Package: forecastbaselines (via r-universe)

June 4, 2026

Type Package

Title R Interface to ForecastBaselines.jl

Version 0.1.0

Description Provides an R interface to the ForecastBaselines.jl Julia package, enabling access to 10 baseline forecasting models including ARMA, ETS, STL, and others. Supports probabilistic forecasting with multiple interval methods and seamless integration with the scoringutils evaluation framework.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports JuliaCall (>= 0.17.0), scoringutils (>= 1.0.0)

SystemRequirements Julia (>= 1.9)

RoxygenNote 7.3.3

URL https://github.com/epiforecasts/forecastbaselines

BugReports https://github.com/epiforecasts/forecastbaselines/issues

Suggests testthat (>= 3.0.0), ggplot2, hubVis, knitr, rmarkdown

Remotes hubverse-org/hubVis

Repository https://epiforecasts.r-universe.dev

Date/Publication 2025-11-26 15:05:39 UTC

RemoteUrl https://github.com/epiforecasts/forecastbaselines

RemoteRef v0.1.0

RemoteSha e900f3d86eb0e39159ca12956b1267b6efc9e224

Contents

`add_intervals` *Add Intervals to Forecast*

Description

Add Intervals to Forecast

Usage

```
add_intervals(forecast, intervals)
```

Arguments

`forecast`	A Forecast object
`intervals`	Interval data structure

Value

Updated Forecast object

`add_median` *Add Median to Forecast*

Description

Add Median to Forecast

Usage

```
add_median(forecast, median)
```

Arguments

`forecast`	A Forecast object
`median`	Numeric vector of median forecasts

Value

Updated Forecast object

add_temporal_info *Add Temporal Information to Forecast*

Description

Add Temporal Information to Forecast

Usage

```
add_temporal_info(forecast, reference_date, target_date, resolution)
```

Arguments

forecast	A Forecast object
reference_date	Reference date
target_date	Target date
resolution	Time resolution

Value

Updated Forecast object

add_trajectories *Add Trajectories to Forecast*

Description

Add Trajectories to Forecast

Usage

```
add_trajectories(forecast, trajectories)
```

Arguments

forecast	A Forecast object
trajectories	Matrix of trajectories

Value

Updated Forecast object

add_truth *Add Truth Values to Forecast*

Description

Add Truth Values to Forecast

Usage

```
add_truth(forecast, truth)
```

Arguments

forecast	A Forecast object
truth	Numeric vector of observed values

Value

Updated Forecast object

ARMAModel *ARMA Model*

Description

Creates an AutoRegressive Moving Average model.

Usage

```
ARMAModel(p = 0L, q = 0L, s = 0L, include_mean = TRUE, include_drift = FALSE)
```

Arguments

p	AR order (default: 0)
q	MA order (default: 0)
s	Seasonal period (default: 0 for no seasonality)
include_mean	Whether to include a mean term (default: TRUE)
include_drift	Whether to include a drift term (default: FALSE)

Value

An ARMAModel object

Examples

```
## Not run:
# AR(1)
model <- ARMAModel(p = 1)

# MA(1)
model <- ARMAModel(q = 1)

# ARMA(2,1) with seasonality
model <- ARMAModel(p = 2, q = 1, s = 12)

## End(Not run)
```

`as_forecast_point.ForecastBaselines_Forecast`
Convert ForecastBaselines Forecast to scoringutils point forecast

---

Description

S3 method to convert a ForecastBaselines_Forecast object to a scoringutils forecast_point object. This allows seamless integration with scoringutils.

Usage

```
## S3 method for class 'ForecastBaselines_Forecast'
as_forecast_point(data, ...)
```

Arguments

`data`	A ForecastBaselines_Forecast object
`...`	Additional arguments (not used)

Value

A forecast_point object from scoringutils

Examples

```
## Not run:
# Convert and validate as point forecast
fc_point <- scoringutils::as_forecast_point(forecast)

## End(Not run)
```

as_forecast_quantile.ForecastBaselines_Forecast *Convert ForecastBaselines Forecast to scoringutils quantile forecast*

Description

S3 method to convert a ForecastBaselines_Forecast object to a scoringutils forecast_quantile object. This allows seamless integration with scoringutils.

