code>.seasonal_ar</code>	Set to 0,
<code>.seasonal_differences</code>	Set to 0,
<code>.seasonal_ma</code>	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_arma_xgboost")` or `set_engine("arma_xgboost")` `modeltime::arma_boost()` `arma_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.arma`) and standard ARIMA (`arma`). The main algorithms are:

- Auto ARIMA + XGBoost Errors (`engine = auto_arma_xgboost`, default)
- ARIMA + XGBoost Errors (`engine = arma_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/arma\\_boost.html](https://business-science.github.io/modeltime/reference/arma_boost.html)

Other Auto Workflowsets: `ts_wfs_auto_arma()`, `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/arma\\_reg.html](https://business-science.github.io/modeltime/reference/arma_reg.html)

Other Auto Workflowsets: `ts_wfs_arma_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_arma("auto_arma", 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

<code>.model_type</code>	This is where you will set your engine. It uses <code>modeltime::exp_smoothing()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>• "ets"</li> <li>• "croston"</li> <li>• "theta"</li> <li>• "smooth_es"</li> <li>• "all_engines" - This will make a model spec for all available engines.</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.seasonal_period</code>	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.
<code>.error</code>	The form of the error term: "auto", "additive", or "multiplicative". If the error is multiplicative, the data must be non-negative.
<code>.trend</code>	The form of the trend term: "auto", "additive", "multiplicative" or "none".
<code>.season</code>	The form of the seasonal term: "auto", "additive", "multiplicative" or "none".
<code>.damping</code>	Apply damping to a trend: "auto", "damped", or "none".
<code>.smooth_level</code>	This is often called the "alpha" parameter used as the base level smoothing factor for exponential smoothing models.
<code>.smooth_trend</code>	This is often called the "beta" parameter used as the trend smoothing factor for exponential smoothing models.
<code>.smooth_seasonal</code>	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](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	<i>Auto Linear Regression Workflowset Function</i>
----------------	--

---

**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 <code>parsnip::linear_reg()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>• "lm"</li> <li>• "glmnet"</li> <li>• "all_engines" - This will make a model spec for all available engines.</li> </ul> Not yet implemented are: <ul style="list-style-type: none"> <li>• "stan"</li> <li>• "spark"</li> <li>• "keras"</li> </ul>
.recipe_list	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
.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/\(workflowsets\)](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
)

wfs_sets <- ts_wfs_lin_reg("all_engines", rec_objs)
wfs_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:

	<ul style="list-style-type: none"> <li>• "earth"</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.num_terms</code>	The number of features that will be retained in the final model, including the intercept.
<code>.prod_degree</code>	The highest possible interaction degree.
<code>.prune_method</code>	The pruning method. This is a character, the default is "backward". You can choose from one of the following: <ul style="list-style-type: none"> <li>• "backward"</li> <li>• "none"</li> <li>• "exhaustive"</li> <li>• "forward"</li> <li>• "seqrep"</li> <li>• "cv"</li> </ul>

### 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, ...)`

<code>.non_seasonal_ar</code>	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
<code>.seasonal_ar</code>	The order of the seasonal auto-regressive (SAR) terms. Often denoted "P" in PDQ-notation.
<code>.hidden_units</code>	An integer for the number of units in the hidden model.
<code>.num_networks</code>	Number of networks to fit with different random starting weights. These are then averaged when producing forecasts.
<code>.penalty</code>	A non-negative numeric value for the amount of weight decay.
<code>.epochs</code>	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](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

<code>.model_type</code>	This is where you will set your engine. It uses <code>modeltime::prophet_reg()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>• "prophet" Or <code>modeltime::prophet_boost()</code> under the hood and can take one of the following:</li> <li>• "prophet_xgboost" You can also choose:</li> <li>• "all_engines" - This will make a model spec for all available engines.</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.growth</code>	String 'linear' or 'logistic' to specify a linear or logistic trend.
<code>.changepoint_num</code>	Number of potential changepoints to include for modeling trend.
<code>.changepoint_range</code>	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.
<code>.seasonality_yearly</code>	One of "auto", TRUE or FALSE. Set to FALSE for prophet_xgboost. Toggles on/off a seasonal component that models year-over-year seasonality.
<code>.seasonality_weekly</code>	One of "auto", TRUE or FALSE. Toggles on/off a seasonal component that models week-over-week seasonality. Set to FALSE for prophet_xgboost
<code>.seasonality_daily</code>	One of "auto", TRUE or FALSE. Toggles on/off a seasonal component that models day-over-day seasonality. Set to FALSE for prophet_xgboost
<code>.season</code>	'additive' (default) or 'multiplicative'.
<code>.prior_scale_changepoints</code>	Parameter modulating the flexibility of the automatic changepoint selection. Large values will allow many changepoints, small values will allow few changepoints.
<code>.prior_scale_seasonality</code>	Parameter modulating the strength of the seasonality model. Larger values allow the model to fit larger seasonal fluctuations, smaller values dampen the seasonality.
<code>.prior_scale_holidays</code>	Parameter modulating the strength of the holiday components model, unless overridden in the holidays input.
<code>.logistic_cap</code>	When growth is logistic, the upper-bound for "saturation".
<code>.logistic_floor</code>	When growth is logistic, the lower-bound for "saturation"
<code>.trees</code>	An integer for the number of trees contained in the ensemble.

