Package: misc (via r-universe)

November 8, 2026

Title Miscellaneous Useful R Functions

Version 0.10.4

Date 2026-02-11

Description Miscellaneous Useful R Functions.

Depends R (>= 3.5.0), MASS, stats, utils, foreach, doSNOW, snow

Suggests knitr, ranger, xgboost, lightgbm, scam, survival, quantreg, tseries, glmnet

License MIT

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

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

Date/Publication 2026-02-11 06:42:08 UTC

RemoteUrl https://github.com/thierrymoudiki/misc

RemoteRef HEAD

RemoteSha 1f8759cfa700e61f75f270b78f5170422b9d8d0b

Contents

compute_finite_difference_gradients 2
compute_linear_shap 3
conformalize 3
correlation_aware_shap 4
coverage_score 4
debug_print 5
direct_sampling 6
extract_coefficients 6
fit_linear_model 7
fit_param_dist 7
fit_predict_shap 8
get_model_predictions 9

get_model_type . . . 9
get_next_date . . . 9
gradient_shap . . . 10
integrate . . . 10
integrated_gradients . . . 11
is_package_available . . . 12
is_wholenumber . . . 13
KL_divergence_hist . . . 13
local_linear_shap . . . 14
monotonic_regression . . . 14
mspl . . . 15
one_hot_encode . . . 16
parfor . . . 16
plot_prediction_interval . . . 18
predict.conformalize . . . 20
print_shap_summary . . . 21
residuals.conformalize . . . 21
rm_zero_cols . . . 22
rmultivariate . . . 22
scale_matrix . . . 23
sims_to_forecast_object . . . 24
simulate.conformalize . . . 25
sort_df . . . 26
split_data . . . 26
splitts . . . 27
time_series_accuracy . . . 27
timeit . . . 29
validate_shap_decomposition . . . 29
vlookup . . . 30
winkler_score . . . 30
winkler_score2 . . . 31

Index **33**

`compute_finite_difference_gradients`
Compute finite difference gradients

Description

Compute finite difference gradients

Usage

```
compute_finite_difference_gradients(f, X, h)
```

compute_linear_shap *Compute SHAP values for linear models*

Description

Compute SHAP values for linear models

Usage

```
compute_linear_shap(model, X, newx, y, account_correlations, n_samples)
```

conformalize *Conformal Prediction for Regression Models*

Description

This function implements conformal prediction for regression models using a split conformal approach.

Usage

```
conformalize(
  formula = NULL,
  x = NULL,
  y = NULL,
  data = NULL,
  fit_func,
  predict_func = predict,
  split_ratio = 0.5,
  seed = NULL,
  ...
)
```

Arguments

formula	A formula specifying the model to be fitted.
x	A numeric matrix of predictors.
y	A numeric vector of responses.
data	A data frame containing the variables in the formula.
fit_func	A function to fit the model. It should take a formula and data as arguments.
predict_func	A function to make predictions. It should take a fitted model and new data as arguments.
split_ratio	The proportion of data to use for training. Default is 0.5.
seed	An optional seed for reproducibility.

Value

A list containing the fitted model, residuals, and other relevant information.

Examples

```
# Example usage with a linear model
library(MASS)
data(Boston)
set.seed(123)
fit_func <- function(formula, data) {
  lm(formula, data = data)
}
predict_func <- function(fit, newdata) {
  predict(fit, newdata)
}
conformalize(formula = medv ~ ., data = Boston, fit_func = fit_func,
predict_func = predict_func, split_ratio = 0.5, seed = 123)
```

---

correlation_aware_shap *Correlation-aware SHAP computation using sampling approach*

Description

Correlation-aware SHAP computation using sampling approach

Usage

```
correlation_aware_shap(coefs, newx_centered, X_train, feature_means, n_samples)
```

---

coverage_score *Calculate Coverage Score for Prediction Intervals*

Description

Computes the proportion of actual values that fall within the prediction intervals, expressed as a percentage. This measures how well the prediction intervals capture the true values.

Usage

```
coverage_score(obj, actual)
```

Arguments

obj	A forecast object containing prediction intervals. Should have either lower and upper components or an intervals matrix with lower bounds in column 1 and upper bounds in column 2.
actual	Numeric vector of actual observed values to compare against the prediction intervals.