Usage

```
## S3 method for class 'ForecastBaselines_Forecast'
as_forecast_quantile(data, ...)
```

Arguments

data	A ForecastBaselines_Forecast object with quantiles
...	Additional arguments (not used)

Value

A forecast_quantile object from scoringutils

Examples

```
## Not run:
# Convert and validate as quantile forecast
fc_quantile <- scoringutils::as_forecast_quantile(forecast)

## End(Not run)
```

as_hubverse *Convert ForecastBaselines Forecast to hubverse format*

Description

Converts a ForecastBaselines_Forecast object to the hubverse format used by hubVis for visualization. Returns a list with model_output and target_data data frames.

Usage

```
as_hubverse(
  data,
  start_date = NULL,
  horizon_unit = c("day", "week", "month"),
  observed_data = NULL
)
```

Arguments

data	A ForecastBaselines_Forecast object with quantiles, or a scoringutils forecast_quantile object
start_date	Optional start date for the time series. If NULL (default), uses horizon numbers as target_date values. Can be a Date object or character string in "YYYY-MM-DD" format.
horizon_unit	Character string specifying the time unit for horizons. One of "day" (default), "week", or "month". Used when start_date is provided.
observed_data	Optional numeric vector of all observed values (training + test). If provided, will be used to populate target_data with complete historical context.

Details

The hubVis and hubUtils packages are not on CRAN and must be installed from GitHub: `remotes::install_github(c("hu "hubverse-org/hubVis"))`

Value

A list with two elements:

model_output Data frame (model_out_tbl) with columns: target_date, value, model_id, output_type, output_type_id

target_data Data frame with columns: date, observation

Examples

```
## Not run:
# With observed data for complete visualization context
hubverse_data <- as_hubverse(forecast,
  start_date = "2024-01-01",
  horizon_unit = "week",
  observed_data = full_data
)

# Install hubVis/hubUtils from GitHub for visualization
# remotes::install_github(c("hubverse-org/hubUtils", "hubverse-org/hubVis"))

# Visualize with hubVis (if installed)
# Note: as_hubverse() automatically converts model_output to model_out_tbl
if (requireNamespace("hubVis", quietly = TRUE)) {
  hubVis::plot_step_ahead_model_output(
    hubverse_data$model_output,
    hubverse_data$target_data
  )
}

## End(Not run)
```

ConstantModel *Constant Model*

Description

Creates a naive forecast model that uses the last observed value as the forecast for all future horizons.

Usage

```
ConstantModel()
```

Value

A ConstantModel object

Examples

```
## Not run:
model <- ConstantModel()

## End(Not run)
```

EmpiricalInterval *Empirical Interval Method*

Description

Creates prediction intervals by bootstrapping from historical forecast errors. This is a non-parametric method that doesn't assume any particular distribution.

Usage

```
EmpiricalInterval(
  n_trajectories = 1000L,
  min_observation = 1L,
  bootstrap_distribution = NULL,
  seed = NULL,
  positivity_correction = "none",
  symmetry_correction = FALSE,
  stepwise = FALSE,
  return_trajectories = FALSE
)
```

Arguments

n_trajectories	Number of bootstrap samples to generate (default: 1000)
min_observation	Minimum number of observations required (default: 1)
bootstrap_distribution	Optional distribution to sample from (default: NULL)
seed	Random seed for reproducibility (default: NULL)
positivity_correction	Method to ensure positive forecasts: "none", "post_clip", "truncate", or "zero_floor" (default: "none")
symmetry_correction	Whether to apply symmetry correction (default: FALSE)
stepwise	Whether to use stepwise intervals (default: FALSE)
return_trajectories	Whether to return forecast trajectories (default: FALSE)

Value

An EmpiricalInterval object

Examples

```
## Not run:
# Basic empirical intervals
method <- EmpiricalInterval()

# With more trajectories and seed
method <- EmpiricalInterval(n_trajectories = 2000, seed = 123)

# With positivity correction for count data
method <- EmpiricalInterval(
  n_trajectories = 1000,
  positivity_correction = "post_clip"
)

# Return trajectories for visualization
method <- EmpiricalInterval(return_trajectories = TRUE)

## End(Not run)
```

ETSModel *ETS Model*

Description

Creates an Error-Trend-Season exponential smoothing model.

