BIODIVERSITY BUILDING BLOCKS FOR POLICY

D5.4 Report of the Indicators of Robustness Assessments

29/04/2026

Author(s): Ward Langeraert, Toon Van Daele

Funded by the European Union

Prepared under contract from the European Commission
Grant agreement No. 101059592
EU Horizon Europe Research and Innovation Action

Project acronym:	**B3**
Project full title:	**Biodiversity Building Blocks for policy**
Project duration:	01.03.2023 – 31.08.2026 (42 months)
Project coordinator:	Dr. Quentin Groom, Agentschap Plantentuin Meise (MeiseBG)
Call:	HORIZON-CL6-2021-GOVERNANCE-01
Deliverable title:	Report of the Indicators of Robustness Assessments
Deliverable n°:	D5.4
WP responsible:	WP5
Nature of the deliverable:	Report
Dissemination level:	Public
Licence of use:	Creative Commons Attribution 4.0 International
Lead partner:	INBO
Recommended citation:	Langeraert, W. & Van Daele, T. (2026). ***Report of the Indicators of Robustness Assessments***. B3 project deliverable D5.4.
Due date of deliverable:	Month n°38
Actual submission date:	Month n°38

Deliverable status:

Version	Status	Date	Author(s)
1.0	Final	29 April 2026	Ward Langeraert (INBO), Toon Van Daele (INBO)

Table of contents

Key takeaway messages....4
Executive summary....4
Non-technical summary....5
List of abbreviations....5
1. Introduction....6
1.1. Background....6
1.2. Methodological challenges in biodiversity indicator development....6
1.3. Analytical framework for indicator evaluation....7
1.4. Objectives and structure of the report....8
2. Methods and software implementation....9
2.1. Overview of the dubicube package....9
2.2. dubicube as a dependency for other packages....11
2.3. Example data and reproducible workflows....13
3. Data variability and exploration....17
3.1. Data quality diagnostics....17
3.2. Group-level sensitivity analysis....29
4. Indicator uncertainty calculation....37
4.1. Bootstrap methodology for data cubes....37
4.2. Calculating bootstrap confidence intervals....47
4.3. Whole-cube bootstrap versus group-specific bootstrap....62
4.4. Integration with the boot package....64
4.5. Smart bootstrap selection....69
5. Indicator interpretation and visualisation....70
5.1. Classifying effects using confidence limits....70
5.2. Visualising temporal trends....74
5.3. Visualising spatial trends....75
6. Software design and future development of dubicube....79
6.1. Computational performance and scalability....79
6.2. Object-oriented design and S3 implementation....80
6.3. Extension of diagnostic rules....81
7. Acknowledgements....83
8. References....84
9. Annex....89
9.1. Data quality diagnostic rules....89

Key takeaway messages

- **Integrated Robustness Framework**: This report provides a structured approach combining data quality diagnostics, group-level sensitivity analysis, and uncertainty quantification for biodiversity indicators calculated from occurrence cubes.
- **Automated Diagnostics**: Features a modular, rule-based system to identify and filter critical data limitations in spatial, temporal, and taxonomic dimensions.
- **Uncertainty Quantification**: Proposes a bootstrap resampling framework to estimate confidence intervals without requiring strong parametric assumptions.
- **Transparent Interpretation**: Introduces effect classification and visualisation tools to communicate indicator trends and reliability.
- **The dubicube R Package**: Implements the robustness framework throughout the occurrence cube indicator calculation workflow, including data exploration and filtering, uncertainty interval calculation via bootstrapping, and uncertainty visualisation and interpretation.

Executive summary

This deliverable documents the conceptual and technical foundations of the **dubicube** R package, a core component of the Biodiversity Building Blocks for Policy (B3) project. As biodiversity data are often fragmented and heterogeneous, transforming raw observations into data cubes to calculate reliable indicators requires a rigorous assessment of robustness. Here, robustness refers to the stability and reliability of results under variations in underlying data and analytical choices.

The report details an integrated evaluation framework organized into three complementary pillars, each addressing a distinct stage of the indicator assessment workflow:

1. **Data Quality and Sensitivity**: An automated, rule-based diagnostic framework evaluates the spatial, temporal, and taxonomic completeness of occurrence cubes, flagging potential data limitations before indicator calculation. Complementary group-level cross-validation then quantifies the influence of specific groups, e.g. taxa or time periods, on indicator values.
2. **Uncertainty Quantification**: To address the frequent omission of variability measures in biodiversity reporting, the package implements a comprehensive bootstrap resampling framework. This allows for the calculation of multiple confidence interval types (percentile, BCa, normal, and basic) tailored to the specific distribution of ecological indicators.
3. **Interpretation and Visualisation**: The framework links quantitative estimates to an effect classification system that categorizes trends (e.g., increase, stable, decrease). Advanced visualisation techniques, including fan plots and spatial transparency mapping, ensure that uncertainty is explicitly communicated to scientific and policy audiences.

By providing these tools, **dubicube** ensures that biodiversity indicators are not merely calculated but are evaluated and communicated in a way that supports confident, evidence-based decision-making.

Non-technical summary

Monitoring global biodiversity is a massive challenge because information often comes from many different sources, ranging from professional surveys to apps used by citizen scientists. While we have more data than ever before, these data can be messy, patchy, or biased toward certain species. If we use these data to create "indicators" (simplified scores used by governments to track nature's health) without checking its quality, we risk making poor policy decisions based on misleading trends.

The **dubicube** software was created to solve this by acting as a "quality control" toolkit for biodiversity indicators. It helps researchers answer three main questions:

1. **Are the data good enough?** The tool automatically checks if there are enough observations over time and across different regions to make a fair assessment.
2. **How certain are we?** Just as a single measurement does not tell the whole story, **dubicube** uses a technique called "bootstrapping" to see how an indicator would change if different samples of the data were used. This highlights the variability in the data, ensuring we do not over-interpret small changes that might just be accidental.
3. **What does it actually mean?** The software helps translate complex statistics into clear messages, such as "strong increase" or "stable," using easy-to-read charts and maps.

Ultimately, this work makes biodiversity monitoring more transparent and reliable, ensuring that policymakers are standing on a firm foundation of evidence when making decisions about environmental protection.

List of abbreviations

B3	Biodiversity Building Blocks for Policy
BCa	Bias-Corrected and Accelerated (Confidence Interval)
CI	Confidence Interval
CV	Cross-Validation
EC	European Commission
EU	European Union
GBIF	Global Biodiversity Information Facility
LOESS	Locally Estimated Scatterplot Smoothing
LOO-CV	Leave-One-Out Cross-Validation
LOSO-CV	Leave-One-Species-Out Cross-Validation
MRE	Mean Relative Error
MSE	Mean Squared Error
OOP	Object-Oriented Programming
RMSE	Root Mean Squared Error

1. Introduction

1.1. Background

The global biodiversity crisis poses a major challenge for environmental policy and management, requiring timely and reliable information to monitor trends and assess the effectiveness of interventions. While an unprecedented volume of biodiversity data is available today, ranging from local monitoring schemes to global databases, these data are often fragmented across sources, heterogeneous in structure, and insufficiently aligned in space and time. This limits their direct usability for ecological modelling and policy support.

The Biodiversity Building Blocks for Policy (B3) project addresses this challenge by transforming raw biodiversity data into standardised and accessible information products (https://b-cubed.eu/). Central to this effort is the concept of occurrence cubes, which aggregate heterogeneous biodiversity observations into a harmonised, multi-dimensional structure. By standardising access across spatial, temporal, and taxonomic dimensions, occurrence cubes provide a consistent and scalable foundation for biodiversity indicators and models.

Through this approach, B3 aims to support policymakers with more rapid, robust, and interpretable insights into biodiversity change. By improving data accessibility and interoperability, the project facilitates more effective monitoring of biodiversity trends and enables evidence-based policy responses at multiple spatial scales.

1.2. Methodological challenges in biodiversity indicator development

While occurrence cubes provide a standardised and scalable framework for aggregating biodiversity data, their use does not automatically guarantee that the resulting indicators are reliable or meaningful (Langeraert, Faulkner, et al., 2026). In practice, biodiversity indicators are often calculated from available data without fully assessing whether those data are sufficient, representative, or appropriate for the intended purpose. This raises several fundamental questions: can the calculated indicator values be interpreted with confidence, and do the underlying data support valid conclusions across species, time periods, and spatial scales?

Addressing these questions requires going beyond the calculation of indicator values alone. In this report, we adopt an integrated perspective that combines three complementary aspects to increase the robustness of data analysis by evaluating the stability and reliability of biodiversity indicators derived from occurrence cubes: (1) sensitivity to data composition, assessing how indicator values respond to the inclusion or exclusion of specific groups of data (e.g. species, sites, or time periods); (2) quantification of uncertainty, focusing on the estimation of variability and confidence around indicator values using resampling approaches, and (3) interpretation and visualisation, which concerns how indicator results and their associated uncertainty are communicated and understood (see also Langeraert & Van Daele, 2025b). Together, these components provide a structured approach to assessing whether biodiversity indicators are suitable for scientific analysis and policy support.

In this report, the term robustness is used in this broad, operational sense to describe the stability and reliability of indicator results under variations in the underlying data and analytical

choices. This definition differs from the classical statistical notion of robustness, which typically refers to the insensitivity of an estimator to outliers or deviations from model assumptions. While related, the focus in this report is on the consistency and interpretability of indicator outcomes in realistic, heterogeneous biodiversity data contexts, rather than on formal robustness properties of specific estimators. To avoid ambiguity, we aim to limit the use of the term throughout the report where more precise terminology can be used, although it is retained in the title in line with the original project proposal.

Within this framework, data quality measures are used to evaluate the applicability and reliability of biodiversity indicators by assessing whether the underlying data are adequate and representative. These measures are therefore properties of the data, calculated alongside indicator values, rather than intrinsic properties of the indicators themselves. This distinction is important, as it emphasises that limitations in indicator interpretation can stem from the data on which they are based. We also provide a framework for group-level sensitivity analysis assessing how indicator values respond to the inclusion or exclusion of specific groups.

In addition, the explicit quantification of indicator uncertainty is essential. Indicator values are sometimes reported without accompanying measures of uncertainty, such as confidence intervals (Rowland et al., 2021), despite their critical role in supporting correct interpretation and informed decision-making (Fischhoff & Davis, 2014; Milner-Gulland & Shea, 2017). For biodiversity indicators derived from occurrence cubes, we propose a flexible framework based on bootstrap resampling, which allows the estimation of variability, bias, and confidence intervals without strong parametric assumptions.

Finally, to support consistent and transparent interpretation, we introduce an interpretation framework that links indicator estimates and their uncertainty to meaningful conclusions. This takes the form of effect classification, where confidence intervals are compared against reference values to categorise trends (e.g. increase, decrease, or stable) (Onkelinx, 2023). Such frameworks are particularly important in policy contexts, where clear and reproducible interpretations are required.

By combining data quality diagnostics, sensitivity analysis, uncertainty quantification, and structured interpretation, biodiversity indicators can be placed in their proper analytical context. This ensures that they are not only calculated, but also evaluated and communicated in a way that supports confident and meaningful decision-making.

1.3. Analytical framework for indicator evaluation

All methods discussed in this deliverable report are implemented in the **dubicube** R package (Langeraert & Van Daele, 2026). This package is part of the **b3verse**, an R-Universe platform that integrates all R package software related to occurrence cubes into a single, open-source framework (Langeraert, Breugelmans, et al., 2026; see also this dedicated guide). A description of earlier development can be found in Langeraert & Van Daele (2025b).

1.4. Objectives and structure of the report

This deliverable documents the conceptual, methodological, and technical foundations of the indicator evaluation framework implemented in **dubicube**. It presents the theoretical principles underlying indicator stability and uncertainty quantification, describes the statistical methods applied, and details their software implementation within the package.

This report can be approached in two complementary ways. Reading the text alone is sufficient to understand the concepts and methodology, as the accompanying code snippets are explained in a descriptive manner and do not require detailed knowledge of the R language. Readers seeking a more hands-on understanding may alternatively execute the provided code snippets in an R environment to reproduce and explore the analyses.

The report is structured as follows. Chapter 2 provides an overview of the **dubicube** package and its role within the broader analytical ecosystem. Chapter 3 addresses data quality and group-level sensitivity assessment. Chapter 4 describes the bootstrap methodology and approaches to indicator uncertainty quantification. Chapter 5 focuses on interpretation and visualisation of results. Finally, Chapter 6 discusses implementation choices, computational performance, and perspectives for further development.

2. Methods and software implementation

2.1. Overview of the dubicube package

The **dubicube** package has been developed to support the assessment of robustness and applicability of biodiversity indicators derived from data cubes. Its conceptual framework is grounded in the recognition that biodiversity indicators are only as reliable as the data and assumptions underlying them. Consequently, **dubicube** focuses on (1) evaluating data quality, (2) quantifying indicator uncertainty, and (3) supporting transparent interpretation.

The package can be installed using base R from the **b3verse** repository.

```
install.packages("dubicube", repos = c("https://b-cubed-eu.r-universe.dev",
"https://cloud.r-project.org"))
```

The package can also be installed from GitHub using **remotes** (Csárdi et al., 2024).

```
# install.packages("remotes")
remotes::install_github("b-cubed-eu/dubicube")
```

Its functionality is useful throughout the indicator calculation workflow (Fig. 1, see also Langeraert, Breugelmans, et al. (2026)). Occurrence cubes can be created from GBIF data using the **rgbif** package (Chamberlain et al., 2025). They are processed using the `process_cube()` function from the **b3gbi** package (Dove, 2026). This ensures data standardisation and verifies that the cube's format is correct. **dubicube** facilitates data exploration and filtering (1), which form part of an iterative process with cube generation and processing. After a number of iterations, data evaluation is successful and the final data cube can be used for indicator calculation. Indicator calculation packages can use **dubicube** as a dependency for uncertainty interval calculation via bootstrapping (2) but the package can also be used on its own. Finally, the package provides tools and tutorials to help with indicator visualisation and interpretation (3).

1. Data exploration and filtering (Chapter 3)
The first step focuses on understanding the structure, completeness, and reliability of the data cube. The `diagnose_cube()` function applies a set of diagnostic rules to identify potential quality issues, while `filter_cube()` enables users to exclude observations that do not meet predefined criteria. In addition, `cross_validate_cube()` implements group-level cross-validation, allowing users to assess how sensitive indicator values are to specific groups such as species, sites, or time periods. This step provides critical insights into data heterogeneity and potential sources of bias.

2. Estimation of indicator uncertainty (Chapter 4)
The second workflow addresses the quantification of uncertainty in biodiversity indicators. The function `bootstrap_cube()` generates bootstrap replicates of the data cube, enabling the estimation of variability, bias, and standard errors associated with indicator values. Building on

these replicates, `calculate_bootstrap_ci()` computes confidence intervals using multiple methods (percentile, bias-corrected and accelerated, normal, and basic intervals), with support for transformations and bias correction. Together, these tools provide a flexible and robust framework for uncertainty estimation in complex ecological datasets.

3. Interpretation and visualisation of indicators (Chapter 5)
The final workflow focuses on translating quantitative results into interpretable outputs for analysis and decision-making. The function `add_effect_classification()` classifies indicator trends (e.g. increase, stable, decrease) based on confidence intervals and predefined thresholds, supporting consistent interpretation. In addition, the package provides guidance and tools for visualising temporal and spatial patterns in indicators, explicitly incorporating uncertainty into these representations. This facilitates the communication of results to both scientific and policy audiences.

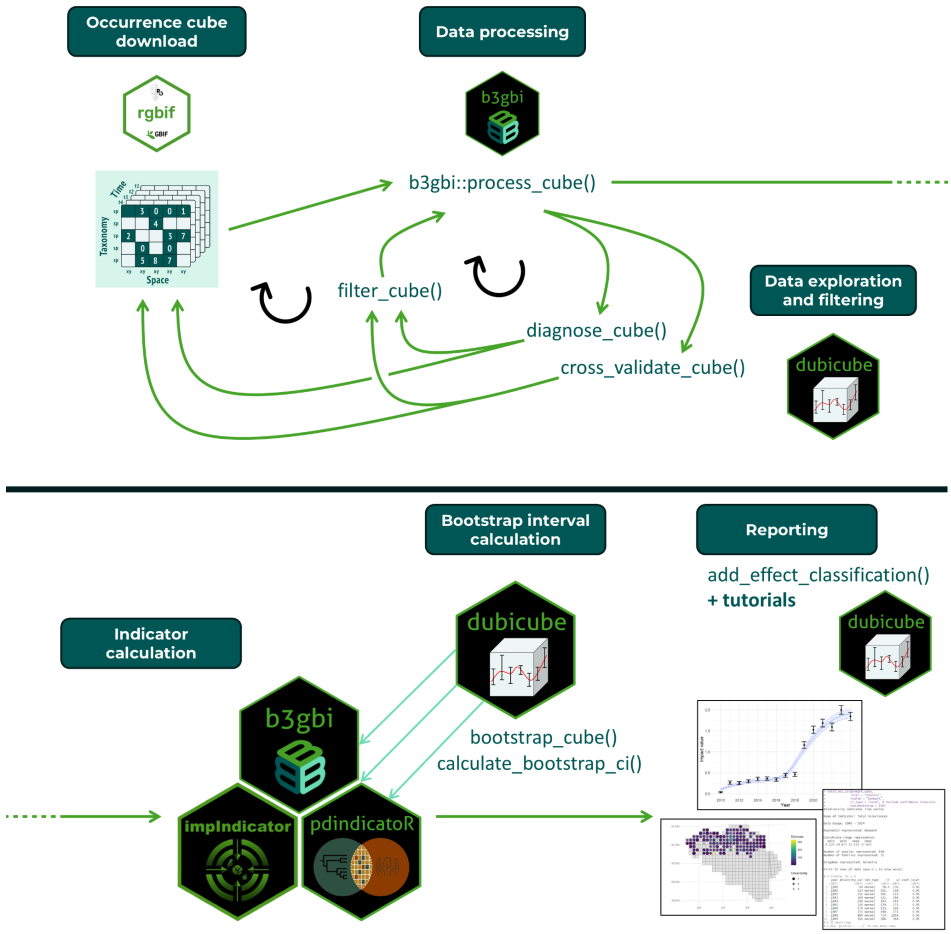


Figure 1: dubicube functionality within the indicator calculation workflow. Data exploration can be done by evaluating diagnostics or group specific sensitivity analysis using cross-validation (Chapter 3). Based on these results, it can be decided to filter the data (Chapter 3) or to generate a new data cube. The package can also be used to calculate confidence intervals for indicators based on bootstrap resampling (Chapter 4) and plays an important role in reporting (Chapter 5).

2.2. dubicube as a dependency for other packages

The **dubicube** package can be used on its own for any functionality, as shown below. However, we also want to allow other packages to integrate **dubicube** as a dependency, for example for bootstrap interval calculation (Fig. 1). We aim to incorporate interval calculation with **dubicube** into the calculation of (1) general biodiversity indicators with **b3gbi** (Dove, 2026), (2) phylogenetic diversity with **pdindicatoR** (Breugelmans et al., 2025), and (3) impact of alien taxa with **impIndicator** (Yahaya et al., 2026) for indicator estimates for which no straightforward uncertainty calculation is available. This is still an ongoing process together with the maintainers of the different packages.

We show a minimal example of the calculation of impact of alien taxa with **impIndicator** (v0.6.0) where we successfully implemented **dubicube** for interval calculation. The **impIndicator** package includes an example occurrence cube (`cube_acacia_SA`) and example EICAT assessment dataset (`eicat_acacia`).

We load the **impIndicator** package (v0.6.0) and the **b3gbi** package (v0.8.17).

```
library(b3gbi)
library(impIndicator)
```

We use the **b3gbi** package to process the cube into a standardised format. We select the data from 2010 until 2024.

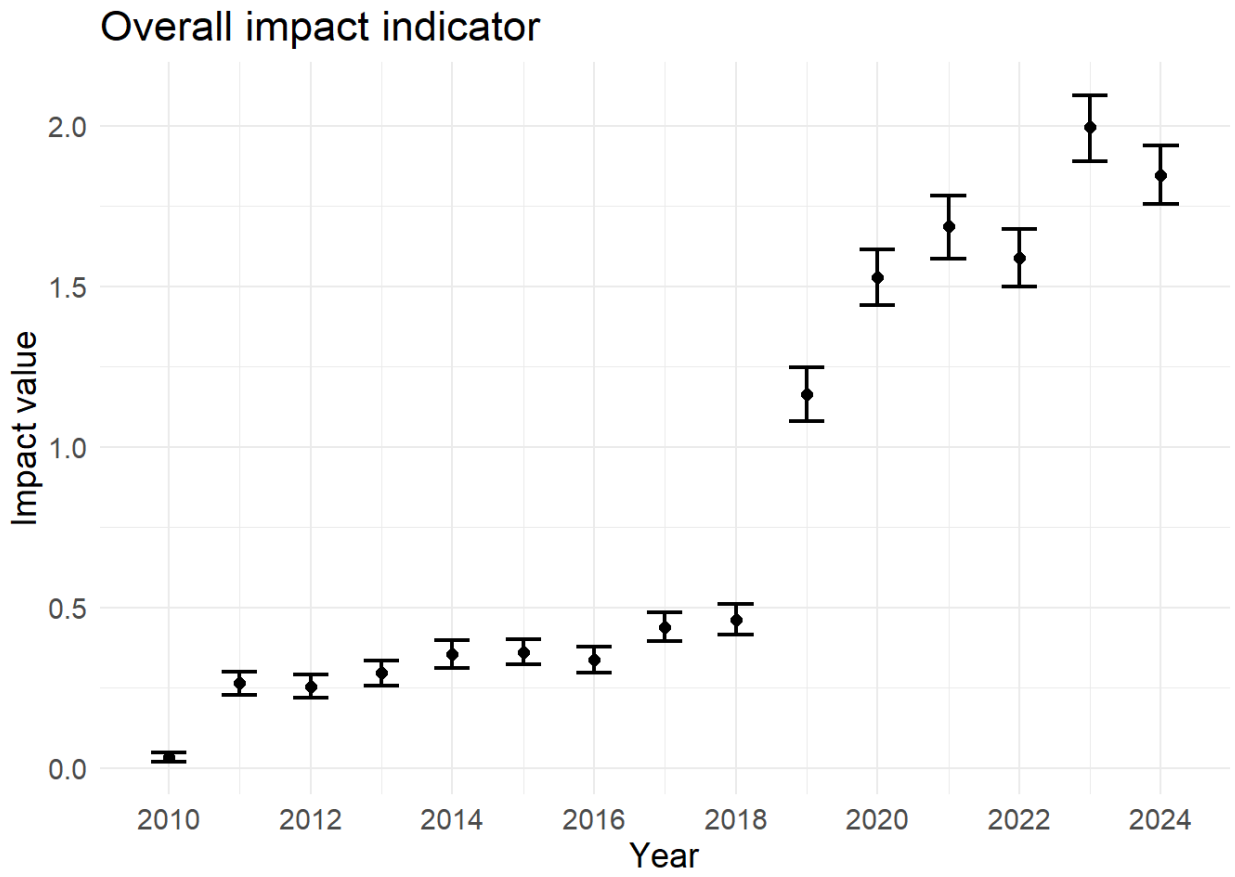
```
acacia_cube <- process_cube(
  cube_name = cube_acacia_SA,
  grid_type = "eqdgc",
  first_year = 2010,
  last_year = 2024
)
```

