Package ‘CorrClim’

April 9, 2024

Title CorrClim : Climatic correction toolbox

Version 1.0.0

Description Climatic correction toolbox for energy consumption.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports data.table,
dplyr,
glue,
httr,
jsonlite,
stringr,
magrittr,
MASS,
R39Toolbox,
R6,
lubridate,
mgcv,
plotly,
MLmetrics,
prophet,
TTR,
rBayesianOptimization

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

`api_get_holidays` *Get holidays from the different years*

Description

Get holidays from the different years

Usage

```
api_get_holidays(start_date, end_date, request_country = "FR")
```

Arguments

`start_date`	`(string)` starting date, format : yyyy/mm/dd
`end_date`	`(string)` end_date, format : yyyy/mm/dd
`request_country`	`(string)` country code of the country requested. Example : 'FR'

Value

holidays `(vector)`

`api_get_stations` *Get Stations list from API Meteo France*

Description

Get Stations list from API Meteo France

Usage

```
api_get_stations(departement)
```

`api_get_token_meteo_france` *Get Token for API Meteo France*

Description

Get Token for API Meteo France

Usage

```
api_get_token_meteo_france()
```

`api_get_weather_enedis` *Process weather data from ENEDIS API*

Description

Process weather data from ENEDIS API

Usage

```
api_get_weather_enedis(start_date, end_date)
```

Arguments

`start_date`	(`string`) starting date, format : yyyy-mm-dd
`end_date`	(`string`) end_date, format : yyyy-mm-dd

`api_get_weather_meteo_france` *Get weather timeseries from api Meteo France*

Description

Get weather timeseries from api Meteo France

Usage

```
api_get_weather_meteo_france(stations_id, date_start, date_end)
```

`api_process_weather_enedis` *Process weather data from ENEDIS API*

Description

Process weather data from ENEDIS API

Usage

```
api_process_weather_enedis(df_weather, start_date, end_date)
```

Arguments

`df_weather`	(`data.table`) Data obtained after api_request_weather_enedis
`start_date`	(`string`) starting date, format : yyyy/mm/dd
`end_date`	(`string`) end_date, format : yyyy/mm/dd

Value

None

`api_request_holidays` *Request Nager API for holidays data*

Description

Request Nager API for holidays data

Usage

```
api_request_holidays(request_year, request_country)
```

Value

holidays (dataframe) for a year and a country

`api_request_weather_enedis` *Request ENEDIS API for weather data*

Description

Request ENEDIS API for weather data

Usage

```
api_request_weather_enedis()
```

Value

df_weather (`data.table`) temperature data from enedis api

`BayesianSmoother` *Bayesian Exponential Smoother*

Description

Perform a Bayesian Optimisation on a Smoother to find best params and smooth using best smoother.

Super class

`CorrClim::Smoother` -> `BayesianSmoother`

Public fields

`bounds (list)` The list of the parameters and the bounds associated. Example : list(alpha = c(0,1))

`score (callable)` Default : correlation_score . A callable to compute the score to maximize. Should return a float and take a data.table with two columns in input.

`smoother (Smoother)` A smoother object not instantiated to perform the gridsearch on.

`n_iter (integer)` The number of iteration of the Bayesian Optimisation.

`init_points (integer)` Number of randomly chosen points to sample the target function before Bayesian Optimization fitting the Gaussian Process.

Methods

Public methods:

- `BayesianSmoother$new()`
- `BayesianSmoother$fit_fun()`
- `BayesianSmoother$smooth_fun()`
- `BayesianSmoother$get_best_params()`
- `BayesianSmoother$get_best_smoother()`
- `BayesianSmoother$clone()`

Method `new()`**:**

Usage:

```
BayesianSmoother$new(bounds, smoother, score = correlation_score, ...)
```

Method `fit_fun()`**:** Fit the smoother

Usage:

```
BayesianSmoother$fit_fun(timeseries, y)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The timeseries to fit on

`y (data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth_fun()`**:** Apply the Smoother

Usage:

`BayesianSmoother$smooth_fun(timeseries)`

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

Returns: `(TimeseriesDT)` The timeseries smoothed

Method `get_best_params()`**:** Get the optimal alpha after the fit method

Usage:

`BayesianSmoother$get_best_params()`

Returns: `(float)` Optimal alpha

Method `get_best_smoother()`**:** Get the best smoother

Usage:

`BayesianSmoother$get_best_smoother()`

Returns: `(Smoother)` Best smoother

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`BayesianSmoother$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`camel_to_snake` *Converts a character from camelCase to snake_case.*

Description

This function takes a character formatted in camelCase and converts it to snake_case.

Usage

```
camel_to_snake(inputString)
```

Arguments

`inputString`	A character character in camelCase format that needs to be converted to snake_case.

Value

`(string)` String in snake_case format.

`ClimaticCorrector` *ClimaticCorrector*

Description

An R6 class which stands as the model structure for each climatic correction model

Public fields

`timeseries_model (TimeseriesModel)` A TimeseriesModel object instantiated.

Methods

Public methods:

- `ClimaticCorrector$new()`
- `ClimaticCorrector$fit()`
- `ClimaticCorrector$apply()`
- `ClimaticCorrector$get_operator()`
- `ClimaticCorrector$export()`
- `ClimaticCorrector$clone()`

Method `new()`:

Usage:

```
ClimaticCorrector$new(timeseries_model, ...)
```

Method `fit()`: Fit the model using the TimeseriesModel object

Usage:

```
ClimaticCorrector$fit(timeseries, weather_observed, fold_varname = NULL)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The output/response data

`weather_observed (data.table | TimeseriesDT)` The input weather observed

`fold_varname (string)` Default NULL. The variable name to use to make a cross validation. Only used for a StdModel and Operator2Moments

Method `apply()`: Apply the model and make the climatic correction using an Operator object

Usage:

```
ClimaticCorrector$apply(timeseries, weather_observed, weather_target)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The output/response data

`weather_observed (data.table | TimeseriesDT)` The input weather observed

`weather_target (data.table | TimeseriesDT)` The input weather target

Returns: `(TimeseriesDT)` The output timeseries corrected from the weather target

Method `get_operator()`**:** Get the operator of the Climatic Corrector

Usage:

```
ClimaticCorrector$get_operator()
```

Returns: (Operator) The Operator of the Climatic Corrector

Method `export()`**:** Export the ClimaticCorrector in RDS.

Usage:

```
ClimaticCorrector$export(path)
```

Arguments:

`path (string)` Path file to write the rds file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ClimaticCorrector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CopyCat` *Copycat*

Description

Performs a Knn - like model.

Super class

`CorrClim::TimeseriesModel` -> `CopyCat`

Methods

Public methods:

- `CopyCat$new()`
- `CopyCat$print()`
- `CopyCat$fit_fun()`
- `CopyCat$predict_fun()`
- `CopyCat$clone()`

Method `new()`**:**

Usage:

```
CopyCat$new(formula = "y ~ temperature", ...)
```

Method `print()`**:**

Usage:

`CopyCat$print()`

Method `fit_fun()`**:** Fit function

Usage:

`CopyCat$fit_fun(model, X)`

Arguments:

`model` `(Any)`

`X` `(data.table | TimeseriesDT)` Timeseries data to fit. Should contains all variables in the formula

Method `predict_fun()`**:** Predict function

Usage:

`CopyCat$predict_fun(model, X)`

Arguments:

`model` `(Any)`

`X` `(data.table | TimeseriesDT)` Timeseries data to predict.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`CopyCat$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`correlation_score` *Compute correlation of a 2 columns data.table*

Description

Compute correlation of a 2 columns data.table

Usage

```
correlation_score(ts)
```

Arguments

`ts`	`(data.table)` The data table to compute the correlation on

Value

`(float)` The correlation between the two columns

`DummySmoother` *Dummy Smoother*

Description

Dummy smoother. Performs nothing.

Super class

`CorrClim::Smoother` -> `DummySmoother`

Methods

Public methods:

- `DummySmoother$new()`
- `DummySmoother$fit_fun()`
- `DummySmoother$smooth_fun()`
- `DummySmoother$clone()`

Method `new()`**:**

Usage:

```
DummySmoother$new(...)
```

Method `fit_fun()`**:** Fit the Smoother

Usage:

```
DummySmoother$fit_fun(timeseries, y = NULL)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to fit on

`y` `(data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth_fun()`**:** Apply the Smoother

Usage:

```
DummySmoother$smooth_fun(timeseries)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

Returns: `(TimeseriesDT)`) The timeseries smoothed

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DummySmoother$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ExponentialSmoother` *Exponential smoother*

Description

This smoother performs an exponential smooth of the timeseries. Can performs a basic exponential smoothing or a exponential smoothing for each timestep of the day, then the comparison is done on the previous day.

Super class

`CorrClim::Smoother` -> `ExponentialSmoother`

Public fields

`alpha (float in [0,1])`. Defines as the weight of the smoothing. s(t) = alpha x(t) + (1 - alpha) s(t-1)

`N (Optional[integer])` Default: 20. Only if granularity = 'days'. The number of passed days used to compute the smoothed timeseries

`granularity (string)` `{'step', 'days'}`. Default : 'step'. If step, performs a classic EMA step by step, if days, performs a EMA by day.

Methods

Public methods:

- `ExponentialSmoother$new()`
- `ExponentialSmoother$fit_fun()`
- `ExponentialSmoother$smooth_fun()`
- `ExponentialSmoother$clone()`

Method `new()`**:**

Usage:

```
ExponentialSmoother$new(...)
```

Method `fit_fun()`**:** Fit the Smoother

Usage:

```
ExponentialSmoother$fit_fun(timeseries, y = NULL)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The timeseries to fit on

`y (data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth_fun()`**:** Apply the Smoother

Usage:

```
ExponentialSmoother$smooth_fun(timeseries)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) The timeseries to smooth

Returns: (`TimeseriesDT`)) The timeseries smoothed

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ExponentialSmoother$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Formula` *Formula*

Description

An R6 class which stands as the formula structure of the library

Public fields

`formula` (`str | formula`) The formula of the model. exemple : y ~ temperature + snow

Methods

Public methods:

- `Formula$new()`
- `Formula$get_formula_str()`
- `Formula$get_formula()`
- `Formula$get_formula_base()`
- `Formula$get_explanatory_variables_formula_base()`
- `Formula$get_all_variables_formula_base()`
- `Formula$get_all_variables()`
- `Formula$shift_formula()`
- `Formula$get_explanatory_variables()`
- `Formula$clone()`

Method `new()`**:**

Usage:

```
Formula$new(f)
```

Method `get_formula_str()`**:** Get the formula formatted as a character

Usage:

```
Formula$get_formula_str()
```

Returns: (`string`) The formula as a character

Method `get_formula()`**:** Get the formula formatted

Usage:

```
Formula$get_formula()
```

Method `get_formula_base()`**:** Get the formula formatted before shift_formula method (if applied)

Usage:

```
Formula$get_formula_base()
```

Method `get_explanatory_variables_formula_base()`**:** Get the explanatory variables from the base formula (before shift method)

Usage:

```
Formula$get_explanatory_variables_formula_base()
```

Method `get_all_variables_formula_base()`**:** Get the all variables from the base formula (before shift method)

Usage:

```
Formula$get_all_variables_formula_base()
```

Method `get_all_variables()`**:** Get all the variables (response and explanatory)

Usage:

```
Formula$get_all_variables()
```

Method `shift_formula()`**:** Set the formula to add a character to each variables

Usage:

```
Formula$shift_formula()
```

Method `get_explanatory_variables()`**:** Get explanatory variables from formula

Usage:

```
Formula$get_explanatory_variables()
```

Returns: The explanatory variables from the formula

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Formula$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

GAM *GAM (Generative Additive Model)*

Description

The Generative Additive Model (GAM) for climate correction.

Super class

`CorrClim::TimeseriesModel` -> `GAM`

Public fields

`formula` (formula) the formula object representing the fit wanted (e.g y ~ temperature)

`by_instant` (boolean). If you want to perform a fit by instant of specific granularity

`granularity` (Optional[string]) {'day', 'month'}. The granularity used to compute instant.

Active bindings

`formula` (formula) the formula object representing the fit wanted (e.g y ~ temperature)

`by_instant` (boolean). If you want to perform a fit by instant of specific granularity

`granularity` (Optional[string]) {'day', 'month'}. The granularity used to compute instant.

Methods

Public methods:

- `GAM$new()`
- `GAM$print()`
- `GAM$fit_fun()`
- `GAM$predict_fun()`
- `GAM$clone()`

Method `new()`**:**

Usage:

