Package: dubicube (via r-universe)

February 17, 2025

Title Calculation and Interpretation of Data Cube Indicator Uncertainty

Version 0.3.0

Description This R package provides functions to explore data cubes using robustness measures and cross-validation techniques. It can also be used for uncertainty calculation using the bootstrap resampling method, and functionality is provided for efficient interpretation and visualisation of uncertainty related to indicators based on biodiversity data cubes.

License MIT + file LICENSE

URL https://github.com/b-cubed-eu/dubicube, https://b-cubed-eu.github.io/dubicube/, https://doi.org/10.5281/zenodo.14850237

BugReports https://github.com/b-cubed-eu/dubicube/issues

Imports assertthat, boot, data.table, dplyr, effectclass, modelr, purrr, rlang, tibble, tidyr

Suggests b3gbi, testthat (>= 3.0.0)

Remotes github::b-cubed-eu/b3gbi, github::inbo/effectclass

Additional_repositories https://inbo.r-universe.dev, https://b-cubed-eu.r-universe.dev/

Config/checklist/communities b3; inbo

Config/checklist/keywords uncertainty quantification; uncertainty visualisation; biodiversity indicators; data cubes

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs make libicu-dev libssl-dev

Repository https://b-cubed-eu.r-universe.dev

RemoteUrl https://github.com/b-cubed-eu/dubicube

RemoteRef HEAD

RemoteSha f597beb2a4f1666b9ee24ed6c12397db288a6e5d

Contents

`add_effect_classification`

Add effect classifications to a dataframe by comparing the confidence intervals with a reference and thresholds

---

Description

This function adds classified effects to a dataframe as ordered factor variables by comparing the confidence intervals with a reference and thresholds.

Usage

```
add_effect_classification(
  df,
  cl_columns,
  threshold,
  reference = 0,
  coarse = TRUE
)
```

Arguments

`df`	A dataframe containing summary data of confidence limits. Two columns are required containing lower and upper limits indicated by the `cl_columns` argument. Any other columns are optional.
`cl_columns`	A vector of 2 column names in `df` indicating respectively the lower and upper confidence limits (e.g. `c("lcl", "ucl")`).
`threshold`	A vector of either 1 or 2 thresholds. A single threshold will be transformed into `reference + c(-abs(threshold), abs(threshold))`.
`reference`	The null hypothesis value to compare confidence intervals against. Defaults to 0.
`coarse`	Logical, defaults to `TRUE`. If `TRUE`, add a coarse classification to the dataframe.

Details

This function is a wrapper around `effectclass::classify()` and `effectclass::coarse_classification()` from the **effectclass** package (Onkelinx, 2023). They classify effects in a stable and transparent manner.

Symbol	Fine effect / trend	Coarse effect / trend	Rule
++	strong positive effect / strong increase	positive effect / increase	confidence interval above the upper thr
+	positive effect / increase	positive effect / increase	confidence interval above reference an
+~	moderate positive effect / moderate increase	positive effect / increase	confidence interval between reference
~	no effect / stable	no effect / stable	confidence interval between threshold
-~	moderate negative effect / moderate decrease	negative effect / decrease	confidence interval between reference
-	negative effect / decrease	negative effect / decrease	confidence interval below reference an
--	strong negative effect / strong decrease	negative effect / decrease	confidence interval below the lower thr
?+	potential positive effect / potential increase	unknown effect / unknown	confidence interval contains reference
?-	potential negative effect / potential decrease	unknown effect / unknown	confidence interval contains reference
?	unknown effect / unknown	unknown effect / unknown	confidence interval contains the lower

Value

The returned value is a modified version of the original input dataframe `df` with additional columns `effect_code` and `effect` containing respectively the effect symbols and descriptions as ordered factor variables. In case of `coarse = TRUE` (by default) also `effect_code_coarse` and `effect_coarse` containing the coarse classification effects.