We calculate the overall impact indicator with 95 % percentile intervals. We use the default arguments for interval calculation but show them here explicitly for illustrative purposes. The package provides logical default arguments for easy and user-friendly interval calculation. Yet, it is also possible to adjust specific bootstrap parameters. The `boot_args` argument is a named list of additional arguments that are internally passed to `dubicube::bootstrap_cube()` (see further in Chapter 4.1.2). By default, 1000 bootstrap samples and no seed are used. The `ci_args` argument is a named list of additional arguments that are internally passed to `dubicube::calculate_bootstrap_ci()` (see further in Chapter 4.2.2). By default, we do not account for bootstrap bias that can occur due to the nature of the impact indicator calculation (cf. species richness example in Chapter 4.2.2).

```
impact_indicator <- compute_impact_indicator(
  # Indicator arguments
  cube = acacia_cube,
  impact_data = eicat_acacia,
  method = "mean_cum",
  # Interval arguments (all default)
  ci_type = "perc",
  confidence_level = 0.95,
  boot_args = list(samples = 1000, seed = NA),
  ci_args = list(no_bias = TRUE)
)
#> [1] "Performing whole-cube bootstrap."
```

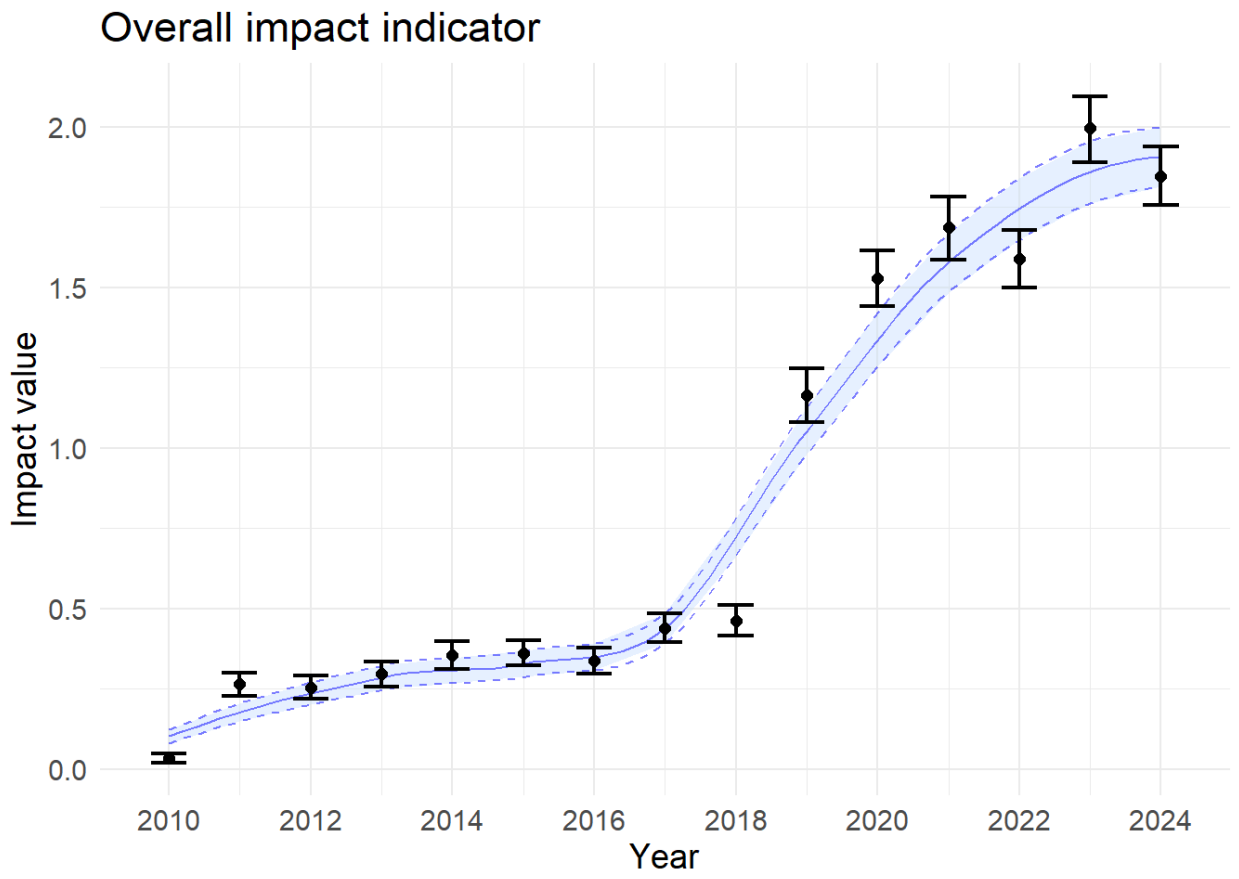
Plotting the indicator object gives us estimates and error bars.

```
plot(impact_indicator)
```



We can add a smoother to the figure to visualise a continuous trend (more information on this in Chapter 5.2).

```
plot(impact_indicator, trend = "smooth")
```



2.3. Example data and reproducible workflows

2.3.1. Package tutorials

Examples and figures in Chapters 3-5 are from the tutorials of **dubicube** v0.12.3 (Table 1), unless mentioned otherwise. They are both published on the website of the package (https://b-cubed-eu.github.io/dubicube) as well as on the B3 documentation website (https://docs.b-cubed.eu/). To run the examples yourself, you need to install and load the **dubicube** package (v0.12.3), the **frictionless** package to load the example data (Desmet et al., 2025), **b3gbi** (v0.8.17) to process the data into a standardised occurrence cube, **dplyr** (Wickham et al., 2023) and **tidyr** (Wickham et al., 2014) for data wrangling, and **ggplot2** for visualisation (Wickham, 2016).

Table 1: dubicube tutorials and their use in this report.

Main topic	Tutorial	Featured in this report
Data exploration and filtering	Data Quality Diagnostics and Filtering for Data Cubes	Directly used in Chapter 3.1.3.
	Group-Level Sensitivity Analysis	Directly used in Chapter 3.2.2.
Estimation of indicator uncertainty	Bootstrap Method for Data Cubes	Directly used in Chapter 4.1.2.
	Calculating Bootstrap Confidence Intervals	Directly used in Chapter 4.2.2.
	Whole-cube Bootstrap versus Group-specific Bootstrap	Parts used in Chapters 4.3, 4.4, and 4.5.
Interpretation and visualisation of indicators	Classifying Effects Using Confidence Limits	Directly used in Chapter 5.1.
	Visualising Temporal Trends	Concepts and ideas from tutorial described in Chapters 4.2.3 and 5.2. Visit the tutorial for more examples.
	Visualising Spatial Trends	Concepts and ideas from tutorial described in Chapter 5.3. Visit the tutorial for more examples.

2.3.2. Loading and processing the example data

For the examples in the following chapters of this report, we use the bird cube data from the **b3data** data package (Langeraert & Van Daele, 2025a) using **frictionless** (see also this guide). It is an occurrence cube for birds in Belgium between 2000 and 2024 using the MGRS grid at 10 km scale.

```
# Read data package
b3data_package <- frictionless::read_package(
  "https://zenodo.org/records/15211029/files/datapackage.json"
)

# Load bird cube data
bird_cube_belgium <- frictionless::read_resource(
  b3data_package,
  "bird_cube_belgium_mgrs10"
)
```

For some examples we use a smaller data cube to reduce computation time (for illustrative purposes only). We will refer to this dataset as the “small dataset” and to the full cube as the “complete dataset”.

```
# Make smaller subset
set.seed(123)
rows <- sample(nrow(bird_cube_belgium), 2000)
bird_cube_belgium_small <- bird_cube_belgium[rows, ]
```

We process the cube (this is the same whether the small or complete dataset is used) with **b3gbi** to put the data in a format ready for analysis. We select the data from 2011–2020.

```
processed_cube_small <- b3gbi::process_cube(
  bird_cube_belgium_small,
  first_year = 2011,
  last_year = 2020,
  cols_occurrences = "n"
)

processed_cube <- b3gbi::process_cube(
  bird_cube_belgium,
  first_year = 2011,
  last_year = 2020,
  cols_occurrences = "n"
)
```

As an example indicator, we calculate the average number of observations per grid cell per year. We create the function `mean_obs()` to calculate this.

```
mean_obs <- function(data) {
  data %>%
    mutate(x = mean(obs), .by = "cellCode") %>%
    summarise(diversity_val = mean(x), .by = "year") %>%
    as.data.frame()
}
```

We get the following results (here based on complete dataset):

```
mean_obs(processed_cube$data)
#>    year diversity_val
#> 1  2011      48.83864
#> 2  2012      48.27942
#> 3  2013      48.13675
#> 4  2014      47.82226
#> 5  2015      47.39197
#> 6  2016      47.32866
#> 7  2017      46.31520
#> 8  2018      45.87515
#> 9  2019      45.93329
#> 10 2020      44.98668
```

3. Data variability and exploration

3.1. Data quality diagnostics

3.1.1. Data quality challenges and conceptual framework

The **dubicube** package implements a rule-based diagnostic framework for evaluating the quality and structural properties of biodiversity data cubes (Fig. 2). The framework assesses the cube across its main conceptual dimensions, including spatial coverage, temporal coverage, taxonomic representation, and overall number of observations. Diagnostics are implemented as modular rule objects which define how a particular metric is calculated from the data cube, as well as how the resulting value should be interpreted.

Each diagnostic rule contains a metric calculation function that derives a quantitative value from the cube, a set of threshold values that define reference ranges for interpreting the metric, and functions that assign a qualitative severity level and generate a human-readable diagnostic message. This modular rule-based design ensures that diagnostic procedures remain transparent, reproducible, and can be easily extended to different datasets or analytical contexts.

Diagnostic rules are evaluated using the `diagnose_cube()` function, which applies a set of rules to a processed occurrence cube and returns a structured diagnostic table. For each rule, the framework calculates the metric value, compares it with predefined threshold values, and assigns a qualitative severity level describing the potential importance of the diagnostic result. The framework distinguishes four severity levels: “ok”, indicating that the metric falls within acceptable limits; “note”, indicating minor limitations that are unlikely to substantially affect analyses; “important”, indicating issues that may influence indicator calculations or interpretation; and “very important”, indicating serious limitations that may compromise the reliability of downstream analyses (Table 2). These severity levels provide an intuitive overview of potential data limitations and allow users to quickly identify aspects of the cube that may require further attention.

Table 2: dubicube diagnostic severity levels.

Category	Colour code	Description
ok	🟢	No issues detected
note	🟡	Minor limitations
important	🟠	Issues that may influence analysis
very important	🔴	Serious limitations that may affect reliability

In addition to identifying potential issues, the **dubicube** framework also supports data filtering based on diagnostic rules (Fig. 2). The `filter_cube()` function applies rule-specific filtering logic to remove or flag observations that do not meet predefined quality criteria. Filtering is implemented through optional record-level filtering functions associated with diagnostic rules, allowing the same rule framework used for diagnostics to be reused for data cleaning. Filtering is optional and enables users to generate alternative versions of the data cube that exclude observations not meeting specific quality criteria. This approach ensures that data preparation

steps remain transparent and reproducible while allowing users to tailor the cube to the requirements of specific analytical workflows. By combining diagnostics with rule-based filtering, **dubicube** supports a structured workflow for assessing, documenting, and improving the quality of biodiversity data cubes prior to indicator calculation.

Figure 2 shows the role of diagnostics in more detail within the first part of an automated, reproducible indicator calculation workflow (Fig. 1). Together with diagnostic rules, the processed cube can be evaluated with the `diagnose_cube()` function which will automatically print an informative evaluation in the console. The resulting `cube_diagnostics` object can then be used by the `filter_cube()` function to filter out any records from the processed cube based on predefined quality criteria. The `filter_cube()` function can also be used with a set of diagnostic rules instead of a `cube_diagnostics` object. A practical example of the diagnostics framework is shown in Chapter 3.1.3. Data evaluation and filtering are iterative steps until a satisfactory cube evaluation is obtained. This final data cube can then be used for further analysis.

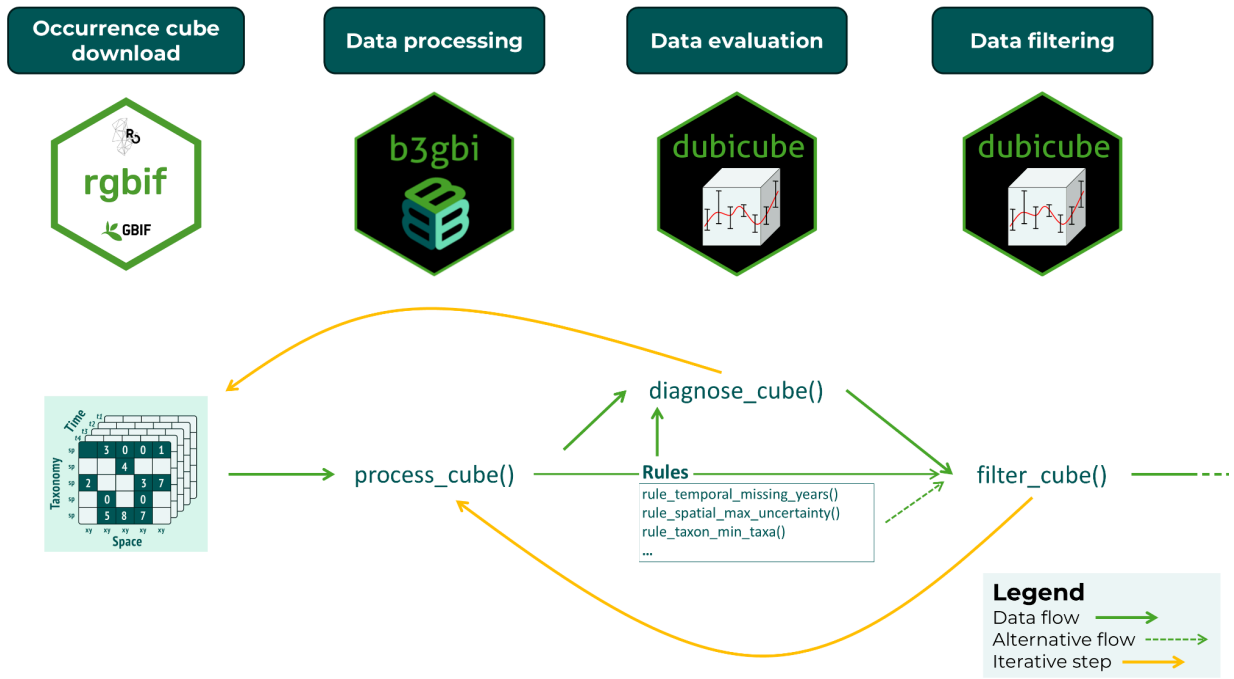


Figure 2: Data quality diagnostics and filtering using dubicube within the indicator calculation workflow. Diagnostics are calculated based on a processed occurrence cube and a list of rules. Filtering can be done with the diagnostic output, or a list of rules (dashed line). Data evaluation and filtering are iterative steps until a satisfactory data cube is obtained for further analysis.

3.1.2. Diagnostic criteria

The diagnostic framework implemented in **dubicube** evaluates a set of quantitative metrics describing the structure and completeness of biodiversity data cubes. They are largely based on the results of Langeraert, Faulkner, et al. (2026) (see also Langeraert & Van Daele, 2025b). These metrics capture key aspects of the dataset that may influence the reliability of biodiversity analyses. By evaluating these different aspects, the framework provides a comprehensive overview of potential limitations in the dataset.

For each diagnostic metric, the calculated value is compared with predefined threshold values that determine the severity of the diagnostic result. These thresholds translate numerical metrics into qualitative diagnostic levels that indicate the potential importance of the issue for subsequent analyses.

The package provides default rule sets, which will be extended further, allowing users to apply a consistent and reproducible set of diagnostics. A complete list of diagnostic metrics and default threshold definitions (to be) implemented in **dubicube** is documented in Annex 1. The diagnostic metrics are organised into three complementary categories:

- **Basic rules** provide a first-level assessment of data sufficiency and essential data quality constraints. These include metrics describing temporal coverage, spatial extent, taxonomic representation, total observation volume, and fundamental data quality aspects such as coordinate uncertainty. Their primary purpose is to identify critical limitations that may prevent meaningful analysis.
- **Observation distribution rules** assess how observations are distributed within each dimension of the cube. These diagnostics detect imbalances in sampling effort across time, space, or taxa, highlighting situations where data are unevenly distributed despite sufficient overall volume. Such imbalances are a common source of bias in biodiversity datasets and can affect the robustness and interpretability of derived indicators.
- **Interaction rules** evaluate the joint structure of multiple dimensions, identifying gaps in coverage that are not apparent when dimensions are considered separately. These metrics assess whether combinations of dimensions (e.g. taxa within time periods or spatial units across years) are sufficiently represented. They are particularly important for detecting structural sparsity and inconsistencies that may compromise analyses relying on integrated spatial–temporal–taxonomic information.

Currently, only basic checks are provided in the package (v0.12.3).

3.1.3. Diagnosing and filtering data with dubicube

3.1.3.1. Data quality checks with `diagnose_cube()`

Running diagnostics

We start by creating a small example data cube which contains a range of common data quality issues, including limited temporal coverage, large coordinate uncertainty, and low sample size. Note that this is for illustrative purposes only, real cubes should be processed with `b3gbi::process_cube()`.

```
cube <- list(
  data = data.frame(
    year = c(2000, 2000, 2001, 2001, 2002, 2002, 2003, 2003, 2003),
    cellCode = c("A", "A", "B", "B", "C", "C", "D", "D", "E"),
    speciesKey = c(1, 1, 1, 2, 2, 3, 3, 3, 3),
    scientificName = paste0("spec_", c(1, 1, 1, 2, 2, 3, 3, 3, 3)),
    occurrences = c(10, 5, 1, 2, 0, 1, 2, 1, 0),
    minCoordinateUncertaintyInMeters = c(
      50, 100, 2000, 70, 5000, 2000, 10, 20, 300
    )
  ),
  resolutions = "1km"
)
class(cube) <- "processed_cube"
```

We can now evaluate the data quality of this cube using `diagnose_cube()`.

```
diag <- diagnose_cube(cube)
#>
#> Data cube diagnostics
#> ---------------------
#> 🟡 NOTE - temporal_min_years
#>    Cube contains observations across 4 years.
#>
#> 🟢 OK - temporal_missing_years
#>    Cube contains 0 missing years.
#>
#> 🟢 OK - spatial_min_cells
#>    Cube contains observations across 5 grid cells.
#>
#> 🟠 IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
#>
#> 🟢 OK - spatial_miss_uncertainty
#>    Cube contains 0 records with missing coordinate uncertainty.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🔴 VERY_IMPORTANT - obs_min_records
#>    Cube contains 9 observation records (rows).
```

```
#>
#> 🟠 IMPORTANT - obs_min_total
#>    Cube contains a total of 22 observations.
```

By default `verbose = TRUE`, so we get a summary of diagnostic results directly in the console and the function returns a structured `cube_diagnostics` object for further inspection.

We can sort the summary with print-arguments.

```
print(diag, sort_summary = "asc")
#>
#> Data cube diagnostics
#> ---------------------
#> 🟢 OK - temporal_missing_years
#>    Cube contains 0 missing years.
#>
#> 🟢 OK - spatial_min_cells
#>    Cube contains observations across 5 grid cells.
#>
#> 🟢 OK - spatial_miss_uncertainty
#>    Cube contains 0 records with missing coordinate uncertainty.
#>
#> 🟡 NOTE - temporal_min_years
#>    Cube contains observations across 4 years.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🟠 IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
#>
#> 🟠 IMPORTANT - obs_min_total
#>    Cube contains a total of 22 observations.
#>
#> 🔴 VERY_IMPORTANT - obs_min_records
#>    Cube contains 9 observation records (rows).
```

We can also filter the summary output based on a minimum severity level.

```
print(diag, sort_summary = "asc", filter_summary = "note")
#>
#> Data cube diagnostics
#> ---------------------
#> 🟡 NOTE - temporal_min_years
#>    Cube contains observations across 4 years.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🟠 IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
#>
#> 🟠 IMPORTANT - obs_min_total
#>    Cube contains a total of 22 observations.
#>
#> 🔴 VERY_IMPORTANT - obs_min_records
#>    Cube contains 9 observation records (rows).
```

Understanding the output

The output object is an object of class `cube_diagnostics`, containing one row per metric with the following columns:

- **dimension**: Dimension of the cube being evaluated (e.g. "spatial", "temporal", "taxonomical").
- **metric**: The diagnostic rule that was evaluated.
- **value**: Computed metric value.
- **severity**: Severity level ("ok", "note", "important", "very_important").
- **message**: Human-readable description of the diagnostic result.

```
diag$dimension
#> [1] "temporal"    "temporal"    "spatial"     "spatial"     "spatial"
#> [6] "taxonomic"   "observation" "observation"

diag$metric
#> [1] "temporal_min_years"      "temporal_missing_years"
#> [3] "spatial_min_cells"        "spatial_max_uncertainty"
#> [5] "spatial_miss_uncertainty" "taxon_min_taxa"
#> [7] "obs_min_records"          "obs_min_total"
```

The rule objects are attached as an attribute of the diagnostics object.

Summarising diagnostics

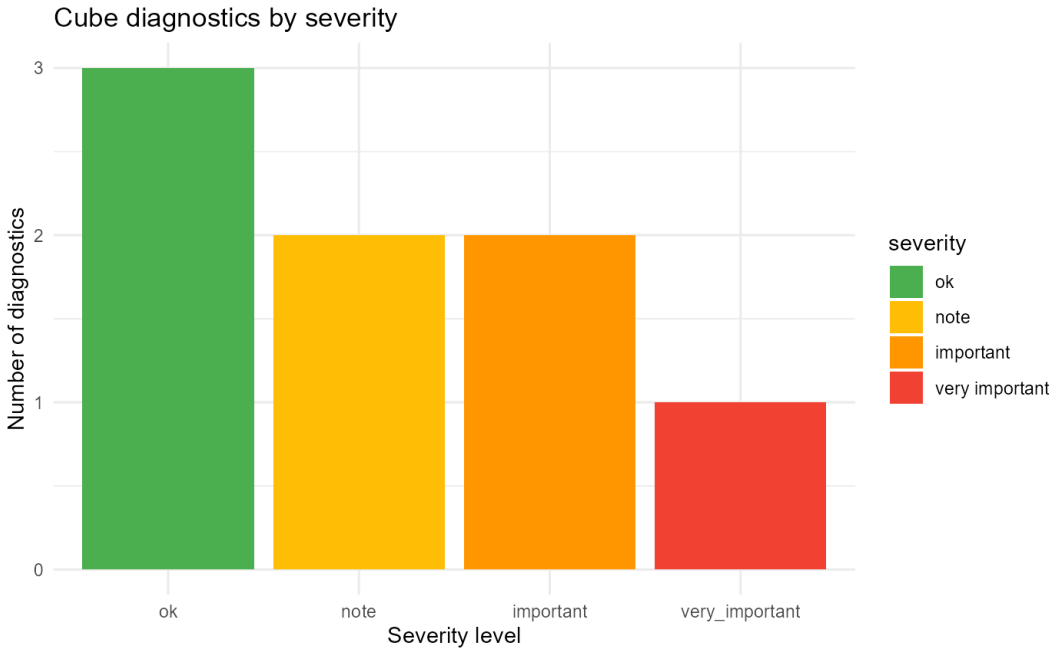
To obtain a concise overview of the results, you can use the `summary()` method:

```
summary(diag)
#> <cube_diagnostics_summary>
#>
#> Rules evaluated: 8
#>
#> Severity levels:
#>      important           note             ok very_important
#>              2              2              3              1
#>
#> Dimensions:
#> observation     spatial   taxonomic    temporal
#>           2           3           1           2
#>
#> Flagged diagnostics:
#>
#> Data cube diagnostics
#> ---------------------
#> 🟡 NOTE - temporal_min_years
#>    Cube contains observations across 4 years.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🟠 IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
#>
#> 🟠 IMPORTANT - obs_min_total
#>    Cube contains a total of 22 observations.
#>
#> 🔴 VERY_IMPORTANT - obs_min_records
#>    Cube contains 9 observation records (rows).
```

This helps to quickly identify which aspects of the data cube may require attention before proceeding with analysis.