```
GAM$new(
  formula = y ~ s(temperature) + s(posan) + jour_semaine + jour_ferie + ponts,
  by_instant = TRUE,
  granularity = "day",
  ...
)
```

Method `print()`**:**

Usage:

```
GAM$print()
```

Method `fit_fun()`**:** Fit function of the model itself

Usage:

`GAM$fit_fun(model, X)`

Arguments:

`model (Any)` The model to fit

`X (data.table | TimeseriesDT)` Timeseries data to fit. Should contains all variables in the formula

Method `predict_fun()`**:** Predict function of the model itself

Usage:

`GAM$predict_fun(model, X)`

Arguments:

`model (Any)` The model trained

`X (data.table | TimeseriesDT)` Timeseries data to predict.

Returns: `(vector)` The output of the prediction of the model on X.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`GAM$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`GamStd` *GamStd (Generative Additive Model for Standard Deviation estimation)*

Description

The Generative Additive Model (GAM) for timeseries conditional standard deviation estimation.

Super classes

`CorrClim::TimeseriesModel` -> `CorrClim::TimeseriesStdModel` -> `GamStd`

Public fields

`formula (formula)` the formula object representing the fit wanted (e.g y ~ temperature)

`by_instant (boolean)`. If you want to perform a fit by instant of specific granularity

`granularity (Optional[string])` {`'day', 'month'`}. The granularity used to compute instant.

Active bindings

`formula (formula)` the formula object representing the fit wanted (e.g y ~ temperature)

`by_instant (boolean)`. If you want to perform a fit by instant of specific granularity

`granularity (Optional[string])` {`'day', 'month'`}. The granularity used to compute instant.

Methods

Public methods:

- `GamStd$new()`
- `GamStd$print()`
- `GamStd$fit_fun()`
- `GamStd$predict_fun()`
- `GamStd$clone()`

Method `new()`**:**

Usage:

```
GamStd$new(
  formula = y ~ s(temperature) + s(posan) + jour_semaine + jour_ferie + ponts,
  by_instant = FALSE,
  granularity = "day",
  ...
)
```

Method `print()`**:**

Usage:

```
GamStd$print()
```

Method `fit_fun()`**:** Fit function of the model itself

Usage:

```
GamStd$fit_fun(model, X)
```

Arguments:

`model (Any)` The model to fit

`X (data.table | TimeseriesDT)` Timeseries data to fit. Should contains all variables in the formula

Method `predict_fun()`**:** Predict function of the model itself

Usage:

```
GamStd$predict_fun(model, X)
```

Arguments:

`model (Any)`

`X (data.table | TimeseriesDT)` Timeseries data to predict.

Returns: `(list)` The output of the prediction of the model on X. Returns the timeseries predicted (then delta = False) or the delta directly (timeseries is then the initial timeseries).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GamStd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

GradDelta *GradDelta*

Description

This class represents a model (GradDelta) for predicting values based on gradients, and linear models. Performs a linear model by instant.

Super class

`CorrClim::TimeseriesModel` -> `GradDelta`

Public fields

`formula` `(formula)` the formula object representing the fit wanted (e.g y ~ temperature)

`N_min` `(integer)`. Default: 30. Minimum sample allowed to perform the linear regression.

`weights` `(float vector)` Weights to pass in the linear regression model. Default is null.

`lm` `(MASS linear model)`. Default: 'robust'. Linear model. Choose between robust, least squares and ridge.

`n_shift` `(integer)` Default: 168. The number of steps in hour to shift your timeseries

`granularity` `(Optional[string])` `{'day', 'month'}`. Default : 'day'. The granularity used to compute instant.

Active bindings

`formula` `(formula)` the formula object representing the fit wanted (e.g y ~ temperature)

`granularity` `(Optional[string])` `{'day', 'month'}`. Default : 'day'. The granularity used to compute instant.

Methods

Public methods:

- `GradDelta$new()`
- `GradDelta$print()`
- `GradDelta$fit_fun()`
- `GradDelta$predict_fun()`
- `GradDelta$get_gradients()`
- `GradDelta$clone()`

Method `new()`**:**

Usage:

```
GradDelta$new(
  formula = y ~ temperature,
  by_instant = FALSE,
  granularity = "day",
  ...
)
```

Method `print()`**:**

Usage:

```
GradDelta$print()
```

Method `fit_fun()`**:** Fit function of the model itself

Usage:

```
GradDelta$fit_fun(model, X)
```

Arguments:

`model` `(Any)` The model to fit

`X` `(data.table | TimeseriesDT)` Timeseries data to fit. Should contains all variables in the formula

Method `predict_fun()`**:** Predict function of the model itself

Usage:

```
GradDelta$predict_fun(model, X)
```

Arguments:

`model` `(Any)` The model trained

`X` `(data.table | TimeseriesDT)` Timeseries data to predict.

Returns: `(vector)` The output of the prediction of the model on X.

Method `get_gradients()`**:** Get gradients from the GradDelta model

Usage:

```
GradDelta$get_gradients()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GradDelta$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GridSearchSmoother` *GridSearch Exponential Smoother*

Description

Perform a Grid search on a Smoother to find best params and smooth using best smoother.

Super class

`CorrClim::Smoother` -> `GridSearchSmoother`

Public fields

`grid (list)` The list of the parameters and the sequences associated to test. Example : list(alpha = seq(0.5, 1, 0.01))

`score (callable)` Default : correlation_score . A callable to compute the score to maximize. Should return a float and take a data.table with two columns in input.

`smoother (Smoother)` A smoother object not instantiated to perform the gridsearch on.

