

# Package ‘inspector’

November 16, 2020

**Type** Package

**Title** Validation of Arguments and Objects in User-Defined Functions

**Version** 1.0.2

**Description** Utility functions that implement and automate common sets of validation tasks.  
These functions are particularly useful to validate inputs, intermediate objects and output values in user-defined functions, resulting in tidier and less verbose functions.

**License** MIT + file LICENSE

**URL** <https://ptfonseca.github.io/inspector/>,  
<https://github.com/ptfonseca/inspector>

**BugReports** <https://github.com/ptfonseca/inspector/issues>

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 2.10)

**Imports** Rdpack (>= 0.7)

**Suggests** covr (>= 3.5.0), knitr (>= 1.28), rmarkdown (>= 2.1),  
testthat (>= 2.3.2), pcal (>= 1.0.0)

**RdMacros** Rdpack

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Pedro Fonseca [aut, cre] (<<https://orcid.org/0000-0002-5708-081X>>),  
FCT [fnd]

**Maintainer** Pedro Fonseca <[pedro.teles.fonseca@phd.iseg.ulisboa.pt](mailto:pedro.teles.fonseca@phd.iseg.ulisboa.pt)>

**Repository** CRAN

**Date/Publication** 2020-11-16 08:10:06 UTC

## R topics documented:

inspect_bfactor . . . . .	2
inspect_bfactor_log . . . . .	4
inspect_bfactor_scale . . . . .	6
inspect_categories . . . . .	7
inspect_character . . . . .	9
inspect_character_match . . . . .	10
inspect_data_categorical . . . . .	12
inspect_data_cat_as_dichotomous . . . . .	14
inspect_data_dichotomous . . . . .	16
inspect_log_base . . . . .	18
inspect_par_bernoulli . . . . .	19
inspect_par_beta . . . . .	21
inspect_par_dirichlet . . . . .	22
inspect_par_haldane . . . . .	24
inspect_par_multinomial . . . . .	25
inspect_prob . . . . .	27
inspect_true_or_false . . . . .	28
<b>Index</b>	<b>30</b>

---

inspect_bfactor	<i>Validate vectors of Bayes factors</i>
-----------------	------------------------------------------

---

### Description

inspect\_bfactor checks if an object is a numeric vector of valid Bayes factor values. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

### Usage

```
inspect_bfactor(x, allow_nas = TRUE, warning_nas = TRUE)
```

### Arguments

x	An arbitrary object.
allow_nas	Logical value. If TRUE then NA and NaN values in x are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in x.
warning_nas	Logical value. If TRUE then the presence of NA or NaN values in x generates a warning message. NA and NaN values pass silently otherwise (if allow_nas is TRUE).

## Details

`inspect_bfactor` conducts a series of tests to check if `x` is a numeric vector of valid Bayes factor values. Namely, `inspect_bfactor` checks if:

- `x` is NULL or empty.
- `x` is an atomic vector.
- `x` is numeric.
- `x` has NA or NaN values.
- The values of `x` are non-negative.

## Value

`inspect_bfactor` does not return any output. There are three possible outcomes:

- The call is silent if:
  - `x` is a numeric vector of valid Bayes factor values and there are no NA or NaN values in `x`.
  - `x` is a numeric vector of valid Bayes factor values, there are some NA or NaN values in `x`, `allow_nas` is set to TRUE and `warning_nas` is set to FALSE.
- An informative warning message is given if `x` is a numeric vector of valid Bayes factor values, there are some NA or NaN values in `x` and both `allow_nas` and `warning_nas` are set to TRUE.
- An informative error message is thrown and the execution is stopped if:
  - `x` is not a numeric vector of valid Bayes factor values.
  - `x` is a numeric vector of valid Bayes factor values, there are some in NA or NaN values in `x` and `allow_nas` is set to FALSE.

## See Also

- [inspect\\_bfactor\\_log](#) to check if an object is a numeric vector of valid logarithmic Bayes factor values.
- [bfactor\\_interpret](#) for the interpretation of Bayes factors.
- [inspect\\_bfactor\\_scale](#) to check if an object is a valid Bayes factor interpretation scale.

## Examples

```
# Calls that pass silently:
x1 <- c(0, 0.5, 1, 10, 50, 100)
x2 <- c(NA, 0.5, 1, 10, 50, 100)
inspect_bfactor(x1)
inspect_bfactor(x2, warning_nas = FALSE)
inspect_bfactor(x2, allow_nas = TRUE, warning_nas = FALSE)

# Call that throws an informative warning message:
y <- c(0.1, 0.2, NA, 0.4, 0.5)
try(inspect_bfactor(y))
try(inspect_bfactor(y, warning_nas = TRUE))
try(inspect_bfactor(y, allow_nas = TRUE, warning_nas = TRUE))
```

```
# Calls that throw informative error messages:
z <- c(-0.9, 0, 0.1, 0.2, 0.3, 0.4, 0.5)
try(inspect_bfactor(z))
mylist <- list(NULL, TRUE, factor(.5), matrix(0.5),
              "0.5", list(0.5), NA, NaN, numeric(0), -0.5, -5)
try(inspect_bfactor(mylist[[1]]))
try(inspect_bfactor(mylist[[2]]))
try(inspect_bfactor(mylist[[3]]))
try(inspect_bfactor(mylist[[4]]))
try(inspect_bfactor(mylist[[5]]))
try(inspect_bfactor(mylist[[6]]))
try(inspect_bfactor(mylist[[7]]))
try(inspect_bfactor(mylist[[8]]))
try(inspect_bfactor(mylist[[9]]))
try(inspect_bfactor(mylist[[10]]))
try(inspect_bfactor(mylist[[11]]))
```

---

inspect\_bfactor\_log    *Validate vectors of logarithmic Bayes factors*

---

## Description

inspect\_bfactor\_log checks if an object is a numeric vector of valid logarithmic Bayes factor values. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

## Usage

```
inspect_bfactor_log(x, allow_nas = TRUE, warning_nas = TRUE)
```

## Arguments

x	An arbitrary object.
allow_nas	Logical value. If TRUE then NA and NaN values in x are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in x.
warning_nas	Logical value. If TRUE then the presence of NA or NaN values in x generates a warning message. NA and NaN values pass silently otherwise (if allow_nas is TRUE).

## Details

inspect\_bfactor\_log conducts a series of tests to check if x is a numeric vector of valid logarithmic Bayes factor values. Namely, inspect\_bfactor\_log checks if:

- x is NULL or empty.
- x is an atomic vector.
- x is numeric.
- x has NA or NaN values.