Usage

```
ETSModel(
  error_type = "A",
  trend_type = "N",
  season_type = "N",
  s = NULL,
  damped = FALSE
)
```

Arguments

error_type	Error type: "A" (additive), "M" (multiplicative), or "N" (none)
trend_type	Trend type: "A" (additive), "M" (multiplicative), "Ad" (damped additive), "Md" (damped multiplicative), or "N" (none)
season_type	Season type: "A" (additive), "M" (multiplicative), or "N" (none)
s	Seasonal period (required if season_type is not "N")
damped	Whether to use damped trend (default: FALSE)

Value

An ETSModel object

Examples

```
## Not run:
# Simple exponential smoothing (A,N,N)
model <- ETSModel(error_type = "A", trend_type = "N", season_type = "N")

# Holt's linear trend (A,A,N)
model <- ETSModel(error_type = "A", trend_type = "A", season_type = "N")

# Holt-Winters additive (A,A,A)
model <- ETSModel(
  error_type = "A", trend_type = "A", season_type = "A", s = 12
)

# Holt-Winters multiplicative (M,M,M)
model <- ETSModel(
  error_type = "M", trend_type = "M", season_type = "M", s = 12
)

## End(Not run)
```

filter_horizons *Filter Forecast to Specific Horizons*

Description

Filter Forecast to Specific Horizons

Usage

```
filter_horizons(forecast, horizons)
```

Arguments

forecast	A Forecast object
horizons	Vector of horizons to keep

Value

Filtered Forecast object

filter_levels *Filter Forecast to Specific Confidence Levels*

Description

Filter Forecast to Specific Confidence Levels

Usage

```
filter_levels(forecast, levels)
```

Arguments

forecast	A Forecast object
levels	Vector of confidence levels to keep

Value

Filtered Forecast object

`fit_baseline` *Fit a Baseline Model*

Description

Fits a baseline forecasting model to observed data.

Usage

```
fit_baseline(x, model, temporal_info = NULL)
```

Arguments

`x`	Numeric vector of time series data
`model`	A model created using ConstantModel(), ARMAModel(), ETSModel(), or other model functions (see ?ConstantModel for available models)
`temporal_info`	Optional TemporalInfo object with start date and resolution

Value

A fitted model object that can be used for forecasting

Examples

```
## Not run:
data <- c(1.2, 2.3, 3.1, 2.8, 3.5, 4.2, 3.9)
model <- ARMAModel(p = 1, q = 1)
fitted <- fit_baseline(data, model)

## End(Not run)
```

`forecast` *Generate Complete Forecast with Intervals*

Description

Generates a complete forecast including point forecasts, prediction intervals, and optionally trajectories.

Usage

```
forecast(
  fitted,
  interval_method = NoInterval(),
  horizon = 1L,
  levels = 0.95,
  include_median = TRUE,
  truth = NULL,
  model_name = ""
)
```

Arguments

fitted	A fitted model object from fit_baseline()
interval_method	Interval method object (NoInterval, EmpiricalInterval, etc.)
horizon	Integer or vector of integers specifying forecast horizons
levels	Numeric vector of confidence levels (default: 0.95)
include_median	Whether to include median forecast (default: TRUE)
truth	Optional vector of true values for evaluation
model_name	Optional name for the model

Value

A Forecast object (list) containing forecasts and metadata

Examples

```
## Not run:
# Point forecast only
fc <- forecast(fitted, interval_method = NoInterval(), horizon = 1:12)

# With prediction intervals
fc <- forecast(fitted,
  interval_method = EmpiricalInterval(n_trajectories = 1000),
  horizon = 1:12,
  levels = c(0.80, 0.95)
)

# With truth for evaluation
fc <- forecast(fitted,
  interval_method = EmpiricalInterval(),
  horizon = 1:12,
  truth = c(3.6, 3.8, 4.1, ...)
)

## End(Not run)
```

has_horizon *Check if Forecast has Horizon*

Description

Check if Forecast has Horizon

Usage

```
has_horizon(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

has_intervals *Check if Forecast has Intervals*

Description

Check if Forecast has Intervals

Usage

```
has_intervals(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

has_mean *Check if Forecast has Mean*

Description

Check if Forecast has Mean

Usage

```
has_mean(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

has_median *Check if Forecast has Median*

Description

Check if Forecast has Median

Usage

```
has_median(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

has_trajectories *Check if Forecast has Trajectories*

Description

Check if Forecast has Trajectories

Usage

```
has_trajectories(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

has_truth *Check if Forecast has Truth Values*

Description

Check if Forecast has Truth Values

Usage

```
has_truth(forecast)
```

Arguments

forecast	A Forecast object

Value

Logical TRUE/FALSE

IDSModel *IDS Model*

Description

Creates an Increase-Decrease-Stable model for trend detection.

Usage

```
IDSModel(threshold = 0, window_size = 3L)
```

Arguments

threshold	Threshold for trend detection (default: 0.0)
window_size	Window size for trend calculation (default: 3)

Value

An IDSModel object

Examples