<code>.min_n</code>	An integer for the minimum number of data points in a node that is required for the node to be split further.
<code>.tree_depth</code>	An integer for the maximum depth of the tree (i.e. number of splits) (specific engines only).
<code>.learn_rate</code>	A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (specific engines only).
<code>.loss_reduction</code>	A number for the reduction in the loss function required to split further (specific engines only).
<code>.stop_iter</code>	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://workflowsets.tidymodels.org/(workflowsets))

[https://business-science.github.io/modeltime/reference/prophet\\_reg.html](https://business-science.github.io/modeltime/reference/prophet_reg.html)

[https://business-science.github.io/modeltime/reference/prophet\\_boost.html](https://business-science.github.io/modeltime/reference/prophet_boost.html)

Other Auto Workflowsets: [ts\\_wfs\\_arma\\_boost\(\)](#), [ts\\_wfs\\_auto\\_arma\(\)](#), [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

<code>.model_type</code>	This is where you will set your engine. It uses <code>parsnip::svm_poly()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>"kernlab"</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.cost</code>	A positive number for the cose of predicting a sample within or on the wrong side of the margin.
<code>.degree</code>	A positive number for polynomial degree.
<code>.scale_factor</code>	A positive number for the polynomial scaling factor.
<code>.margin</code>	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](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

<code>.model_type</code>	This is where you will set your engine. It uses <code>parsnip::svm_rbf()</code> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>"kernlab"</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.cost</code>	A positive number for the cost of predicting a sample within or on the wrong side of the margin.
<code>.rbf_sigma</code>	A positive number for the radial basis function.
<code>.margin</code>	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](https://parsnip.tidymodels.org/reference/svm_rbf.html)

Other Auto Workflowsets: `ts_wfs_arma_boost()`, `ts_wfs_auto_arma()`, `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