References

Onkelinx, T. (2023). effectclass: Classification and visualisation of effects [Computer software]. https://inbo.github.io/effectclass/

See Also

Other uncertainty: `bootstrap_cube()`, `calculate_bootstrap_ci()`

Examples

```
# Example dataset
ds <- data.frame(
  mean = c(0, 0.5, -0.5, 1, -1, 1.5, -1.5, 0.5, -0.5, 0),
  sd = c(1, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.5)
)
ds$lcl <- qnorm(0.05, ds$mean, ds$sd)
ds$ucl <- qnorm(0.95, ds$mean, ds$sd)

add_effect_classification(
 df = ds,
 cl_columns = c("lcl", "ucl"),
 threshold = 1,
 reference = 0,
 coarse = TRUE)
```

bootstrap_cube *Perform bootstrapping over a data cube for a calculated statistic*

Description

This function generate `samples` bootstrap replicates of a statistic applied to a data cube. It resamples the data cube and computes a statistic `fun` for each bootstrap replicate, optionally comparing the results to a reference group (`ref_group`).

Usage

```
bootstrap_cube(
  data_cube,
  fun,
  ...,
  grouping_var,
  samples = 1000,
  ref_group = NA,
  seed = NA,
  progress = FALSE
)
```

Arguments

`data_cube`	A data cube object (class 'processed_cube' or 'sim_cube', see `b3gbi::process_cube()`) or a dataframe (from `$data` slot of 'processed_cube' or 'sim_cube'). To limit runtime, we recommend using a dataframe with custom function as `fun`.
`fun`	A function which, when applied to `data_cube` returns the statistic(s) of interest. This function must return a dataframe with a column `diversity_val` containing the statistic of interest.
`...`	Additional arguments passed on to `fun`.
`grouping_var`	A string specifying the grouping variable(s) for the bootstrap analysis. The output of `fun(data_cube)` returns a row per group.
`samples`	The number of bootstrap replicates. A single positive integer. Default is 1000.
`ref_group`	A string indicating the reference group to compare the statistic with. Default is `NA`, meaning no reference group is used.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.
`progress`	Logical. Whether to show a progress bar. Set to `TRUE` to display a progress bar, `FALSE` (default) to suppress it.

Details

Bootstrapping is a statistical technique used to estimate the distribution of a statistic by resampling with replacement from the original data (Davison & Hinkley, 1997; Efron & Tibshirani, 1994). In the case of data cubes, each row is sampled with replacement. Below are the common notations used in bootstrapping:

1. **Original Sample Data**: $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$
   - The initial set of observed data points. Here, n is the sample size. This corresponds to the number of cells in a data cube or the number of rows in tabular format.
2. **Statistic of Interest**: θ
   - The parameter or statistic being estimated, such as the mean $\bar{X}$, variance σ^2, or a biodiversity indicator. Let $\hat{\theta}$ denote the estimated value of θ calculated from the complete dataset $\mathbf{X}$.
3. **Bootstrap Sample**: $\mathbf{X}^* = \{X_1^*, X_2^*, \ldots, X_n^*\}$
   - A sample of size n drawn with replacement from the original sample $\mathbf{X}$. Each X_i^* is drawn independently from $\mathbf{X}$.
   - A total of B bootstrap samples are drawn from the original data. Common choices for B are 1000 or 10,000 to ensure a good approximation of the distribution of the bootstrap replications (see further).
4. **Bootstrap Replication**: $\hat{\theta}_b^*$
   - The value of the statistic of interest calculated from the b-th bootstrap sample $\mathbf{X}_b^*$. For example, if θ is the sample mean, $\hat{\theta}_b^* = \bar{X}_b^*$.
5. **Bootstrap Statistics**:

- **Bootstrap Estimate of the Statistic**: $\hat{\theta}_{\text{boot}}$
  - The average of the bootstrap replications:

$$\hat{\theta}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$$

- **Bootstrap Bias**: $\text{Bias}_{\text{boot}}$
  - This bias indicates how much the bootstrap estimate deviates from the original sample estimate. It is calculated as the difference between the average bootstrap estimate and the original estimate:

$$\text{Bias}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} (\hat{\theta}_b^* - \hat{\theta}) = \hat{\theta}_{\text{boot}} - \hat{\theta}$$

- **Bootstrap Standard Error**: SE_{boot}
  - The standard deviation of the bootstrap replications, which estimates the variability of the statistic.