Methods

Public methods:

- `GridSearchSmoother$new()`
- `GridSearchSmoother$fit_fun()`
- `GridSearchSmoother$smooth_fun()`
- `GridSearchSmoother$get_best_params()`
- `GridSearchSmoother$get_best_smoother()`
- `GridSearchSmoother$clone()`

Method `new()`**:**

Usage:

```
GridSearchSmoother$new(grid, smoother, score = correlation_score, ...)
```

Method `fit_fun()`**:** Fit the smoother

Usage:

```
GridSearchSmoother$fit_fun(timeseries, y)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The timeseries to fit on

`y (data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth_fun()`**:** Apply the Smoother

Usage:

```
GridSearchSmoother$smooth_fun(timeseries)
```

Arguments:

`timeseries` (`data.table` | `TimeseriesDT`) The timeseries to smooth

Returns: (`TimeseriesDT`)) The timeseries smoothed

Method `get_best_params()`**:** Get the best parameters

Usage:

```
GridSearchSmoother$get_best_params()
```

Returns: (`float`) Optimal alpha

Method `get_best_smoother()`**:** Get the best smoother

Usage:

```
GridSearchSmoother$get_best_smoother()
```

Returns: (`Smoother`) Best smoother

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GridSearchSmoother$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Metrics` *Metrics*

Description

Metrics model-agnostic

Methods

Public methods:

- `Metrics$new()`
- `Metrics$plot_timeseries()`
- `Metrics$RMSE()`
- `Metrics$MAPE()`
- `Metrics$BIAS()`
- `Metrics$plot_bar_volume_correction()`
- `Metrics$plot_aggregated_cc_evol_by_category()`
- `Metrics$plot_scatter_timeseries()`
- `Metrics$clone()`

Method `new()`**:**

Usage:

```
Metrics$new()
```

Method `plot_timeseries()`**:** Plot timeseries before and after the climate correction

Usage:

```
Metrics$plot_timeseries(
  timeseries,
  prediction,
  ts_to_compare_with = NULL,
  variables_to_compare = NULL,
  granularity = "day",
  fun = mean,
  ylabel = "Timeseries",
  ylabel_to_compare = NULL,
  title = "Timeseries comparison with and without climate correction",
  legend = c("Timeseries input", "Timeseries climate corrected"),
  legend_difference = "Correction climatique",
  width = 1000,
  height = 700
)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` timeseries with measures

`prediction` `(data.table | TimeseriesDT)` timeseries with forecasts/climate corrections

`ts_to_compare_with` `(data.table | TimeseriesDT)` The timeseries to compare with. Builds a subplots if feeded.

`variables_to_compare` (vector of string) The variables in the ts_to_compare_with data to plot.

`granularity` `(string)` The granularity to aggregate on. Choose between hour, days, week, month, year

`fun` `(callable)` The aggregation function to apply.

`ylabel` `(string)` The label for y axis.

`ylabel_to_compare` `(string)` The label for the y axis timeseries to compare

`title` `(string)` Title of the graph

`legend` (vector of string) The legend for timeseries and prediction args

`legend_difference` `(string)` The legend for the difference between timeseries and prediction

`width` `(string)` The width of the graph

`height` `(integer)` The height of the graph

Returns: `(plotly graph)` The before/after timeseries

Method `RMSE()`**:** Root Mean Squared Error

Usage:

```
Metrics$RMSE(
  timeseries,
  prediction,
  plot = TRUE,
  granularity = "day",
  fun = mean,
  width = 1000,
  height = 700
)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) timeseries with measures

`prediction` (`data.table | TimeseriesDT`) timeseries with forecasts/climate corrections

`plot` (`boolean`) Whether to plot or not the evolution of RSE

`granularity` (`string`) The granularity to aggregate the timeseries on (used only for plots)

`fun` (`callable`) An aggregation function for the timeseries (used only for plots)

`width` (`integer`) The width of the graph

`height` (`integer`) The height of the graph

Returns: `(float | plotly graph)` RMSE plot or RMSE float

Method `MAPE()`**:** Mean Absolute Pourcentage Error

Usage:

```
Metrics$MAPE(
  timeseries,
  prediction,
  plot = TRUE,
  granularity = "day",
  fun = mean,
  width = 1000,
  height = 700
)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) timeseries with measures

`prediction` (`data.table | TimeseriesDT`) timeseries with forecasts/climate corrections

`plot` (`boolean`) Whether to plot or not the evolution of APE

`granularity` (`string`) The granularity to aggregate the timeseries on (used only for plots)

`fun` (`callable`) An aggregation function for the timeseries (used only for plots)

`width` (`integer`) The width of the graph

`height` (`integer`) The height of the graph

Returns: `(float | plotly graph)` MAPE plot or MAPE float

Method `BIAS()`**:** Bias (error = measure - forecast/cc)

Usage:

```
Metrics$BIAS(
  timeseries,
  prediction,
  plot = TRUE,
  granularity = "day",
  fun = mean,
  width = 1000,
  height = 700
)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) timeseries with measures

`prediction` (`data.table | TimeseriesDT`) timeseries with forecasts/climate corrections

`plot` (`boolean`) Whether to plot or not the evolution of APE

`granularity` (`string`) The granularity to aggregate the timeseries on (used only for plots)

`fun` (`callable`) An aggregation function for the timeseries (used only for plots)

`width` (`integer`) The width of the graph

`height` (`integer`) The height of the graph

Returns: `(float | plotly graph)` BIAS plot or BIAS float

Method `plot_bar_volume_correction()`**:** Plot the volume of climate correction aggregated on a specific granularity

Usage:

```
Metrics$plot_bar_volume_correction(
  timeseries,
  prediction,
  granularity = "month",
  fun = mean,
  title = "Climate correction volume",
  ylabel = "Energy",
  width = 1000,
  height = 700
)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) timeseries with measures

`prediction` (`data.table | TimeseriesDT`) timeseries with forecasts/climate corrections

`granularity` (`string`) The granularity to aggregate the timeseries on (used only for plots)

`fun` (`callable`) An aggregation function for the timeseries (used only for plots)

`title` (`string`) The graph title

`ylabel` (`string`) The label for y axis.