Details

The function first checks if the forecast object has lower and upper components. If not, it assumes the intervals are stored in a matrix format where column 1 contains lower bounds and column 2 contains upper bounds.

Value

Numeric value representing the coverage percentage (0-100). Higher values indicate better interval coverage.

Examples

```
## Not run:
# Example with lower/upper components
forecast_obj <- list(lower = c(10, 15, 20), upper = c(20, 25, 30))
actual_values <- c(12, 18, 25)
coverage_score(forecast_obj, actual_values)

# Example with intervals matrix
forecast_obj2 <- list(intervals = matrix(c(10, 15, 20, 20, 25, 30), ncol = 2))
coverage_score(forecast_obj2, actual_values)

## End(Not run)
```

---

debug_print *Debug print*

Description

Debug print

Usage

```
debug_print(x)
```

Arguments

x	An object to be printed

Examples

```
misc::debug_print(1:10)
misc::debug_print("Hello, world!")
```

`direct_sampling` *Direct sampling*

Description

Direct sampling

Usage

```
direct_sampling(
  data = NULL,
  n = 100L,
  method = c("kde", "surrogate", "bootstrap"),
  kde = NULL,
  seed = NULL,
  ...
)
```

Arguments

`data`	A numeric vector or matrix.
`n`	The number of samples to draw.
`method`	The method to use for sampling.
`kde`	The kernel density estimate to use for sampling.
`seed`	The seed to use for sampling.
`...`	Additional arguments to pass to the density function.

`extract_coefficients` *Extract coefficients and intercept from various model types*

Description

Extract coefficients and intercept from various model types

Usage

```
extract_coefficients(model, model_type)
```

fit_linear_model *Fit linear model based on specified type*

Description

Fit linear model based on specified type

Usage

```
fit_linear_model(X, y, model_type, lambda, tau, alpha)
```

fit_param_dist *Fit multiple parametric distributions, compute KL divergence, simulate best fit*

Description

Fit multiple parametric distributions, compute KL divergence, simulate best fit

Usage

```
fit_param_dist(vector, num_bins = 30, verbose = TRUE)
```

Arguments

vector	Numeric vector of data to fit
num_bins	Number of bins for the empirical histogram
verbose	Logical indicating whether to print results

Value

Function to simulate data from the best-fitting distribution

Examples

```
set.seed(123)
n <- 1000
vector <- rnorm(n)

start <- proc.time()[3]
simulate_function <- fit_param_dist(vector)
end <- proc.time()[3]
print(paste("Time taken:", end - start))
simulated_data <- simulate_function(n)  # Generate 100 samples from the best-fit distribution
par(mfrow = c(1, 2))
hist(vector, main = "Original Data", xlab = "Value", ylab = "Frequency")
hist(simulated_data, main = "Simulated Data", xlab = "Value", ylab = "Frequency")
```

`fit_predict_shap` *Fit a linear model and compute SHAP values with response centering*

Description

Fit a linear model and compute SHAP values with response centering

Usage

```
fit_predict_shap(
  X,
  y,
  newx,
  model_type = "lm",
  center_response = TRUE,
  account_correlations = FALSE,
  n_samples = 100,
  lambda = 0.01,
  tau = 0.5,
  alpha = 1
)
```

Arguments

`X`	Training feature matrix or data frame
`y`	Training response vector
`newx`	New feature matrix or data frame for prediction/explanation
`model_type`	Type of model: "lm", "rlm", "rq", "glmnet"
`center_response`	Whether to center the response variable
`account_correlations`	Whether to use correlation-aware SHAP
`n_samples`	Number of samples for correlation-aware computation
`lambda`	Regularization parameter for glmnet
`tau`	Quantile for quantile regression
`alpha`	Elastic net mixing parameter for glmnet

Value

List with model, predictions, SHAP values, and validation results

get_model_predictions *Get predictions from various model types*

Description

Get predictions from various model types

Usage

```
get_model_predictions(model, newx, model_type)
```

get_model_type *Get model type string*

Description

Get model type string

Usage

```
get_model_type(model)
```

get_next_date *Get next date for a time series object*

Description

Get next date for a time series object

Usage

```
get_next_date(y)
```

Arguments

y	A time series object

Examples

```
get_next_date(AirPassengers)
get_next_date(lynx)
get_next_date(USAccDeaths)
```

gradient_shap *Gradient-Based SHAP for Any Differentiable Function*

Description

This implements SHAP-like explanations using finite differences to approximate gradients for any function f, extending the linear SHAP concept to nonlinear models. Compute gradient-based SHAP values for any function