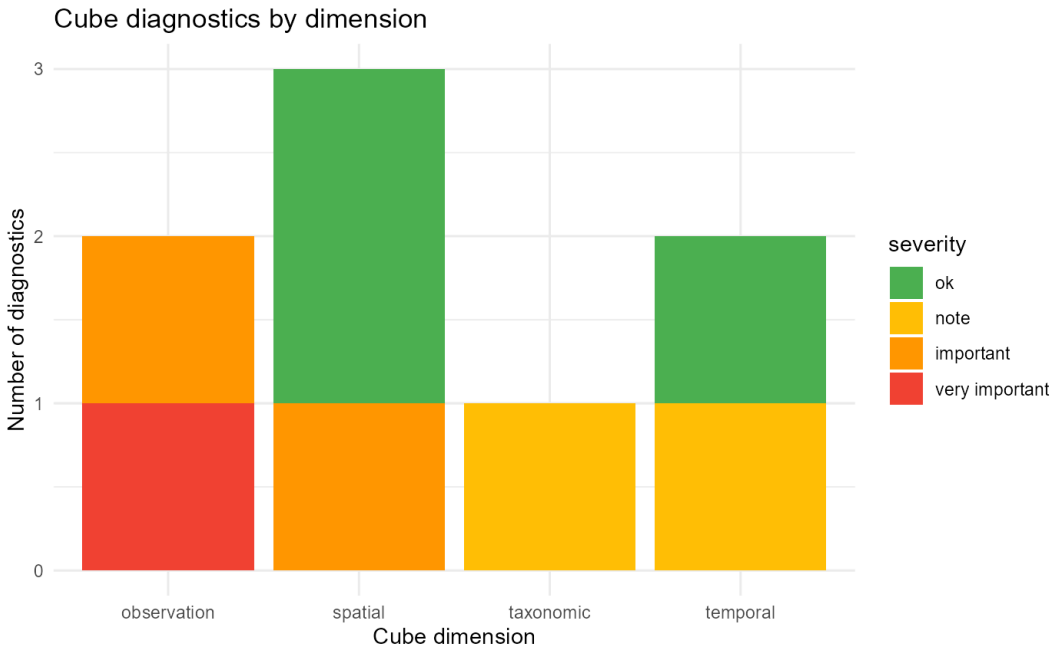
Optionally, diagnostics can also be visualised:

```
plot(diag)
```

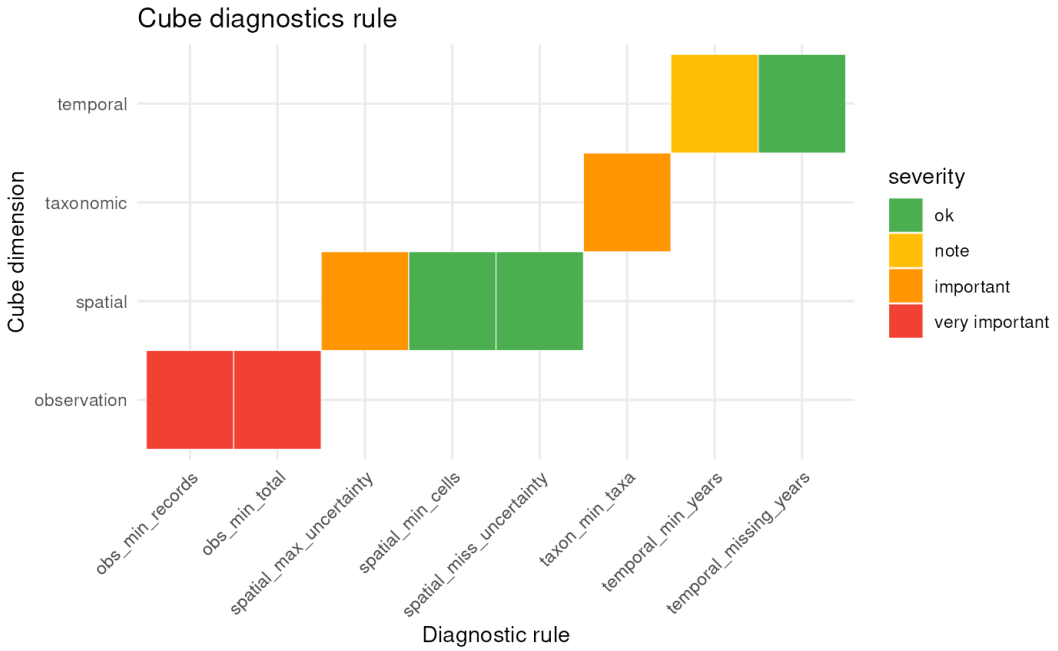


This provides a graphical summary of diagnostic results across severity levels and cube dimensions. The diagnostics can also be plotted by dimension or as a heat map:

```
plot(diag, type = "dimension")
```



```
plot(diag, type = "rule")
```



Selecting diagnostic rules to evaluate

By default, `diagnose_cube()` uses the basic rules to evaluate.

```
basic_cube_rules()
#> <cube_rule_list>
#>   rules: 8
#>
#>   - temporal_min_years ( temporal )
#>   - temporal_missing_years ( temporal )
#>   - spatial_min_cells ( spatial )
#>   - spatial_max_uncertainty ( spatial )
#>   - spatial_miss_uncertainty ( spatial )
#>   - taxon_min_taxa ( taxonomic )
#>   - obs_min_records ( observation )
#>   - obs_min_total ( observation )
```

You can consult the default rule thresholds by printing the rule functions.

```
rule_spatial_max_uncertainty()
#> <cube_rule>
#>   id:        spatial_max_uncertainty
```

```
#>   dimension: spatial
#>   thresholds: ok = 0, note = 1, important = 3, very_important = 5
#>
#> Functions:
#>   compute()     metric calculation
#>   severity()    severity assignment
#>   message()     diagnostic message
#>   filter_fn()   filter rows
```

You can refer to built-in rule sets using a string, you can provide a list of `cube_rule` objects, or combine both.

```
diagnose_cube(cube, rules = "basic")
#>
#> Data cube diagnostics
#> ---------------------
#> 🟡 NOTE - temporal_min_years
#>    Cube contains observations across 4 years.
#>
#> 🟢 OK - temporal_missing_years
#>    Cube contains 0 missing years.
#>
#> 🟢 OK - spatial_min_cells
#>    Cube contains observations across 5 grid cells.
#>
#> 🟠 IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
#>
#> 🟢 OK - spatial_miss_uncertainty
#>    Cube contains 0 records with missing coordinate uncertainty.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🔴 VERY_IMPORTANT - obs_min_records
#>    Cube contains 9 observation records (rows).
#>
#> 🟠 IMPORTANT - obs_min_total
#>    Cube contains a total of 22 observations.
```

If you provide a list of rule objects, you can also change the thresholds for the severity values.

```
diagnose_cube(
  cube,
  rules = list(
    rule_temporal_missing_years(),
    rule_spatial_max_uncertainty(
      thresholds = c(ok = 0, note = 1, important = 2, very_important = 3)
    ),
    rule_taxon_min_taxa()
  ),
  sort_summary = "asc"
)
#>
#> Data cube diagnostics
#> ---------------------
#> 🟢 OK - temporal_missing_years
#>    Cube contains 0 missing years.
#>
#> 🟡 NOTE - taxon_min_taxa
#>    Cube contains observations across 3 taxon keys.
#>
#> 🔴 VERY_IMPORTANT - spatial_max_uncertainty
#>    Cube contains 3 records where the coordinate uncertainty is larger
than the grid cell resolution.
```

3.1.3.2. Filtering based on diagnostics with `filter_cube()`

After the diagnostic evaluation we might want to generate a new data cube, or we might want to filter the data. Some `cube_rule` objects have `filter_fn()` functions that return a logical vector indicating which rows should be removed. For the basic rule set, `rule_spatial_max_uncertainty()` and `rule_spatial_miss_uncertainty()` will filter out respectively rows with minimal coordinate uncertainty larger than the grid cell resolution, and rows with missing minimal coordinate uncertainty.

We can pass the full `cube_diagnostics` object. Only rules with a `filter_fn()` function will be applied, while rules without a filtering function are ignored.

After filtering, **dubicube** attempts to rebuild the cube using **b3gbi** to ensure that all metadata remain consistent. In this example, that step initially fails because our synthetic dataset was not created with `b3gbi::process_cube()` (this was done for illustrative purposes only). However, we can still force a successful rebuild by explicitly passing additional arguments to `process_cube()`. For example, setting `grid_type = "none"` allows the cube to be

processed again without requiring a spatial grid. This ensures that the filtered cube is properly reconstructed, including updated metadata.

```
filtered_cube <- filter_cube(
  cube,
  diagnostics = diag,
  process_cube_args = list(grid_type = "none")
)
```

We see that the data with minimal coordinate uncertainty larger than the grid cell resolution (= 1000 m) are removed.

```
cube$data
#>   year cellCode taxonKey obs minCoordinateUncertaintyInMeters
#> 1 2000        A        1  10                              50
#> 2 2000        A        1   5                             100
#> 3 2001        B        1   1                            2000
#> 4 2001        B        2   2                              70
#> 5 2002        C        2   0                            5000
#> 6 2002        C        3   1                            2000
#> 7 2003        D        3   2                              10
#> 8 2003        D        3   1                              20
#> 9 2003        E        3   0                             300

filtered_cube$data
#>   year cellCode taxonKey obs minCoordinateUncertaintyInMeters
#> 1 2000        A        1  10                              50
#> 2 2000        A        1   5                             100
#> 4 2001        B        2   2                              70
#> 7 2003        D        3   2                              10
#> 8 2003        D        3   1                              20
#> 9 2003        E        3   0                             300
```

You can also use the rules directly for filtering.

```
filtered_cube2 <- filter_cube(
  cube,
  rules = list(rule_spatial_max_uncertainty()),
  process_cube_args = list(grid_type = "none")
)
```

3.2. Group-level sensitivity analysis

Biodiversity indicators derived from data cubes inherently aggregate information across multiple dimensions, such as species, grid cells, and time periods. While this aggregation is essential for producing policy-relevant metrics, it can obscure the influence of individual groups on the final indicator values. Group-level sensitivity analysis addresses this challenge by explicitly quantifying how much each group contributes to, or biases, an indicator.

3.2.1. Conceptual basis and methodological approaches

The conceptual basis of group-level sensitivity analysis lies in a resampling procedure, where subsets of the data are systematically left out and the resulting variation in indicator values is evaluated. In the context of biodiversity data cubes, this approach allows analysts to move beyond point estimates and assess the structural robustness of indicator values with respect to their underlying data composition.

A key rationale for this analysis is that biodiversity indicators are rarely uniformly informed by all observations. Instead, they may be disproportionately influenced by specific species (e.g. dominant or well-sampled taxa), particular years (e.g. anomalous climatic conditions), or certain habitats (e.g. overrepresented sampling strata). As a result, indicator values may change substantially when such influential groups are removed. Identifying these dependencies is essential for interpreting indicator values and trends and ensuring their reliability for decision-making.

To operationalise this concept, the **dubicube** package implements group-level cross-validation through the `cross_validate_cube()` function. This function adapts classical cross-validation techniques to the structure of biodiversity data cubes.

Consider a data cube $\mathbf{X}$ from which we want to calculate a statistic θ. The data cube can be grouped, e.g. by `taxon`, which contains multiple categories, e.g. `species1`, `species2`, `species3` ...

- **Original Sample Data**: $\mathbf{X} = X_{11}, X_{12}, X_{13}, \ldots, X_{sn}$
  - The initial set of data points, where there are s different categories in a group (e.g. $s = 10$ species in group `taxon`) 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.
- **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 $\mathbf{X}$.

From $\mathbf{X}$, multiple data cubes $\mathbf{X}_{-s_j}$ are created where in each new dataset a different category is removed compared to the original data cube $\mathbf{X}$. For example in dataset 1, `species1` is excluded; in dataset 2 `species2`; in dataset 3 `species3`; and so on. For each new dataset, the statistic θ is again calculated: $\hat{\theta}_{-s_j}$.

- **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 $\hat{\theta}$.
- **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, $\hat{\theta}_{-s_j} = \bar{X}_{-s_j}$.

From this, we can calculate different error measures that help inform further analyses or decision-making steps.

- **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, normalized 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.

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

- **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.

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

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

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

Several methodological approaches can be applied:

- **Leave-one-out cross-validation (LOO-CV):**
  The most direct approach consists of systematically excluding one group at a time and recalculating the indicator. A prominent example is *leave-one-species-out cross-validation* (LOSO-CV), where each species is removed in turn. This allows the assessment of how strongly individual species influence the overall indicator. Large deviations between the full-data estimate and the reduced-data estimates indicate high sensitivity to specific species. An example of LOSO-CV with **dubicube** is provided in Chapter 3.2.2. *Leave-one-dataset-out cross-validation* was used by Langeraert, Faulkner, et al. (2026) to investigate the influence of component datasets on species prevalence.
- **k-fold cross-validation:**
  Instead of removing one group at a time, groups can be partitioned into *k* subsets (folds). Each fold is omitted in turn, and the indicator is recalculated. This approach is particularly useful when the number of groups is large or when computational efficiency is a concern. It also enables the assessment of variability at broader aggregation levels. This method is experimental and at this moment it is not clear how informative it is compared to LOO-CV.

The interpretation of group-level sensitivity results focuses on the magnitude and consistency of indicator changes under these resampling schemes. Key aspects include:

- **Magnitude of change:** Large deviations from the original indicator value indicate strong dependence on specific groups and potential lack of robustness.
- **Direction of change:** Systematic increases or decreases when certain groups are removed can reveal structural biases.
- **Variability across groups:** High variability in cross-validated estimates suggests uneven data contributions and potential instability.
- **Identification of influential groups:** Groups that cause substantial changes can be flagged for further investigation, such as data quality checks or ecological interpretation.

3.2.2. Assessing sensitivity with dubicube

3.2.2.1. Loading and processing the data

For this example, we use the complete dataset. See Chapter 2.3.

3.2.2.2. Leave-one-species-out cross-validation

We use the `cross_validate_cube()` function to do this. It relies on the following arguments:

- **`data_cube`**:
  The input data, either a processed data cube (from `b3gbi::process_cube()`), or just the dataframe inside (i.e. `processed_cube$data`, see further).

- **`fun`**:
  A user-defined function that computes the statistic(s) of interest from `data_cube`. This function should return a dataframe that includes a column named `diversity_val`, containing the statistic to evaluate.

- **`grouping_var`**:
  The column(s) used for grouping the output of `fun()`. For example, if `fun()` returns one value per year, use `grouping_var = "year"`.

- **`out_var`**:
  The variable used to leave out one group at a time during cross-validation. The default is `"taxonKey"` (typically representing species), which results in leave-one-species-out cross-validation.

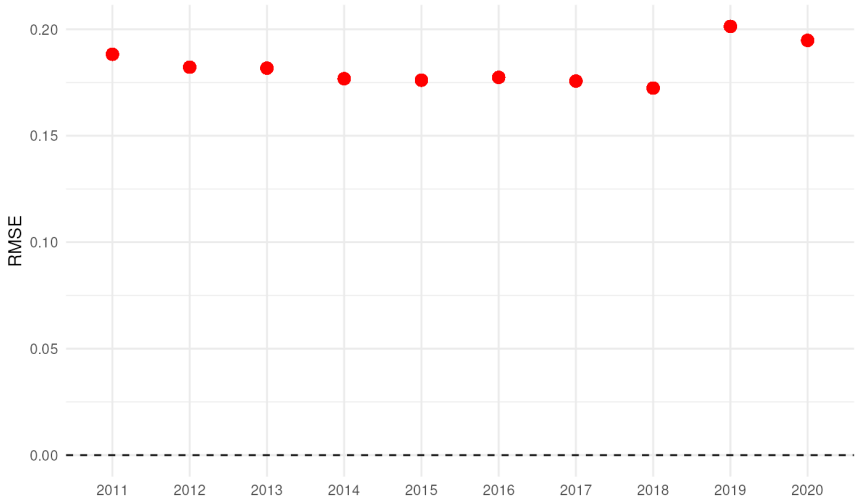
- **`progress`**:
  Logical flag to show a progress bar. Set to `TRUE` to enable progress reporting; default is `FALSE`.

```
cv_results <- cross_validate_cube(
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  out_var = "taxonKey"
)

head(cv_results)
#>   id_cv year taxonkey_out   rep_cv est_original       error     sq_error
#> 1     1 2011      2474051 48.85551     48.83864  0.01686866 0.0002845518
#> 2     2 2011      2474377 48.49272     48.83864 -0.34592150 0.1196616847
#> 3     3 2011      2474831 48.91447     48.83864  0.07583503 0.0057509524
#> 4     4 2011      2478523 48.66861     48.83864 -0.17002678 0.0289091060
```

```
#> 5     5 2011      2480242 48.56884     48.83864 -0.26979464 0.0727891466
#> 6     6 2011      2480332 48.97265     48.83864  0.13401280 0.0179594315
#>    abs_error    rel_error perc_error         mre        mse      rmse
#> 1 0.01686866 0.0003453959 0.03453959 0.001122516 0.03544285 0.1882627
#> 2 0.34592150 0.0070829473 0.70829473 0.001122516 0.03544285 0.1882627
#> 3 0.07583503 0.0015527672 0.15527672 0.001122516 0.03544285 0.1882627
#> 4 0.17002678 0.0034813989 0.34813989 0.001122516 0.03544285 0.1882627
#> 5 0.26979464 0.0055242048 0.55242048 0.001122516 0.03544285 0.1882627
#> 6 0.13401280 0.0027439914 0.27439914 0.001122516 0.03544285 0.1882627
```

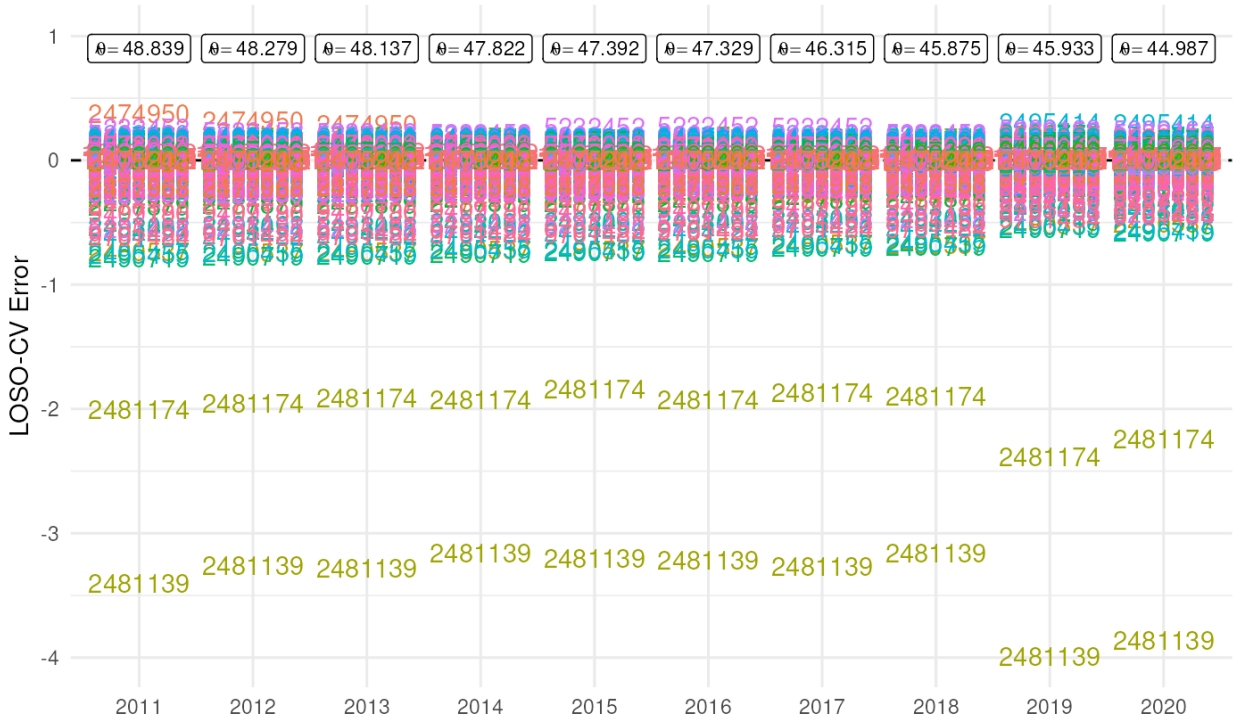
The RMSE is an average error measure we obtain for each year. It remains very similar over the years.

```
# Visualise mean errors
ggplot(cv_results, aes(x = as.factor(year))) +
  # Reference line
  geom_hline(yintercept = 0, colour = "black", linetype = "dashed") +
  # Plot RMSE
  geom_point(aes(y = rmse), colour = "red", size = 3) +
  # Settings
  labs(x = "", y = "RMSE") +
  theme_minimal()
```



Indeed, looking at the individual error values reveals similar patterns for every year. The species with taxon key 2481174 has an error of -2 in year 2011. This means that without this species the average number of occurrences per grid cell would be 2 units lower than the estimate based on the full dataset (in this case around 46.8 instead of 48.8).

```
# Get original estimates (based on full data cube)
original_estimates <- mean_obs(processed_cube$data)
original_estimates$label <- paste(
  "hat(theta) ==", round(original_estimates$diversity_val, 3)
)

# Visualise errors
ggplot(cv_results, aes(x = as.factor(year))) +
  # Reference line
  geom_hline(yintercept = 0, colour = "black", linetype = "dashed") +
  # Plot species keys
  geom_text(aes(y = error, colour = as.factor(taxonkey_out),
                label = as.factor(taxonkey_out))) +
  # Plot original estimates
  geom_label(data = original_estimates, aes(label = label),
             y = 0.9, parse = TRUE, size = 3) +
  # Settings
  scale_y_continuous(limits = c(NA, 1)) +
  labs(x = "", y = "LOSO-CV Error") +
  theme_minimal() +
  theme(legend.position = "")
```



There are two species that have a big effect on the calculation of our statistic.

- 2481174: *Larus fuscus* Linnaeus, 1758
- 2481139: *Larus argentatus* Pontoppidan, 1763

If we go back to the source of the data, we see that the data cube includes bird tracking data from two large datasets strictly dedicated to these two species (2481174, 2481139).

```
index <- match("bird_cube_belgium_mgrs10", resources(b3data_package))
b3data_package$resources[[index]]$sources
#> [[1]]
#> [[1]]$title
#> [1] "GBIF Occurrence Download"
#>
#> [[1]]$path
#> [1] "https://doi.org/10.15468/dl.y3wpwk"
```

This suggests that the two species are overrepresented in the data cube, likely due to targeted monitoring efforts. Their disproportionate influence inflates indicator values, potentially distorting the true diversity signal. Group-level sensitivity analysis like this helps uncover such sampling biases, ensuring that biodiversity indicators reflect ecological reality rather than artifacts in the data.

3.2.2.3. Advanced usage of `cross_validate_cube()`

Cross-validate simple dataframes

As stated in the documentation, it is also possible to cross-validate over a dataframe. In this case, set the argument `processed_cube = FALSE`. This is implemented to allow for flexible use of simple dataframes, while still encouraging the use of `b3gbi::process_cube()` as default functionality.

```
cv_results_df <- cross_validate_cube(
  data_cube = processed_cube$data,      # data.frame object
  fun = mean_obs,
  grouping_var = "year",
  out_var = "taxonKey",
  processed_cube = FALSE
)
```

Optional arguments

The `cross_validate_cube()` function includes several optional parameters that give you more control over how the cross-validation is performed. These arguments are not required for basic use, but can be helpful in specific scenarios.

- **Cross-validation Method**: **`crossv_method`**
  Determines how data is partitioned for cross-validation.
  - `"loo"` (default):
    Leave-one-out cross-validation. One category in `out_var` (e.g. one species) is excluded one at a time.

  - `"kfold"`:
    K-fold cross-validation. All categories in `out_var` are split into `k` subsets, and each subset is excluded once while the others are used for analysis.
    Note: this method is experimental and results should be interpreted with caution.