**Value**

inspect\_bfactor\_log does not return any output. There are three possible outcomes:

- The call is silent if:
  - x is a numeric vector of valid logarithmic Bayes factor values and there are no NA or NaN values in x.
  - x is a numeric vector of valid logarithmic Bayes factor values, there are some NA or NaN values in x, allow\_nas is set to TRUE and warning\_nas is set to FALSE.
- An informative warning message is given if x is a numeric vector of valid logarithmic Bayes factor values, there are some NA or NaN values in x and both allow\_nas and warning\_nas are set to TRUE.
- An informative error message is thrown and the execution is stopped if:
  - x is not a numeric vector of valid logarithmic Bayes factor values.
  - x is a numeric vector of valid logarithmic Bayes factor values, there are some NA or NaN values in x and allow\_nas is set to FALSE.

**See Also**

- [inspect\\_bfactor](#) to check if an object is a numeric vector of valid Bayes factor values.
- [bfactor\\_log\\_interpret](#) for the interpretation of the logarithms of Bayes factors.
- [inspect\\_bfactor\\_scale](#) to check if an object is a Bayes factor interpretation scale.
- [inspect\\_log\\_base](#) to check if an object is an eligible logarithmic base.

**Examples**

```
# Calls that pass silently:
x1 <- c(0, 0.5, 1, 10, 50, 100)
x2 <- c(NA, 0.5, 1, 10, 50, 100)
inspect_bfactor_log(x1)
inspect_bfactor_log(x2, warning_nas = FALSE)
inspect_bfactor_log(x2, allow_nas = TRUE, warning_nas = FALSE)

# Call that throws an informative warning message:
y <- c(0.1, 0.2, NA, 0.4, 0.5)
try(inspect_bfactor_log(y))
try(inspect_bfactor_log(y, warning_nas = TRUE))
try(inspect_bfactor_log(y, allow_nas = TRUE, warning_nas = TRUE))

# Calls that throw informative error messages:
mylist <- list(NULL, TRUE, factor(.5), matrix(0.5),
              "0.5", list(0.5), numeric(0), NA, NaN)
try(inspect_bfactor_log(mylist[[1]]))
try(inspect_bfactor_log(mylist[[2]]))
try(inspect_bfactor_log(mylist[[3]]))
try(inspect_bfactor_log(mylist[[4]]))
try(inspect_bfactor_log(mylist[[5]]))
try(inspect_bfactor_log(mylist[[6]]))
try(inspect_bfactor_log(mylist[[7]]))
```

```
try(inspect_bfactor_log(mylist[[8]]))  
try(inspect_bfactor_log(mylist[[9]]))
```

---

inspect\_bfactor\_scale *Validate Bayes factor interpretation scales*

---

## Description

inspect\_bfactor\_scale checks if an object is a character vector of [length](#) 1 that is eligible to represent one of the Bayes factor interpretation scales available in the pcal package. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_bfactor_scale(x)
```

## Arguments

x                    An arbitrary object.

## Details

inspect\_bfactor\_scale conducts a series of tests to check if x is a character vector of [length](#) 1 that is eligible to represent one of the Bayes factor interpretation scales available in the pcal package. Namely, inspect\_bfactor\_scale checks if:

- x is NULL or empty.
- x is NA or NaN.
- x is an atomic vector of [length](#) 1
- The [typeof](#) x is character
- The value of x is either "Jeffreys" or "Kass-Raftery" (not case sensitive).

## Value

inspect\_bfactor\_scale does not return any output. There are two possible scenarios:

- The call is silent if x is a character vector of [length](#) 1 that is eligible to represent one of the Bayes factor interpretation scales available in the pcal package.
- An informative error message is thrown otherwise.

## See Also

- [bfactor\\_interpret](#) for the interpretation of Bayes factors.
- [bfactor\\_log\\_interpret](#) for the interpretation of the logarithms of Bayes factors.
- [inspect\\_bfactor](#) to check if an object is a numeric vector of valid Bayes factor values.
- [inspect\\_bfactor\\_log](#) to check if an object is a numeric vector of valid logarithmic Bayes factor values.

## Examples

```
# Calls that pass silently:
x1 <- "Jeffreys"
x2 <- "jeffreys"
x3 <- "kass-raftery"
x4 <- "Kass-Raftery"
inspect_bfactor_scale(x1)
inspect_bfactor_scale(x2)
inspect_bfactor_scale(x3)
inspect_bfactor_scale(x4)

# Calls that throw informative error messages:
mylist <- list(NULL, NA, NaN, 10, "Bayes", "Jeff",
              "kassraftery", c("jeffreys", "kass-raftery"))
try(inspect_bfactor_scale(mylist[[1]]))
try(inspect_bfactor_scale(mylist[[2]]))
try(inspect_bfactor_scale(mylist[[3]]))
try(inspect_bfactor_scale(mylist[[4]]))
try(inspect_bfactor_scale(mylist[[5]]))
try(inspect_bfactor_scale(mylist[[6]]))
try(inspect_bfactor_scale(mylist[[7]]))
try(inspect_bfactor_scale(mylist[[8]]))
```

---

inspect\_categories      *Validate factor levels*

---

## Description

inspect\_categories checks if an object is eligible to be used as the levels of a factor. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_categories(x)
```

## Arguments

x                      An arbitrary object.

## Details

inspect\_categories conducts a series of tests to check if x is eligible to be used as the levels of a factor. Namely, inspect\_categories checks if:

- x is NULL or empty.
- x is atomic.
- x has an eligible data type (logical, integer, double, character).
- There are NA or NaN values in x.
- There are repeated values in x.

**Value**

inspect\_categories does not return any output. There are two possible outcomes:

- The call is silent if `x` is eligible to be used as the levels of a factor.
- An informative error message is thrown otherwise.

**See Also**

- [inspect\\_data\\_dichotomous](#) to validate dichotomous data.
- [inspect\\_data\\_categorical](#) and [inspect\\_data\\_cat\\_as\\_dichotomous](#) to validate categorical data.
- [inspect\\_par\\_bernoulli](#) to validate Bernoulli/Binomial proportions.
- [inspect\\_par\\_multinomial](#) to validate vectors of Multinomial proportions.
- [inspect\\_character](#) to validate character vectors.
- [inspect\\_character\\_match](#) to validate character vectors with predefined allowed values.