Value

A dataframe containing the bootstrap results with the following columns:

- `sample`: Sample ID of the bootstrap replicate
- `est_original`: The statistic based on the full dataset per group
- `rep_boot`: The statistic based on a bootstrapped dataset (bootstrap replicate)
- `est_boot`: The bootstrap estimate (mean of bootstrap replicates per group)
- `se_boot`: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- `bias_boot`: The bias of the bootstrap estimate per group

References

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap (1st ed.). Chapman and Hall/CRC. doi:10.1201/9780429246593

See Also

Other uncertainty: `add_effect_classification()`, `calculate_bootstrap_ci()`

Examples

```
# Get example data
# install.packages("remotes")
# remotes::install_github("b-cubed-eu/b3gbi")
library(b3gbi)
cube_path <- system.file(
  "extdata", "denmark_mammals_cube_eqdgc.csv",
  package = "b3gbi")
denmark_cube <- process_cube(
  cube_path,
  first_year = 2014,
  last_year = 2020)

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(data) {
  out_df <- aggregate(obs ~ year, data, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(denmark_cube$data)

# Perform bootstrapping

bootstrap_mean_obs <- bootstrap_cube(
  data_cube = denmark_cube$data,
```

```
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  progress = FALSE)
head(bootstrap_mean_obs)
```

`calculate_bootstrap_ci` *Calculate confidence intervals for a dataframe with bootstrap replicates*

Description

This function calculates confidence intervals for a dataframe containing bootstrap replicates based on different methods, including percentile (`perc`), bias-corrected and accelerated (`bca`), normal (`norm`), and basic (`basic`).

Usage

```
calculate_bootstrap_ci(
  bootstrap_samples_df,
  grouping_var,
  type = c("perc", "bca", "norm", "basic"),
  conf = 0.95,
  aggregate = TRUE,
  data_cube = NA,
  fun = NA,
  ...,
  ref_group = NA,
  jackknife = ifelse(is.element("bca", type), "usual", NA),
  progress = FALSE
)
```

Arguments

`bootstrap_samples_df`
: A dataframe containing the bootstrap replicates, where each row represents a bootstrap sample. As returned by `bootstrap_cube()`. Apart from the `grouping_var` column, the following columns should be present:

  - `est_original`: The statistic based on the full dataset per group
  - `rep_boot`: The statistic based on a bootstrapped dataset (bootstrap replicate)

`grouping_var`
: A string specifying the grouping variable(s) used for the bootstrap analysis. This variable is used to split the dataset into groups for separate confidence interval calculations.

`type` A character vector specifying the type(s) of confidence intervals to compute. Options include:

- `"perc"`: Percentile interval
- `"bca"`: Bias-corrected and accelerated interval
- `"norm"`: Normal interval
- `"basic"`: Basic interval
- `"all"`: Compute all available interval types (default)

`conf` A numeric value specifying the confidence level of the intervals. Default is `0.95` (95 % confidence level).

`aggregate` Logical. If `TRUE` (default), the function returns distinct confidence limits per group. If `FALSE`, the confidence limits are added to the original bootstrap dataframe `bootstrap_samples_df`.

`data_cube` Only used when `type = "bca"`. A data cube object (class 'processed_cube' or 'sim_cube', see `b3gbi::process_cube()`) or a dataframe (from `$data` slot of 'processed_cube' or 'sim_cube'). As used by `bootstrap_cube()`. To limit runtime, we recommend using a dataframe with custom function as `fun`.

`fun` Only used when `type = "bca"`. A function which, when applied to `data_cube` returns the statistic(s) of interest. This function must return a dataframe with a column `diversity_val` containing the statistic of interest. As used by `bootstrap_cube()`.

`...` Additional arguments passed on to `fun`.

`ref_group` Only used when `type = "bca"`. A string indicating the reference group to compare the statistic with. Default is `NA`, meaning no reference group is used. As used by `bootstrap_cube()`.

`jackknife` Only used when `type = "bca"`. A string specifying the jackknife resampling method for BCa intervals.

- `"usual"`: Negative jackknife (default if BCa is selected).
- `"pos"`: Positive jackknife

`progress` Logical. Whether to show a progress bar for jackknifing. Set to `TRUE` to display a progress bar, `FALSE` (default) to suppress it.