- **Number of Folds**: **`k`**
  Specifies the number of folds when using `"kfold"` cross-validation. The default is `5`. Only used when `crossv_method = "kfold"`.

- **Maximum Number of Categories**: **`max_out_cats`**
  Sets an upper limit on the number of unique categories in `out_var` that will be excluded one-by-one during cross-validation. The default is `1000`. This helps to prevent long runtimes for datasets with many unique categories. You can increase this if needed, but expect slower computation.

4. Indicator uncertainty calculation

To quantify indicator uncertainty, we followed the guidelines of Rowland et al. (2021) that advise the selection of an appropriate approach to present uncertainty in biodiversity indicators (Fig. 3). Based on these guidelines, we propose the calculation of confidence intervals for indicators calculated from occurrence cubes using the bootstrap resampling method (hereafter, bootstrapping).

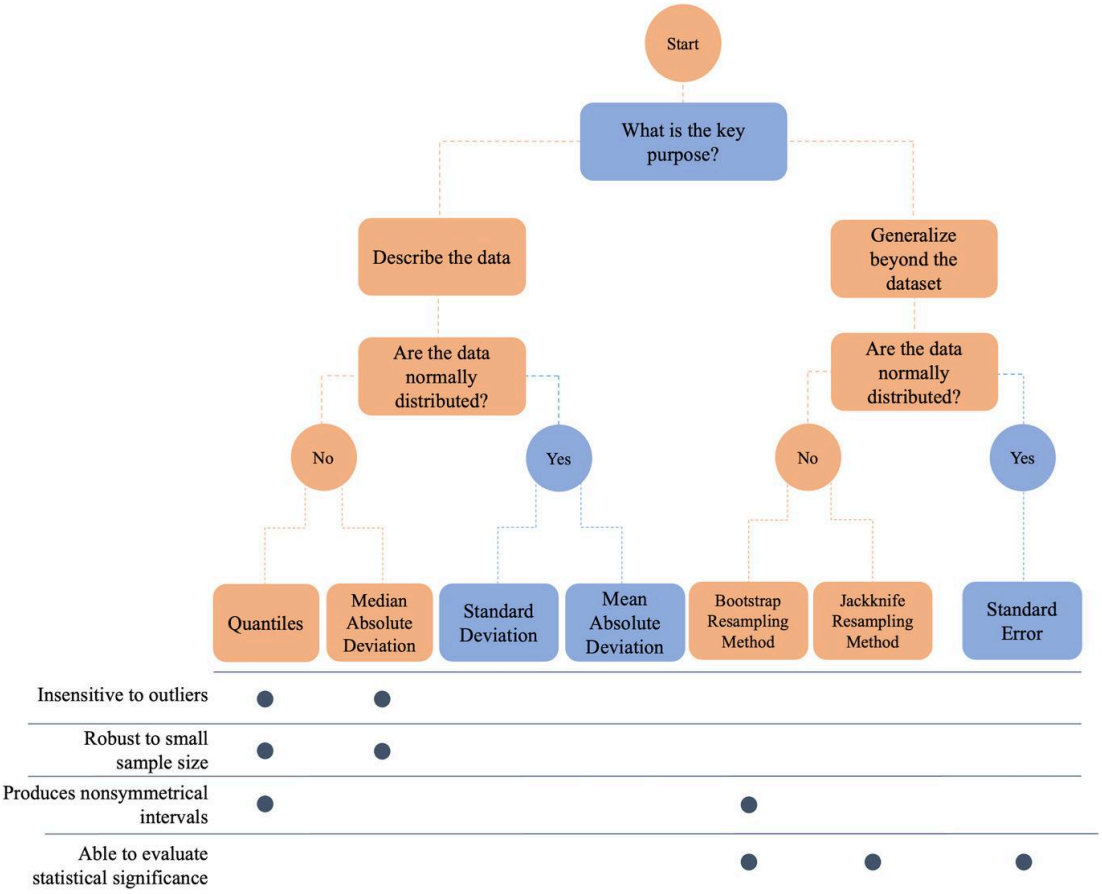


Figure 3: Figure 1 from Rowland et al. (2021): "Decision tree for presenting data variability or uncertainty in biodiversity indicators with interval methods. This list of approaches is not exhaustive."

4.1. Bootstrap methodology for data cubes

4.1.1. Concept and rationale of the bootstrap approach

Bootstrapping provides a flexible, non-parametric approach to estimate the variability of indicators without relying on strong assumptions about the underlying data distribution (Davison & Hinkley, 1997; Efron & Tibshirani, 1994). This flexibility is particularly useful in this context, as both occurrence cube datasets and the derived indicators can be very distinct in nature (Dixon, 2001). Each occurrence cube has a unique spatial, temporal, taxonomic … scope, and spatial and temporal indicators are developed related to prevalence, abundance, phylogenetic diversity, impact of alien species, etc. (Breugelmans et al., 2025; Dove, 2026; Yahaya et al., 2026). In

bootstrapping, the dataset is resampled multiple times with replacement. For each resampled dataset, the statistic of interest, e.g. a biodiversity indicator, is calculated. Based on the distribution of these indicator values, we get an idea about indicator uncertainty, e.g. via calculation of confidence intervals (Fig. 4). Note that we make a distinction between whole-cube bootstrap and group-specific bootstrap strategies. This is further explained in Chapter 4.3.

Consider a data cube $\mathbf{X}$ from which we want to calculate a statistic θ.

- **Original Sample Data**: $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$
    - The initial set of 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.
- **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 $\mathbf{X}$.

From $\mathbf{X}$, multiple resampled datasets $\mathbf{X}^*$ are created by sampling rows (cells) with replacement until each new dataset is equally large as $\mathbf{X}$. This process is repeated B times, and each resample yields a new estimate $\hat{\theta}_b^*$ of the statistic.

- **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 below).
- **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, $\hat{\theta}_b^* = \bar{X}_b^*$.

The set of bootstrap replications forms the bootstrap distribution, which can be used to estimate bootstrap statistics and construct confidence intervals (see below).

- **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.

$$\text{SE}_{\text{boot}} = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}_b^* - \hat{\theta}_{\text{boot}})^2}$$

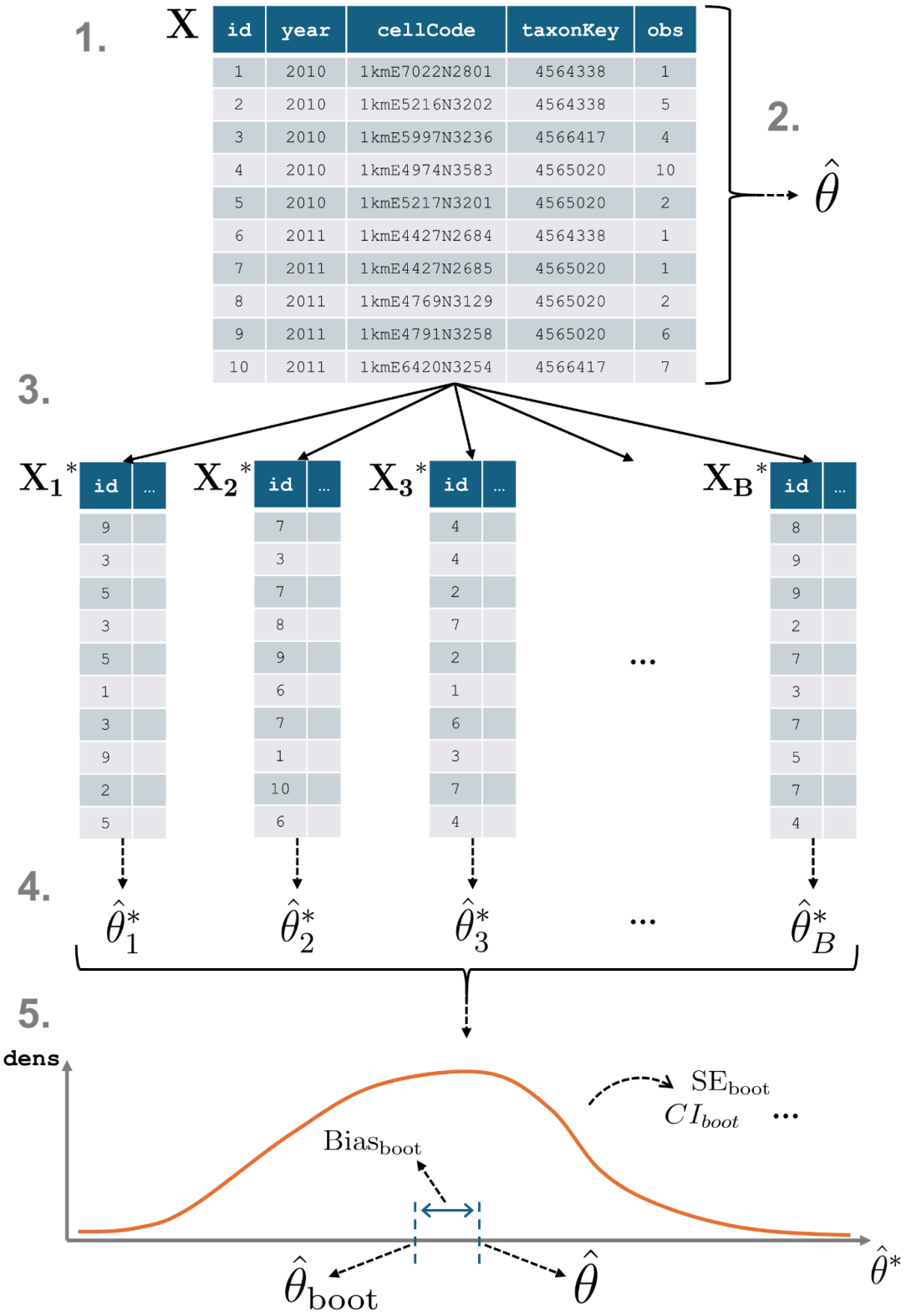


Figure 4: Use of bootstrapping for the quantification of indicator uncertainty. See the report text for an explanation of the mathematical notation.

4.1.2. Bootstrapping with dubicube

4.1.2.1. Loading and processing the data

For this example, we use the small dataset. See Chapter 2.3.

4.1.2.2. Bootstrapping

We use the `bootstrap_cube()` function to do this. It relies on the following arguments:

- **`data_cube`**:
  The input data as a processed data cube (from `b3gbi::process_cube()`).

- **`fun`**:
  A user-defined function that computes the statistic(s) of interest from `data_cube$data`. This function should return a dataframe that includes a column named `diversity_val`, containing the statistic to evaluate.

- **`grouping_var`**:
  The column(s) used for grouping the output of `fun()`. For example, if `fun()` returns one value per year, use `grouping_var = "year"`.

- **`samples`**:
  The number of bootstrap samples to draw. Common values are 1000 or more for reliable estimates of variability and confidence intervals.

- **`seed`**:
  An optional numeric seed to ensure reproducibility of the bootstrap resampling. Set this to a fixed value to get consistent results across runs.

- **`progress`**:
  Logical flag to show a progress bar. Set to `TRUE` to enable progress reporting; default is `FALSE`.

```
bootstrap_results <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)
#> [1] "Performing whole-cube bootstrap with `boot::boot()`."
```

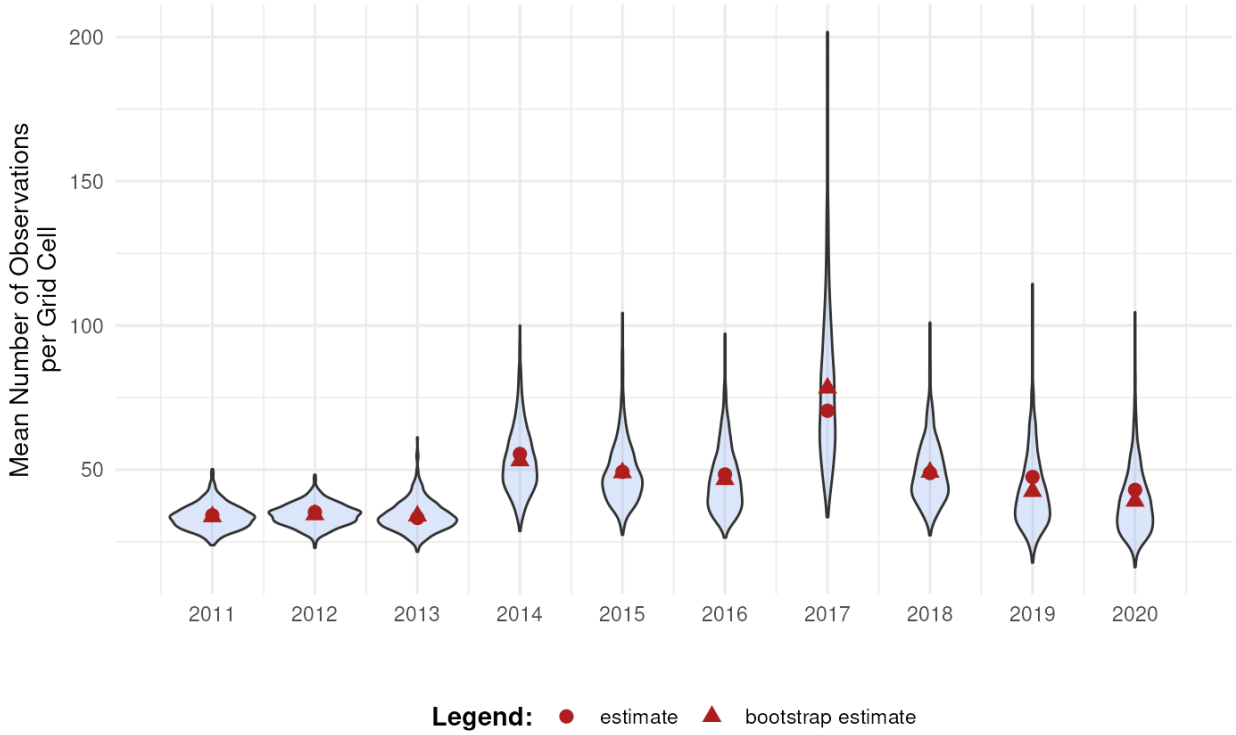
This returned a list of `boot` objects. We can convert this to a dataframe. The informative message and output type will be clarified below.

```
bootstrap_results_df <- boot_list_to_dataframe(
  boot_list = bootstrap_results,
  grouping_var = "year"
) %>%
  mutate(year = as.numeric(year))

head(bootstrap_results_df)
#>   sample year est_original rep_boot est_boot  se_boot  bias_boot
#> 1      1 2011     34.17777 31.15094  33.7093 4.304673 -0.4684721
#> 2      2 2011     34.17777 32.45489  33.7093 4.304673 -0.4684721
#> 3      3 2011     34.17777 29.31098  33.7093 4.304673 -0.4684721
#> 4      4 2011     34.17777 34.10232  33.7093 4.304673 -0.4684721
#> 5      5 2011     34.17777 25.46354  33.7093 4.304673 -0.4684721
#> 6      6 2011     34.17777 33.85088  33.7093 4.304673 -0.4684721
```

We can visualise the bootstrap distributions using a violin plot.

```
# Get bias values
bias_mean_obs <- bootstrap_results_df %>%
  distinct(year, estimate = est_original, `bootstrap estimate` = est_boot)

# Get estimate values
estimate_mean_obs <- bias_mean_obs %>%
  pivot_longer(cols = c("estimate", "bootstrap estimate"),
               names_to = "Legend", values_to = "value") %>%
  mutate(
    Legend = factor(Legend, levels = c("estimate", "bootstrap estimate"),
                    ordered = TRUE)
  )

# Visualise bootstrap distributions
bootstrap_results_df %>%
  ggplot(aes(x = year)) +
  # Distribution
  geom_violin(aes(y = rep_boot, group = year),
              fill = alpha("cornflowerblue", 0.2)) +
  # Estimates and bias
  geom_point(data = estimate_mean_obs, aes(y = value, shape = Legend),
             colour = "firebrick", size = 2.5) +
```

```
# Settings
labs(y = "Mean Number of Observations\nper Grid Cell",
     x = "", shape = "Legend:") +
scale_x_continuous(breaks = sort(unique(bootstrap_results_df$year))) +
theme_minimal() +
theme(legend.position = "bottom",
      legend.title = element_text(face = "bold"))
```



4.1.2.3. Advanced usage of `bootstrap_cube()`

Bootstrap simple dataframes

As stated in the documentation, it is also possible to bootstrap a dataframe. In this case, set the argument `processed_cube = FALSE`. This option is implemented to allow for flexible use of simple dataframes, while still encouraging the use of `b3gbi::process_cube()` as default functionality.

```
bootstrap_results_df <- bootstrap_cube(
  data_cube = processed_cube_small$data,      # data.frame object
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  method = "whole_cube",
  seed = 123,
  processed_cube = FALSE
)
```

Comparison with a reference group
A particularly insightful approach is comparing indicator values to a reference group. In time series analyses, this often means comparing each year's indicator to a baseline year (e.g., the first or last year in the series).

To do this, we perform bootstrapping over the differences between indicator values:

1. Resample the dataset with replacement
2. Calculate the indicator for each group (e.g., each year)
3. For each non-reference group, compute the difference between its indicator value and that of the reference group
4. Repeat steps 1–3 across all bootstrap iterations

This process yields bootstrap replicate distributions of differences in indicator values.

```
bootstrap_results_ref <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  ref_group = 2011,
  seed = 123
)
#> [1] "Performing whole-cube bootstrap."
```

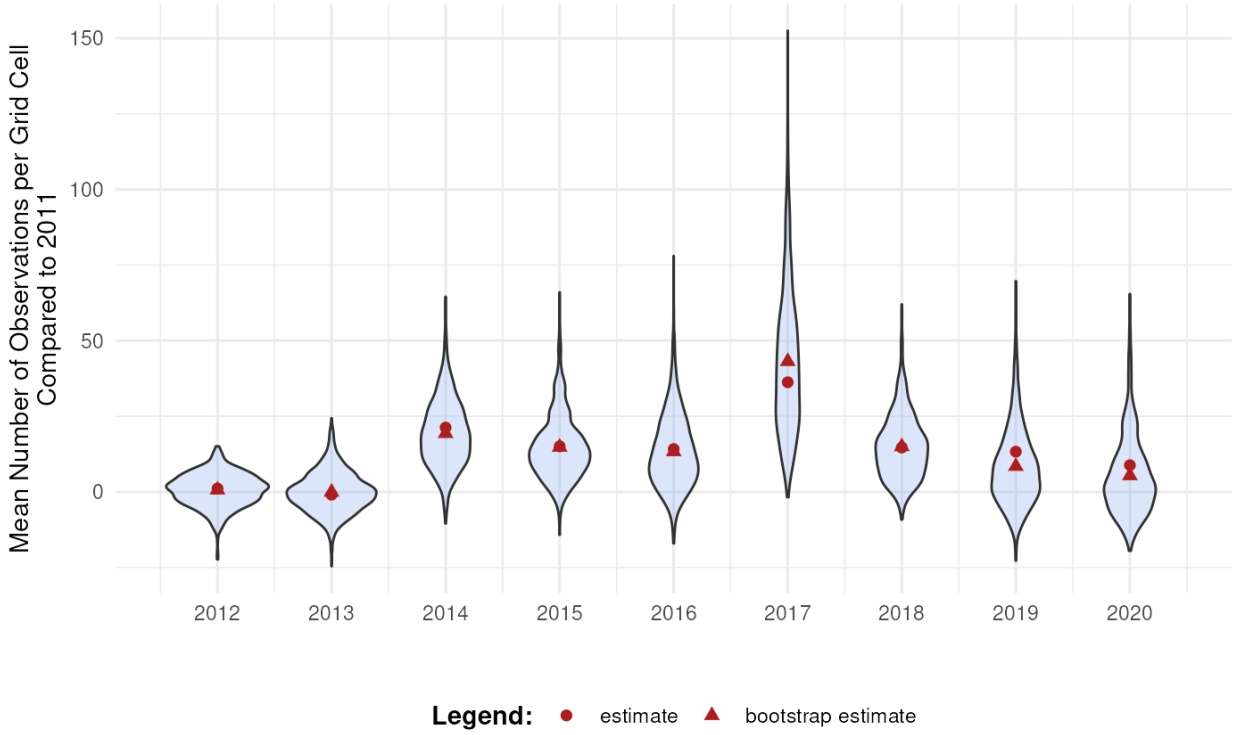
This time, a dataframe is returned since the **boot** package cannot be used with a reference group (see below).

```
head(bootstrap_results_ref)
#>   sample year est_original   rep_boot  est_boot  se_boot  bias_boot
#> 1      1 2012     1.094245  8.1881078 0.6583191 5.475053 -0.4359261
#> 2      2 2012     1.094245  7.6061946 0.6583191 5.475053 -0.4359261
```

```
#> 3      3 2012     1.094245 -4.6058908 0.6583191 5.475053 -0.4359261
#> 4      4 2012     1.094245  2.4102039 0.6583191 5.475053 -0.4359261
#> 5      5 2012     1.094245  6.2626545 0.6583191 5.475053 -0.4359261
#> 6      6 2012     1.094245 -0.1577162 0.6583191 5.475053 -0.4359261
```

We see that the mean number of observations is higher in most years compared to 2011. At what point can we say these differences are significant? This will be further explored in Chapter 5.1.

```
# Get bias values
bias_mean_obs <- bootstrap_results_ref %>%
  distinct(year, estimate = est_original, `bootstrap estimate` = est_boot)

# Get estimate values
estimate_mean_obs <- bias_mean_obs %>%
  pivot_longer(cols = c("estimate", "bootstrap estimate"),
               names_to = "Legend", values_to = "value") %>%
  mutate(
    Legend = factor(Legend, levels = c("estimate", "bootstrap estimate"),
                    ordered = TRUE)
  )

# Visualise bootstrap distributions
bootstrap_results_ref %>%
  ggplot(aes(x = year)) +
  # Distribution
  geom_violin(aes(y = rep_boot, group = year),
              fill = alpha("cornflowerblue", 0.2)) +
  # Estimates and bias
  geom_point(data = estimate_mean_obs, aes(y = value, shape = Legend),
             colour = "firebrick", size = 2) +
  # Settings
  labs(y = "Mean Number of Observations per Grid Cell\nCompared to 2011",
       x = "", shape = "Legend:") +
  scale_x_continuous(breaks = sort(unique(bootstrap_results_ref$year))) +
  theme_minimal() +
  theme(legend.position = "bottom",
        legend.title = element_text(face = "bold"))
```



Note that the choice of the reference year should be well considered. Keep in mind which comparisons should be made, and what the motivation is behind the reference period. A high or low value in the reference period relative to other periods, e.g. an exceptional bad or good year, can affect the magnitude and direction of the calculated differences. Whether this should be avoided or not, depends on the motivation behind the choice and the research question. A reference period can be determined by legislation, or by the start of a monitoring campaign. A specific research question can determine the periods that need to be compared. Furthermore, the variability of the estimate of reference period affects the width of confidence intervals for the differences. A more variable reference period will propagate greater uncertainty. In the case of GBIF data, more data will be available in recent years than in earlier years. If this is the case, it could make sense to select the last period as a reference period. In a way, this also avoids the arbitrariness of choosing the reference period. You compare previous situations with the current situation (last year), where you could repeat this comparison annually, for example. Finally, when comparing multiple indicators, we recommend using a consistent reference period to maintain comparability.

Bootstrap method

The **dubicube** package supports multiple bootstrap strategies depending on the type of indicator, the structure of the data, and whether a reference group is used. Users rarely need to specify the method explicitly, as the default `method = "smart"` automatically selects the appropriate approach. This is further explained in Chapter 4.3.

4.2. Calculating bootstrap confidence intervals

4.2.1. Concept and different types of bootstrap confidence intervals

Bootstrapping can be used to assess the uncertainty about an indicator estimate via confidence intervals (Davison & Hinkley, 1997; Dixon, 2001; Rowland et al., 2021). 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 smooth implementation in software. They are based on the definitions provided by Davison & Hinkley (1997, Chapter 5) (see also DiCiccio & Efron, 1996; Efron, 1987).