<code>.model_type</code>	This is where you will set your engine. It uses <a href="#">parsnip::boost_tree</a> under the hood and can take one of the following: <ul style="list-style-type: none"> <li>"xgboost"</li> </ul>
<code>.recipe_list</code>	You must supply a list of recipes. <code>list(rec_1, rec_2, ...)</code>
<code>.trees</code>	The number of trees (type: integer, default: 15L)
<code>.min_n</code>	Minimal Node Size (type: integer, default: 1L)
<code>.tree_depth</code>	Tree Depth (type: integer, default: 6L)
<code>.learn_rate</code>	Learning Rate (type: double, default: 0.3)
<code>.loss_reduction</code>	Minimum Loss Reduction (type: double, default: 0.0)
<code>.sample_size</code>	Proportion Observations Sampled (type: double, default: 1.0)
<code>.stop_iter</code>	The number of iterations 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://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
```

# Index

- \* **Augment Function**
  - ts\_acceleration\_augment, 17
  - ts\_velocity\_augment, 104
- \* **Auto Workflowsets**
  - ts\_wfs\_arima\_boost, 107
  - ts\_wfs\_auto\_arima, 110
  - ts\_wfs\_ets\_reg, 111
  - ts\_wfs\_lin\_reg, 114
  - ts\_wfs\_mars, 115
  - ts\_wfs\_nnetar\_reg, 117
  - ts\_wfs\_prophet\_reg, 119
  - ts\_wfs\_svm\_poly, 122
  - ts\_wfs\_svm\_rbf, 124
  - ts\_wfs\_xgboost, 126
- \* **Boiler Plate**
  - ts\_auto\_arima, 21
  - ts\_auto\_arima\_xgboost, 23
  - ts\_auto\_croston, 25
  - ts\_auto\_exp\_smoothing, 27
  - ts\_auto\_glmnet, 30
  - ts\_auto\_lm, 32
  - ts\_auto\_mars, 34
  - ts\_auto\_nnetar, 36
  - ts\_auto\_prophet\_boost, 38
  - ts\_auto\_prophet\_reg, 40
  - ts\_auto\_smooth\_es, 44
  - ts\_auto\_svm\_poly, 47
  - ts\_auto\_svm\_rbf, 49
  - ts\_auto\_theta, 51
  - ts\_auto\_xgboost, 53
- \* **Boilerplate Utility**
  - ts\_extract\_auto\_fitted\_workflow, 64
- \* **Clustering**
  - ts\_feature\_cluster, 65
  - ts\_feature\_cluster\_plot, 67
- \* **Data Generator**
  - tidy\_fft, 15
  - ts\_brownian\_motion, 55
  - ts\_brownian\_motion\_augment, 57
  - ts\_geometric\_brownian\_motion, 71
  - ts\_geometric\_brownian\_motion\_augment, 72
  - ts\_random\_walk, 93
- \* **Helper**
  - ts\_model\_spec\_tune\_template, 88
- \* **Model Tuning**
  - ts\_model\_auto\_tune, 80
  - ts\_model\_spec\_tune\_template, 88
- \* **Plot**
  - ts\_brownian\_motion\_plot, 58
  - ts\_event\_analysis\_plot, 62
  - ts\_qq\_plot, 91
  - ts\_scedacity\_scatter\_plot, 97
- \* **Recipes**
  - step\_ts\_acceleration, 11
  - step\_ts\_velocity, 13
- \* **SVM**
  - ts\_auto\_svm\_poly, 47
  - ts\_auto\_svm\_rbf, 49
- \* **Simulator**
  - ts\_arima\_simulator, 19
  - ts\_forecast\_simulator, 69
- \* **Statistic**
  - ci\_hi, 5
  - ci\_lo, 6
- \* **Time Filtering**
  - ts\_compare\_data, 61
  - ts\_time\_event\_analysis\_tbl, 101
- \* **Utility**
  - calibrate\_and\_plot, 3
  - internal\_ts\_backward\_event\_tbl, 7
  - internal\_ts\_both\_event\_tbl, 8
  - internal\_ts\_forward\_event\_tbl, 9
  - model\_extraction\_helper, 10
  - ts\_get\_date\_columns, 74
  - ts\_info\_tbl, 75
  - ts\_is\_date\_class, 76

- ts\_lag\_correlation, 77
  - ts\_model\_auto\_tune, 80
  - ts\_model\_compare, 84
  - ts\_model\_rank\_tbl, 86
  - ts\_model\_spec\_tune\_template, 88
  - ts\_qq\_plot, 91
  - ts\_scedacity\_scatter\_plot, 97
  - ts\_to\_tbl, 103
  - \* **Vector Function**
    - ts\_acceleration\_vec, 18
    - ts\_velocity\_vec, 105
  - \* **arima\_xgboost**
    - ts\_auto\_arima\_xgboost, 23
  - \* **arima**
    - ts\_auto\_arima, 21
  - \* **croston**
    - ts\_auto\_croston, 25
  - \* **ets**
    - ts\_auto\_exp\_smoothing, 27
  - \* **exp\_smoothing**
    - ts\_auto\_croston, 25
    - ts\_auto\_exp\_smoothing, 27
    - ts\_auto\_smooth\_es, 44
    - ts\_auto\_theta, 51
  - \* **glmnet**
    - ts\_auto\_glmnet, 30
  - \* **lm**
    - ts\_auto\_lm, 32
  - \* **mars**
    - ts\_auto\_mars, 34
  - \* **nnetar**
    - ts\_auto\_nnetar, 36
  - \* **poly**
    - ts\_auto\_svm\_poly, 47
  - \* **prophet**
    - ts\_auto\_prophet\_boost, 38