Usage

```
gradient_shap(
  f,
  X_train,
  X_new,
  h = 1e-05,
  baseline_method = "mean",
  scale_by_range = FALSE
)
```

Arguments

f	Function to explain (should accept matrix input, return vector output)
X_train	Training data (for computing baseline and feature means)
X_new	New observations to explain
h	Step size for finite differences (default: 1e-5)
baseline_method	Method for computing baseline: "mean", "median", "zero", or custom vector
scale_by_range	Whether to scale gradients by feature ranges

Value

List with gradient-SHAP values, baseline, and predictions

integrate *Numerical Integration Using Trapezoidal or Simpson's Rule*

Description

Computes the definite integral of a function using either the trapezoidal rule or Simpson's rule for numerical approximation.

Usage

```
integrate(f, a, b, n = 10000, method = "simpson")
```

Arguments

f	A function to be integrated (must be vectorized)
a	Lower bound of integration (numeric)
b	Upper bound of integration (numeric, must be > a)
n	Number of subdivisions (default = 10000, must be >= 2)
method	Integration method: "trapezoidal" or "simpson" (default)

Value

The approximated value of the integral (numeric)

Examples

```
# Integrate x^2 from 0 to 1 (exact answer = 1/3)
integrate(function(x) x^2, 0, 1)

# Compare methods
integrate(sin, 0, pi, method = "trapezoidal")
integrate(sin, 0, pi, method = "simpson")

# Vectorized function example
integrate(dnorm, -1.96, 1.96) # Approx 0.95
```

---

integrated_gradients *Integrated Gradients for Model Explanations*

Description

Computes feature attributions using the Integrated Gradients method. This integrates gradients along the straight-line path from a baseline to each input.

Usage

```
integrated_gradients(
  f,
  X_train,
  X_new,
  baseline_method = "mean",
  n_steps = 50,
  h = 1e-05
)
```

Arguments

f	Function to explain (should accept matrix input, return vector output)
X_train	Training data (for computing baseline)
X_new	New observations to explain
baseline_method	Method for computing baseline: "mean", "median", or "zero"
n_steps	Number of steps for the integration path (default: 50)
h	Step size for finite differences (default: 1e-5)

Value

List with attributions, integrated gradients, baseline, predictions, and residual diagnostics

Examples

```
# integrated_gradients(f, X_train, X_new)
```

---

is_package_available *Check if a package is available*

Description

Check if a package is available

Usage

```
is_package_available(pkg_name)
```

Arguments

pkg_name	A package name

Value

A logical value

Examples

```
misc::is_package_available("dplyr")
```

is_wholenumber *Check if a number is a whole number*

Description

Check if a number is a whole number

Usage

```
is_wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

x	A number
tol	A tolerance level

Value

A logical value

Examples

```
is_wholenumber(1)
is_wholenumber(1.1)
is_wholenumber(1L)
```

KL_divergence_hist *Function to calculate KL divergence for continuous distributions using histograms*

Description

Function to calculate KL divergence for continuous distributions using histograms

Usage

```
KL_divergence_hist(P, Q)
```

Arguments

P	Numeric vector representing the empirical distribution
Q	Numeric vector representing the theoretical distribution

Value

KL divergence between P and Q

Examples

```
P <- c(0.2, 0.3, 0.5)
Q <- c(0.1, 0.4, 0.5)
misc::KL_divergence_hist(P, Q)
```

`local_linear_shap` *LIME-style local linear approximation using gradients*

Description

LIME-style local linear approximation using gradients

Usage

```
local_linear_shap(f, X_train, X_new, n_samples = 1000, sigma = 0.1, h = 1e-05)
```

`monotonic_regression` *Monotonic Regression using XGBoost, LightGBM, or SCAM*

Description

Monotonic Regression using XGBoost, LightGBM, or SCAM

Usage

```
monotonic_regression(
  x,
  y,
  method = c("xgboost", "lightgbm", "scam"),
  monotonicity = 1,
  prediction = TRUE,
  strict = FALSE,
  ...
)
```

Arguments

`x`	Numeric vector or matrix of predictor variables
`y`	Numeric vector of response variable
`method`	Either "xgboost", "lightgbm", or "scam"
`monotonicity`	1 for non-decreasing, -1 for non-increasing
`prediction`	A boolean; obtain predictions on training set (TRUE) or return model object
`strict`	If TRUE, enforces strict monotonicity (for xgboost/lightgbm only; scam is always strict)
`...`	Additional parameters passed to the underlying algorithm

Value

A list with fitted values and model object

Examples