Details

We consider four different types of intervals (with confidence level α). The choice for confidence interval types and their calculation is in line with the **boot** package in R (Canty & Ripley, 1999) to ensure ease of implementation. They are based on the definitions provided by Davison & Hinkley (1997, Chapter 5) (see also DiCiccio & Efron, 1996; Efron, 1987).

1. **Percentile**: Uses the percentiles of the bootstrap distribution.

$$CI_{perc} = \left[\hat{\theta}^*_{(\alpha/2)}, \hat{\theta}^*_{(1-\alpha/2)}\right]$$

where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

2. **Bias-Corrected and Accelerated (BCa)**: Adjusts for bias and acceleration

   Bias refers to the systematic difference between the observed statistic from the original dataset and the center of the bootstrap distribution of the statistic. The bias correction term is calculated as follows:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#(\hat{\theta}_b^* < \hat{\theta})}{B}\right)$$

   where $\#$ is the counting operator and Φ^{-1} the inverse cumulative density function of the standard normal distribution.

   Acceleration quantifies how sensitive the variability of the statistic is to changes in the data.

   - $a = 0$: The statistic's variability does not depend on the data (e.g., symmetric distribution)
   - $a > 0$: Small changes in the data have a large effect on the statistic's variability (e.g., positive skew)
   - $a < 0$: Small changes in the data have a smaller effect on the statistic's variability (e.g., negative skew).

   The acceleration term is calculated as follows:

$$\hat{a} = \frac{1}{6}\frac{\sum_{i=1}^{n}(I_i^3)}{\left(\sum_{i=1}^{n}(I_i^2)\right)^{3/2}}$$

   where I_i denotes the influence of data point x_i on the estimation of θ. I_i can be estimated using jackknifing. Examples are (1) the negative jackknife: $I_i = (n-1)(\hat{\theta} - \hat{\theta}_{-i})$, and (2) the positive jackknife $I_i = (n+1)(\hat{\theta}_{-i} - \hat{\theta})$ (Frangos & Schucany, 1990). Here, $\hat{\theta}_{-i}$ is the estimated value leaving out the i'th data point x_i. The **boot** package also offers infinitesimal jackknife and regression estimation. Implementation of these jackknife algorithms can be explored in the future.

   The bias and acceleration estimates are then used to calculate adjusted percentiles.
   $\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{\alpha/2})}\right)$, $\alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{1-\alpha/2})}\right)$
   So, we get

$$CI_{bca} = \left[\hat{\theta}^*_{(\alpha_1)}, \hat{\theta}^*_{(\alpha_2)}\right]$$

3. **Normal**: Assumes the bootstrap distribution of the statistic is approximately normal

$$CI_{norm} = \left[\hat{\theta} - \text{Bias}_{\text{boot}} - \text{SE}_{\text{boot}} \times z_{1-\alpha/2}, \hat{\theta} - \text{Bias}_{\text{boot}} + \text{SE}_{\text{boot}} \times z_{1-\alpha/2}\right]$$

   where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard normal distribution.

4. **Basic**: Centers the interval using percentiles

$$CI_{basic} = \left[2\hat{\theta} - \hat{\theta}^*_{(1-\alpha/2)}, 2\hat{\theta} - \hat{\theta}^*_{(\alpha/2)}\right]$$

   where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

Value

A dataframe containing the bootstrap results with the following columns:

- `est_original`: The statistic based on the full dataset per group
- rep_boo
- `est_boot`: The bootstrap estimate (mean of bootstrap replicates per group)
- `se_boot`: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- `bias_boot`: The bias of the bootstrap estimate per group
- `int_type`: The interval type
- `ll`: The lower limit of the confidence interval
- `ul`: The upper limit of the confidence interval
- `conf`: The confidence level of the interval When `aggregate = FALSE`, the dataframe contains the columns from `bootstrap_samples_df` with one row per bootstrap replicate.