**Examples**

```
# Calls that pass silently:
x1 <- 1:5
x2 <- c("yes", "no")
x3 <- c(TRUE, FALSE)
x4 <- factor(c("smoker", "non-smoker"))
x5 <- factor(c("yes", "no", "yes"))
inspect_categories(x1)
inspect_categories(x2)
inspect_categories(x3)
inspect_categories(x4)
inspect_categories(levels(x5))

# Calls that throw informative error messages:
y1 <- c(1, 1:5)
y2 <- c("yes", "no", "yes")
y3 <- factor(c("yes", "no", "yes"))
try(inspect_categories(y1))
try(inspect_categories(y2))
try(inspect_categories(y3))
try(mylist <- list(NULL, numeric(0),
  complex(1), list(10), NaN, NA))
try(inspect_categories(mylist[[1]]))
try(inspect_categories(mylist[[2]]))
try(inspect_categories(mylist[[3]]))
try(inspect_categories(mylist[[4]]))
try(inspect_categories(mylist[[5]]))
try(inspect_categories(mylist[[6]])
```

---

inspect_character	<i>Validate character vectors</i>
-------------------	-----------------------------------

---

### Description

`inspect_character` checks if an object is a character vector. This can be useful to validate inputs in user-defined functions.

### Usage

```
inspect_character(x, allow_nas = TRUE, warning_nas = FALSE)
```

### Arguments

<code>x</code>	An arbitrary object.
<code>allow_nas</code>	Logical value. If TRUE then NA and NaN values in <code>x</code> are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in <code>x</code> .
<code>warning_nas</code>	Logical value. If TRUE then the presence of NA or NaN values in <code>x</code> generates a warning message. NA and NaN values pass silently otherwise (if <code>allow_nas</code> is set to TRUE).

### Details

`inspect_character` conducts a series of tests to check if `x` is a character vector. Namely, `inspect_character` checks if:

- `x` is NULL or empty.
- `x` is an atomic vector.
- The `typeof` `x` is character.
- There are NA or NaN values in `x`.

### Value

`inspect_character` does not return any output. There are three possible outcomes:

- The call is silent if:
  - `x` is a character vector and there are no NA or NaN values in `x`.
  - `x` is a character vector, there are some NA or NaN values in `x`, `allow_nas` is set to TRUE and `warning_nas` is set to FALSE.
- An informative warning message is thrown if `x` is a character vector, there are some NA or NaN values in `x` and both `allow_nas` and `warning_nas` are set to TRUE.
- An informative error message is thrown if:
  - `x` is not a character vector.
  - `x` is a character vector, there are some NA or NaN values in `x` and `allow_nas` is set to FALSE.

**See Also**

- [inspect\\_character\\_match](#) to validate character vectors with predefined allowed values.
- [inspect\\_true\\_or\\_false](#) to check if an object is a non-missing logical value.

**Examples**

```
# Calls that pass silently:
x1 <- "Kass"
x2 <- c("Kass", "Raftery")
x3 <- c("Kass", "Raftery", NA)
x4 <- letters
inspect_character(x1)
inspect_character(x2)
inspect_character(x3)
inspect_character(x4)

# Call that throws an informative warning message
y <- c("Kass", "Raftery", NA)
try(inspect_character(y, warning_nas = TRUE))

# Calls that throw informative error messages
try(inspect_character(y, allow_nas = FALSE))
mylist <- list(NULL, character(0), 1,
  c(1, 2), factor(c(1,2)), list(c(1,2)), NaN, NA)
try(inspect_character(mylist[[1]]))
try(inspect_character(mylist[[2]]))
try(inspect_character(mylist[[3]]))
try(inspect_character(mylist[[4]]))
try(inspect_character(mylist[[5]]))
try(inspect_character(mylist[[6]]))
try(inspect_character(mylist[[7]]))
try(inspect_character(mylist[[8]]))
```

---

inspect\_character\_match

*Validate character values*

---

**Description**

`inspect_character_match` checks if an object is a character vector of `length` 1 that belongs to a set of allowed values. This can be useful to validate inputs in user-defined functions.

**Usage**

```
inspect_character_match(x, allowed, case_sensitive = FALSE)
```

**Arguments**

x	An arbitrary object.
allowed	A character vector.
case_sensitive	A non-missing logical value.

**Details**

inspect\_character\_match conducts a series of tests to check if x is a character vector of [length 1](#) whose value belongs to the set of allowed values. Namely, inspect\_character\_match checks if:

- x is NULL or empty.
- x is an atomic vector of [length 1](#).
- The [typeof](#) x is character.
- x is NA or NaN.
- x is one of the allowed values (as specified in the allowed argument).

By default, the comparison of x with allowed is not case sensitive. If you only want case sensitive matches of x to allowed set case\_sensitive to TRUE.

**Value**

inspect\_character\_match does not return any output. There are two possible outcomes:

- The call is silent if x is a character vector of [length 1](#) whose value belongs to the set of allowed values.
- An informative error message is thrown otherwise.

**See Also**

- [inspect\\_character](#) to validate character vectors with arbitrary allowed values.
- [inspect\\_true\\_or\\_false](#) to check if an object is a non-missing logical value.

**Examples**

```
# Calls that pass silently:
x1 <- "Kass"
x2 <- "kass"
inspect_character_match(x1, allowed = c("Kass", "Raftery"))
inspect_character_match(x2, allowed = c("Kass", "Raftery"))

# Calls that throw informative error messages:
y1 <- "kasss"
y2 <- "kass"
try(inspect_character_match(y1, allowed = c("Kass", "Raftery")))
try(inspect_character_match(y2, allowed = c("Kass", "Raftery"),
  case_sensitive = TRUE))
mylist <- list(NULL, character(0), c("abc", "abcd"),
  c("abc", "abc"), "ab", list("abc"), factor("abc"), NaN, NA)
```

```
try(inspect_character_match(mylist[[1]], "abc"))
try(inspect_character_match(mylist[[2]], "abc"))
try(inspect_character_match(mylist[[3]], "abc"))
try(inspect_character_match(mylist[[4]], "abc"))
try(inspect_character_match(mylist[[5]], "abc"))
try(inspect_character_match(mylist[[6]], "abc"))
try(inspect_character_match(mylist[[7]], "abc"))
try(inspect_character_match(mylist[[8]], "abc"))
try(inspect_character_match(mylist[[9]], "abc"))
```