4.2.1.1. Interval types

1. Percentile
Uses the percentiles of the bootstrap distribution.

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

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 centre of the bootstrap distribution of the statistic. The bias correction term is calculated as:

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

where $\#$ is the counting operator, counting the number of times $\hat{\theta}^*_b$ is smaller than $\hat{\theta}$, and Φ^{-1} is the inverse cumulative density function of the standard normal distribution. B is the number of bootstrap samples.

Acceleration quantifies how sensitive the variability of the statistic is to changes in the data. See further for how this is calculated:

- $a = 0$: 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 bias and acceleration estimates are then used to calculate adjusted percentiles:

$$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z_{\alpha/2})}\right), \quad \alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z_{1-\alpha/2})}\right)$$

So, we get:

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

3. Normal
Assumes the bootstrap replications are normally distributed:

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

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

4. Basic
Centres the interval using percentiles:

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

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.

4.2.1.2. Acceleration

The acceleration 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.

In the case of the BCa interval, `calculate_bootstrap_ci()` uses the function `calculate_acceleration()` to calculate acceleration (unless a `boot_*` method is used, see further). The latter can also be used on its own to calculate acceleration values to quantify

the sensitivity of a statistic's variability to changes in the dataset. For jackknifing, it uses the `perform_jackknifing()` helper function which is not exported by **dubicube**.

If the BCa interval is calculated and a reference group is used, jackknifing is implemented differently. Consider $\hat{\theta} = \hat{\theta}_1 - \hat{\theta}_2$ where $\hat{\theta}_1$ is the estimate for the indicator value of a non-reference period (sample size n_1) and $\hat{\theta}_2$ is the estimate for the indicator value of a reference period (sample size n_2).

The acceleration is now calculated as follows:

$$\hat{a} = \frac{1}{6} \frac{\sum_{i=1}^{n_1+n_2} (I_i^3)}{\left(\sum_{i=1}^{n_1+n_2} (I_i^2)\right)^{3/2}}$$

I_i can be calculated using the negative or positive jackknife. Such that

$\hat{\theta}_{-i} = \hat{\theta}_{1,-i} - \hat{\theta}_2$ for $i = 1, \ldots, n_1$, and

$\hat{\theta}_{-i} = \hat{\theta}_1 - \hat{\theta}_{2,-i}$ for $i = n_1 + 1, \ldots, n_1 + n_2$

Therefore, if you want to calculate the BCa intervals using `calculate_bootstrap_ci()`, you also need to provide a `ref_group` argument. Since we are not working with `boot` objects, we need to specify `data_cube` and `fun` as well. This might feel rather cumbersome for users and will no longer be necessary once we adopt an object-oriented approach (Chapter 6.2).

4.2.2. Calculating bootstrap confidence intervals with dubicube

4.2.2.1. Loading and processing the data

For this example, we use the small dataset. See Chapter 2.3.

4.2.2.2. Bootstrapping

We use the `bootstrap_cube()` function to perform bootstrapping as introduced before.

```
bootstrap_results <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)
#> [1] "Performing whole-cube bootstrap with `boot::boot()`."
```

4.2.2.3. Interval calculation

Now we can use the `calculate_bootstrap_ci()` function to calculate confidence limits. It relies on the following arguments:

- **`bootstrap_results`**: A dataframe containing the bootstrap replicates, where each row represents a bootstrap sample. As returned by `bootstrap_cube()`.

- **`grouping_var`**: The column(s) used for grouping the output of `fun()`. For example, if `fun()` returns one value per year, use `grouping_var = "year"`.

- **`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`**: The confidence level of the intervals. Default is `0.95` (95 % confidence level).

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

- **`data_cube`**: Only used when `type = "bca"` and no boot method is used (see further). The input data as a processed data cube (from `b3gbi::process_cube()`).

- **`fun`**: Only used when `type = "bca"` and no boot method is used. A user-defined function that computes the statistic(s) of interest from `data_cube$data`. This function should return a dataframe containing a column named `diversity_val` with the statistic to be evaluated.

- **`progress`**: Logical flag to show a progress bar. Set to `TRUE` to enable progress reporting; default is `FALSE`.

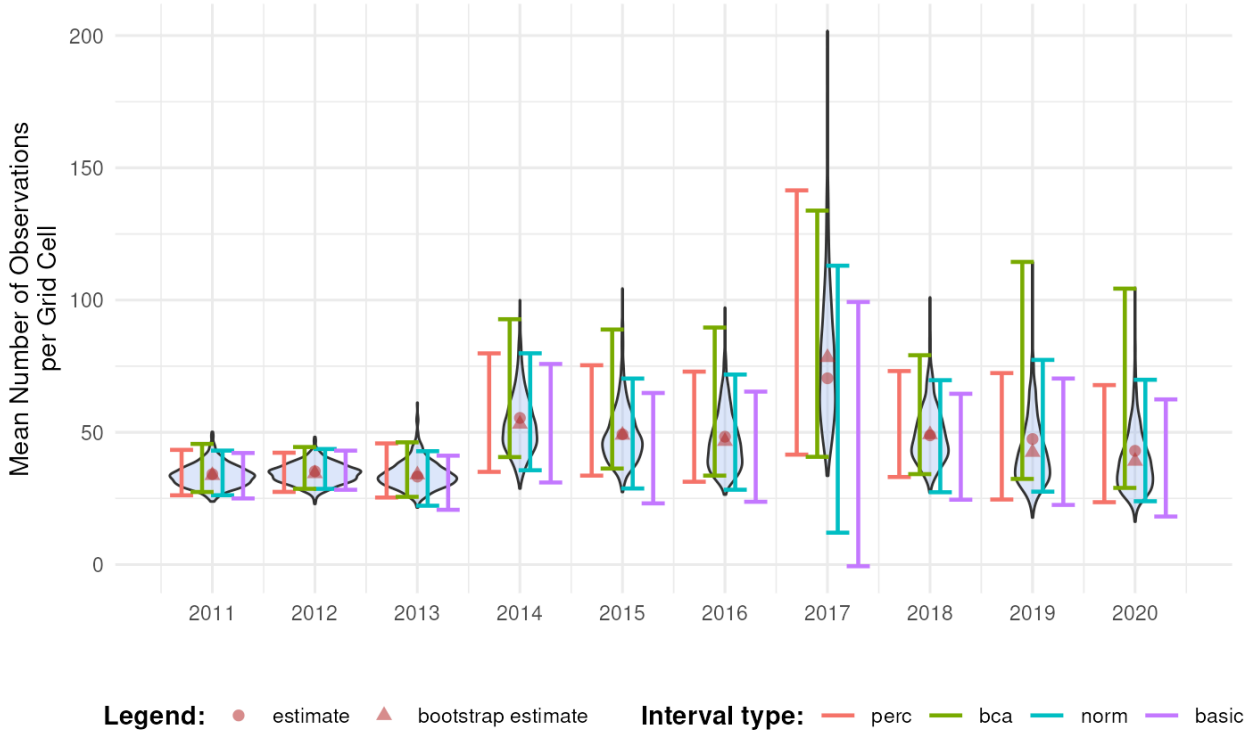
We get a warning message for BCa calculation because we are using a relatively small dataset. Since we are working with `boot` objects, we do not need to specify `data_cube` or `fun`.

```
ci_mean_obs <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_results,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic"),
  conf = 0.95
)
#> Warning in norm.inter(t, adj.alpha): extreme order statistics used as
endpoints

head(ci_mean_obs)
#>   year est_original int_type       ll       ul conf
#> 1 2011     34.17777     norm 26.20924 43.08325 0.95
#> 2 2011     34.17777    basic 24.99408 42.17612 0.95
#> 3 2011     34.17777     perc 26.17942 43.36146 0.95
#> 4 2011     34.17777      bca 27.43845 45.60900 0.95
#> 5 2012     35.27201     norm 28.62332 43.67756 0.95
#> 6 2012     35.27201    basic 28.26708 43.07078 0.95
```

We visualise the distribution of the bootstrap replicates and the confidence intervals.

```
# Make interval type a factor
ci_mean_obs <- ci_mean_obs %>%
  mutate(
    year = as.numeric(year),
    int_type = factor(
      int_type, levels = c("perc", "bca", "norm", "basic")
    )
  )

# Convert bootstrap replicates to dataframe
bootstrap_results_df <- boot_list_to_dataframe(
  boot_list = bootstrap_results,
  grouping_var = "year"
) %>%
  mutate(year = as.numeric(year))
```

```r
# Get bias values
bias_mean_obs <- bootstrap_results_df %>%
  distinct(year, estimate = est_original, `bootstrap estimate` = est_boot)

# Get estimate values
estimate_mean_obs <- bias_mean_obs %>%
  pivot_longer(cols = c("estimate", "bootstrap estimate"),
               names_to = "Legend", values_to = "value") %>%
  mutate(
    Legend = factor(Legend, levels = c("estimate", "bootstrap estimate"),
                    ordered = TRUE)
  )

# Visualise
bootstrap_results_df %>%
  ggplot(aes(x = year)) +
  # Distribution
  geom_violin(aes(y = rep_boot, group = year),
              fill = alpha("cornflowerblue", 0.2)) +
  # Estimates and bias
  geom_point(data = estimate_mean_obs, aes(y = value, shape = Legend),
             colour = "firebrick", size = 2, alpha = 0.5) +
  # Intervals
  geom_errorbar(data = ci_mean_obs,
                aes(ymin = ll, ymax = ul, colour = int_type),
                position = position_dodge(0.8), linewidth = 0.8) +
  # Settings
  labs(y = "Mean Number of Observations\nper Grid Cell",
       x = "", shape = "Legend:", colour = "Interval type:") +
  scale_x_continuous(breaks = sort(unique(bootstrap_results_df$year))) +
  theme_minimal() +
  theme(legend.position = "bottom",
        legend.title = element_text(face = "bold"))
```



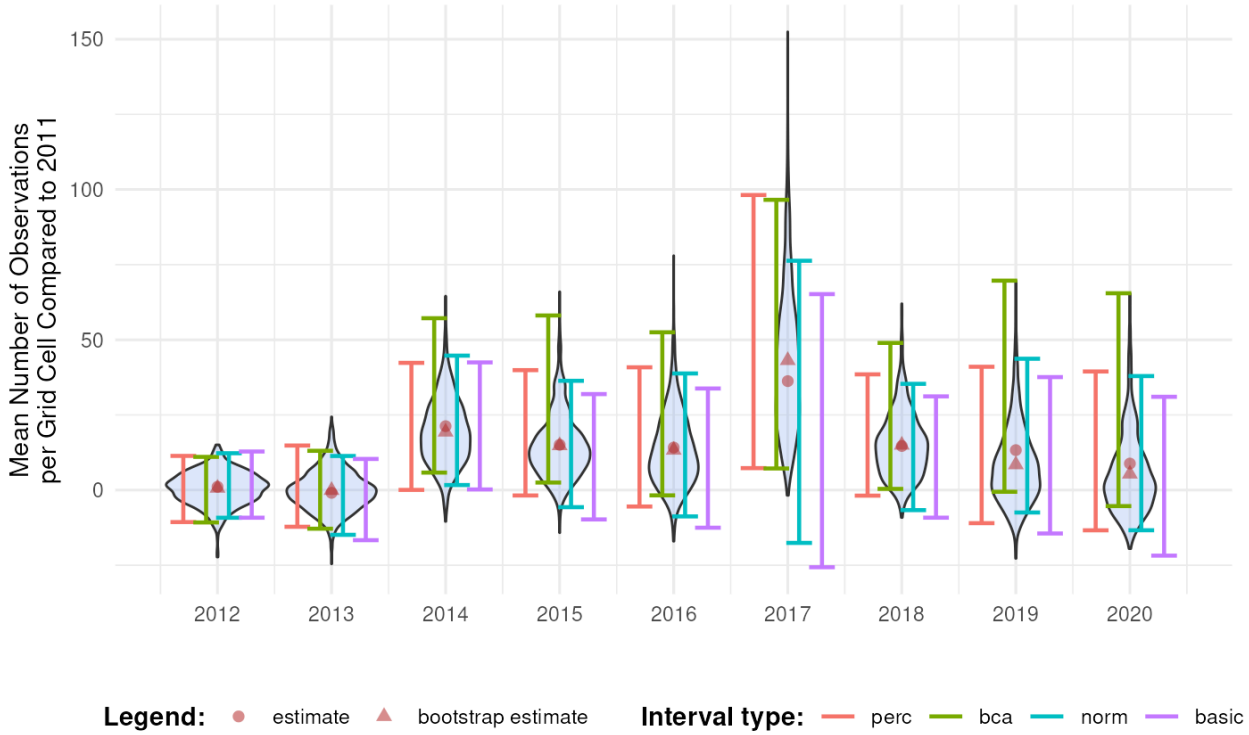
4.2.2.4. Advanced usage of `calculate_bootstrap_ci()`

Comparison with a reference group

As discussed earlier, we can also compare indicator values to a reference group. In this case we need to provide the `ref_group` argument.

```
bootstrap_results_ref <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  ref_group = 2011,
  seed = 123
)
#> [1] "Performing whole-cube bootstrap."
```

And the results are now the difference with our reference year.



Transformations

Consider the calculation of Pielou's evenness on a random subset of the data from 2011-2015. We take only a small subset of the dataset and we artificially create a community with high evenness. This is an indicator that has values between 0 and 1. Higher evenness values indicate a more balanced community (a value of 1 means that all species are equally abundant), while low values indicate a more unbalanced community (a value of 0 means that one species dominates completely).

```
set.seed(123)

# Make dataset smaller
rows <- sample(nrow(bird_cube_belgium), 1000)
bird_cube_belgium_even <- bird_cube_belgium[rows, ]

# Make dataset even
bird_cube_belgium_even$n <- rnbinom(nrow(bird_cube_belgium_even),
                                    size = 2, mu = 100)
```

```
# Process cube
processed_cube_even <- b3gbi::process_cube(
  bird_cube_belgium_even,
  first_year = 2011,
  last_year = 2015,
  cols_occurrences = "n"
)
```

We create a custom function to calculate evenness:

```
calc_evenness <- function(data) {
  data %>%
    # Calculate number of observations
    group_by(year, scientificName) %>%
    summarise(obs = sum(obs), .groups = "drop_last") %>%
    # Calculate evenness by year
    mutate(
      tot = sum(obs),
      p = obs / tot,
      p_ln_p = p * log(p),
      ln_S = log(n_distinct(scientificName)),
      diversity_val = (-sum(p_ln_p)) / ln_S
    ) %>%
    ungroup() %>%
    # Get distinct values
    distinct(year, diversity_val)
}
```

We perform bootstrapping as before. Note that you can also perform bootstrapping of `processed_cube_even` using the **b3gbi** function `pielou_evenness_ts()`.

```
bootstrap_results_evenness <- bootstrap_cube(
  data_cube = processed_cube_even,
  fun = calc_evenness,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)
#> [1] "Performing group-specific bootstrap with `boot::boot()`."
```

We calculate the percentile, BCa, normal and basic intervals with `calculate_bootstrap_ci()`.

```
ci_evenness <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_results_evenness,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic")
)
```

We make a similar visualisation as before. We notice that the normal and basic intervals have limits larger than 1 which is an impossible value for evenness. This is because their intervals are symmetrical around $\hat{\theta} - \text{Bias}_{\text{boot}}$. We can use transformation functions to account for this.

When using a transformation, the intervals are calculated on the scale of `h` and the inverse function `hinv` are applied to the resulting intervals. For values between 0 and 1, we can use the logit function and its inverse:

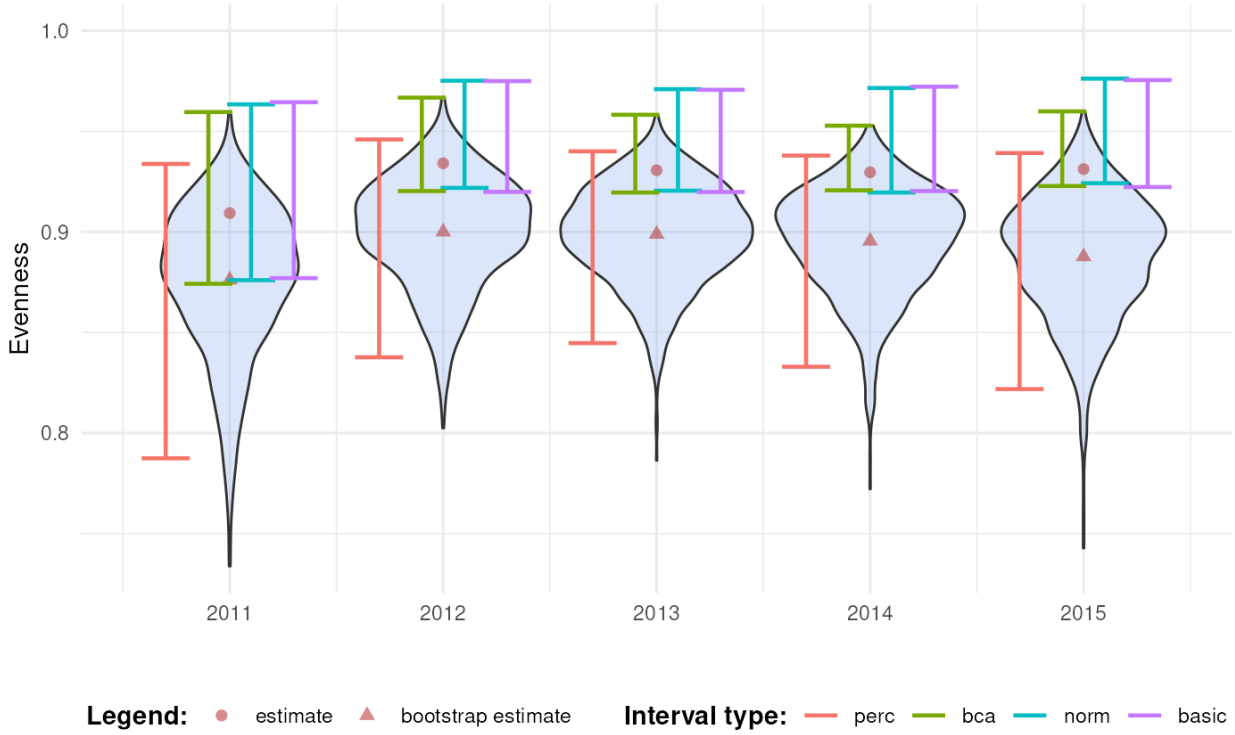
```
# Logit transformation
logit <- function(p) {
  log(p / (1 - p))
}
```

```
# Inverse logit transformation
inv_logit <- function(l) {
  exp(l) / (1 + exp(l))
}
```

We enter them through `calculate_bootstrap_ci()`.

```
ci_evenness_trans <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_results_evenness,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic"),
  h = logit,
  hinv = inv_logit
)
```

Now we see that all the intervals fall within the expected range.



Issues with bias correction for species richness indicators
Consider the calculation of observed species richness on the same subset used in the previous part. We create a custom function to calculate richness:

```
calc_richness <- function(data) {
  data %>%
    group_by(year) %>%
    summarise(diversity_val = n_distinct(scientificName),
              .groups = "drop")
}
```

We perform bootstrapping as before. We will not use any boot method (`method = "group_specific"`, see further).

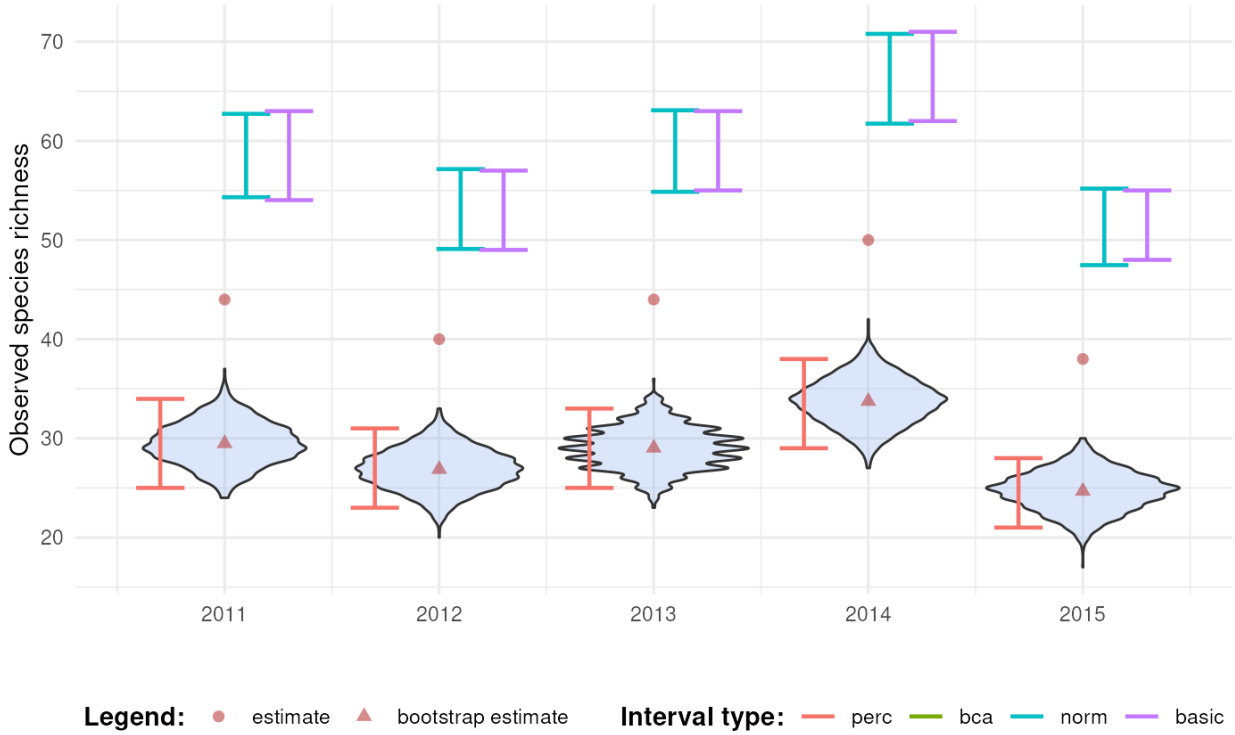
```
bootstrap_results_richness <- bootstrap_cube(
  data_cube = processed_cube_even,
  fun = calc_richness,
  grouping_var = "year",
  samples = 1000,
  method = "group_specific",
  seed = 123
)
#> [1] "Performing group-specific bootstrap."
```

We calculate the percentile, BCa, normal and basic intervals with `calculate_bootstrap_ci()`. We get a warning message for BCa calculation. The bias is infinite such that the BCa intervals cannot be calculated.

```
ci_richness <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_results_richness,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic"),
  data_cube = processed_cube_even,
  fun = calc_richness
)
#> Warning in bca_ci(t0 = unique(df$est_original), t = df$rep_boot, a = a,:
#> Estimated adjustment 'z0' is infinite.
#> Warning in bca_ci(t0 = unique(df$est_original), t = df$rep_boot, a = a,:
#> Estimated adjustment 'z0' is infinite.
#> Warning in bca_ci(t0 = unique(df$est_original), t = df$rep_boot, a = a,:
#> Estimated adjustment 'z0' is infinite.
```

```
#> Warning in bca_ci(t0 = unique(df$est_original), t = df$rep_boot, a = a,:
#> Estimated adjustment 'z0' is infinite.
#> Warning in bca_ci(t0 = unique(df$est_original), t = df$rep_boot, a = a,:
#> Estimated adjustment 'z0' is infinite.
```

We notice that none of the intervals cover the estimate. The percentile interval does not account for bias, the BCa interval cannot be calculated because the bias is too large and the normal and basic intervals have overcompensated because of the large bootstrap bias.