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

DiCiccio, T. J., & Efron, B. (1996). Bootstrap confidence intervals. Statistical Science, 11(3). doi:10.1214/ss/1032280214

Efron, B. (1987). Better Bootstrap Confidence Intervals. Journal of the American Statistical Association, 82(397), 171–185. doi:10.1080/01621459.1987.10478410

Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap (1st ed.). Chapman and Hall/CRC. doi:10.1201/9780429246593

Frangos, C. C., & Schucany, W. R. (1990). Jackknife estimation of the bootstrap acceleration constant. Computational Statistics & Data Analysis, 9(3), 271–281. doi:10.1016/01679473(90)90109U

See Also

Other uncertainty: `add_effect_classification()`, `bootstrap_cube()`

Examples

```
# Get example data
# install.packages("remotes")
# remotes::install_github("b-cubed-eu/b3gbi")
library(b3gbi)
cube_path <- system.file(
  "extdata", "denmark_mammals_cube_eqdgc.csv",
  package = "b3gbi")
denmark_cube <- process_cube(
  cube_path,
  first_year = 2014,
```

```
  last_year = 2020)

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(data) {
  out_df <- aggregate(obs ~ year, data, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(denmark_cube$data)

# Perform bootstrapping

bootstrap_mean_obs <- bootstrap_cube(
  data_cube = denmark_cube$data,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  progress = FALSE)
head(bootstrap_mean_obs)

# Calculate confidence limits
# Percentile interval
ci_mean_obs1 <- calculate_bootstrap_ci(
  bootstrap_samples_df = bootstrap_mean_obs,
  grouping_var = "year",
  type = "perc",
  conf = 0.95,
  aggregate = TRUE)
ci_mean_obs1

# All intervals
ci_mean_obs2 <- calculate_bootstrap_ci(
  bootstrap_samples_df = bootstrap_mean_obs,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic"),
  conf = 0.95,
  aggregate = TRUE,
  data_cube = denmark_cube$data, # Required for BCa
  fun = mean_obs,                # Required for BCa
  progress = FALSE)
ci_mean_obs2
```

`cross_validate_cube` *Leave-one-out cross-validation for data cubes*

Description

This function performs leave-one-out (LOO) or k-fold (experimental) cross-validation (CV) on a biodiversity data cube to assess the performance of a specified indicator function. It partitions the data by a specified variable, calculates the specified indicator on training data, and compares it with the true values to evaluate the influence of one or more categories on the final result.

Usage

```
cross_validate_cube(
  data_cube,
  fun,
  ...,
  grouping_var,
  out_var = "taxonKey",
  crossv_method = c("loo", "kfold"),
  k = ifelse(crossv_method == "kfold", 5, NA),
  max_out_cats = 1000,
  progress = FALSE
)
```

Arguments

data_cube	A data cube object (class 'processed_cube' or 'sim_cube', see `b3gbi::process_cube()`) or a dataframe (from `$data` slot of 'processed_cube' or 'sim_cube'). To limit runtime, we recommend using a dataframe with custom function as `fun`.
fun	A function which, when applied to `data_cube` returns the statistic(s) of interest. This function must return a dataframe with a column `diversity_val` containing the statistic of interest.
...	Additional arguments passed on to `fun`.
grouping_var	A string specifying the grouping variable(s) for `fun`. The output of `fun(data_cube)` returns a row per group.
out_var	A string specifying the column by which the data should be left out iteratively. Default is `"taxonKey"` which can be used for leave-one-species-out CV.
crossv_method	Method of data partitioning. If `crossv_method = "loo"` (default), `S = number of unique values in ou` training partitions are created containing `S - 1` rows each. If `crossv_method = "kfold"`, the aggregated data is split the data into `k` exclusive partitions containing `S / k` rows each. K-fold CV is experimental and results should be interpreted with caution.
k	Number of folds (an integer). Used only if `crossv_method = "kfold"`. Default 5.
max_out_cats	An integer specifying the maximum number of unique categories in `out_var` to leave out iteratively. Default is `1000`. This can be increased if needed, but keep in mind that a high number of categories in `out_var` may significantly increase runtime.
progress	Logical. Whether to show a progress bar. Set to `TRUE` to display a progress bar, `FALSE` (default) to suppress it.