---

inspect\_data\_categorical

*Validate categorical data*

---

## Description

`inspect_data_categorical` checks if an object contains data that is eligible to have been generated by a Multinomial distribution. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_data_categorical(data, allow_nas = TRUE, warning_nas = FALSE)
```

## Arguments

<code>data</code>	An arbitrary object.
<code>allow_nas</code>	Logical value. If TRUE then NA and NaN values in data are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in data.
<code>warning_nas</code>	Logical value. If TRUE then the presence of NA or NaN values in data generates a warning message. NA and NaN values pass silently otherwise (if <code>allow_nas</code> is set to TRUE).

## Details

`inspect_data_categorical` conducts a series of tests to check if data is eligible to have been generated by a Multinomial distribution. Namely, `inspect_data_categorical` checks if:

- data is NULL or empty.
- data is atomic and have an eligible data type (logical, integer, double, character).
- data has NA or NaN values.

**Value**

`inspect_data_categorical` does not return any output. There are three possible outcomes:

- The call is silent if:
  - data is eligible to have been generated by a Multinomial distribution and there are no NA or NaN values in data.
  - data is eligible to have been generated by a Multinomial distribution, there are some NA or NaN values in data and `warning_nas` is set to `FALSE`.
- An informative warning message is thrown if: data is eligible to have been generated by a Multinomial distribution, there are some NA or NaN values in data and `warning_nas` is set to `TRUE`.
- An informative error message is thrown and the execution is stopped if:
  - data is not eligible to have been generated by a Multinomial distribution.
  - data is eligible to have been generated by a Multinomial distribution, there are some NA or NaN values in data and `allow_nas` is set to `TRUE`.

**See Also**

- [inspect\\_data\\_cat\\_as\\_dichotomous](#) to validate categorical data as dichotomous.
- [inspect\\_par\\_multinomial](#) to validate vectors of Multinomial proportions.
- [inspect\\_data\\_dichotomous](#) to validate dichotomous data.
- [inspect\\_par\\_bernoulli](#) to validate Bernoulli/Binomial proportions.

**Examples**

```
# Calls that pass silently:
x1 <- c(1, 0, 0, 1, 2)
x2 <- c(FALSE, FALSE, TRUE, NA)
x3 <- c("yes", "no", "yes", "maybe")
x4 <- factor(c("yes", "no", "yes", "maybe"))
x5 <- c(1, 0, 0, 1, 0, NA, 2)
inspect_data_categorical(x1)
inspect_data_categorical(x2)
inspect_data_categorical(x3)
inspect_data_categorical(x4)
inspect_data_categorical(x5)
inspect_data_categorical(x5)

# Call that throws an informative warning message:
y1 <- c(1, 1, NA, 0, 0, 2)
try(inspect_data_categorical(y1, warning_nas = TRUE))

# Calls that throw an informative error message:
z <- c(1, 1, NA, 0, 0, 2)
try(inspect_data_categorical(z, allow_nas = FALSE))
try(inspect_data_categorical(NULL))
try(inspect_data_categorical(list(1,0)))
try(inspect_data_categorical(numeric(0)))
```

```
try(inspect_data_categorical(NaN))  
try(inspect_data_categorical(NA))
```

---

inspect\_data\_cat\_as\_dichotomous

*Validate categorical data as dichotomous*

---

## Description

inspect\_data\_cat\_as\_dichotomous checks if an object contains valid categorical data that is eligible to be used as dichotomous data. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_data_cat_as_dichotomous(  
  data,  
  success,  
  allow_nas = TRUE,  
  warning_nas = FALSE  
)
```

## Arguments

data, success	Arbitrary objects. success is meant to indicate the value of data that corresponds to a success.
allow_nas	Logical value. If TRUE then NA and NaN values in data are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in data.
warning_nas	Logical value. If TRUE then the presence of NA or NaN values in data generates a warning message. NA and NaN values pass silently otherwise (if allow_nas is set to TRUE).

## Details

inspect\_data\_cat\_as\_dichotomous conducts a series of tests to check if data contains valid categorical data that is eligible to be used as dichotomous data. Namely, inspect\_data\_cat\_as\_dichotomous checks if:

- data and success are NULL or empty.
- data and success are atomic and have an eligible data type (logical, integer, double, character).
- data and success have NA or NaN values.
- success has `length 1`.
- success is observed in data.

**Value**

`inspect_data_cat_as_dichotomous` does not return any output. There are three possible outcomes:

- The call is silent if:
  - data contains valid categorical data that is eligible to be used as dichotomous data and there are no NA or NaN values in data.
  - data contains valid categorical data that is eligible to be used as dichotomous data, there are some NA or NaN values in data, `allow_nas` is set to TRUE and `warning_nas` is set to FALSE.
- An informative warning message is thrown if:
  - data contains valid categorical data that is eligible to be used as dichotomous data and success is not observed in data.
  - data contains valid categorical data that is eligible to be used as dichotomous data, there are NA or NaN values in data and both `allow_nas` and `warning_nas` are set to TRUE.
- An informative error message is thrown and the execution is stopped if:
  - data does not contain valid categorical data that is eligible to be used as dichotomous data.
  - data contains valid categorical data that is eligible to be used as dichotomous data, there are some NA or NaN values in data and `allow_nas` is set to FALSE.

**See Also**

- [inspect\\_data\\_categorical](#) to validate categorical.
- [inspect\\_par\\_multinomial](#) to validate vectors of Multinomial proportions.
- [inspect\\_data\\_dichotomous](#) to validate dichotomous data.
- [inspect\\_par\\_bernoulli](#) to validate Bernoulli/Binomial proportions.