This issue arises because bootstrap resampling cannot introduce new species that were not present in the original sample (Dixon, 2001, p. 287). As a result, the observed species richness, which is simply the count of unique species, is negatively biased in bootstrap replicates. This leads to an extreme mismatch between the original estimate and the distribution of bootstrap replicates. In such cases, the BCa intervals may fail altogether (e.g., due to infinite bias correction factors), and other bootstrap intervals (normal, basic) may overcorrect.

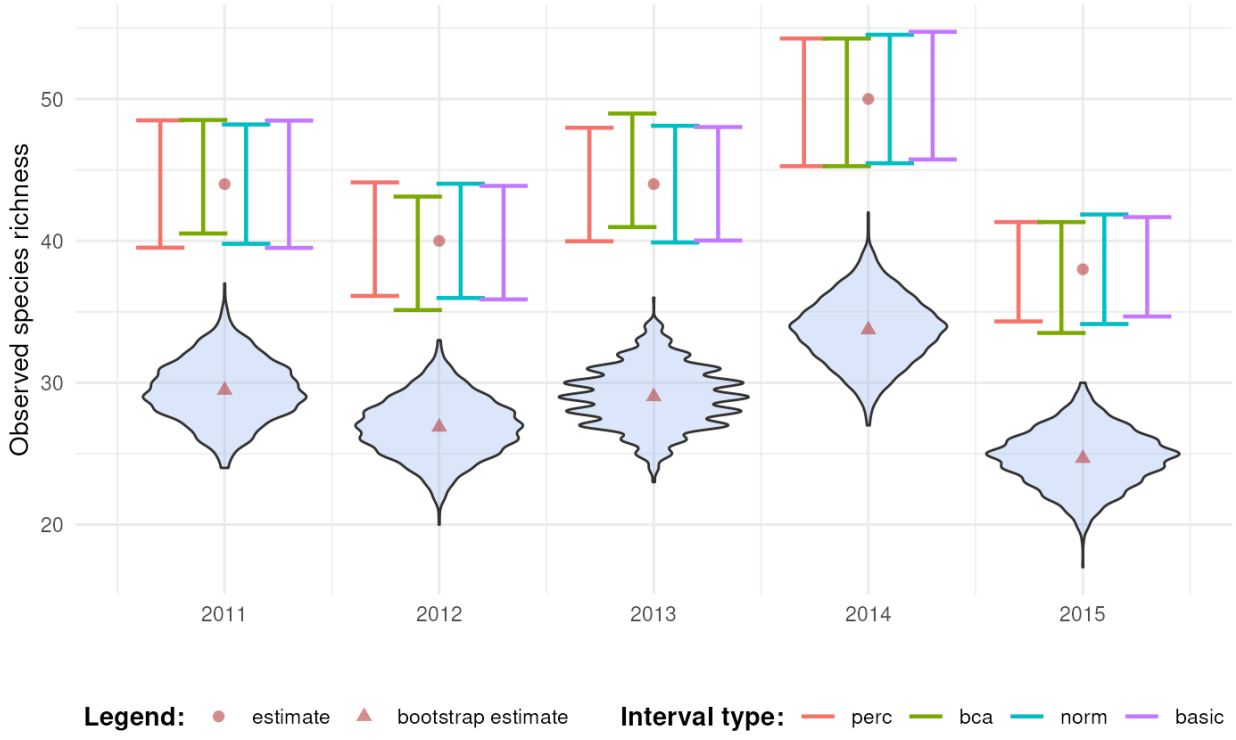
There is an option within `calculate_bootstrap_ci()` to centre the confidence limits around the original estimate (`no_bias = TRUE`). This means the bootstrap distribution is used to calculate confidence intervals, except for the bootstrap bias. While it may "solve" technical problems with interval calculation (like infinite or undefined corrections), it does so at the cost of ignoring bootstrap bias. This approach should only be used with caution and clear justification, such as when the bootstrap bias is known to be an artifact of a sampling limitation and not of the underlying data structure.

Because of this inherent limitation, alternative richness estimators that account for undetected species are preferred when uncertainty quantification is needed. The **vegan** (Oskanen et al., 2024) and **iNEXT** (Hsieh et al., 2016) R packages provide such estimators, including Chao, Jackknife, and coverage-based rarefaction/extrapolation, all of which are designed to handle unseen species and provide meaningful uncertainty estimates.

Some of these estimators are also implemented directly for occurrence cubes in recent versions of **b3gbi** (≥ v0.4.0), offering integration into existing cube-based workflows. However, it is important to note that these are alternative estimators. They are not equivalent to observed richness and will yield different values by design.

```
ci_richness_no_bias <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_results_richness,
  grouping_var = "year",
  type = c("perc", "bca", "norm", "basic"),
  no_bias = TRUE,
  data_cube = processed_cube_even,
  fun = calc_richness
)
```

Indeed, the intervals are now centred around the original estimate.



4.2.3. Deciding between confidence interval types

A critical step in the interpretation of bootstrap-based uncertainty estimates is the selection of an appropriate confidence interval type. Different interval types rely on different assumptions and may yield different results. A practical starting point is to do a comparative evaluation of multiple interval types by computing and visualising different confidence intervals (percentile, BCa, normal, and basic intervals) alongside the bootstrap distribution and the bootstrap bias (cf. examples above). This combined inspection helps to reveal whether the bootstrap distribution is symmetric or skewed, whether bias is present, and how sensitive interval estimates are to methodological assumptions.

Given the diversity of biodiversity indicators and data structures, we generally recommend the use of percentile or BCa intervals. These methods have the advantage of making relatively weak assumptions about the shape of the bootstrap distribution. The percentile interval is straightforward to compute and interpret, as it directly uses quantiles of the bootstrap distribution. The BCa interval extends this approach by explicitly correcting for both bias and skewness, making it theoretically more accurate in many practical situations. Despite its advantages, the BCa method comes with increased computational cost. The estimation of the acceleration parameter relies on a jackknife procedure, which can be time-consuming, especially for large data cubes or complex indicator functions. As a result, there is a trade-off between statistical reliability of the interval estimates and computational efficiency that needs to be considered in practice.

In contrast, normal and basic confidence intervals rely more strongly on assumptions about the underlying distribution, particularly the assumption of approximate normality. These methods are generally not recommended for biodiversity indicators, as this assumption is often violated. However, they may still be useful in specific contexts, for example when combined with appropriate transformations (e.g. log or logit transformations) or when results are truncated to respect known bounds. The validity of the normality assumption can be formally assessed using diagnostic tools such as Q–Q plots of the bootstrap replicates (Davison & Hinkley, 1997).

Overall, the selection of confidence interval types should be guided by a combination of empirical diagnostics, theoretical considerations, and practical constraints (Table 3). While no single method is universally optimal, the use of percentile or BCa intervals provides a robust choice for most applications. The comparison of multiple interval types, together with inspection of the bootstrap distribution and bias, offers a transparent and reproducible framework for making informed decisions about uncertainty quantification in biodiversity indicator analyses. It should be noted that the overview of Table 3 is not exhaustive, but is based on established literature and practical experience within the context of biodiversity data cubes.

Table 3: Overview of the advantages and disadvantages between the considered confidence interval (CI) types.

Interval type	Advantages	Disadvantages	References
Normal	- Simplicity - Understanding of bootstrap and CI theory	- Assumes bootstrap distribution is normal - Often transformation needed for more accurate results - Erratic coverage error in practise	(Davison & Hinkley, 1997, Chapter 5; Efron & Tibshirani, 1994, Chapter 13; Hesterberg, 2015)
Basic	- Simplicity - Understanding of bootstrap and CI theory	- Assumes symmetric bootstrap distribution - Typically substantial coverage error	(Carpenter & Bithell, 2000; Davison & Hinkley, 1997, Chapter 5; Hesterberg, 2015)
Percentile	- Simplicity - No assumptions about bootstrap distribution - Implicitly uses the existence of a good transformation	- Does not take bias into account - Substantial coverage error if the distribution is not nearly symmetric	(Carpenter & Bithell, 2000; Davison & Hinkley, 1997, Chapter 5; Efron & Tibshirani, 1994, Chapter 13)
BCa	- No assumptions about bootstrap distribution - Implicitly uses the existence of a good transformation - Adjusts for bias - Adjusts for skewness - Smaller coverage error than the other methods	- Involved calculation of acceleration parameter a - Unstable coverage when sample size is small or a is small ($a < 0.025$)	(Carpenter & Bithell, 2000; Davison & Hinkley, 1997, Chapter 5; Dixon, 2001; Efron & Tibshirani, 1994, Chapter 14)

4.3. Whole-cube bootstrap versus group-specific bootstrap

In **dubicube**, bootstrapping can be done in two ways:

- **Whole-cube bootstrapping**: resampling all rows in the cube, regardless of grouping.
- **Group-specific bootstrapping**: resampling rows only within a group of interest (e.g., a species, year, or habitat).

The choice between these two methods directly affects how confidence intervals should be interpreted:

- Some indicators **combine information across groups** (e.g., community richness, turnover, or multi-species metrics). These require whole-cube bootstrapping to preserve correlations.
- Other indicators are **calculated independently per group** (e.g., species-specific or year-specific metrics). For these, group-specific bootstrapping is usually more appropriate.

4.3.1. Whole-cube bootstrap

Definition: Resample all rows in the cube, regardless of species, year, or other grouping.

Advantages:

- Preserves correlations between groups (e.g., species co-occurrence, temporal dependencies).
- Appropriate for indicators that depend on multiple groups together (community-level metrics, multi-species diversity).

Disadvantages:

- Rare groups may end up with zero rows in some bootstrap replicates, leading to wider or undefined CIs.
- Variance for small groups may be inflated.

Use case examples:

- Community richness per site or habitat.
- Multi-species indicators (e.g., average occupancy across species).
- Temporal turnover indicators that rely on multiple years.

Implementation in `dubicube`

- Use `method = "whole_cube"` in the `bootstrap_cube()` function.

```
bootstrap_mean_whole <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  method = "whole_cube"
)
#> [1] "Performing whole-cube bootstrap."
```

4.3.2. Group-specific bootstrap

Definition: Subset the cube by the group of interest (e.g., species or year), then resample rows only within that group.

Advantages:

- Guarantees each replicate has rows for each group → stable CIs.
- Reflects within-group variability only.

Disadvantages:

- Ignores correlations with other groups.
- Variance may be slightly underestimated if the group's presence is correlated with other groups.

Use case examples:

- Species-specific occupancy or habitat preference metrics.
- Year-specific indicators (e.g., annual richness).
- Small or rare groups where zero-row replicates would be problematic.

Implementation in `dubicube`

- Use `method = "group_specific"` in the `bootstrap_cube()` function.

```
bootstrap_mean_group <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  method = "group_specific"
)
#> [1] "Performing group-specific bootstrap."
```

4.4. Integration with the boot package

dubicube can delegate the bootstrap resampling procedure to the well-established **boot** package. By default (`method = "smart"`), dubicube will automatically choose the bootstrapping method which is best for the given configuration. However the user can specify a specific method manually with `method = "boot_whole_cube"`, or `method = "boot_group_specific"`. In these cases, `bootstrap_cube()` returns the native objects produced by `boot::boot()`, rather than a summarised dataframe.

The differences in performance and the smart selection of the appropriate method are discussed below.

4.4.1. Whole-cube bootstrap with boot

When whole-cube bootstrapping is performed via **boot**, `bootstrap_cube()` returns a named list of `boot` objects, one per group.

```
bootstrap_mean_boot_whole <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  method = "boot_whole_cube"
)
#> [1] "Performing whole-cube bootstrap with `boot::boot()`."

sapply(bootstrap_mean_boot_whole, class)
#>   2011   2012   2013   2014   2015   2016   2017   2018   2019   2020
#> "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot"
```

4.4.2. Group-specific bootstrap with boot

For group-specific bootstrapping via **boot**, `bootstrap_cube()` returns a named list of `boot` objects, one per group.

```
bootstrap_mean_boot_group <- bootstrap_cube(
  data_cube = processed_cube_small,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123,
  method = "boot_group_specific"
)
#> [1] "Performing group-specific bootstrap with `boot::boot()`."

sapply(bootstrap_mean_boot_group, class)
#>   2011   2012   2013   2014   2015   2016   2017   2018   2019   2020
#> "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot" "boot"
```

4.4.3. Comparison between methods

We perform bootstrapping and confidence interval calculation for the period 2015–2018, following the same procedure as in previous examples. The results show that the bootstrap distributions and confidence intervals are very similar, regardless of whether the **boot** package is used or the **dubicube** internal code (Fig. 5). The label “boot” indicates that **dubicube** relies on the **boot** package for resampling and interval calculation, whereas “no boot” denotes that the same task is performed using **dubicube**'s internal code paths.

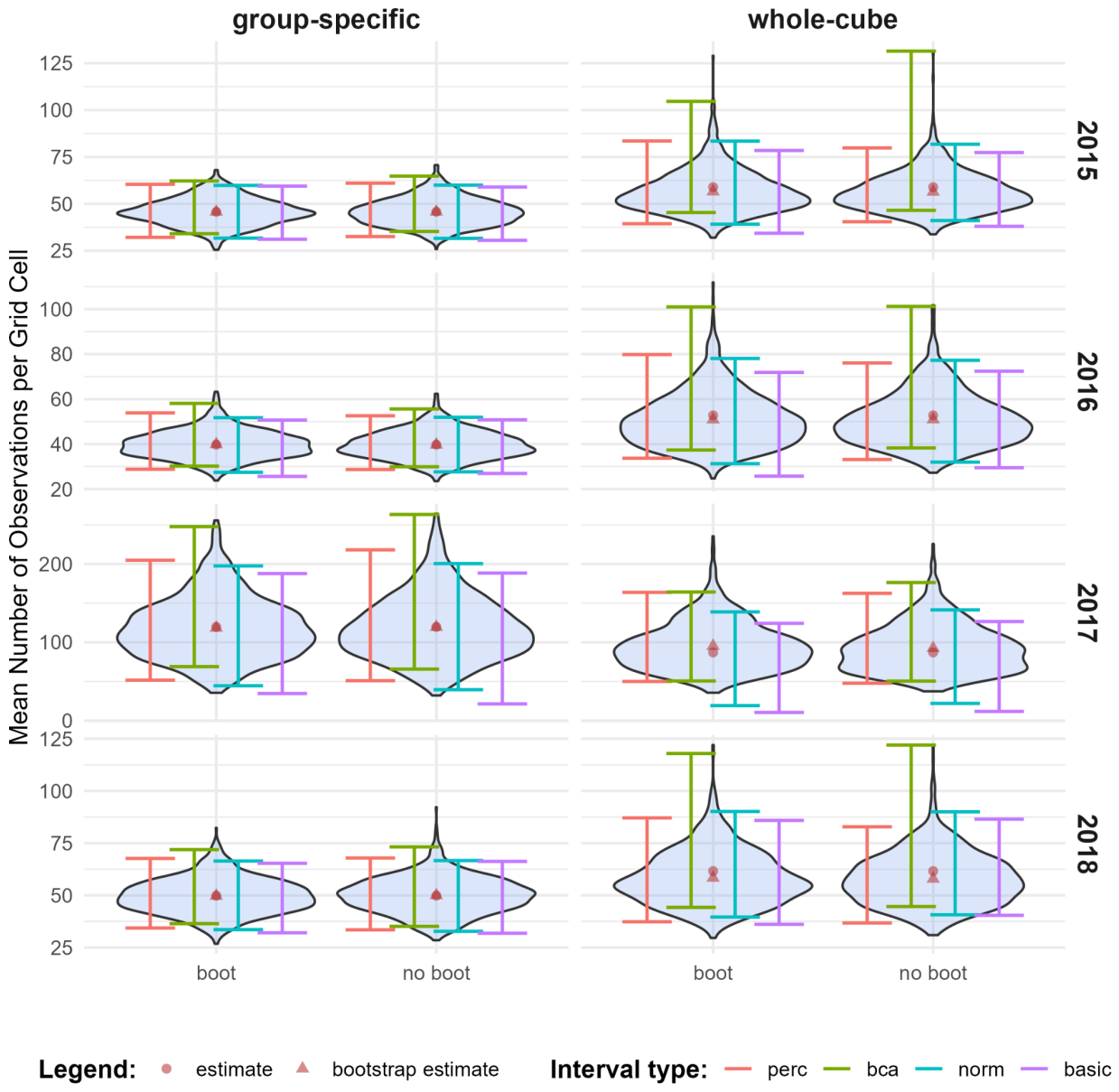


Figure 5: Comparison of bootstrap results between internal dubicube implementations “no boot” and delegation to the boot package “boot” for both group-specific and whole-cube bootstrap implementations. Based on the small dataset (see Chapter 2.3).

To illustrate the computational performance of the different bootstrapping strategies, we use the classic `iris` dataset. As a simple example indicator, we calculate the mean sepal length per species and quantify uncertainty using bootstrap-based confidence intervals.

```
mean_sepal_length <- function(x) {
  out_df <- aggregate(Sepal.Length ~ Species, x, mean)
  names(out_df) <- c("Species", "diversity_val")
  out_df
}

mean_sepal_length(iris)
#>      Species diversity_val
#> 1     setosa         5.006
#> 2 versicolor         5.936
#> 3  virginica         6.588
```

We benchmark execution time using the **microbenchmark** package (Mersmann, 2011). Two computational steps are evaluated separately:

1. Bootstrap resampling using `bootstrap_cube()` (1,000 bootstrap samples)
2. Confidence interval calculation using `calculate_bootstrap_ci()` (all interval types)

For each step, we compare:

- Whole-cube vs group-specific resampling strategies, and
- Internal **dubicube** implementations vs delegation to the **boot** package.

Each method is executed 20 times and timings are reported in milliseconds. All benchmarks are run with identical inputs and a fixed random seed to ensure comparability. The code is available in the source of the “Whole-cube Bootstrap versus Group-specific Bootstrap” tutorial (Table 1).

We observe that using the **boot** package results in faster performance for both bootstrapping and confidence interval calculation in the case of group-specific bootstrapping (Fig. 6). For whole-cube bootstrapping, while interval calculation is faster with **boot**, the bootstrapping step itself is slower, which leads to a higher total computation time overall.

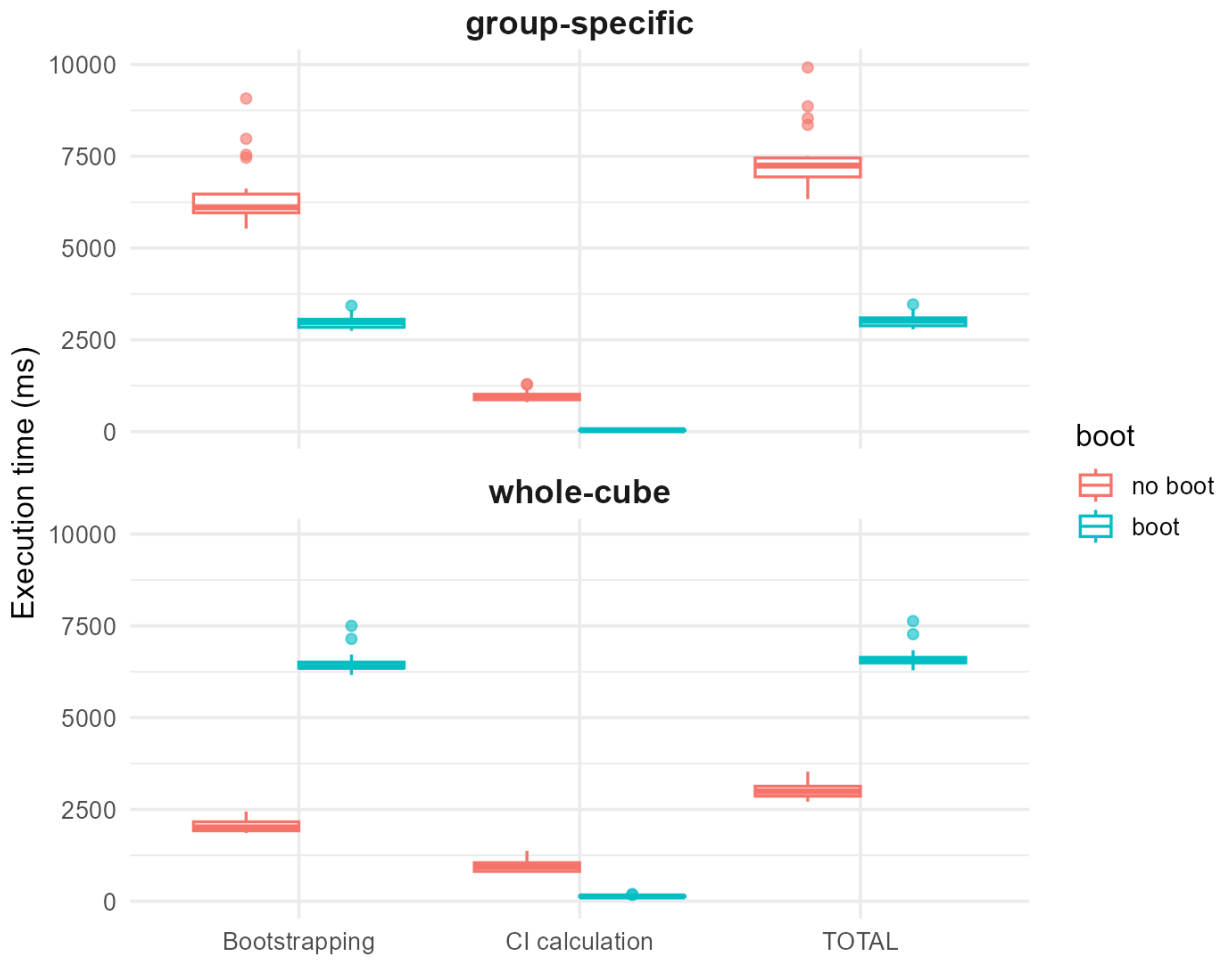


Figure 6: Comparison of execution time between internal dubicube implementations "no boot" and delegation to the boot package "boot" for both group-specific and whole-cube bootstrap implementations.

4.5. Smart bootstrap selection

In practice, users rarely need to set the `method` argument explicitly. The default `method = "smart"` reliably selects an appropriate and valid bootstrap strategy (Table 4). The package uses the `resolve_bootstrap_method()` helper function for this. Explicitly setting the bootstrap method is mainly useful for advanced workflows, debugging, or methodological comparisons. With the smart method option, **dubicube** automatically selects the most appropriate bootstrap strategy based on:

- **The scope of the indicator** Whether indicator values for each group are independent (group-specific) or depend on the full dataset (whole-cube). This is inferred by comparing indicator values computed on progressively smaller subsets of the data.

- **The presence of a reference group** If a reference group is specified (`ref_group` not `NA`), bootstrapping via the **boot** package is disabled, because reference-based indicators require explicit resampling logic.

- **The number of grouping variables** If more than one grouping variable is supplied, bootstrapping via the **boot** package is never used, even when `ref_group = NA`. In this case, `method = "smart"` always resolves to a non-boot method (`group_specific` or `whole_cube`), because multi-dimensional grouping is not compatible with the `boot` delegation used internally.

Table 4: Overview of internal bootstrap method selection rules.

Scope of indicator calculation	Reference group used?	Number of grouping variables	Method chosen by `method = "smart"`	Uses **boot** package?
Group-specific	No	1	`boot_group_specific`	yes
Whole cube	No	1	`boot_whole_cube`	yes
Group-specific	Yes	any	`group_specific`	no
Whole cube	Yes	any	`whole_cube`	no
Group-specific	No	> 1	`group_specific`	no
Whole cube	No	> 1	`whole_cube`	no

5. Indicator interpretation and visualisation

When interpreting indicators, too much emphasis is often placed on relatively small differences in indicator values. Such differences are to be expected due to natural variability and limited sample size. To avoid over-interpretation of changes in indicator values, it is important to include confidence limits around the calculated values in the final evaluation. This chapter presents a general approach to interpreting effects by providing a general framework for interpreting changes (increases or decreases) for the various indicators developed (Chapter 5.1), and clear visualisation of estimates with associated uncertainties for both temporal (Chapter 5.2) and spatial trends (Chapter 5.3)

5.1. Classifying effects using confidence limits

5.1.1. Concept of effect classification

Interpretation of the results can be done using effect classification for which we base on the **effectclass** package (Onkelinx, 2023). If the confidence interval is above/under the reference line, we call it an increase/decrease. Threshold values can be provided to make a distinction between stable (confidence interval covering the reference line) and uncertain trends (confidence interval covering the reference line and at least one of the threshold lines). The thresholds can also be used to classify even further (Table 5). Threshold values should be manually selected around the reference line at a level deemed negligible, allowing trends within this range to be classified as “no effect”.

dubicube’s `add_effect_classification()` function takes a dataframe with confidence limits and adds a classification column (see further). It uses the **effectclass** package internally to perform the classification logic.