```
## Not run:
model <- IDSModel()
model <- IDSModel(threshold = 0.1, window_size = 5)

## End(Not run)
```

INARCHModel *INARCH Model*

Description

Creates an Integer-valued ARCH model for count time series.

Usage

```
INARCHModel(p = 1L)
```

Arguments

p	Order of the INARCH model

Value

An INARCHModel object

Examples

```
## Not run:
model <- INARCHModel(p = 1)

## End(Not run)
```

`interval_forecast` *Generate Interval Forecasts*

Description

Generates prediction intervals from a fitted model.

Usage

```
interval_forecast(fitted, method, horizon = 1L, levels = 0.95)
```

Arguments

`fitted`	A fitted model object from fit_baseline()
`method`	Interval method object
`horizon`	Integer or vector of integers specifying forecast horizons
`levels`	Numeric vector of confidence levels

Value

List containing point forecasts, median, intervals, and trajectories

Examples

```
## Not run:
intervals <- interval_forecast(fitted,
  method = EmpiricalInterval(),
  horizon = 1:12,
  levels = c(0.80, 0.95)
)

## End(Not run)
```

inverse_transform_data *Apply Inverse Transformation*

Description

Applies the inverse of a transformation to data (e.g., exp for log).

Usage

```
inverse_transform_data(y, transformation)
```

Arguments

y	Numeric vector of transformed data
transformation	A transformation object

Value

Original-scale numeric vector

Examples

```
## Not run:
transformed <- c(0, 0.693, 1.099, 1.386, 1.609)
trans <- LogTransform()
original <- inverse_transform_data(transformed, trans)

## End(Not run)
```

is_setup *Check if Julia and ForecastBaselines.jl are set up*

Description

Tests whether Julia is configured and ForecastBaselines.jl is accessible.

Usage

```
is_setup()
```

Value

TRUE if setup is complete, FALSE otherwise

Examples

```
## Not run:
if (is_setup()) {
  # Use forecasting functions
} else {
  setup_ForecastBaselines()
}

## End(Not run)
```

KDEModel *KDE Model*

Description

Creates a kernel density estimation model for non-parametric forecasting.

Usage

```
KDEModel(bandwidth = NULL, kernel = "gaussian")
```

Arguments

bandwidth	Bandwidth for KDE (default: NULL for automatic selection)
kernel	Kernel function to use (default: "gaussian")

Value

A KDEModel object

Examples

```
## Not run:
model <- KDEModel()
model <- KDEModel(bandwidth = 0.5)

## End(Not run)
```

LogPlusOneTransform *Log Plus One Transformation*

Description

Creates a log(x + c) transformation. Useful for data with zeros.

Usage

```
LogPlusOneTransform(c = 1)
```

Arguments

c	Constant to add before taking log (default: 1)

Value

A LogPlusOneTransform object

Examples

```
## Not run:
# Standard log(x + 1)
trans <- LogPlusOneTransform()

# Custom constant
trans <- LogPlusOneTransform(c = 0.5)

## End(Not run)
```

LogTransform *Log Transformation*

Description

Creates a natural logarithm transformation. Data must be positive.

Usage

```
LogTransform()
```

Value

A LogTransform object

Examples

```
## Not run:
trans <- LogTransform()

## End(Not run)
```

`LSDModel` *Last Similar Dates (LSD) Model*

Description

Creates a seasonal forecasting model based on similar historical dates.

Usage

```
LSDModel(s, window_width = 1L, trend_correction = FALSE)
```

Arguments

`s`	Seasonal period (e.g., 7 for weekly, 12 for monthly)
`window_width`	Width of the window for averaging similar dates (default: 1)
`trend_correction`	Whether to apply trend correction (default: FALSE)

Value

An LSDModel object

Examples

```
## Not run:
# Weekly seasonality
model <- LSDModel(s = 7)

# Monthly seasonality with window
model <- LSDModel(s = 12, window_width = 2)

## End(Not run)
```

MarginalModel *Marginal Model*

Description

Creates a forecast based on the empirical marginal distribution using the mean of the p most recent observations.

Usage

```
MarginalModel(p = NULL)
```

Arguments

p	Number of most recent observations to use (default: all observations)

Value

A MarginalModel object

Examples

```
## Not run:
model <- MarginalModel(p = 10)

## End(Not run)
```

ModelTrajectoryInterval *Model Trajectory Interval Method*

Description

Creates prediction intervals by simulating trajectories from the fitted model. This method uses the model's own simulation mechanism.

Usage