**Examples**

```
# Calls that pass silently:
x1 <- c(1, 0, 0, 1, 0)
x2 <- c(FALSE, FALSE, TRUE)
x3 <- c("yes", "no", "yes")
x4 <- factor(c("yes", "no", "yes"))
x5 <- c(1, 0, 0, 1, 0, NA)
inspect_data_cat_as_dichotomous(x1, success = 1)
inspect_data_cat_as_dichotomous(x2, success = TRUE)
inspect_data_cat_as_dichotomous(x3, success = "yes")
inspect_data_cat_as_dichotomous(x4, success = "yes")
inspect_data_cat_as_dichotomous(x5, success = 1)

# Calls that throw an informative warning message:
y1 <- c(1, 1, NA, 0, 0)
y2 <- c(0, 0)
success <- 1
try(inspect_data_cat_as_dichotomous(y1, success = 1, warning_nas = TRUE))
```

```

try(inspect_data_cat_as_dichotomous(y2, success = success))

# Calls that throw an informative error message:
try(inspect_data_cat_as_dichotomous(y1, 1, allow_nas = FALSE))
try(inspect_data_cat_as_dichotomous(NULL, 1))
try(inspect_data_cat_as_dichotomous(c(1,0), NULL))
try(inspect_data_cat_as_dichotomous(list(1,0), 1))
try(inspect_data_cat_as_dichotomous(c(1,0), list(1)))
try(inspect_data_cat_as_dichotomous(numeric(0), 0))
try(inspect_data_cat_as_dichotomous(1, numeric(0)))
try(inspect_data_cat_as_dichotomous(NaN, 1))
try(inspect_data_cat_as_dichotomous(NA, 1))
try(inspect_data_cat_as_dichotomous(c(1,0), NA))
try(inspect_data_cat_as_dichotomous(c(1,0), NaN))
try(inspect_data_cat_as_dichotomous(c(1,0), 2))

```

---

inspect\_data\_dichotomous

*Validate dichotomous data*

---

## Description

inspect\_data\_dichotomous checks if an object contains data that is eligible to have been generated by a series of Bernoulli trials. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_data_dichotomous(data, success, allow_nas = TRUE, warning_nas = FALSE)
```

## Arguments

data, success	Arbitrary objects. success is meant to indicate the value of data that corresponds to a success.
allow_nas	Logical value. If TRUE then NA and NaN values in data are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in data.
warning_nas	Logical value. If TRUE then the presence of NA or NaN values in data generates a warning message. NA and NaN values pass silently otherwise (if allow_nas is set to TRUE).

## Details

inspect\_data\_dichotomous conducts a series of tests to check if data is eligible to have been generated by a series of Bernoulli trials. Namely, inspect\_data\_dichotomous checks if:

- data and success are NULL or empty.

- data and success are atomic and have an eligible data type (logical, integer, double, character).
- data and success have NA or NaN values.
- The number of unique values in data and success are adequate.
- success has `length 1`.
- success is observed in data.

## Value

`inspect_data_dichotomous` does not return any output. There are three possible outcomes:

- The call is silent if:
  - data is eligible to have been generated by a series of Bernoulli trials and there are no NA or NaN values in data.
  - data is eligible to have been generated by a series of Bernoulli trials, there are some NA or NaN values in data, `allow_nas` is set to TRUE and `warning_nas` is set to FALSE.
- An informative warning message is thrown if:
  - data is eligible to have been generated by a series of Bernoulli trials and success is not observed in data.
  - data is eligible to have been generated by a series of Bernoulli trials, there are NA or NaN values in data and both `allow_nas` and `warning_nas` are set to TRUE.
- An informative error message is thrown and the execution is stopped if:
  - data is not eligible to have been generated by a series of Bernoulli trials.
  - data is eligible to have been generated by a series of Bernoulli trials, there are some NA or NaN values in data and `allow_nas` is set to FALSE.

## See Also

- [inspect\\_par\\_bernoulli](#) to validate Bernoulli/Binomial proportions.
- [inspect\\_data\\_categorical](#) and [inspect\\_data\\_cat\\_as\\_dichotomous](#) to validate categorical data.
- [inspect\\_par\\_multinomial](#) to validate vectors of Multinomial proportions.

## Examples

```
# Calls that pass silently:
x1 <- c(1, 0, 0, 1, 0)
x2 <- c(FALSE, FALSE, TRUE)
x3 <- c("yes", "no", "yes")
x4 <- factor(c("yes", "no", "yes"))
x5 <- c(1, 0, 0, 1, 0, NA)
inspect_data_dichotomous(x1, success = 1)
inspect_data_dichotomous(x2, success = TRUE)
inspect_data_dichotomous(x3, success = "yes")
inspect_data_dichotomous(x4, success = "yes")
inspect_data_dichotomous(x5, success = 1)
```

```

# Calls that throw an informative warning message:
y1 <- c(1, 1, NA, 0, 0)
y2 <- c(0, 0)
success <- 1
try(inspect_data_dichotomous(y1, success = 1, warning_nas = TRUE))
try(inspect_data_dichotomous(y2, success = success))

# Calls that throw an informative error message:
try(inspect_data_dichotomous(NULL, 1))
try(inspect_data_dichotomous(c(1,0), NULL))
try(inspect_data_dichotomous(list(1,0), 1))
try(inspect_data_dichotomous(c(1,0), list(1)))
try(inspect_data_dichotomous(numeric(0), 0))
try(inspect_data_dichotomous(1, numeric(0)))
try(inspect_data_dichotomous(NaN, 1))
try(inspect_data_dichotomous(NA, 1))
try(inspect_data_dichotomous(c(1,0), NA))
try(inspect_data_dichotomous(c(1,0), NaN))
try(inspect_data_dichotomous(c(1,0), 2))

```

---

inspect\_log\_base

*Validate logarithmic bases*


---

## Description

inspect\_log\_base checks if an object is a valid a logarithmic base. This can be useful to validate inputs in user-defined functions.

## Usage

```
inspect_log_base(x)
```

## Arguments

x                    An arbitrary object.

## Details

inspect\_log\_base conducts a series of tests to check if x is a valid logarithmic base. Namely, inspect\_log\_base checks if:

- x is NULL or empty.
- x is an atomic vector of [length 1](#).
- x is numeric.
- x is NA or NaN.
- x is positive.

**Value**

inspect\_log\_base does not return any output. There are two possible outcomes:

- The call is silent if `x` is a numeric vector of [length](#) 1 that is a valid logarithmic base.
- An informative error message is thrown otherwise.

**See Also**

- [bfactor\\_log\\_interpret](#) for the interpretation of the logarithms of Bayes factors.
- [inspect\\_bfactor\\_log](#) to check if an object is a numeric vector of valid logarithmic Bayes factor values.

**Examples**

```
# Calls that pass silently:
x1 <- 10
x2 <- exp(1)
x3 <- 0.5
inspect_log_base(x1)
inspect_log_base(x2)
inspect_log_base(x3)