Table 5: Rules for effect classification to aid the interpretation of indicator effects/trends (Onkelinx, 2023).

Symbol	Fine effect/trend	Coarse effect/trend	Rule
++	strong positive effect/strong increase	positive effect/ increase	confidence interval above the upper threshold
+	positive effect/ increase	positive effect/ increase	confidence interval above reference and contains the upper threshold
+~	moderate positive effect/ moderate increase	positive effect/ increase	confidence interval between reference and the upper threshold
~	no effect/stable	no effect/stable	confidence interval between thresholds and contains reference
−~	moderate negative effect/moderate decrease	negative effect/ decrease	confidence interval between reference and the lower threshold

-	negative effect/ decrease	negative effect/ decrease	confidence interval below reference and contains the lower threshold
--	strong negative effect/strong decrease	negative effect/ decrease	confidence interval below the lower threshold
?+	potential positive effect/potential increase	unknown effect/ unknown	confidence interval contains reference and the upper threshold
?-	potential negative effect/potential decrease	unknown effect/ unknown	confidence interval contains reference and the lower threshold
?	unknown effect/ unknown	unknown effect/ unknown	confidence interval contains the lower and upper threshold

5.1.2. Effect classification with dubicube

The input data is a dataframe which contains lower and upper confidence limits, e.g. `"lcl"` and `"ucl"`. Consider this synthetic dataset of 10 observations.

```
# Simulated means and standard deviations
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)
)

# Compute 90% confidence intervals
ds$lcl <- qnorm(0.05, ds$mean, ds$sd)
ds$ucl <- qnorm(0.95, ds$mean, ds$sd)

# View the dataset
ds
#>    mean   sd         lcl         ucl
#> 1   0.0 1.00 -1.64485363  1.64485363
#> 2   0.5 0.50 -0.32242681  1.32242681
#> 3  -0.5 0.50 -1.32242681  0.32242681
#> 4   1.0 0.50  0.17757319  1.82242681
#> 5  -1.0 0.50 -1.82242681 -0.17757319
#> 6   1.5 0.25  1.08878659  1.91121341
#> 7  -1.5 0.25 -1.91121341 -1.08878659
#> 8   0.5 0.25  0.08878659  0.91121341
#> 9  -0.5 0.25 -0.91121341 -0.08878659
#> 10  0.0 0.50 -0.82242681  0.82242681
```

We use the `add_effect_classification()` function for effect classification. It relies on the following arguments:

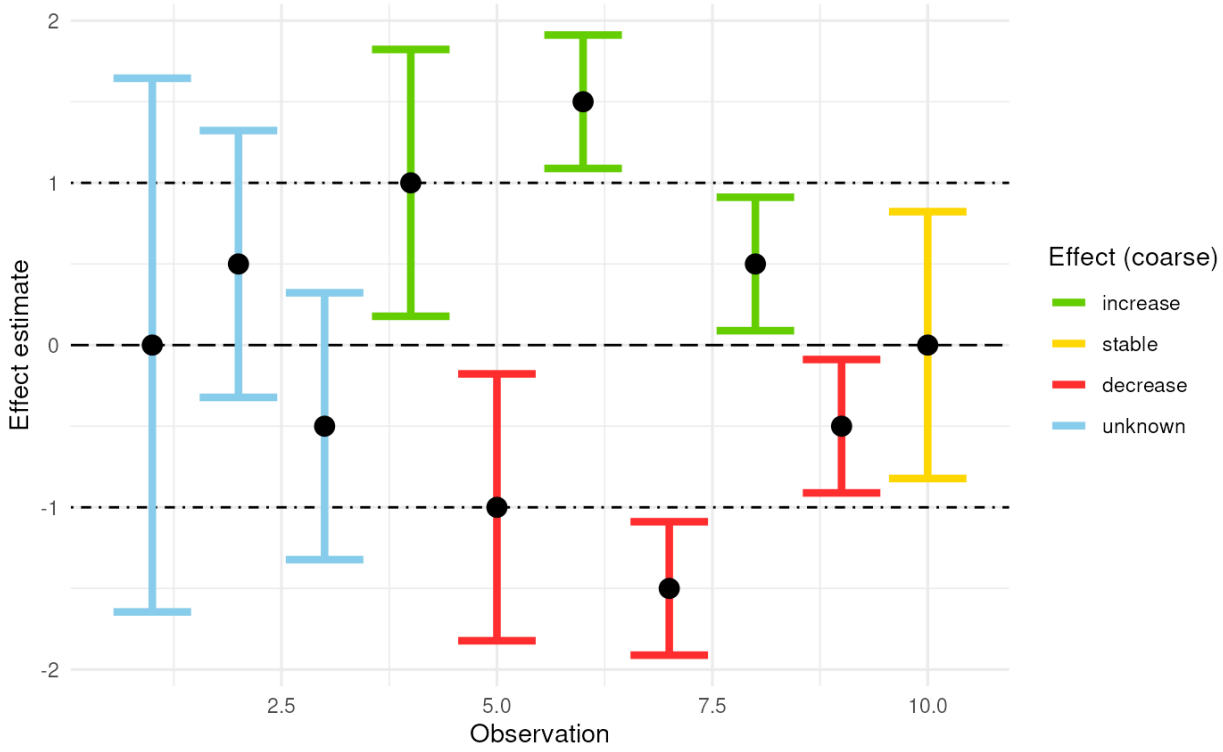
- **`df`**: The input data, a dataframe containing confidence limits, e.g. the result of bootstrap confidence interval calculation as introduced earlier.
- **`cl_columns`**: A character vector of length 2 specifying the column names in `df` that contain the **lower** and **upper** confidence limits. For example: `cl_columns = c("lcl", "ucl")`.
- **`threshold`**: A numeric vector defining the effect size threshold(s) used to distinguish between "no effect" and meaningful increases or decreases. This can be either one number (interpreted symmetrically around the reference) or two (explicit lower and upper thresholds).
- **`reference`**: A single numeric value representing the **null hypothesis** or **no-effect** level (e.g. 0). The position of the confidence interval relative to this value determines the direction of the effect.
- **`coarse`**: A logical flag indicating whether a simplified classification should also be added. If `TRUE` (default), additional columns `effect_code_coarse` and `effect_coarse` are included, summarizing effects into broader categories (increase, decrease, stable, unknown).

Let's classify the effects using a threshold of `1` (will be expanded to `[-1, 1]` around the reference) and a reference of `0`.

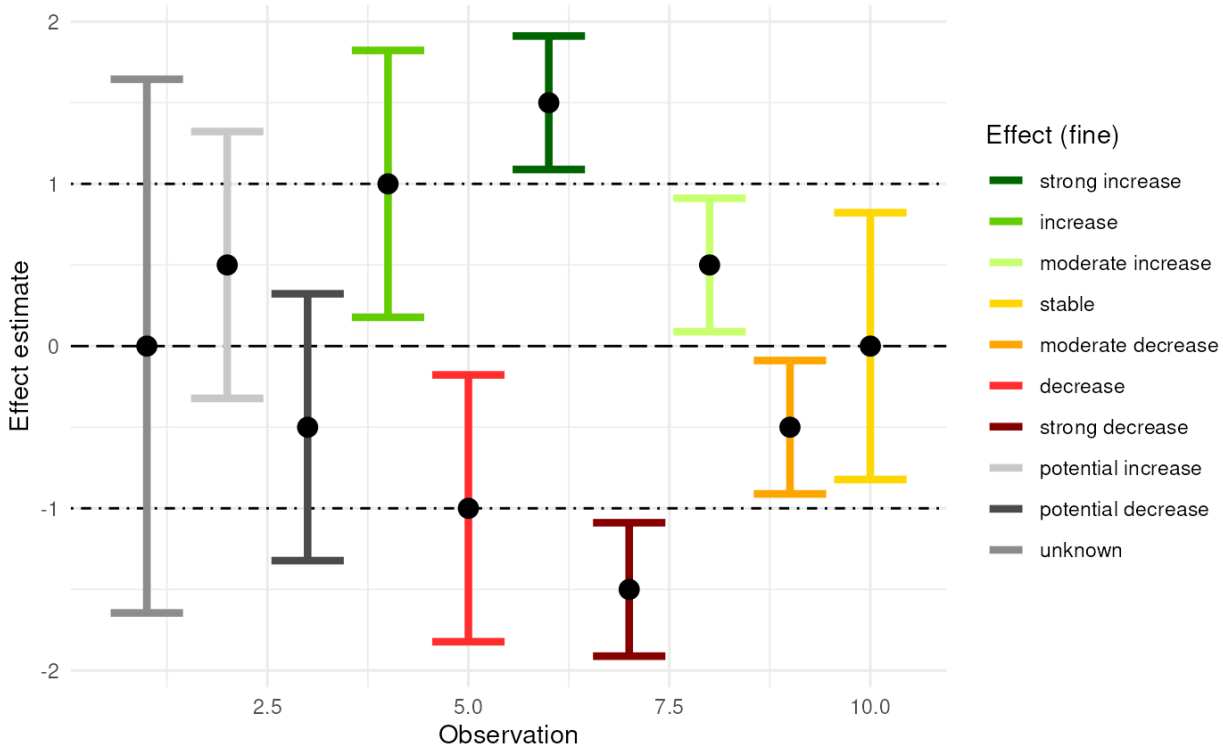
```
# Perform effect classification
result <- add_effect_classification(
  df = ds,
  cl_columns = c("lcl", "ucl"),
  threshold = 1,
  reference = 0,
  coarse = TRUE
)

# View the result
result
#>    mean   sd         lcl         ucl effect_code effect_code_coarse
#> 1   0.0 1.00 -1.64485363  1.64485363           ?                   ?
#> 2   0.5 0.50 -0.32242681  1.32242681          ?+                   ?
#> 3  -0.5 0.50 -1.32242681  0.32242681          ?-                   ?
#> 4   1.0 0.50  0.17757319  1.82242681           +                   +
```

```
#> 5  -1.0 0.50 -1.82242681 -0.17757319          -                   -
#> 6   1.5 0.25  1.08878659  1.91121341         ++                   +
#> 7  -1.5 0.25 -1.91121341 -1.08878659         --                   -
#> 8   0.5 0.25  0.08878659  0.91121341         +~                   +
#> 9  -0.5 0.25 -0.91121341 -0.08878659         -~                   -
#> 10  0.0 0.50 -0.82242681  0.82242681          ~                   ~
#>                effect effect_coarse
#> 1             unknown       unknown
#> 2  potential increase       unknown
#> 3  potential decrease       unknown
#> 4            increase      increase
#> 5            decrease      decrease
#> 6      strong increase      increase
#> 7      strong decrease      decrease
#> 8    moderate increase      increase
#> 9    moderate decrease      decrease
#> 10             stable        stable
```

We visualise this using **ggplot2**. The following plot shows point estimates and their 90% confidence intervals, coloured by the coarse effect classification.



The plot below provides a more detailed view using the fine-grained classification.



5.2. Visualising temporal trends

The visualisation of temporal trends is a key step in the interpretation of biodiversity indicators, particularly when communicating results to both scientific and policy audiences. It is essential that graphical representations explicitly incorporate uncertainty rather than presenting only point estimates. Several complementary approaches can be used to achieve this, each with its own strengths and limitations. For more detailed, applied examples, readers are referred to the online tutorial “Visualising Temporal Trends” listed in Table 1.

A straightforward and widely used method is the inclusion of error bars around indicator estimates. In this approach, point estimates (e.g. annual indicator values) are plotted together with their corresponding confidence intervals (Fig. 7A). Error bars provide a clear and transparent representation of uncertainty and allow readers to quickly assess the statistical significance and variability of changes over time. This method is particularly suitable for discrete time series, such as yearly indicators derived from occurrence cubes, and is generally easy to interpret even for non-specialist audiences.

To facilitate the identification of broader temporal patterns, it can be useful to complement discrete representations with smoothed trend lines. One commonly applied technique is LOESS (Locally Estimated Scatterplot Smoothing), which fits a series of local regressions to subsets of the data. This results in a flexible curve that captures underlying trends while reducing short-term fluctuations. When applied to both the indicator estimates and their confidence limits, LOESS smoothing provides a more continuous representation of temporal dynamics and can help highlight gradual increases, declines, or non-linear patterns (Fig. 7).

However, the use of smoothing techniques requires careful consideration. Biodiversity data cubes are inherently based on aggregated, categorical time units (e.g. years), and smoothing introduces an additional layer of modelling that may not fully reflect the original data structure. As a result, smoothed trends can differ from patterns that would emerge from analyses conducted on non-aggregated data. For this reason, smoothed visualisations should be interpreted as descriptive tools rather than inferential results. They are particularly useful for long time series, where the objective is to convey general patterns rather than year-to-year variability.

A more advanced approach to representing uncertainty over time is the use of fan plots (Onkelinx, 2023). Fan plots display multiple confidence intervals (e.g. 50%, 75%, 95%) as shaded bands around the central estimate, creating a “fan-like” appearance that illustrates how uncertainty expands or contracts across time. These plots can be constructed in either a discrete or continuous manner and provide a rich visual summary of uncertainty distributions (Fig. 7B). They are especially valuable when conveying the full range of variability derived from bootstrap analyses.

Despite their advantages, fan plots may be less intuitive for audiences unfamiliar with probabilistic visualisations. Their interpretation requires some understanding of confidence intervals and uncertainty gradients, which can limit their accessibility in policy contexts. Therefore, their use should be carefully considered and, where applied, accompanied by clear explanations.

5.3. Visualising spatial trends

The visualisation of spatial patterns in biodiversity indicators is essential for identifying geographic gradients, hotspots, and areas of concern. As with temporal analyses, it is crucial that spatial visualisations explicitly incorporate uncertainty, as indicator estimates derived from biodiversity data cubes are often subject to variability due to sampling effort, data availability, and ecological heterogeneity. Effective spatial visualisation should therefore communicate both the magnitude of the indicator and the associated uncertainty in a clear and interpretable manner.

A common starting point is the separate visualisation of point estimates and confidence limits. Spatial maps can be produced for the estimated indicator values (e.g. mean observations per grid cell), as well as for the lower and upper bounds of the confidence intervals. This approach allows users to explore the spatial distribution of both central estimates and uncertainty components independently. While this separation provides clarity, it requires the comparison of multiple maps and may limit the ability to quickly synthesise information.

To address this, it is often desirable to combine estimates and uncertainty into a single visual representation. This requires the definition of an appropriate uncertainty metric (Table 6). A straightforward measure is the width of the confidence interval, calculated as the difference between the upper and lower bounds, which reflects absolute uncertainty. However, because indicator values may vary substantially across space, relative measures of uncertainty are often more informative. For example, the relative confidence interval half-width expresses uncertainty as a proportion of the estimate, facilitating comparison between spatial units with different

magnitudes. Similarly, bootstrap-derived standard errors can be used in absolute or relative form to quantify variability.

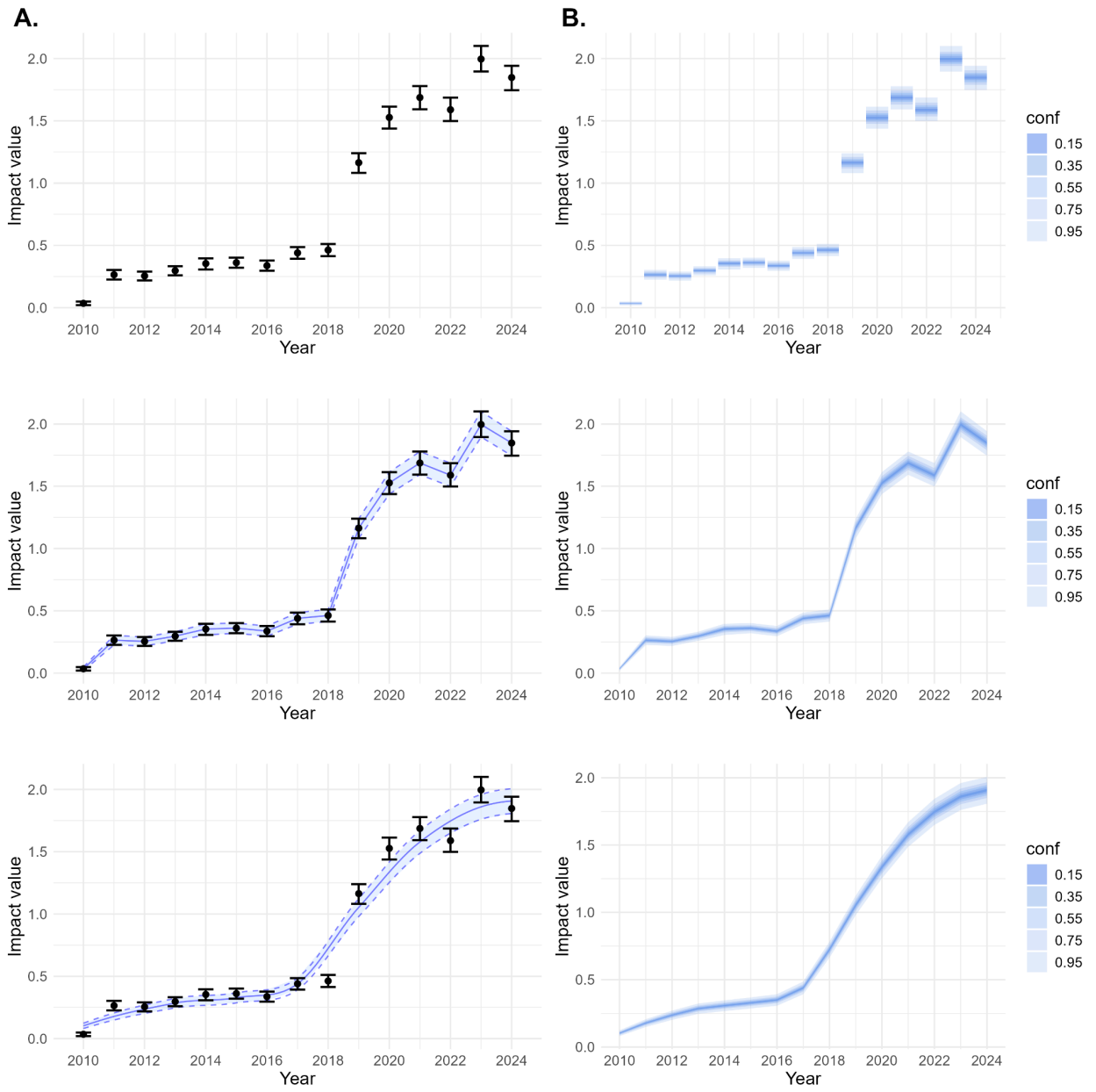


Figure 7: Visualisation of impact indicator values with uncertainty estimates (percentile method) based on the impIndicator package: error bars (A), and fan plots (B) with categorical (first row), linear (second row), and LOESS (third row) trends. The code for creating this figure is provided in Langeraert, Dove, et al. (2026).

Table 6: Uncertainty measures for spatial visualisation

Measure	Formula	Description
CI width	`ul - ll`	Absolute uncertainty
Relative CI width	`(ul - ll) / estimate`	Total CI width scaled by estimate
Relative CI half-width	`(ul - ll) / (2 × estimate)`	Margin of error relative to estimate
Bootstrap SE	`sd(bootstrap replicates)`	Standard deviation of bootstrap samples
Relative bootstrap SE	`sd(...) / estimate`	Standard error relative to estimate

Once an uncertainty measure has been defined, several visual encoding strategies can be applied to integrate it with spatial estimates. One effective approach is the use of transparency (alpha blending). In this representation, the colour of each spatial unit (e.g. grid cell or centroid point) reflects the indicator value, while the level of transparency represents uncertainty (Fig. 8C). Typically, higher uncertainty is associated with greater transparency, making uncertain areas visually less prominent. Empirical studies have shown that this approach performs well in terms of user accuracy and interpretation speed (Kinkeldey et al., 2014; MacEachren et al., 2005, 2012).

In addition to transparency, symbol size can be used as a complementary visual cue (Fig. 8D). By scaling the size of symbols according to uncertainty, areas with higher uncertainty can be emphasised or de-emphasised depending on the chosen design. Combining size and transparency can further enhance interpretability by providing redundant encoding of uncertainty, thereby reducing the likelihood of misinterpretation. However, care must be taken to avoid overly complex visualisations that may overwhelm the reader.

An alternative approach is the use of blurriness to represent uncertainty. In this context, more uncertain spatial estimates are displayed with a blurred or diffuse appearance, whereas more certain estimates appear sharp (Fig. 8A-B). This method aligns with intuitive visual metaphors of clarity and uncertainty and can therefore be effective for communication (Kinkeldey et al., 2014; MacEachren et al., 2005, 2012). However, it is not natively supported in most standard R visualisation tools and typically requires custom implementations. Existing approaches often simulate blur by layering semi-transparent symbols, which may limit flexibility and control over the visual effect.

Each of these visualisation techniques involves trade-offs between interpretability, accuracy, and complexity. Transparency and size are relatively easy to implement and interpret, making them suitable for most applications. Blurriness offers intuitive appeal but currently requires more advanced or custom implementations. Regardless of the chosen method, it is important to clearly document how uncertainty is quantified and visualised, and to provide appropriate legends and explanations.

For more detailed, applied examples based on real data, including a reproducible workflow, readers are referred to the online tutorial “Visualising Spatial Trends” listed in Table 1.