```
ModelTrajectoryInterval(
  n_trajectories = 1000L,
  seed = NULL,
  positivity_correction = "none",
  return_trajectories = FALSE
)
```

Arguments

n_trajectories	Number of trajectories to simulate (default: 1000)
seed	Random seed for reproducibility (default: NULL)
positivity_correction	Method to ensure positive forecasts: "none", "post_clip", "truncate", or "zero_floor" (default: "none")
return_trajectories	Whether to return forecast trajectories (default: FALSE)

Value

A ModelTrajectoryInterval object

Examples

```
## Not run:
# Basic model-based intervals
method <- ModelTrajectoryInterval()

# With more trajectories
method <- ModelTrajectoryInterval(n_trajectories = 2000, seed = 456)

# Return trajectories for analysis
method <- ModelTrajectoryInterval(
  n_trajectories = 1000,
  return_trajectories = TRUE
)

## End(Not run)
```

NoInterval *No Interval Method*

Description

Creates an interval method that produces only point forecasts without prediction intervals.

Usage

```
NoInterval()
```

Value

A NoInterval object

Examples

```
## Not run:
method <- NoInterval()
fc <- forecast(fitted, interval_method = method, horizon = 1:12)

## End(Not run)
```

`NoTransform` *No Transformation*

Description

Creates a transformation that applies no changes to the data.

Usage

```
NoTransform()
```

Value

A NoTransform object

Examples

```
## Not run:
trans <- NoTransform()

## End(Not run)
```

`OLSModel` *OLS Model*

Description

Creates an ordinary least squares model with polynomial trend.

Usage

```
OLSModel(degree = 1L, differencing = 0L)
```

Arguments

`degree`	Polynomial degree (default: 1 for linear trend)
`s`	Seasonal period (default: NULL for no seasonality)

Value

An OLSModel object

Examples

```
## Not run:
# Linear trend
model <- OLSModel(degree = 1)

# Quadratic trend with seasonality
model <- OLSModel(degree = 2, s = 12)

## End(Not run)
```

ParametricInterval *Parametric Interval Method*

Description

Creates prediction intervals using parametric assumptions based on the model's distribution (e.g., assuming normality for ARMA models).

Usage

```
ParametricInterval(positivity_correction = "none")
```

Arguments

positivity_correction	Method to ensure positive forecasts: "none" or "post_clip" (default: "none")

Value

A ParametricInterval object

Examples

```
## Not run:
# Standard parametric intervals
method <- ParametricInterval()

# With positivity correction
method <- ParametricInterval(positivity_correction = "post_clip")

## End(Not run)
```

point_forecast *Generate Point Forecasts*

Description

Generates point forecasts from a fitted model.

Usage

```
point_forecast(fitted, horizon = 1L)
```

Arguments

fitted	A fitted model object from fit_baseline()
horizon	Integer or vector of integers specifying forecast horizons

Value

Numeric vector of point forecasts

Examples

```
## Not run:
forecasts <- point_forecast(fitted, horizon = 1:12)

## End(Not run)
```

PowerPlusOneTransform *Power Plus One Transformation*

Description

Creates a power transformation with a constant: (x + c)^lambda. Useful for Box-Cox transformations with zeros.

Usage

```
PowerPlusOneTransform(lambda, constant = 1)
```

Arguments

lambda	Power parameter
constant	Constant to add before transformation (default: 1.0)

Value

A PowerPlusOneTransform object

Examples

```
## Not run:
# Box-Cox transformation
trans <- PowerPlusOneTransform(lambda = 0.3)

# Custom constant
trans <- PowerPlusOneTransform(lambda = 0.5, constant = 0.5)

## End(Not run)
```

PowerTransform *Power Transformation*

Description

Creates a power transformation: x^lambda (Box-Cox family).

Usage

```
PowerTransform(lambda)
```

Arguments

lambda	Power parameter (lambda = 0 is log, lambda = 0.5 is sqrt)

Value

A PowerTransform object

Examples

```
## Not run:
# Square root (equivalent to SquareRootTransform)
trans <- PowerTransform(lambda = 0.5)

# Cube root
trans <- PowerTransform(lambda = 1 / 3)

## End(Not run)
```

`print.ForecastBaselines_Forecast`
Print method for Forecast objects

Description

Print method for Forecast objects

Usage