# Calls that throw informative error messages:
mylist <- list(NULL, numeric(0), TRUE, factor(10),
              list(10), matrix(10), NaN, NA, -1, 0)
try(inspect_log_base(mylist[[1]]))
try(inspect_log_base(mylist[[2]]))
try(inspect_log_base(mylist[[3]]))
try(inspect_log_base(mylist[[4]]))
try(inspect_log_base(mylist[[5]]))
try(inspect_log_base(mylist[[6]]))
try(inspect_log_base(mylist[[7]]))
try(inspect_log_base(mylist[[8]]))
try(inspect_log_base(mylist[[9]]))
try(inspect_log_base(mylist[[10]]))
```

---

inspect\_par\_bernoulli *Validate parameters for the Bernoulli/Binomial distributions*

---

**Description**

inspect\_par\_bernoulli checks if an object is an eligible Bernoulli/Binomial proportion. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

**Usage**

```
inspect_par_bernoulli(x)
```

## Arguments

x                    An arbitrary object.

## Details

`inspect_par_bernoulli` conducts a series of tests to check if `x` is an eligible Bernoulli/Binomial proportion. Namely, `inspect_par_bernoulli` checks if:

- `x` is NULL or empty.
- `x` is an atomic vector
- `x` is numeric
- `x` has `length 1`
- `x` is NA or NaN.
- `x` is in the (0, 1) interval.

## Value

`inspect_par_bernoulli` does not return any output. There are two possible outcomes:

- The call is silent if `x` is an eligible Bernoulli/Binomial proportion.
- An informative error message is thrown otherwise.

## See Also

- [inspect\\_par\\_multinomial](#) to validate parameters for the Multinomial distribution.
- [inspect\\_par\\_beta](#) to validate parameters for the Beta distribution.
- [inspect\\_par\\_dirichlet](#) to validate parameters for the Dirichlet distribution.
- [inspect\\_par\\_haldane](#) to validate parameters for the Haldane distribution.
- [inspect\\_data\\_dichotomous](#) to validate dichotomous data.
- [inspect\\_prob](#) to check if an object is a numeric vector of valid probability values.

## Examples

```
# Calls that pass silently:
x <- 0.5
inspect_par_bernoulli(x)
inspect_par_bernoulli(0.1)

# Calls that throw an informative error message:
mylist <- list(NULL, TRUE, factor(.5), matrix(0.5), "0.5",
  list(0.5), NA, NaN, numeric(0), c(0.1, 0.5), -0.5, 1.1)
try(inspect_par_bernoulli(mylist[[1]]))
try(inspect_par_bernoulli(mylist[[2]]))
try(inspect_par_bernoulli(mylist[[3]]))
try(inspect_par_bernoulli(mylist[[4]]))
try(inspect_par_bernoulli(mylist[[5]]))
try(inspect_par_bernoulli(mylist[[6]]))
```

```
try(inspect_par_bernoulli(mylist[[7]]))
try(inspect_par_bernoulli(mylist[[8]]))
try(inspect_par_bernoulli(mylist[[9]]))
try(inspect_par_bernoulli(mylist[[10]]))
try(inspect_par_bernoulli(mylist[[11]]))
try(inspect_par_bernoulli(mylist[[12]]))
```

---

inspect_par_beta	<i>Validate parameters for the Beta distribution</i>
------------------	------------------------------------------------------

---

## Description

inspect\_par\_beta checks if an object is an eligible vector of parameters for the Beta distribution. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

## Usage

```
inspect_par_beta(x)
```

## Arguments

x                    An arbitrary object.

## Details

inspect\_par\_beta conducts a series of tests to check if x is an eligible vector of parameters for the Beta distribution. Namely, inspect\_par\_beta checks if:

- x is NULL or empty.
- x is an atomic vector
- x is numeric
- x has `length 2`
- x has NA or NaN values.
- All elements of x are positive.

## Value

inspect\_par\_beta does not return any output. There are two possible outcomes:

- The call is silent if x is an eligible vector of parameters for the Beta distribution.
- An informative error message is thrown otherwise.

**See Also**

- [inspect\\_par\\_bernoulli](#) to validate parameters for the Bernoulli/Binomial distribution.
- [inspect\\_par\\_multinomial](#) to validate parameters for the Multinomial distribution.
- [inspect\\_par\\_dirichlet](#) to validate parameters for the Dirichlet distribution.
- [inspect\\_par\\_haldane](#) to validate parameters for the Haldane distribution.

**Examples**

```
# Calls that pass silently:
x1 <- c(1, 1)
x2 <- c(2, 5)
inspect_par_beta(x1)
inspect_par_beta(x2)

# Calls that throw an informative error message:
mylist <- list(NULL, 1, factor(1, 1),
  matrix(c(1, 1)), c("1", "1"), list(1, 1), c(1, NA),
  c(1, NaN), c(TRUE, FALSE), numeric(0), c(-1, 1))
try(inspect_par_beta(mylist[[1]]))
try(inspect_par_beta(mylist[[2]]))
try(inspect_par_beta(mylist[[3]]))
try(inspect_par_beta(mylist[[4]]))
try(inspect_par_beta(mylist[[5]]))
try(inspect_par_beta(mylist[[6]]))
try(inspect_par_beta(mylist[[7]]))
try(inspect_par_beta(mylist[[8]]))
try(inspect_par_beta(mylist[[9]]))
try(inspect_par_beta(mylist[[10]]))
try(inspect_par_beta(mylist[[11]]))
```

---

inspect\_par\_dirichlet *Validate parameters for the Dirichlet distribution*

---

**Description**

inspect\_par\_dirichlet checks if an object is an eligible vector of parameters for the Dirichlet distribution. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

**Usage**

```
inspect_par_dirichlet(x)
```

**Arguments**

x                    An arbitrary object.

## Details

`inspect_par_dirichlet` conducts a series of tests to check if `x` is an eligible vector of parameters for the Dirichlet distribution. Namely, `inspect_par_dirichlet` checks if:

- `x` is NULL or empty.
- `x` is an atomic vector
- `x` is numeric
- `x` has NA or NaN values.
- All elements of `x` are positive.

## Value

`inspect_par_dirichlet` does not return any output. There are two possible outcomes:

- The call is silent if `x` is an eligible vector of parameters for the Dirichlet distribution.
- An informative error message is thrown otherwise.

## See Also

- [inspect\\_par\\_bernoulli](#) to validate parameters for the Bernoulli/Binomial distribution.
- [inspect\\_par\\_multinomial](#) to validate parameters for the Multinomial distribution.
- [inspect\\_par\\_beta](#) to validate parameters for the Beta distribution.
- [inspect\\_par\\_haldane](#) to validate parameters for the Haldane distribution.

## Examples

```
# Calls that pass silently:
x1 <- c(1, 1, 1)
x2 <- c(2, 5)
inspect_par_dirichlet(x1)
inspect_par_dirichlet(x2)