```
# Strict decreasing example with xgboost
x <- 1:5
y <- c(0.98, 0.99, 0.97, 0.95, 0.8)
result <- monotonic_regression(x, y, method = "xgboost", monotonicity = -1)
print(result$fitted)
```

---

mspl *Mean Scaled Pinball Loss for Quantile Forecasts*

Description

Calculates the Mean Scaled Pinball Loss (MSPL) for quantile forecasts, scaled by the Mean Absolute Error (MAE) of the median forecast.

Usage

```
mspl(
  y_true,
  y_pred,
  probs = c(0.005, 0.025, 0.165, 0.25, 0.5, 0.75, 0.835, 0.975, 0.995)
)
```

Arguments

`y_true`	Numeric vector of observed values
`y_pred`	Numeric matrix or data.frame of predicted quantiles (rows = observations, columns = quantiles)
`probs`	Numeric vector of quantile levels (must be between 0 and 1) Default `c(0.005, 0.025, 0.165, 0.25, 0.5, 0.75, 0.835, 0.975, 0.995)`

Value

Numeric scalar representing the MSPL

Examples

```
y_true <- c(1, 2, 3, 4, 5)
y_pred <- matrix(c(0.5, 1.5, 2.5, 3.5, 4.5,
                   1.0, 2.0, 3.0, 4.0, 5.0,
                   1.5, 2.5, 3.5, 4.5, 5.5), ncol = 3)
mspl(y_true, y_pred, probs=c(0.25, 0.5, 0.75))
```

one_hot_encode *One-hot encoding*

Description

One-hot encoding

Usage

```
one_hot_encode(y)
```

Arguments

y	A vector of class labels
n_classes	The number of classes

Value

A matrix of one-hot encoded labels

Examples

```
y <- as.factor(c(1, 2, 1, 1, 2))
misc::one_hot_encode(y)
```

parfor *Parallel For Loop with Progress Bar*

Description

Execute a function over a list of arguments in parallel or sequentially, with optional progress tracking and flexible result combination.

Usage

```
parfor(
  what,
  args,
  cl = NULL,
  combine = "list",
  errorhandling = c("stop", "remove", "pass"),
  verbose = FALSE,
  show_progress = TRUE,
  export = NULL,
  packages = NULL,
```

```
  cl_type = "PSOCK",
  use_do_call = FALSE,
  ...
)
```

Arguments

what	A function to apply to each element of args.
args	A list of arguments to iterate over. Each element will be passed to what.
cl	Number of cores to use for parallel execution. NULL (default) for sequential execution, -1 to use all available cores, or a positive integer specifying the number of cores.
combine	Method for combining results. Can be: • A string: "list" (default), "c", "cbind", "rbind", or arithmetic operators ("+", etc.) • A function: custom function taking ... arguments
errorhandling	How to handle errors. One of: • "stop": stop execution on first error (default) • "remove": remove failed iterations from results • "pass": return error objects for failed iterations
verbose	Logical. If TRUE, print verbose output from foreach.
show_progress	Logical. If TRUE (default), display a progress bar.
export	Character vector of variable names to export to parallel workers.
packages	Character vector of package names to load on parallel workers.
cl_type	Type of cluster to create. One of "PSOCK" (default), "FORK", "MPI", or "NWS".
use_do_call	Logical. If TRUE, use do.call to apply what to arguments. Useful when args contains lists of arguments.
...	Additional arguments passed to what.

Details

This function provides a unified interface for sequential and parallel iteration with progress tracking. When cl is NULL, execution is sequential. Otherwise, a parallel cluster is created using the parallel and doSNOW packages.

The combine parameter determines how results are aggregated:

- "list": Keep each result as a separate list element (safest default)
- "c": Concatenate results with c()
- "rbind"/"cbind": Bind data frames or matrices by row/column
- Custom function: Provide your own combining logic

Value

Combined results from applying what to each element of args. The structure depends on the combine parameter.

Examples

```
# Sequential execution with list output
parfor(sqrt, list(1, 4, 9, 16))