`width` (`integer`) The width of the graph

`height` (`integer`) The height of the graph

Returns: `(plotly graph)` Barplot of the climate correction volume

Method `plot_aggregated_cc_evol_by_category()`**:** Plot the evolution through time of climate correction aggregated at a specific granularity, for different categories

Usage:

```
Metrics$plot_aggregated_cc_evol_by_category(
  prediction,
  category_varname,
  granularity = "year",
  fun = sum,
  title = "Climate correction evolution by category",
  ylabel = "Energy",
  width = 1000,
  height = 700
)
```

Arguments:

`prediction` (data.table | TimeseriesDT | list there of) list of timeseries with forecasts/climate corrections (1 element for each category)

`granularity` `(string)` The granularity to aggregate the timeseries on (used only for plots)

`fun` `(callable)` An aggregation function for the timeseries (used only for plots)

`title` `(string)` The graph title

`ylabel` `(string)` The label for y axis.

`width` `(integer)` The width of the graph

`height` `(integer)` The height of the graph

Returns: `(plotly graph)` Barplot of the climate correction volume

Method `plot_scatter_timeseries():` Plot a scatter of y_ts vs x_ts. Add second_y_ts if you want two scatter group.

Usage:

```
Metrics$plot_scatter_timeseries(
  y_ts,
  x_ts,
  xlabel,
  ylabel,
  title,
  second_y_ts = NULL,
  legend = c("First Timeseries", "Second Timeseries"),
  opacity = 0.7,
  granularity = "day",
  width = 1000,
  height = 700,
  size = 6,
  fun = mean
)
```

Arguments:

`y_ts` `(data.table | TimeseriesDT)` The timeseries to plot in y axis

`x_ts` `(data.table | TimeseriesDT)` The timeseries to plot in x axis

`xlabel` `(string)` The label for x axis

`ylabel` `(string)` The label for y axis

`title` `(string)` The graph title

`second_y_ts` `(data.table | TimeseriesDT)` The second timeseries to plot in y axis

`granularity` `(string)` The granularity to aggregate the timeseries

`width` `(integer)` The width of the graph

`height` `(integer)` The height of the graph

`size` `(integer)` Dot size of the scatter plot

`fun` `(callable)` An aggregation function for the timeseries

Returns: `(float | plotly graph)` Scatter plot for timeseries

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Metrics$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`MultiSmoother` *MultiSmoother*

Description

A MultiSmoother class to encapsulates the role of multiple smoothing terms. Smoothes only temperature variables, and consider one smoother for one temperature variable.

Super class

`CorrClim::Smoother` -> `MultiSmoother`

Public fields

`smoothers` (vector of Smoother object). The smoothers to apply.

`variables` (vector of strings). The variables to smooth.

Methods

Public methods:

- `MultiSmoother$new()`
- `MultiSmoother$fit_fun()`
- `MultiSmoother$smooth_fun()`
- `MultiSmoother$get_smoothers()`
- `MultiSmoother$clone()`

Method `new()`**:**

Usage:

```
MultiSmoother$new(smoothers, variables)
```

Method `fit_fun()`**:** Apply the Smoother

Usage:

```
MultiSmoother$fit_fun(timeseries, y)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

`y` `(data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth_fun()`**:** Apply the Smoother

Usage:

```
MultiSmoother$smooth_fun(timeseries)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

Returns: `(TimeseriesDT)`) The timeseries smoothed

Method `get_smoothers()`**:** Get smoothers of the MultiSmoother object

Usage:

```
MultiSmoother$get_smoothers()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MultiSmoother$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Operator` *Operator*

Description

Represents the structure of Climate Correction Operator.

Methods

Public methods:

- `Operator$new()`
- `Operator$apply()`
- `Operator$clone()`

Method `new()`**:**

Usage:

```
Operator$new(...)
```

Method `apply()`**:** Apply the Operator on the output to make the climatic correction

Usage:

```
Operator$apply(
  timeseries,
  y_pred_observed,
  y_pred_target,
  y_std_observed = NULL,
  y_std_target = NULL
)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The timeseries to apply climate correction on.

`y_pred_observed (vector)` The inference made on the observed weather.

`y_pred_target (vector)` The inference made on the target weather

`y_std_observed (vector)` The inference of standard deviation made on the observed weather

`y_std_target (vector)` The inference of standard deviation made on the target weather

Returns: `(TimeseriesDT)` The output climate corrected

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Operator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Operator2Moments` *Operator2Moments*

Description

Operator to perform climatic correction over the 2 first distribution moments (expectation and standard deviation)

Super class

`CorrClim::Operator` -> `Operator2Moments`

Methods

Public methods:

- `Operator2Moments$new()`
- `Operator2Moments$apply_fun()`
- `Operator2Moments$clone()`

Method `new()`**:**

Usage:

```
Operator2Moments$new(...)
```

Method `apply_fun()`**:** Apply the 2-moments Operator

Usage:

```
Operator2Moments$apply_fun(
  timeseries,
  y_pred_observed,
  y_pred_target,
  y_std_observed,
  y_std_target
)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) The timeseries to apply climate correction on.

`y_pred_observed` (`vector`) The inference made on the observed weather.

`y_pred_target` (`vector`) The inference made on the target weather

`y_std_observed` (`vector`) The inference of standard deviation made on the observed weather

`y_std_target` (`vector`) The inference of standard deviation made on the target weather

Returns: (`TimeseriesDT`) The output climate corrected

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Operator2Moments$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`OperatorAdditive` *OperatorAdditive*

Description

Additive Operator. Performs an addition of the climate part of the timeseries

Super class

`CorrClim::Operator` -> `OperatorAdditive`

Methods

Public methods:

- `OperatorAdditive$new()`
- `OperatorAdditive$apply_fun()`
- `OperatorAdditive$clone()`

Method `new()`**:**

Usage:

```
OperatorAdditive$new(...)
```

Method `apply_fun()`**:** Apply function of the Additive operator

Usage:

```
OperatorAdditive$apply_fun(timeseries, y_pred_observed, y_pred_target, ...)
```

Arguments:

`timeseries` (`data.table | TimeseriesDT`) The timeseries to apply climate correction on.

`y_pred_observed` (`vector`) The inference made on the observed weather.

`y_pred_target` (`vector`) The inference made on the target weather

`...` Additional arguments

Returns: `(TimeseriesDT)` The output climate corrected

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OperatorAdditive$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`OperatorMultiplicative` *OperatorMultiplicative*

Description

Multiplicative Operator. Apply a share of the climate part of the timeseries

Super class

`CorrClim::Operator` -> `OperatorMultiplicative`

Methods

Public methods:

- `OperatorMultiplicative$new()`
- `OperatorMultiplicative$apply_fun()`
- `OperatorMultiplicative$clone()`

Method `new()`**:**

Usage:

```
OperatorMultiplicative$new(...)
```

Method `apply_fun()`**:** Apply the Multiplicative Operator

Usage:

```
OperatorMultiplicative$apply_fun(
  timeseries,
  y_pred_observed,
  y_pred_target,
  ...
)
```

Arguments:

`timeseries (data.table | TimeseriesDT)` The timeseries to apply climate correction on.

`y_pred_observed (vector)` The inference made on the observed weather.

`y_pred_target (vector)` The inference made on the target weather

`...` Additional arguments

Returns: `(TimeseriesDT)` The output climate corrected

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OperatorMultiplicative$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`OperatorTarget` *OperatorTarget*

Description

Operator returning model estimation at target weather as a climatic correction

Super class

`CorrClim::Operator` -> `OperatorTarget`

Methods

Public methods:

- `OperatorTarget$new()`
- `OperatorTarget$apply_fun()`
- `OperatorTarget$clone()`

Method `new()`**:**

Usage:

```
OperatorTarget$new(...)
```

Method `apply_fun()`**:** Apply function of the Target operator

Usage:

```
OperatorTarget$apply_fun(y_pred_target, ...)
```

Arguments:

`y_pred_target` `(data.table | TimeseriesDT)` The inference made on the target weather

`...` Additional arguments

Returns: `(TimeseriesDT)` The output climate corrected

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OperatorTarget$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Prophet` *Prophet Model*

Description

This class represents a model (Prophet) for time series forecasting using the Prophet package.

Super class

`CorrClim::TimeseriesModel` -> `Prophet`

Public fields

`formula` (`formula`) the formula object representing the fit wanted (e.g y ~ temperature)

`yearly_seasonality` Boolean, default TRUE. Indicates whether to include yearly seasonality.

`weekly_seasonality` Boolean, default TRUE. Indicates whether to include weekly seasonality.

`daily_seasonality` Boolean, default TRUE. Indicates whether to include daily seasonality.

`interval_width` Numeric, default 0. Indicates the uncertainty interval width.

Active bindings

`formula` (`formula`) the formula object representing the fit wanted (e.g y ~ temperature)

Methods

Public methods:

- `Prophet$new()`
- `Prophet$print()`
- `Prophet$fit_fun()`
- `Prophet$predict_fun()`
- `Prophet$clone()`

Method `new()`**:**

Usage:

```
Prophet$new(formula = y ~ temperature, ...)
```

Method `print()`**:**

Usage:

```
Prophet$print()
```

Method `fit_fun()`**:** Fit function of the model itself

Usage:

```
Prophet$fit_fun(model, X)
```

Arguments:

`model` (`Any`)

`X` (`data.table | TimeseriesDT`) Timeseries data to fit. Should contains all variables in the formula

Method `predict_fun()`**:** Predict function of the model itself

Usage:

```
Prophet$predict_fun(model, X)
```

Arguments:

`model` (`Any`) The model trained

`X` (`data.table | TimeseriesDT`) Timeseries data to predict.

Returns: (`vector`) The output of the prediction of the model on X.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Prophet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Smoother` *Smoother*

Description

Base class for any smoother object.

Methods

Public methods:

- `Smoother$new()`
- `Smoother$fit()`
- `Smoother$smooth()`
- `Smoother$fit_smooth()`
- `Smoother$export()`
- `Smoother$clone()`

Method `new()`**:**

Usage:

```
Smoother$new(...)
```

Method `fit()`**:** Fit the Smoother

Usage:

```
Smoother$fit(timeseries, y = NULL)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to fit on
`y` `(data.table | TimeseriesDT)` The response timeseries to compare with

Method `smooth()`**:** Apply the Smoother

Usage:

```
Smoother$smooth(timeseries)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

Returns: `(TimeseriesDT)` The timeseries smoothed

Method `fit_smooth()`**:** Fit and apply the smoother. Calls the fit and smooth method.

Usage:

```
Smoother$fit_smooth(timeseries, y = NULL)
```

Arguments:

`timeseries` `(data.table | TimeseriesDT)` The timeseries to smooth

Returns: `(TimeseriesDT)` The timeseries smoothed

Method `export()`**:** Export the Smoother in RDS.

Usage:

```
Smoother$export(path)
```

Arguments:

`path` `(string)` Path file to write the rds file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Smoother$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`TimeseriesDT` *TimeseriesDT*

Description

An R6 class which stands as the timeseries structure described in the library

Public fields

`timeseries` `(data.table)` A data.table which contains a time column and N value columns.

Methods

Public methods:

Method `new()`:

Usage:

```
TimeseriesDT$new(
  timeseries,
  is_output = FALSE,
  format_date = "%Y-%m-%d %H:%M:%S",
  timezone = "UTC"
)
```

Method `remove_na()`**:** Remove NA from timeseries data

Usage:

`TimeseriesDT$remove_na(inplace = TRUE)`

Arguments:

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `set_timeseries()`**:** Set the timeseries of a TimeseriesDT existing object. Use with caution.

Usage:

`TimeseriesDT$set_timeseries(timeseries)`

Arguments:

`timeseries (data.table)`

Method `get_timeseries()`**:** Get the timeseries data.table

Usage:

`TimeseriesDT$get_timeseries()`

Returns: `(data.table)` TimeseriesDT data.table

Method `check_pattern_in_column()`**:** Look for patterns in a character

Usage:

`TimeseriesDT$check_pattern_in_column(patterns, column)`

Arguments:

`patterns (string vector)`

`column (string vector)`

Returns: (boolean vector) A vector asserting is yes or no the pattern is contained in the column arg

Method `set_format_date()`**:** Set a new format for date

Usage:

`TimeseriesDT$set_format_date(format_date = NULL)`

Arguments:

`format_date (string)` A valid format for dates

Method `set_timezone()`**:** Set a new timezone

Usage:

`TimeseriesDT$set_timezone(timezone)`

Method `get_variables_from_pattern()`**:** Get variables from a Timeseries matching patterns

Usage:

`TimeseriesDT$get_variables_from_pattern(patterns)`

Arguments:

`patterns (string vector)`

Method `get_variables_name()`: Get variables from the dataset.

Usage:

```
TimeseriesDT$get_variables_name()
```

Method `get()`: Get variable from timeseries. Similar as dt$var or dt[var]

Usage:

```
TimeseriesDT$get(var)
```

Method `add_calendar()`: Generate calendar from a timeseries

Usage:

```
TimeseriesDT$add_calendar(
  variables = c("date", "Annee", "Mois", "Jour", "Heure", "Minute", "Posan", "Tendance",
    "JourSemaine", "JourFerie", "Ponts", "isoSemaine", "isoAnnee"),
   inplace = TRUE
)
```

Arguments:

`variables (string vector)`variables to add to the calendar.

Returns: The calendar generated

Method `shift()`: Add shifted timeseries to existing timeseries

Usage:

```
TimeseriesDT$shift(variables, n = 168, inplace = TRUE)
```

Arguments:

`variables (string vector)`variables names to shift

`n (integer)` number of steps to shift on

`inplace (boolean)` If TRUE or FALSE you want to add to existing timeseries or return a new object

Method `nrows()`: Get the row number of the timeseries

Usage:

```
TimeseriesDT$nrows()
```

Method `compute_instant()`: Add an instant column to existing timeseries

Usage:

```
TimeseriesDT$compute_instant(granularity = "day", inplace = TRUE)
```

Arguments:

`inplace (boolean)` If TRUE or FALSE you want to add to existing timeseries or return a new object

Method `add_suffix()`: Add a suffix to explanatory variables

Usage:

```
TimeseriesDT$add_suffix(variables, suffix, inplace = TRUE)
```

Arguments:

`variables (string vector)` The variables to add suffix on

`suffix (string)` The suffix you want to add

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `sort()`**:** Sort timeseries accordin to a variable, ascending mode.

Usage:

```
TimeseriesDT$sort(variable, inplace = TRUE)
```

Arguments:

`variable (string)` The variable to use to order (ascending mode only) the timeseries

Method `compute_period_start()`**:** Compute the period start for a specific granularity

Usage:

```
TimeseriesDT$compute_period_start(granularity, inplace = TRUE)
```

Arguments:

`granularity (string)` {`'hour', 'day', 'week', 'month', "year"`}. Aggregation level.

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `aggregate()`**:** Aggregates data according to the 'time' column at different levels. Returns datetime in the time column

Usage:

```
TimeseriesDT$aggregate(granularity, fun = mean, inplace = TRUE)
```

Arguments:

`granularity (string)` {`'hour', 'day', 'month', 'year', 'week'`} Aggregation level.

`fun (callable)` Aggregation function. 'mean' by default.

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `groupby()`**:** Groupby data according to the 'time' column at different levels. Doesn't return datetime in time column

Usage:

```
TimeseriesDT$groupby(granularity, fun = mean)
```

Arguments:

`granularity (string)` {`'hour', 'wday', 'month', 'week', 'year'`}. Aggregation level.

`fun (callable)` Aggregation function. 'mean' by default.

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `select()`**:** Select variables from timeseries

Usage:

```
TimeseriesDT$select(variables, inplace = TRUE)
```

Arguments:

`variables (string vector)` The variables you want to select into the TimeseriesDT.

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `remove_duplicated()`**:** Remove duplicated values from timeseries

Usage:

```
TimeseriesDT$remove_duplicated(variables = "time", inplace = TRUE)
```

Arguments:

`variables (string vector)` Default : 'time'. The variables on which you want to compare uniqueness

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `assign()`**:** A vector to your timeseries

Usage:

```
TimeseriesDT$assign(name, vector, inplace = TRUE)
```

Arguments:

`name (string)` The name of your new column

`vector (vector)` The vector to add to your timeseries

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `merge()`**:** Merge TimeseriesDT with an other data.table or TimeseriesDT

Usage:

```
TimeseriesDT$merge(
  timeseries_to_merge,
  by = "time",
  how = "inner",
  suffixes = c(".x", ".y"),
  inplace = TRUE
)
```

Arguments:

`by (string vector)` Default: 'time'. Column to merge on

`how (string)` Default : 'inner'. Either inner left or all

`suffixes (string vector)`Suffixes to add in case of overlapping columns names

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `remove_variables()`**:** Remove variables from timeseries

Usage:

```
TimeseriesDT$remove_variables(variables, inplace = TRUE)
```

Arguments:

`variables (string vector)` Variables to remove from timeseries `inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `align()`**:** Align timeseries to ensure same length. Returns input aligned

Usage:

```
TimeseriesDT$align(to_align, second_to_align = NULL)
```

Arguments:

`to_align (data.table | TimeseriesDT)` The timeseries to align with

`second_to_align (data.table | TimeseriesDT)` A third timeseries to align with

Method `compute_degre_days()`**:** Compute degree days using a threshold for heating or cooling days

Usage:

```
TimeseriesDT$compute_degre_days(
  temperature_column = NULL,
  all = TRUE,
  cooling = FALSE,
  threshold_cooling = 18,
  threshold_heating = 15,
  inplace = TRUE
)
```

Arguments:

`temperature_column (string)` The temperature column

`all (boolean)` Default: TRUE If TRUE, heating and cooling degree days are computed

`cooling (boolean)` If TRUE, cooling is the only to be computed

`threshold_cooling (float)` Default: 18. The threshold value for cooling

`threshold_heating (float)` Default: 15. The threshold value for heating

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `get_granularity()`**:** Returns the granularity of your timeseries in the specified unit

Usage:

```
TimeseriesDT$get_granularity(unit = "hour")
```

Arguments:

`unit (string)` The unit of time you want the granularity. Default is hour

Method `rename()`**:** Renames columns

Usage:

```
TimeseriesDT$rename(old_cols, new_cols, inplace = TRUE)
```

Arguments:

`old_cols (string vector)` Old columns names

`new_cols (string vector)` New columns names

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

Method `filter_dataset()`**:** Returns a filtered dataset

Usage:

```
TimeseriesDT$filter_dataset(
  y_shifted,
  var,
  var_shifted,
  threshold,
  q_max = 0.8,
  q_min = 0.2,
  IC_width = 1.5,
  inferior = TRUE,
  inplace = TRUE
)
```

Arguments:

`var (string)` Explanatory variables to filter on

`var_shifted (string)` Explanatory variables shifted to filter on

`threshold (float)` Threshold for this explanatory variables

`q_max (float)` Quantile maximal for filtering inputs

`q_min (float)` Quantile minimum for filtering inputs

`IC_width (float)` Interval confidence to filder input

`inferior (boolean)` If TRUE or FALSE the var has to be inferior of the threshold

`inplace (boolean)` If TRUE or FALSE a new instance is wanted or modifiying existing one is sufficient.

`y (string)` Response variable

Method `export()`**:** Export your timeseries to file

Usage:

```
TimeseriesDT$export(path, as_data_table = TRUE, format = "csv")
```

Arguments:

`path (string)` A valid path to store the timeseries

`as_data_table (boolean)` Whether you want to export the object itself or the data.

`format (string)` {`'csv', 'rds'`} The format to export. Optional if self = TRUE.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TimeseriesDT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`TimeseriesModel` *TimeseriesModel*

Description

An R6 class which stands as the model structure for each climatic correction model

Public fields

`formula` `(formula)` the formula object representing the fit wanted (e.g y ~ temperature)

`model` `(Any)` a model object obtained such as GAM, prophet

`smoothers` `(MultiSmoother)` a MultiSmoother object to handle smoothing variables.

`by_instant` `(list)` Hyperparameter of the TimeseriesModel to take into account a fit by an instant granularity or not

Methods

Public methods:

- `TimeseriesModel$new()`
- `TimeseriesModel$check_timeseries()`
- `TimeseriesModel$get_formula()`
- `TimeseriesModel$get_model()`
- `TimeseriesModel$get_smoothers()`
- `TimeseriesModel$fit()`
- `TimeseriesModel$predict()`
- `TimeseriesModel$cv_predict()`
- `TimeseriesModel$export()`
- `TimeseriesModel$clone()`

Method `new()`**:**

Usage:

```
TimeseriesModel$new(formula, by_instant, granularity, ...)
```

Method `check_timeseries()`**:** Check if dataset in input has the variables in formula and had variables if possible.

Usage:

```
TimeseriesModel$check_timeseries(X, is_fitting = TRUE)
```

Arguments:

`X` `(data.table | TimeseriesDT)` The data to test

Returns: X `(data.table | TimeseriesDT)` The data tested and with calendar or shifted inputs if needed.

Method `get_formula()`**:** Get the formula of the model

Usage:

`TimeseriesModel$get_formula()`

Method `get_model()`**:** Get the model itself

Usage:

`TimeseriesModel$get_model()`

Method `get_smoothers()`**:** Get the smoothers of the MultiSmoother object

Usage:

`TimeseriesModel$get_smoothers()`

Method `fit()`**:** Fit the model using the TimeseriesModel object

Usage:

`TimeseriesModel$fit(outputs, inputs)`

Arguments:

`outputs (data.table | TimeseriesDT)` The output/response data

`inputs (data.table | TimeseriesDT)` The input data to fit on.

`fold_varname (string)` the name of the variable in `inputs` to define CV folds from

Method `predict()`**:** Predict the model and make the climatic correction using the an Operator Object

Usage:

`TimeseriesModel$predict(X)`

Arguments:

`X (data.table | TimeseriesDT)` The timeseries data to make prediction on

Returns: `(vector)` The output timeseries as a vector from the model prediction

Method `cv_predict()`**:** Make CV predictions from the model

Usage:

`TimeseriesModel$cv_predict(outputs, inputs, fold_varname)`

Arguments:

`outputs (data.table | TimeseriesDT)` The output/response data

`inputs (data.table | TimeseriesDT)` The input data

`fold_varname (string)` the name of the variable in `inputs` to define CV folds from

Returns: `(vector)` The output timeseries as a vector from the model CV prediction

Method `export()`**:** Export the TimeseriesModel in RDS.

Usage:

`TimeseriesModel$export(path)`

Arguments:

`path (string)` Path file to write the rds file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`TimeseriesModel$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`TimeseriesStdModel` *TimeseriesStdModel*

Description

An R6 class which stands as the model structure for each conditional standard deviation timeseries model

Super class

`CorrClim::TimeseriesModel` -> `TimeseriesStdModel`

Public fields

`formula (formula)` the formula object representing the fit wanted (e.g value ~ temperature)

`model (Any)` a model object obtained such as GAM, prophet

`smoothers (Smoother)` a Smoother object

`conditional_expectation_model (TimeseriesModel)` the model to compute conditional expectation from

Active bindings

`formula (formula)` the formula object representing the fit wanted (e.g value ~ temperature)

`model (Any)` a model object obtained such as GAM, prophet

`smoothers (Smoother)` a Smoother object

Methods

Public methods:

- `TimeseriesStdModel$new()`
- `TimeseriesStdModel$fit()`
- `TimeseriesStdModel$predict()`
- `TimeseriesStdModel$clone()`

Method `new()`**:**

Usage:

```
TimeseriesStdModel$new(formula, by_instant, granularity, ...)
```

Method `fit()`**:** Fit the conditional variance model based on the conditional expectation one Relies on CV squared residuals computed from the latter to do so

Usage:

```
TimeseriesStdModel$fit(outputs, inputs, fold_varname)
```

Arguments:

`outputs (data.table | TimeseriesDT)` The output/response data `inputs` `(data.table | TimeseriesDT)` The input data with data for both conditional expectation and conditional variance models

`fold_varname` `(string)` the name of the variable in `inputs` to define CV folds from

Method `predict()`**:** Predict the conditional standard deviation

Usage:

```
TimeseriesStdModel$predict(inputs)
```

Arguments:

`inputs` `(data.table | TimeseriesDT)` The timeseries data to make prediction on

Returns: `(vector)` The output timeseries as a vector from the model prediction

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TimeseriesStdModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index