# Calls that throw an informative error message:
mylist <- list(NULL, factor(1, 1, 1),
  matrix(c(1, 1, 1)), c("1", "1", "1"), list(1, 1, 1), c(1, NA),
  c(1, NaN, 1), c(TRUE, FALSE), numeric(0), c(-1, 1, 1))
try(inspect_par_dirichlet(mylist[[1]]))
try(inspect_par_dirichlet(mylist[[2]]))
try(inspect_par_dirichlet(mylist[[3]]))
try(inspect_par_dirichlet(mylist[[4]]))
try(inspect_par_dirichlet(mylist[[5]]))
try(inspect_par_dirichlet(mylist[[6]]))
try(inspect_par_dirichlet(mylist[[7]]))
try(inspect_par_dirichlet(mylist[[8]]))
try(inspect_par_dirichlet(mylist[[9]]))
try(inspect_par_dirichlet(mylist[[10]]))
```

---

inspect\_par\_haldane    *Validate parameters for the Haldane distribution*

---

### Description

inspect\_par\_haldane checks if an object is an eligible vector of parameters for the Haldane distribution. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

### Usage

```
inspect_par_haldane(x)
```

### Arguments

x                    An arbitrary object.

### Details

inspect\_par\_haldane conducts a series of tests to check if x is an eligible vector of parameters for the Haldane distribution. Namely, inspect\_par\_haldane checks if:

- x is NULL or empty.
- x is an atomic vector
- x is numeric
- x has NA or NaN values.
- All elements of x equal to 0.

### Value

inspect\_par\_haldane does not return any output. There are two possible outcomes:

- The call is silent if x is an eligible vector of parameters for the Haldane distribution.
- An informative error message is thrown otherwise.

### See Also

- [inspect\\_par\\_bernoulli](#) to validate parameters for the Bernoulli/Binomial distribution.
- [inspect\\_par\\_multinomial](#) to validate parameters for the Multinomial distribution.
- [inspect\\_par\\_beta](#) to validate parameters for the Beta distribution.
- [inspect\\_par\\_dirichlet](#) to validate parameters for the Dirichlet distribution.

## Examples

```
# Calls that pass silently:
x1 <- c(0, 0, 0)
x2 <- c(0, 0)
inspect_par_haldane(x1)
inspect_par_haldane(x2)

# Calls that throw an informative error message:
mylist <- list(NULL, factor(0, 0, 0),
  matrix(c(0, 0, 0)), c("0", "0", "0"), list(0, 0, 0), c(0, NA),
  c(0, NaN, 0), c(TRUE, FALSE), numeric(0), c(1, 0, 0))
try(inspect_par_haldane(mylist[[1]]))
try(inspect_par_haldane(mylist[[2]]))
try(inspect_par_haldane(mylist[[3]]))
try(inspect_par_haldane(mylist[[4]]))
try(inspect_par_haldane(mylist[[5]]))
try(inspect_par_haldane(mylist[[6]]))
try(inspect_par_haldane(mylist[[7]]))
try(inspect_par_haldane(mylist[[8]]))
try(inspect_par_haldane(mylist[[9]]))
try(inspect_par_haldane(mylist[[10]]))
```

---

inspect\_par\_multinomial

*Validate parameters for the Multinomial distribution*

---

## Description

inspect\_par\_multinomial checks if an object is an eligible vector of Multinomial proportions. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

## Usage

```
inspect_par_multinomial(x)
```

## Arguments

x                    An arbitrary object.

## Details

inspect\_par\_multinomial conducts a series of tests to check if x is an eligible vector of Multinomial proportions. Namely, inspect\_par\_multinomial checks if:

- x is NULL or empty.
- x is an atomic vector
- x is numeric

- `x` has NA or NaN values.
- All elements of `x` are in the (0, 1) interval.
- `x` sums to 1.

### Value

`inspect_par_multinomial` does not return any output. There are two possible outcomes:

- The call is silent if `x` is an eligible vector of Multinomial proportions.
- An informative error message is thrown otherwise.

### See Also

- [inspect\\_par\\_bernoulli](#) to validate parameters for the Bernoulli/Binomial distribution.
- [inspect\\_par\\_beta](#) to validate parameters for the Beta distribution.
- [inspect\\_par\\_dirichlet](#) to validate parameters for the Dirichlet distribution.
- [inspect\\_par\\_haldane](#) to validate parameters for the Haldane distribution.
- [inspect\\_data\\_categorical](#) and [inspect\\_data\\_cat\\_as\\_dichotomous](#) to validate categorical data.
- [inspect\\_prob](#) to check if an object is a numeric vector of valid probability values.

### Examples

```
# Calls that pass silently:
x1 <- c(0.5, 0.5)
x2 <- rep(1/5, 5)
inspect_par_multinomial(x1)
inspect_par_multinomial(x2)

# Calls that throw an informative error message:
mylist <- list(NULL, TRUE, factor(0.5, 0.5),
  matrix(c(0.5, 0.5)), c("0.5", "0.5"), list(0.5, 0.5),
  c(0.9, NA), c(0.9, NaN), numeric(0), NA, c(0.9, 0.6), c(-0.1, 0.9))
try(inspect_par_multinomial(mylist[[1]]))
try(inspect_par_multinomial(mylist[[2]]))
try(inspect_par_multinomial(mylist[[3]]))
try(inspect_par_multinomial(mylist[[4]]))
try(inspect_par_multinomial(mylist[[5]]))
try(inspect_par_multinomial(mylist[[6]]))
try(inspect_par_multinomial(mylist[[7]]))
try(inspect_par_multinomial(mylist[[8]]))
try(inspect_par_multinomial(mylist[[9]]))
try(inspect_par_multinomial(mylist[[10]]))
try(inspect_par_multinomial(mylist[[11]]))
try(inspect_par_multinomial(mylist[[12]]))
```

---

inspect_prob	<i>Validate vectors of probabilities</i>
--------------	------------------------------------------

---

### Description

inspect\_prob checks if an object is a numeric vector of valid probability values. This can be useful to validate inputs, intermediate calculations or outputs in user-defined functions.

### Usage