# Parallel execution on 2 cores, combine with c()
parfor(sqrt, list(1, 4, 9, 16), cl = 2, combine = "c")

# Use do.call for functions expecting multiple arguments
args_list <- list(list(x = 1, y = 2), list(x = 3, y = 4))
parfor(function(x, y) x + y, args_list, use_do_call = TRUE)

# Custom combine function
parfor(sqrt, list(1, 4, 9, 16), combine = sum)

# Combine data frames by row
parfor(function(x) data.frame(x = x, y = x^2),
       list(1, 2, 3), combine = "rbind")
```

---

plot_prediction_interval

Plot Prediction Interval With Optional Past Observations

Description

This function creates a time series plot showing prediction intervals, including optional past observations. The forecast is visualized with a confidence interval band and mean prediction line.

Usage

```
plot_prediction_interval(
  mean_pred,
  lower_pred,
  upper_pred,
  col_past = "black",
  col_mean = "red",
  col_future = "blue",
  col_ci = rgb(0.8, 0.8, 0.8, 0.5),
  legend_position = "topleft",
  x_past = NULL,
  x_future = NULL,
  ...
)
```

Arguments

mean_pred	Numeric vector of mean forecasted values. Required.
lower_pred	Numeric vector of lower bounds of the prediction interval. Required.

upper_pred	Numeric vector of upper bounds of the prediction interval. Required.
col_past	Color for past observations. Default is '"black"'.
col_mean	Color for the mean forecast line. Default is '"red"'.
col_future	Color for true future value. Default is '"blue"'.
col_ci	Color for the confidence interval polygon. Default is 'rgb(0.8, 0.8, 0.8, 0.5)'.
legend_position	Position of the legend. Passed to the 'legend()' function. Default is '"topleft"'.
x_past	Numeric vector of past observations. If 'NULL', the plot will contain only forecasted values. Default is 'NULL'.
x_future	Numeric vector of future observations.
...	Additional arguments passed to the 'plot()' function.

Value

No return value. Called for side effects (generates a plot).

Examples

```
# Without past observations:
plot_prediction_interval(
  mean_pred = sin(1:10/3) + 1,
  lower_pred = sin(1:10/3) + 1 - 0.5,
  upper_pred = sin(1:10/3) + 1 + 0.5,
  main = "Forecast Without History"
)

# With past observations:
plot_prediction_interval(
  x_past = sin(1:20/6),
  mean_pred = sin(21:30/6) + 0.1 * (1:10),
  lower_pred = sin(21:30/6) + 0.1 * (1:10) - 0.3,
  upper_pred = sin(21:30/6) + 0.1 * (1:10) + 0.3,
  main = "Forecast With History"
)

# With past observations:
set.seed(123)
mean_pred <- sin(21:30/6) + 0.1 * (1:10)
plot_prediction_interval(
  x_past = sin(1:20/6),
  mean_pred = mean_pred,
  x_future = mean_pred + runif(length(mean_pred)),
  lower_pred = sin(21:30/6) + 0.1 * (1:10) - 0.3,
  upper_pred = sin(21:30/6) + 0.1 * (1:10) + 0.3,
  main = "Forecast With History"
)
```

predict.conformalize *Predict method for conformalize objects*

Description

This function generates prediction intervals for new data using the fitted conformal model.

Usage

```
## S3 method for class 'conformalize'
predict(
  object,
  newdata,
  predict_func = predict,
  level = 0.95,
  method = c("splitconformal", "kde", "surrogate", "bootstrap"),
  n_sim = 250L,
  seed = 123L,
  ...
)
```

Arguments

object	A conformalize object.
newdata	A data frame or matrix of new data for prediction.
level	The confidence level for the prediction intervals. Default is 0.95.
method	The method to use for prediction intervals. Options are "split" or "simulation".
n_sim	The number of simulations to perform if using the "simulation" method. Default is 1000.
seed	An optional seed for reproducibility.
...	Additional arguments to pass to the predict function.

Value

A matrix with predictions and prediction intervals.

Examples

```
## Not run:
# Define fit and predict functions
fit_func <- function(formula, data, ...) stats::glm(formula, data = data, ...)
predict_func <- function(fit, newdata, ...) predict(fit, newdata, ...)