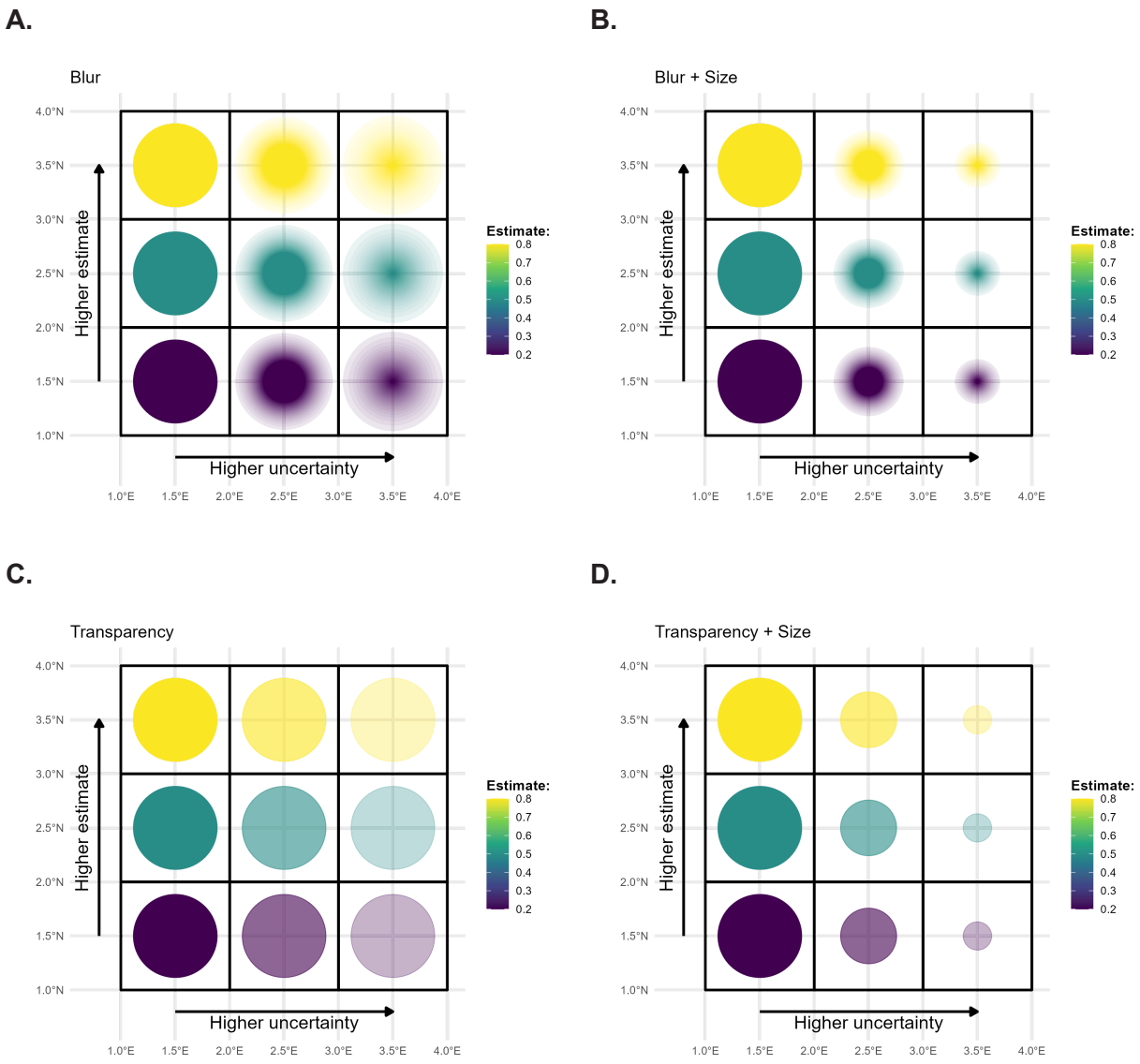


Figure 8: Visualisation of indicator estimate and uncertainty within a spatial grid. A: Blur. B: Blur and size. C: Transparency. D: Transparency and size. The code for creating this figure is provided in Langeraert, Dove, et al. (2026).

6. Software design and future development of dubicube

While the **dubicube** package already provides a comprehensive framework for assessing data variability, uncertainty, and interpretation of biodiversity indicators, several aspects of its software design can be further improved to enhance performance, usability, and interoperability. This chapter discusses key considerations for future development, focusing on computational scalability, the adoption of a more structured object-oriented design, and the further extension of the data quality diagnostics framework. These developments are particularly important in the context of increasing data volumes and the integration of **dubicube** as a dependency in other analytical packages. The planned release of version 1.0.0 is therefore contingent on the implementation of these improvements, most notably the introduction of S3-based object-oriented structures, which will provide a stable and extensible foundation for future work.

6.1. Computational performance and scalability

The application of resampling-based methods such as cross-validation and bootstrapping to biodiversity data cubes introduces important computational challenges. These methods require repeated recalculation of indicators across potentially large and high-dimensional datasets, which can lead to substantial demands in terms of both memory usage and computation time. As biodiversity monitoring increasingly relies on large-scale datasets, addressing these performance constraints becomes essential for the practical applicability of the **dubicube** package.

One of the main challenges identified during development is the high memory demand of cross-validation procedures. In particular, `cross_validate_cube()` can become difficult to run on standard hardware when applied to large datasets. Empirical observations show that reducing dataset size can significantly improve performance. Strategies are required to enable analyses to be performed on large datasets without exceeding available system resources.

A potential approach is the implementation of chunk-based processing, in which computations are divided into smaller subsets that are processed sequentially. By introducing a `chunk_size` argument, both cross-validation and bootstrapping procedures could operate on manageable portions of the data, thereby reducing peak memory usage. This approach also opens the possibility of writing intermediate results to disk and combining them at a later stage. Such functionality can be designed in a way that preserves the existing user interface, allowing users to scale analyses without fundamentally changing their workflows.

In addition to memory constraints, computation time is an important factor. Functions such as `cross_validate_cube()`, `bootstrap_cube()`, and `calculate_bootstrap_ci()` can be computationally intensive, particularly when a large number of bootstrap replicates is required or when complex interval estimation methods are used. Parallel computing offers a potential solution to reduce computation time. In the R ecosystem, this can be achieved using the **future** framework (Bengtsson, 2021), which allows users to define parallel execution strategies through `future::plan()`. Experimental results indicate that parallelisation can substantially reduce runtime for certain components, such as the jackknife procedure, with near-linear improvements observed when increasing the number of workers. However, parallelisation is not universally beneficial. For functions operating on large dataframes, such as

cross-validation and bootstrapping, the overhead associated with copying data between processes seems to outweigh the performance gains.

An important development in this context is the integration of the **boot** package (Chapter 4.4), which allows **dubicube** to delegate bootstrap resampling to a well-established and optimised implementation. This results in improved performance in several scenarios, particularly for group-specific resampling where both resampling and confidence interval calculation are faster. For whole-cube resampling, performance gains are more nuanced: while interval calculation remains faster, the resampling step itself can be slower, potentially increasing overall computation time. Importantly, the resulting bootstrap distributions and confidence intervals are highly comparable between the internal **dubicube** implementation and the delegated **boot** approach, ensuring consistency of analytical results. To facilitate efficient use without requiring detailed methodological choices, **dubicube** implements a smart bootstrap selection mechanism (Chapter 4.5) that automatically selects the most appropriate strategy based on the indicator structure, the presence of a reference group, and the number of grouping variables. Where possible, computation is delegated to **boot** to leverage performance advantages, while more complex scenarios rely on internal implementations to retain flexibility, ensuring both efficiency and robustness across a wide range of applications.

Overall, performance considerations remain a central aspect of ongoing development.

6.2. Object-oriented design and S3 implementation

The current implementation of the **dubicube** package relies primarily on returning standard dataframes as outputs for its core analytical functions. While this approach ensures transparency and compatibility with existing R workflows, it presents limitations in terms of structure, usability, and extensibility. These limitations become particularly apparent in more complex workflows involving cross-validation, bootstrapping, and confidence interval estimation. To address these issues, the adoption of an object-oriented programming (OOP) approach, based on S3 classes in R, is identified as an important direction for further development. This has already been successfully implemented for the data quality diagnostics functionality (Chapter 3.1).

At present, outputs from functions such as `cross_validate_cube()`, `bootstrap_cube()`, and `calculate_bootstrap_ci()` are not formally structured beyond their tabular representation. As a result, users must manually interpret the contents of these dataframes, and there is no standardised mechanism for summarising or visualising results. In addition, the absence of formal object classes prevents the use of method dispatch, which would allow for tailored `print()`, `summary()`, and `plot()` methods. Introducing dedicated S3 classes for different result types, such as `cube_cv`, `cube_bootstrap`, `bootstrap_ci`, and `effect_classification`, would provide a consistent and self-describing structure in which both results and associated metadata are stored together.

Such an object-oriented design would also allow for more efficient and user-friendly workflows. Currently, several functions require a large number of arguments that need to be specified repeatedly, including grouping variables, the statistic of interest, and additional options. By embedding this information within structured objects, subsequent functions could directly access

the required metadata, thereby reducing the number of explicit arguments. This would simplify function interfaces, reduce the risk of user error, and improve reproducibility. For example, confidence interval calculations could be performed directly on a `cube_bootstrap` object without the need to repeat any arguments that were already specified during bootstrapping (e.g. for BCa interval calculation).

An additional advantage of this approach is improved consistency in data handling. In the current implementation, certain technical issues may arise when working with list-based outputs, such as those returned by bootstrap procedures. For instance, grouping variables stored as list names may be implicitly converted to character values, even when they represent numeric variables such as years. By explicitly storing grouping variables and their original types within structured objects, these inconsistencies can be avoided, ensuring that downstream analyses remain correct and interpretable.

Importantly, the introduction of S3 classes is also expected to facilitate the integration of **dubicube** as a dependency in other R packages, as discussed in Chapter 2.2. A well-defined object structure with clear metadata and standardised methods makes it easier for external packages to interface with **dubicube** functionality. This is particularly relevant for ongoing efforts to incorporate bootstrap-based uncertainty estimation into packages such as **b3gbi**, **pdindicatoR**, and **implndicator**. In such contexts, structured outputs can serve as stable interfaces between packages, reducing the need for *ad hoc* data manipulation and improving interoperability. As these integrations are further developed, a consistent object-oriented design will be essential to ensure that **dubicube** can function as a reliable backend for uncertainty estimation across multiple biodiversity indicator workflows.

To support this transition, a modular file structure separating user-facing functions, object definitions, and method implementations is recommended. In addition, appropriate unit tests will be required to verify correct class behaviour and method dispatch, while maintaining compatibility with existing workflows through coercion methods such as conversion to dataframes.

6.3. Extension of diagnostic rules

While the current implementation of the data quality diagnostics framework provides a structured and extensible rule-based system (Chapter 3.1), its present implementation in **dubicube** remains limited to a core set of basic checks. Although these diagnostics provide an important first-level assessment of dataset suitability, they do not yet fully capture more complex structural properties of biodiversity data cubes. Further development of the diagnostic framework will focus on the implementation of additional rule categories, in particular observation distribution rules and interaction rules.

In addition to expanding the set of predefined rules, an important planned development is the introduction of a rule constructor function that allows users to define their own diagnostic rules. Such functionality would enable users to specify custom metric calculation functions, filter functions, threshold values, severity classifications, and diagnostic messages, fully aligned with the existing rule-based framework. This will make it possible to tailor diagnostics to specific datasets, indicator types, or policy contexts, while preserving the transparency and reproducibility of the workflow.

Future work may, depending on the extent of future use and contributions, address the refinement and empirical calibration of threshold values as new insights and applications emerge in practice. Expanding the default rule sets, improving their configurability, and enabling user-defined extensions would further strengthen the diagnostic framework as a flexible and scalable component of **dubicube**, ensuring that increasingly complex biodiversity datasets can be evaluated in a robust and transparent manner.

7. Acknowledgements

The authors would like to thank Sandra MacFadyen for the careful review of this deliverable report and for the constructive feedback provided. We also gratefully acknowledge Shawn Dove, Lissa Breugelmans, and Mukhtar Muhammed Yahaya for the valuable discussions and close collaboration that supported the implementation of bootstrapping approaches within other packages and the occurrence cube framework in general. Their input and cooperation have been instrumental in ensuring the robustness and practical applicability of the methods presented in this work. Additionally, we appreciate the input of Emma Cartuyvels, Toon Westra, Floris Vanderhaeghe, and Shawn Dove regarding the visualisation of spatial uncertainty. Finally, we would like to thank Peter Desmet for coming up with the R package name for **dubicube**.

8. References

Bengtsson, H. (2021). A Unifying Framework for Parallel and Distributed Processing in R using Futures. *The R Journal*, *13*(2), 208. https://doi.org/10.32614/RJ-2021-048

Breugelmans, L., Trekels, M., & Hendrickx, L. (2025). *pdindicatoR: Calculate and visualize phylogenetic diversity indicators based on species occurrence data cubes* [Computer software]. https://github.com/b-cubed-eu/pdindicatoR

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

Carpenter, J., & Bithell, J. (2000). Bootstrap confidence intervals: When, which, what? A practical guide for medical statisticians. *Statistics in Medicine*, *19*(9), 1141–1164. https://doi.org/10.1002/(SICI)1097-0258(20000515)19:9%3C1141::AID-SIM479%3E3.0.CO;2-F

Chamberlain, S., Barve, V., McGlinn, D., Oldoni, D., Desmet, P., Geffert, L., & Ram, K. (2025). *rgbif: Interface to the Global Biodiversity Information Facility API* [Computer software]. https://CRAN.R-project.org/package=rgbif

Csárdi, G., Hester, J., Wickham, H., Chang, W., Morgan, M., & Tenenbaum, D. (2024). *remotes: R Package Installation from Remote Repositories, Including 'GitHub'* [Computer software]. https://CRAN.R-project.org/package=remotes

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

Desmet, P., Oldoni, D., Huybrechts, P., & Govaert, S. (2025). *frictionless: Read and Write Frictionless Data Packages* [Computer software]. https://doi.org/10.32614/CRAN.package.frictionless

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

Dixon, P. M. (2001). The Bootstrap and the Jackknife: Describing the Precision of Ecological Indices. In S. M. Scheiner & J. Gurevitch (Eds), *Design and Analysis of Ecological Experiments* (Second Edition, pp. 267–288). Oxford University PressNew York, NY. https://doi.org/10.1093/oso/9780195131871.003.0014

Dove, S. (2026). *b3gbi: General Biodiversity Indicators for Biodiversity Data Cubes* [Computer software]. https://github.com/b-cubed-eu/b3gbi

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

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

Fischhoff, B., & Davis, A. L. (2014). Communicating scientific uncertainty. *Proceedings of the National Academy of Sciences*, *111*(supplement_4), 13664–13671. https://doi.org/10.1073/pnas.1317504111

Frangos, C. C., & Schucany, W. R. (1990). Jackknife estimation of the bootstrap acceleration constant. *Computational Statistics & Data Analysis*, *9*(3), 271–281. https://doi.org/10.1016/0167-9473(90)90109-U

Hesterberg, T. C. (2015). What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum. *The American Statistician*, *69*(4), 371–386. https://doi.org/10.1080/00031305.2015.1089789

Hsieh, T. C., Ma, K. H., & Chao, A. (2016). iNEXT: An R package for rarefaction and extrapolation of species diversity Hill numbers). *Methods in Ecology and Evolution*, *7*(12), 1451–1456. https://doi.org/10.1111/2041-210X.12613

Kinkeldey, C., MacEachren, A. M., & Schiewe, J. (2014). How to Assess Visual Communication of Uncertainty? A Systematic Review of Geospatial Uncertainty Visualisation User Studies. *The Cartographic Journal*, *51*(4), 372–386. https://doi.org/10.1179/1743277414Y.0000000099

Langeraert, W., Breugelmans, L., Desmet, P., Dove, S., Kumschick, S., MacFadyen, S., Oldoni, D., Seebens, H., Trekels, M., Yahaya, M. M., & Van Daele, T. (2026). *R Package and Documentation (Vignettes) for the Calculation of the Indicators* (Deliverables B-Cubed D5.5). https://b-cubed.eu/library

Langeraert, W., Dove, S., & Van Daele, T. (2026). *Investigate indicator uncertainty* (Version 2.0.0) [Computer software]. https://doi.org/10.5281/zenodo.19819479

Langeraert, W., Faulkner, K., Cartuyvels, E., Groom, Q., & Van Daele, T. (2026). *Report on the criteria for data quality and species characteristics for estimating species status and trends* (Deliverables B-Cubed D4.3). https://b-cubed.eu/library

Langeraert, W., & Van Daele, T. (2025a). *b3data: Data resources for the b3verse* [Data set]. Research Institute for Nature and Forest (INBO). https://doi.org/10.5281/ZENODO.15181097

Langeraert, W., & Van Daele, T. (2025b). *Design of data and indicator robustness measures* (Milestones B-Cubed No. MS26). https://b-cubed.eu/library

Langeraert, W., & Van Daele, T. (2026). *dubicube: Calculation and Interpretation of Data Cube Indicator Uncertainty* [Computer software]. https://github.com/b-cubed-eu/dubicube

MacEachren, A. M., Robinson, A., Hopper, S., Gardner, S., Murray, R., Gahegan, M., & Hetzler, E. (2005). Visualizing Geospatial Information Uncertainty: What We Know and What We Need to Know. *Cartography and Geographic Information Science*, *32*(3), 139–160. https://doi.org/10.1559/1523040054738936

MacEachren, A. M., Roth, R. E., O'Brien, J., Li, B., Swingley, D., & Gahegan, M. (2012). Visual Semiotics & Uncertainty Visualization: An Empirical Study. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2496–2505. https://doi.org/10.1109/TVCG.2012.279

Mersmann, O. (2011). *microbenchmark: Accurate Timing Functions* (p. 1.5.0) [Data set]. https://doi.org/10.32614/CRAN.package.microbenchmark

Milner-Gulland, E. J., & Shea, K. (2017). Embracing uncertainty in applied ecology. *The Journal of Applied Ecology*, *54*(6), 2063–2068. https://doi.org/10.1111/1365-2664.12887

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

Oskanen, J., Simpson, G., Blanchet, F., Kindt, R., Legendre, P., Minchin, P., O'Hara, R., Solymos, P., Stevens, M., Szoecs, E., Wagner, H., Barbour, M., Bedward, M., Bolker, B., Borcard, D., Carvalho, G., Chirico, M., De Caceres, M., Durand, S., … Weedon, J. (2024). *vegan: Community Ecology Package* [Computer software]. https://CRAN.R-project.org/package=vegan

Rowland, J. A., Bland, L. M., James, S., & Nicholson, E. (2021). A guide to representing variability and uncertainty in biodiversity indicators. *Conservation Biology*, *35*(5), 1669–1682. https://doi.org/10.1111/cobi.13699

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *dplyr: A Grammar of Data Manipulation* [Computer software]. https://CRAN.R-project.org/package=dplyr

Wickham, H., Vaughan, D., & Girlich, M. (2014). *tidyr: Tidy Messy Data* (p. 1.3.2) [Data set]. https://doi.org/10.32614/CRAN.package.tidyr

Yahaya, M. M., Kumschick, S., MacFadyen, S., Landi, P., & Hui, C. (2026). *impIndicator: Impact Indicators of Alien Taxa* [Computer software]. https://github.com/b-cubed-eu/impIndicator

9. Annex

9.1. Data quality diagnostic rules

9.1.1. Basic rules

These metrics evaluate overall completeness and minimal coverage along a single dimension. See `basic_cube_rules()` or use `rules = "basic"` in `diagnose_cube()`.

9.1.1.1. Temporal

Minimal number of time points (`rule_temporal_min_years()`)
The number of time points (years) is largely determined during cube specification. While not always critical for indicator calculation, very low numbers may indicate issues (e.g. incorrect filtering or aggregation).

category	
≥ 5 time points	🟢
3-4 time points	🟡
<3 time points	🟠
	🔴

Missing time points (`rule_temporal_missing_years()`)
Number of missing years within the full time range of the cube (from first to last observed year).

category	
0 missing years	🟢
1-2 missing years	🟡
≥ 3 missing years	🟠
	🔴

9.1.1.2. Spatial

Spatial range (`rule_spatial_min_cells()`)
Number of grid cells with observations, representing spatial coverage of the cube.

category	
≥ 5 grid cells	🟢
3-4 grid cells	🟡
< 3 grid cells	🟠
	🔴

Coordinate uncertainty vs grid resolution (`rule_spatial_max_uncertainty()`)
Number of records where coordinate uncertainty exceeds the spatial resolution of the grid.

category	
0 records	🟢
1-2 records	🟡
3-4 records	🟠
≥ 5 records	🔴

Overall spatial clustering (*planned*)

While ecological data are expected to have a considerable degree of structure, extreme values for clustering or dispersion may indicate underlying problems with the dataset. The degree of spatial clustering of the data in the data cube will be investigated using spatial autocorrelation analysis. High spatial autocorrelation means that neighbouring cells tend to have similar values, whereas low spatial autocorrelation means that values are very different when moving from one cell to another.

category	
-0.5 < x < 0.8	🟢
x < -0.5 or x > 0.8	🟡
x < -0.7 or x > 0.9	🟠
x < -0.8 or x > 0.95	🔴

Moran's I will be used to summarise spatial autocorrelation at varying lag distances for a geographical area. It indicates whether the data are dispersed, random or clustered with values between -1 and 1.

Local indicators for spatial association (*planned*)
To gain more insight into where local spatial patterns occur, Local Indicators of Spatial Association (LISA) will be used. They can be used to identify areas with statistically significant spatial patterns, unusual high or low values, detect spatial outliers with values significantly different from their surroundings and local variations.

Several indicators are possible, but at least the most common Local Moran's I and Geary ratio will be implemented.

9.1.1.3. Taxonomic

Minimal number of taxa (`rule_taxon_min_taxa()`)
Number of unique taxa represented in the cube.

category	
≥ 5 taxa	🟢
3-4 taxa	🟡
< 3 taxa	🟠
	🔴

9.1.1.4. Observation

Minimal number of records (`rule_obs_min_records()`)
Number of observation records (rows) in the cube.

category	
≥ 40 records	🟢
30-39 records	🟡
20-29 records	🟠
< 20 records	🔴

Minimal total number of observations (`rule_obs_min_total()`)
Total number of observations (e.g. counts summed across all records).

category	
≥ 40 records	🟢
30-39 records	🟡
20–29 records	🟠
< 20 records	🔴

9.1.2. Observation distribution rules

These metrics detect imbalances in observations along a single dimension. They are not yet implemented in **dubicube**.

9.1.2.1. Temporal

Minimal number of observations per time point
Flag when not enough observations per time point.

category	
> 30 observations	🟢
20-30 observations	🟡
10-20 observations	🟠
< 10 observations	🔴

Relative difference of number of observations
We can calculate the relative difference between the number of observations per time point and the median number of observations over all time points. If the relative difference is too large for one or more time points, they can be flagged.

category	
< 10 %	🟢
10-50 %	🟡
50-100 %	🟠
> 100 %	🔴

9.1.2.2. Spatial

Minimal number of observations per grid cell
Flag when not enough observations per grid cell.

category	
> 30 observations	🟢
20-30 observations	🟡
10-20 observations	🟠
< 10 observations	🔴

Relative difference of number of observations
We can calculate the relative difference between the number of observations per grid cell and the median number of observations over all grid cells. If the relative difference is too large for one or more taxa, they can be flagged.

category	
< 10 %	🟢
10-50 %	🟡
50-100 %	🟠
> 100 %	🔴

9.1.2.3. Taxonomic

Minimal number of observations per taxon
Flag when not enough observations per taxon.

category	
> 30 observations	🟢
20-30 observations	🟡
10-20 observations	🟠
< 10 observations	🔴

Relative difference of number of observations
We can calculate the relative difference between the number of observations per taxon and the median number of observations over all taxa. If the relative difference is too large for one or more taxa, they can be flagged.

category	
< 10 %	🟢
10-50 %	🟡
50-100 %	🟠
> 100 %	🔴

9.1.3. Interaction rules

These metrics detect coverage gaps across multiple dimensions. They are not yet implemented in **dubicube**.

Spatial similarity for consecutive time steps (temporal + spatial)
The spatial distribution of occurrences naturally fluctuates, but abrupt changes may indicate underlying issues in the dataset. To assess spatial changes over time, we calculate the similarity of presence/absence maps (where presence is defined as occurrences > 0, and absence as no occurrences) between successive time points. This is measured using the Jaccard similarity index:

Jaccard similarity = (number of cells with occurrences in both time points) / (number of cells with occurrences in both sets)

The index ranges from 0 (no similarity) to 1 (complete similarity). It can be computed for the entire data cube or for individual species. Results can be visualized as a time series with (time period - 1) steps or summarized as the mean similarity over the full time period.

category	
> 0.8	🟢
0.5 > x < 0.8	🟡
> 0.2 x < 0.5	🟠
< 0.2	🔴

Minimal number of taxa per time point (temporal + taxonomic)
The user specifies a species group from which to make an occurrence cube. It is possible to get a cube with less taxa (e.g. species) than expected because data for only a few taxa in this group are available on GBIF. The number of taxa may or may not be important for indicator calculation depending on the indicator function.

category	
> 5 taxa	🟢
3-5 taxa	🟡
2-3 taxa	🟠
1 taxon	🔴

Minimal number of taxa per grid cell (spatial + taxonomic)
The user specifies a species group from which to make an occurrence cube. It is possible to get a cube with less taxa (e.g. species) than expected because data for only a few taxa in this group are available on GBIF. The number of taxa may or may not be important for indicator calculation depending on the indicator function.

category	
> 5 taxa	🟢
3-5 taxa	🟡
2-3 taxa	🟠
1 taxon	🔴