```
inspect_prob(x, allow_nas = TRUE, warning_nas = TRUE)
```

### Arguments

x	An arbitrary object.
allow_nas	Logical value. If TRUE then NA and NaN values in x are allowed. If FALSE, execution is stopped and an error message is thrown in case there are NA or NaN values in x.
warning_nas	Logical value. If TRUE then the presence of NA or NaN values in x generates a warning message. NA and NaN values pass silently otherwise (if allow_nas is set to TRUE).

### Details

inspect\_prob conducts a series of tests to check if x is a numeric vector of valid probability values. Namely, inspect\_prob checks if:

- x is NULL or empty.
- x is an atomic vector.
- x is numeric.
- x has NA or NaN values.
- The values of x are in the [0, 1] interval.

### Value

inspect\_prob does not return any output. There are three possible outcomes:

- The call is silent if:
  - x is a numeric vector of valid probability values and there are no NA or NaN values in x.
  - x is a numeric vector of valid probability values, there are some NA or NaN values in x, allow\_nas is set to TRUE and warning\_nas is set to FALSE.
- An informative warning message is thrown if x is a numeric vector of valid probability values, there are some NA or NaN values in x and both allow\_nas and warning\_nas are set to TRUE.
- An informative error message is thrown and the execution is stopped if:
  - x is not a numeric vector of valid probability values.
  - x is a numeric vector of valid probability values, there are some NA or NaN values in x and allow\_nas is set to FALSE.

**See Also**

- [inspect\\_par\\_bernoulli](#) to check if an object is a valid Bernoulli/Binomial proportion.
- [inspect\\_par\\_multinomial](#) to check if an object is a numeric vector of valid Multinomial proportions.

**Examples**

```
# Calls that pass silently:
x1 <- c(0.1, 0.2, 0.3, 0.4, 0.5)
x2 <- c(0.1, 0.2, 0.3, 0.4, 0.5, NA)
inspect_prob(x1)
inspect_prob(x2, warning_nas = FALSE)
inspect_prob(x2, allow_nas = TRUE, warning_nas = FALSE)

# Calls that throw an informative warning message:
y <- c(0.1, 0.2, NA, 0.4, 0.5)
try(inspect_prob(y))
try(inspect_prob(y, allow_nas = TRUE))
try(inspect_prob(y, allow_nas = TRUE, warning_nas = TRUE))

# Calls that throw an informative error message:
z1 <- c(-0.9, 0, 0.1, 0.2, 0.3, 0.4, 0.5)
try(inspect_prob(z1))
z2 <- c(NA, 0, 0.1, 0.2, 0.3, 0.4, 0.5)
try(inspect_prob(z2, allow_nas = FALSE))
mylist <- list(NULL, TRUE, factor(.5), matrix(0.5),
              "0.5", list(0.5), NA, NaN, numeric(0), 1.1, -0.5)
try(inspect_prob(mylist[[1]]))
try(inspect_prob(mylist[[2]]))
try(inspect_prob(mylist[[3]]))
try(inspect_prob(mylist[[4]]))
try(inspect_prob(mylist[[5]]))
try(inspect_prob(mylist[[6]]))
try(inspect_prob(mylist[[7]]))
try(inspect_prob(mylist[[8]]))
try(inspect_prob(mylist[[9]]))
try(inspect_prob(mylist[[10]]))
try(inspect_prob(mylist[[11]]))
```

---

inspect\_true\_or\_false *Validate non-missing logical values*

---

**Description**

`inspect_true_or_false` checks if an object is a non-missing logical vector of [length](#) 1. This can be useful to validate inputs in user-defined functions.

**Usage**

```
inspect_true_or_false(x)
```

**Arguments**

x                    An arbitrary object.

**Details**

inspect\_true\_or\_false conducts a series of tests to check if x is a non-missing logical vector of [length 1](#). Namely, inspect\_true\_or\_false checks if:

- x is NULL or empty.
- x is an atomic vector of [length 1](#).
- The [typeof](#) x is logical.
- x is NA or NaN.

**Value**

inspect\_true\_or\_false does not return any output. There are two possible scenarios:

- The call is silent if x is a non-missing logical vector of [length 1](#).
- An informative error message is thrown otherwise.

**See Also**

- [inspect\\_character](#) to validate character vectors.
- [inspect\\_character\\_match](#) to validate character vectors with predefined allowed values.

**Examples**

```
# Calls that pass silently:
x <- TRUE
y <- FALSE
inspect_true_or_false(x)
inspect_true_or_false(y)

# Calls that throw informative error messages:
mylist <- list(NULL, NA, NaN, 1, 0, "TRUE")
try(inspect_true_or_false(mylist[[1]]))
try(inspect_true_or_false(mylist[[2]]))
try(inspect_true_or_false(mylist[[3]]))
try(inspect_true_or_false(mylist[[4]]))
try(inspect_true_or_false(mylist[[5]]))
try(inspect_true_or_false(mylist[[6]]))
```

# Index

`bfactor_interpret`, [3](#), [6](#)  
`bfactor_log_interpret`, [5](#), [6](#), [19](#)

`inspect_bfactor`, [2](#), [5](#), [6](#)  
`inspect_bfactor_log`, [3](#), [4](#), [6](#), [19](#)  
`inspect_bfactor_scale`, [3](#), [5](#), [6](#)  
`inspect_categories`, [7](#)  
`inspect_character`, [8](#), [9](#), [11](#), [29](#)  
`inspect_character_match`, [8](#), [10](#), [10](#), [29](#)  
`inspect_data_cat_as_dichotomous`, [8](#), [13](#),  
[14](#), [17](#), [26](#)  
`inspect_data_categorical`, [8](#), [12](#), [15](#), [17](#),  
[26](#)  
`inspect_data_dichotomous`, [8](#), [13](#), [15](#), [16](#),  
[20](#)  
`inspect_log_base`, [5](#), [18](#)  
`inspect_par_bernoulli`, [8](#), [13](#), [15](#), [17](#), [19](#),  
[22–24](#), [26](#), [28](#)  
`inspect_par_beta`, [20](#), [21](#), [23](#), [24](#), [26](#)  
`inspect_par_dirichlet`, [20](#), [22](#), [22](#), [24](#), [26](#)  
`inspect_par_haldane`, [20](#), [22](#), [23](#), [24](#), [26](#)  
`inspect_par_multinomial`, [8](#), [13](#), [15](#), [17](#), [20](#),  
[22–24](#), [25](#), [28](#)  
`inspect_prob`, [20](#), [26](#), [27](#)  
`inspect_true_or_false`, [10](#), [11](#), [28](#)

`length`, [6](#), [10](#), [11](#), [14](#), [17–21](#), [28](#), [29](#)

`typeof`, [6](#), [9](#), [11](#), [29](#)