# Apply conformalize using the training data
conformal_model_boston <- misc::conformalize(
  formula = medv ~ .,
```

```
  data = train_data,
  fit_func = fit_func,
  predict_func = predict_func,
  seed = 123
)

# Predict with split conformal method on the test data
predictions_boston <- predict(
  conformal_model_boston,
  newdata = test_data,
  level = 0.95,
  method = "split"
)

head(predictions_boston)

## End(Not run)
```

`print_shap_summary` *Print summary of SHAP results*

Description

Print summary of SHAP results

Usage

```
print_shap_summary(result)
```

`residuals.conformalize`
Residuals method for conformalize objects

Description

This function extracts the residuals from a conformalize object.

Usage

```
## S3 method for class 'conformalize'
residuals(object, ...)
```

Arguments

`object`	A conformalize object.
`...`	Additional arguments (not used).

Value

A numeric vector of residuals.

`rm_zero_cols` *Removing columns containing only zeros*

Description

Removing columns containing only zeros

Usage

```
rm_zero_cols(X)
```

Arguments

`X`	A matrix or data frame

Value

A matrix or data frame

Examples

```
X <- matrix(c(1, 0, 3, 0, 5, 0, 0, 0), nrow = 2)
print(misc::rm_zero_cols(X))
```

`rmultivariate` *Simulate multivariate data*

Description

Simulate multivariate data

Usage

```
rmultivariate(
  data,
  method = c("bootstrap", "block-bootstrap"),
  n = 100L,
  block_size = 5
)
```

Arguments

data	A numeric vector or matrix.
method	The method to use for sampling.
n	The number of samples to draw.
block_size	The size of the blocks to use for the block bootstrap.
...	Additional arguments to pass to the density function.

---

scale_matrix *Scale matrix*

Description

Scale matrix

Usage

```
scale_matrix(X, X_mean = NULL, X_sd = NULL)
```

Arguments

X	A matrix
X_mean	Mean of each column
X_sd	Standard deviation of each column

Value

A list containing the scaled matrix, mean of each column, and standard deviation of each column

Examples

```
X <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
(X_scaled <- misc::scale_matrix(X))
(X_scaled <- misc::scale_matrix(X, X_mean = colMeans(X), X_sd = apply(X, 2, stats::sd)))
print(colMeans(X_scaled$X))
print(apply(X_scaled$X, 2, stats::sd))
```

`sims_to_forecast_object` *Convert Simulation Matrix to Forecast Object*

Description

Transforms a matrix of simulated future paths into a `forecast` class object containing point forecasts (means) and prediction intervals, while preserving time series properties.

Usage

```
sims_to_forecast_object(y, sims, level = 95, keep_first_row = TRUE)
```

Arguments

`y`	The original time series (of class `ts`) used to generate simulations.
`sims`	A matrix of simulated future paths where each column represents one simulation path and each row represents a time period. Must have same frequency as `y`.
`level`	The confidence level for prediction intervals (default: 95).
`keep_first_row`	Logical indicating whether to retain the first row of `sims`. If `FALSE` (default), first row is dropped (assumed to be present-day value).

Details

The function:

1. Calculates point forecasts as row means of `sims`
2. Computes prediction intervals using quantiles at `(1-level)/2` and `1-(1-level)/2`
3. Converts all outputs to `ts` objects with correct temporal alignment
4. Optionally handles initial condition rows via `keep_first_row`

Value

An object of class `forecast` containing:

- `x`: Original time series
- `mean`: Point forecasts (row means of simulations)
- `lower`: Lower prediction bounds
- `upper`: Upper prediction bounds
- `sims`: Full simulation matrix (as time series)
- `level`: Confidence level used

All time series outputs maintain the original frequency and start at the next observation after `y`.

See Also

forecast for standard forecast objects, quantile for interval calculation method

Examples

```
## Not run:
library(forecast)
y <- ts(rnorm(20), frequency=4)
sims <- matrix(rnorm(200), nrow=10, ncol=20) # 10 steps, 20 simulations
fc <- sims_to_forecast_object(y, sims, level=80)
autoplot(fc)  # Requires ggplot2 and forecast packages

## End(Not run)
```

---

simulate.conformalize *simulate method for conformalize objects*

Description

This function generates simulations for new data using the fitted conformal model.

Usage