```
## S3 method for class 'ForecastBaselines_Forecast'
print(x, ...)
```

Arguments

`x`	A Forecast object
`...`	Additional arguments (ignored)

`setup_ForecastBaselines`
Setup Julia and load ForecastBaselines.jl

Description

This function initializes Julia, installs ForecastBaselines.jl if needed, and loads the package. Must be called before using any forecasting functions.

Usage

```
setup_ForecastBaselines(
  JULIA_HOME = NULL,
  install_package = TRUE,
  rebuild = FALSE,
  verbose = TRUE
)
```

Arguments

`JULIA_HOME`	Path to Julia installation (optional, will auto-detect if not provided)
`install_package`	Whether to install ForecastBaselines.jl if not already installed
`rebuild`	Whether to rebuild the Julia system image
`verbose`	Whether to print verbose output during setup

Value

Invisibly returns TRUE if setup was successful

Examples

```
## Not run:
# Basic setup (auto-detect Julia)
setup_ForecastBaselines()

# Specify Julia location
setup_ForecastBaselines(JULIA_HOME = "/usr/local/julia/bin")

## End(Not run)
```

SquareRootTransform *Square Root Transformation*

Description

Creates a square root transformation. Useful for count data and variance stabilization.

Usage

```
SquareRootTransform()
```

Value

A SquareRootTransform object

Examples

```
## Not run:
trans <- SquareRootTransform()

## End(Not run)
```

STLModel *STL Model*

Description

Creates a Seasonal-Trend decomposition using Loess model.

Usage

```
STLModel(s, trend = TRUE, robust = FALSE)
```

Arguments

s	Seasonal period
trend	Whether to include trend component (default: TRUE)
robust	Whether to use robust fitting (default: FALSE)

Value

An STLModel object

Examples

```
## Not run:
# Monthly seasonality
model <- STLModel(s = 12)

# Robust STL
model <- STLModel(s = 12, robust = TRUE)

## End(Not run)
```

TemporalInfo *Create Temporal Information Object*

Description

Creates a TemporalInfo object to track dates and time resolution.

Usage

```
TemporalInfo(start = 1, resolution = 1)
```

Arguments

start	Start date (Date, POSIXct, or integer)
resolution	Time resolution (integer for generic, or period object)

Value

A TemporalInfo object

Examples

```
## Not run:
# Simple integer indexing
temp_info <- TemporalInfo(start = 1, resolution = 1)

# Date-based
temp_info <- TemporalInfo(start = as.Date("2024-01-01"), resolution = 1)

## End(Not run)
```

---

transform_data *Apply Data Transformation*

Description

Applies a transformation to data.

Usage

```
transform_data(x, transformation)
```

Arguments

x	Numeric vector of data
transformation	A transformation object

Details

Recommended: Use R's built-in transformation functions instead for better reliability and flexibility. This function has limitations due to bugs in ForecastBaselines.jl (e.g., SquareRootTransform fails).

See 'vignette("transformations")' for the recommended R approach.

Value

Transformed numeric vector

Examples

```
## Not run:
# Julia approach (limited):
data <- c(1, 2, 3, 4, 5)
trans <- LogTransform()
transformed <- transform_data(data, trans)

# Recommended R approach:
transformed <- log(data)

## End(Not run)
```

`transform_model` *Apply Transformation to Model*

Description

Wraps a model with a data transformation.

Usage

```
transform_model(model, transformation)
```

Arguments

`model`	A model object
`transformation`	A transformation object

Details

Not Implemented: This function does not work because ForecastBaselines.jl does not implement ‘transform()‘ for model types.

Recommended approach: Transform your data manually in R before fitting:

“‘r # Instead of transform_model(): log_data <- log(data) fitted <- fit_baseline(log_data, model) fc <- forecast(fitted, ...) fc$mean <- exp(fc$mean) # Back-transform “‘

See ‘vignette("transformations")‘ for complete examples.

Value

A transformed model object

Examples

```
## Not run:
# This will fail - not implemented in Julia package
model <- ARMAModel(p = 1, q = 1)
trans <- LogTransform()
transformed_model <- transform_model(model, trans) # Error!

# Use manual transformation instead:
log_data <- log(data)
fitted <- fit_baseline(log_data, model)
fc <- forecast(fitted, interval_method = NoInterval(), horizon = 1:5)
fc$mean <- exp(fc$mean)

## End(Not run)
```

`truncate_horizon` *Truncate Forecast Horizon*

Description

Truncate Forecast Horizon

Usage

```
truncate_horizon(forecast, max_h)
```

Arguments

`forecast`	A Forecast object
`max_h`	Maximum horizon to keep

Value

Truncated Forecast object

Index