    - ts\_auto\_prophet\_reg, 40
  - \* **rbf**
    - ts\_auto\_svm\_rbf, 49
  - \* **smooth\_es**
    - ts\_auto\_smooth\_es, 44
  - \* **theta**
    - ts\_auto\_theta, 51
  - \* **xgboost**
    - ts\_auto\_xgboost, 53
- Arima, 10, 69
- auto.arima, 10, 69
- calibrate\_and\_plot, 3, 8–10, 74, 76, 78, 83, 85, 87, 90, 92, 98, 104
- ci\_hi, 5, 6
- ci\_lo, 6, 6
- color\_blind, 7
- dials::grid\_latin\_hypercube(), 81
- ets, 10, 69
- forecast::simulate.Arima(), 70
- internal\_ts\_backward\_event\_tbl, 4, 7, 9, 10, 74, 76, 78, 83, 85, 87, 90, 92, 98, 104
- internal\_ts\_both\_event\_tbl, 4, 8, 8, 9, 10, 74, 76, 78, 83, 85, 87, 90, 92, 98, 104
- internal\_ts\_forward\_event\_tbl, 4, 8, 9, 9, 10, 74, 76, 78, 83, 85, 87, 90, 92, 98, 104
- model\_extraction\_helper, 4, 8, 9, 10, 74, 76, 78, 83, 85, 87, 90, 92, 98, 104
- modeltime::arima\_boost(), 108, 109
- modeltime::arima\_reg(), 110
- modeltime::default\_forecast\_accuracy\_metric\_set(), 22, 24, 26, 28, 31, 35, 37, 39, 41, 45, 48, 50, 54
- modeltime::exp\_smoothing(), 112
- modeltime::modeltime\_calibrate(), 3
- modeltime::nnetar\_reg(), 117, 118
- modeltime::plot\_modeltime\_forecast(), 3
- modeltime::prophet\_boost(), 120
- modeltime::prophet\_reg(), 120
- nnetar, 10, 69
- parsnip::boost\_tree, 126
- parsnip::boost\_tree(), 127
- parsnip::linear\_reg(), 114
- parsnip::mars(), 115
- parsnip::set\_engine(), 89
- parsnip::svm\_poly(), 122, 123
- parsnip::svm\_rbf(), 124, 125
- plotly::ggplotly(), 60
- recipes::step\_nzv(), 44
- recipes::step\_rm(), 12, 14
- recipes::step\_YeoJohnson(), 44

- `rlang::enquo()`, [17](#), [104](#)
- `simulate`, [69](#)
- `stats::arima.sim`, [20](#)
- `stats::arima.sim()`, [20](#)
- `stats::fft()`, [15](#)
- `stats::median()`, [19](#)
- `step_ts_acceleration`, [11](#), [14](#)
- `step_ts_velocity`, [12](#), [13](#)
- `tidy_fft`, [15](#), [56](#), [58](#), [72](#), [73](#), [94](#)
- `timetk::slidify_vec()`, [99](#)
- `timetk::step_fourier()`, [43](#)
- `timetk::step_timeseries_signature()`, [43](#)
- `timetk::time_series_cv()`, [22](#), [24](#), [26](#), [28](#), [31](#), [35](#), [37](#), [39](#), [41](#), [45](#), [48](#), [50](#), [54](#), [81](#), [82](#)
- `timetk::tk_index()`, [69](#)
- `timetk::tk_tbl()`, [103](#)
- `ts_acceleration_augment`, [17](#), [105](#)
- `ts_acceleration_augment()`, [18](#)
- `ts_acceleration_vec`, [18](#), [106](#)
- `ts_arima_simulator`, [19](#), [70](#)
- `ts_auto_arima`, [21](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_arima_xgboost`, [22](#), [23](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_croston`, [22](#), [24](#), [25](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_exp_smoothing`, [22](#), [24](#), [27](#), [27](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_glmnet`, [22](#), [24](#), [27](#), [29](#), [30](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_lm`, [22](#), [24](#), [27](#), [29](#), [31](#), [32](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_mars`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [34](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_nnetar`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [36](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_prophet_boost`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [38](#), [42](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_prophet_reg`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [40](#), [46](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_recipe`, [42](#)
- `ts_auto_smooth_es`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [44](#), [48](#), [50](#), [52](#), [54](#)
- `ts_auto_svm_poly`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [47](#), [50](#), [52](#), [54](#)
- `ts_auto_svm_rbf`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [49](#), [52](#), [54](#)
- `ts_auto_theta`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [51](#), [54](#)
- `ts_auto_xgboost`, [22](#), [24](#), [27](#), [29](#), [31](#), [33](#), [35](#), [37](#), [39](#), [42](#), [46](#), [48](#), [50](#), [52](#), [53](#)