```
## S3 method for class 'conformalize'
simulate(
  object,
  newdata,
  method = c("kde", "surrogate", "bootstrap"),
  n_sim = 250L,
  seed = NULL,
  ...
)
```

Arguments

object	A object of class conformalize.
newdata	A data frame or matrix of new data for prediction.
method	The method to use for prediction intervals. Options are "split" or "simulation".
n_sim	The number of simulations to perform if using the "simulation" method. Default is 1000.
seed	An optional seed for reproducibility.
...	Additional arguments to pass to the predict function.

Value

A matrix with predictions and prediction intervals.

sort_df *Sort data frame*

Description

Sort data frame

Usage

```
sort_df(df, by, decreasing = FALSE)
```

Arguments

df	data frame
by	column to sort by
decreasing	logical. Should sorting be decreasing?

Value

A sorted data frame

Examples

```
df <- data.frame(a = c(2, 4, 3), b = c(3, 5, 1))
misc::sort_df(df, "a")
misc::sort_df(df, "b", decreasing = TRUE)
```

split_data *Split a dataset*

Description

Split a dataset

Usage

```
split_data(y, p = 0.5, seed = 123, type_split = c("stratify", "sequential"))
```

Arguments

y	A vector of labels
p	A proportion of the dataset to split
seed	An integer to set the seed
type_split	A character string specifying the type of split

Value

A vector of indices

Examples

```
set.seed(123)
(y <- rnorm(10))
misc::split_data(y, 0.5)
misc::split_data(y, 0.5, type_split = "sequential")
```

`splitts` *Partition a time series object*

Description

Partition a time series object

Usage

```
splitts(y, split_prob = 0.5, return_indices = FALSE)
```

Arguments

`y`	A time series object
`split_prob`	Splitting ratio
`return_indices`	if TRUE, returns series' indices, otherwise, time series objects

Examples

```
misc::splitts(ts(1:10))
```

`time_series_accuracy` *Calculate Multiple Accuracy Metrics for Forecasts*

Description

Computes a comprehensive set of accuracy metrics to evaluate forecast performance, including error measures, scaled metrics, and interval evaluation scores.

Usage

```
time_series_accuracy(obj, actual, level = 95)
```

Arguments

`obj`	A forecast object containing at minimum a `mean` component with point forecasts and an `x` component with the original time series. Should also contain prediction intervals (either `lower`/`upper` or `intervals` matrix) for coverage and Winkler score calculations.
`actual`	Numeric vector of actual observed values to compare against the forecasts.
`level`	Numeric value specifying the confidence level of the prediction intervals (default: 95). Used for Winkler score calculation.

Details

The function calculates both point forecast accuracy metrics (ME, RMSE, MAE, MPE, MAPE, MASE) and interval forecast metrics (coverage, Winkler score).

The MASE (Mean Absolute Scaled Error) uses seasonal naive forecasts as the benchmark, with the scaling factor determined by the frequency of the original time series. For seasonal data (frequency > 1), it uses seasonal differences; for non-seasonal data, it uses first differences.

Value

Named numeric vector containing the following accuracy metrics:

me Mean Error - average of forecast errors

rmse Root Mean Square Error - square root of average squared errors

mae Mean Absolute Error - average of absolute forecast errors

mpe Mean Percentage Error - average of percentage errors

mape Mean Absolute Percentage Error - average of absolute percentage errors

mase Mean Absolute Scaled Error - MAE scaled by seasonal naive forecast MAE

coverage Coverage score - percentage of actuals within prediction intervals

winkler Winkler score - penalized interval width and coverage failures

See Also

`coverage_score`, `winkler_score2`

Examples

```
## Not run:
# Example with a forecast object
library(forecast)
ts_data <- ts(rnorm(50), frequency = 12)
fit <- auto.arima(ts_data[1:40])
fc <- forecast(fit, h = 10)
actual_test <- ts_data[41:50]

# Calculate all accuracy metrics
metrics <- accuracy(fc, actual_test)
print(metrics)
```

```
## End(Not run)
```

`timeit` *Timing an expression*

Description

Timing an expression

Usage

```
timeit(expr, times = 1, ...)
```

Arguments

`expr`	an R expression
`times`	number of repetitions
`...`	additional arguments passed to `base::eval`

Value

the elapsed time in seconds

Examples

```
timeit(1 + 1)
timeit(1 + 1, times = 10)
```

`validate_shap_decomposition` *Validate SHAP decomposition*

Description

Validate SHAP decomposition

Usage