Details

This function assesses the influence of each category in `out_var` on the indicator value by iteratively leaving out one category at a time, similar to leave-one-out cross-validation. K-fold CV works in a similar fashion but is experimental and will not be covered here.

1. **Original Sample Data**: $\mathbf{X} = \{X_{11}, X_{12}, X_{13}, \ldots, X_{sn}\}$
   - The initial set of observed data points, where there are s different categories in `out_var` and n total samples across all categories (= the sample size). n corresponds to the number of cells in a data cube or the number of rows in tabular format.
2. **Statistic of Interest**: θ
   - The parameter or statistic being estimated, such as the mean $\bar{X}$, variance σ^2, or a biodiversity indicator. Let $\hat{\theta}$ denote the estimated value of θ calculated from the complete dataset $\mathbf{X}$.
3. **Cross-Validation (CV) Sample**: $\mathbf{X}_{-s_j}$
   - The full dataset $\mathbf{X}$ excluding all samples belonging to category j. This subset is used to investigate the influence of category j on the estimated statistic $\hat{\theta}$.
4. **CV Estimate for Category j**: $\hat{\theta}_{-s_j}$
   - The value of the statistic of interest calculated from $\mathbf{X}_{-s_j}$, which excludes category j. For example, if θ is the sample mean, $\hat{\theta}_{-s_j} = \bar{X}_{-s_j}$.
5. **Error Measures**:
   - The **Error** is the difference between the statistic estimated without category j ($\hat{\theta}_{-s_j}$) and the statistic calculated on the complete dataset ($\hat{\theta}$).

$$\text{Error}_{s_j} = \hat{\theta}_{-s_j} - \hat{\theta}$$

   - The **Relative Error** is the absolute error, normalised by the true estimate $\hat{\theta}$ and a small error term $\epsilon = 10^{-8}$ to avoid division by zero.

$$\text{Rel. Error}_{s_j} = \frac{|\hat{\theta}_{-s_j} - \hat{\theta}|}{\hat{\theta} + \epsilon}$$

   - The **Percent Error** is the relative error expressed as a percentage.

$$\text{Perc. Error}_{s_j} = \text{Rel. Error}_{s_j} \times 100\%$$

6. **Summary Measures**:
   - The **Mean Relative Error (MRE)** is the average of the relative errors over all categories.

$$\text{MRE} = \frac{1}{s} \sum_{j=1}^{s} \text{Rel. Error}_{s_j}$$

   - The **Mean Squared Error (MSE)** is the average of the squared errors.

$$\text{MSE} = \frac{1}{s} \sum_{j=1}^{s} (\text{Error}_{s_j})^2$$

- The **Root Mean Squared Error (RMSE)** is the square root of the MSE.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Value

A dataframe containing the cross-validation results with the following columns:

- Cross-Validation id (`id_cv`)
- The grouping variable `grouping_var` (e.g., year)
- The category left out during each cross-validation iteration (specified `out_var` with suffix '_out' in lower case)
- The computed statistic values for both training (`rep_cv`) and true datasets (`est_original`)
- Error metrics: error (`error`), squared error (`sq_error`), absolute difference (`abs_error`), relative difference (`rel_error`), and percent difference (`perc_error`)
- Error metrics summarised by `grouping_var`: mean relative difference (`mre`), mean squared error (`mse`) and root mean squared error (`rmse`)

See Details section on how these error metrics are calculated.

Examples

```
# Get example data
# install.packages("remotes")
# remotes::install_github("b-cubed-eu/b3gbi")
library(b3gbi)
cube_path <- system.file(
  "extdata", "denmark_mammals_cube_eqdgc.csv",
  package = "b3gbi")
denmark_cube <- process_cube(
  cube_path,
  first_year = 2014,
  last_year = 2020)

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(data) {
  out_df <- aggregate(obs ~ year, data, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(denmark_cube$data)

# Perform leave-one-species-out CV

cv_mean_obs <- cross_validate_cube(
  data_cube = denmark_cube$data,
  fun = mean_obs,
  grouping_var = "year",
  out_var = "taxonKey",
```

```
  crossv_method = "loo",
  progress = FALSE)
head(cv_mean_obs)
```

Index