- `ts_brownian_motion`, [16](#), [55](#), [58](#), [72](#), [73](#), [94](#)
- `ts_brownian_motion_augment`, [16](#), [56](#), [57](#), [72](#), [73](#), [94](#)
- `ts_brownian_motion_plot`, [58](#), [63](#), [92](#), [98](#)
- `ts_calendar_heatmap_plot`, [59](#)
- `ts_compare_data`, [61](#), [102](#)
- `ts_event_analysis_plot`, [59](#), [62](#), [92](#), [98](#)
- `ts_extract_auto_fitted_workflow`, [64](#)
- `ts_feature_cluster`, [65](#), [68](#)
- `ts_feature_cluster_plot`, [66](#), [67](#)
- `ts_forecast_simulator`, [20](#), [69](#)
- `ts_geometric_brownian_motion`, [16](#), [56](#), [58](#), [71](#), [73](#), [94](#)
- `ts_geometric_brownian_motion_augment`, [16](#), [56](#), [58](#), [72](#), [72](#), [94](#)
- `ts_get_date_columns`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_info_tbl`, [4](#), [8–10](#), [74](#), [75](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_is_date_class`, [4](#), [8–10](#), [74](#), [76](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_lag_correlation`, [4](#), [8–10](#), [74](#), [76](#), [77](#), [83](#), [85](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_ma_plot`, [79](#)
- `ts_model_auto_tune`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [80](#), [85](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_model_auto_tune()`, [88](#)
- `ts_model_compare`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [83](#), [84](#), [87](#), [90](#), [92](#), [98](#), [104](#)
- `ts_model_rank_tbl`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [83](#), [85](#), [86](#), [90](#), [92](#), [98](#), [104](#)
- `ts_model_spec_tune_template`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [83](#), [85](#), [87](#), [88](#), [92](#), [98](#), [104](#)
- `ts_model_spec_tune_template()`, [80](#), [82](#)
- `ts_qc_run_chart`, [90](#)
- `ts_qq_plot`, [4](#), [8–10](#), [59](#), [63](#), [74](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#), [91](#), [98](#), [104](#)
- `ts_random_walk`, [16](#), [56](#), [58](#), [72](#), [73](#), [93](#)
- `ts_random_walk()`, [95](#)
- `ts_random_walk_ggplot_layers`, [95](#)
- `ts_scale_color_colorblind`, [96](#)
- `ts_scale_fill_colorblind`, [96](#)

`ts_scedacity_scatter_plot`, [4](#), [8–10](#), [59](#),  
[63](#), [74](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#), [92](#), [97](#),  
[104](#)

`ts_sma_plot`, [99](#)

`ts_splits_plot`, [100](#)

`ts_time_event_analysis_tbl`, [62](#), [101](#)

`ts_to_tbl`, [4](#), [8–10](#), [74](#), [76](#), [78](#), [83](#), [85](#), [87](#), [90](#),  
[92](#), [98](#), [103](#)

`ts_velocity_augment`, [17](#), [104](#)

`ts_velocity_augment()`, [105](#)

`ts_velocity_vec`, [19](#), [105](#)

`ts_vva_plot`, [106](#)

`ts_wfs_arima_boost`, [107](#), [111](#), [113](#), [114](#),  
[116](#), [118](#), [121](#), [123](#), [125](#), [127](#)

`ts_wfs_auto_arima`, [109](#), [110](#), [113](#), [114](#), [116](#),  
[118](#), [121](#), [123](#), [125](#), [127](#)

`ts_wfs_ets_reg`, [109](#), [111](#), [111](#), [114](#), [116](#),  
[118](#), [121](#), [123](#), [125](#), [127](#)

`ts_wfs_lin_reg`, [109](#), [111](#), [113](#), [114](#), [116](#),  
[118](#), [121](#), [123](#), [125](#), [127](#)

`ts_wfs_mars`, [109](#), [111](#), [113](#), [114](#), [115](#), [118](#),  
[121](#), [123](#), [125](#), [127](#)

`ts_wfs_nnetar_reg`, [109](#), [111](#), [113](#), [114](#), [116](#),  
[117](#), [121](#), [123](#), [125](#), [127](#)

`ts_wfs_prophet_reg`, [109](#), [111](#), [113](#), [114](#),  
[116](#), [118](#), [119](#), [123](#), [125](#), [127](#)

`ts_wfs_svm_poly`, [109](#), [111](#), [113](#), [114](#), [116](#),  
[118](#), [121](#), [122](#), [125](#), [127](#)

`ts_wfs_svm_rbf`, [109](#), [111](#), [113](#), [114](#), [116](#),  
[118](#), [121](#), [123](#), [124](#), [127](#)

`ts_wfs_xgboost`, [109](#), [111](#), [113](#), [114](#), [116](#),  
[118](#), [121](#), [123](#), [125](#), [126](#)

`xts::plot.xts()`, [79](#)