```
validate_shap_decomposition(shap_values, base_value, predictions)
```

vlookup *VLOOKUP*

Description

A simple implementation similar to the VLOOKUP function in Excel.

Usage

```
vlookup(this, df, key, value)
```

Arguments

this	The value to look up
df	A data frame
key	The column to look up
value	The column to return

Value

The value in the ‘value‘ column corresponding to the ‘key‘ column

Examples

```
df <- data.frame(key = c("a", "b", "c"), value = c(1, 2, 3))
print(misc::vlookup("b", df, "key", "value"))
```

winkler_score *Winkler score for probabilistic forecasts*

Description

Winkler score for probabilistic forecasts

Usage

```
winkler_score(actual, lower, upper, level = 95, scale = FALSE)
```

Arguments

actual	numeric vector of actual values
lower	numeric vector of lower bounds
upper	numeric vector of upper bounds
level	numeric level of confidence
scale	logical, if TRUE, the score is scaled by the range of the bounds

Value

numeric score

Examples

```
actual <- c(1, 2, 3, 4, 5)
lower <- c(0, 1, 2, 3, 4)
upper <- c(2, 3, 4, 5, 6)
winkler_score(actual, lower, upper)
winkler_score(actual, lower, upper, scale = TRUE)
winkler_score(actual, lower, upper, level = 99)
winkler_score(actual, lower, upper, level = 99, scale = TRUE)
```

---

winkler_score2 *Calculate Winkler Score for Prediction Intervals*

Description

Computes the Winkler score, which evaluates the quality of prediction intervals by penalizing both the width of intervals and coverage failures. Lower scores indicate better performance.

Usage

```
winkler_score2(obj, actual, level = 95)
```

Arguments

obj	A forecast object containing prediction intervals. Should have either lower and upper components or an intervals matrix with lower bounds in column 1 and upper bounds in column 2.
actual	Numeric vector of actual observed values to compare against the prediction intervals.
level	Numeric value specifying the confidence level of the prediction intervals (default: 95). Used to calculate the penalty factor alpha.

Details

The Winkler score is calculated as:

$$S = (U - L) + \frac{2}{\alpha}\max(L - y, 0) + \frac{2}{\alpha}\max(y - U, 0)$$

where:

- U and L are the upper and lower bounds of the prediction interval
- y is the actual observed value
- alpha = 1 - level/100 is the significance level

The score penalizes wide intervals (first term) and coverage failures (second and third terms). The penalty for coverage failures increases as the confidence level increases.

Value

Numeric value representing the mean Winkler score. Lower values indicate better interval forecasts.

References

Winkler, R. L. (1972). A decision-theoretic approach to interval estimation. Journal of the American Statistical Association, 67(337), 187-191.

Examples

```
## Not run:
# Example usage
forecast_obj <- list(lower = c(10, 15, 20), upper = c(20, 25, 30))
actual_values <- c(12, 18, 35)  # Note: 35 is outside the last interval
winkler_score2(forecast_obj, actual_values, level = 95)

# With 80% confidence intervals
winkler_score2(forecast_obj, actual_values, level = 80)

## End(Not run)
```

Index

base::eval, *29*

compute_finite_difference_gradients, 2
compute_linear_shap, 3
conformalize, 3
correlation_aware_shap, 4
coverage_score, 4, *28*

debug_print, 5
direct_sampling, 6

extract_coefficients, 6

fit_linear_model, 7
fit_param_dist, 7
fit_predict_shap, 8
forecast, *25*

get_model_predictions, 9
get_model_type, 9
get_next_date, 9
gradient_shap, 10

integrate, 10
integrated_gradients, 11
is_package_available, 12
is_wholenumber, 13

KL_divergence_hist, 13

local_linear_shap, 14

monotonic_regression, 14
mspl, 15

one_hot_encode, 16

parfor, 16
plot_prediction_interval, 18
predict.conformalize, 20
print_shap_summary, 21

quantile, *25*

residuals.conformalize, 21
rm_zero_cols, 22
rmultivariate, 22

scale_matrix, 23
sims_to_forecast_object, 24
simulate.conformalize, 25
sort_df, 26
split_data, 26
splitts, 27

time_series_accuracy, 27
timeit, 29

validate_shap_decomposition, 29
vlookup, 30

winkler_score, 30
winkler_score2, *28*, 31