Package: assertr (via r-universe)

March 8, 2025

Type Package

Title Assertive Programming for R Analysis Pipelines

Version 3.0.1

Description Provides functionality to assert conditions that have to be met so that errors in data used in analysis pipelines can fail quickly. Similar to 'stopifnot()' but more powerful, friendly, and easier for use in pipelines.

URL https://docs.ropensci.org/assertr/ (website)

https://github.com/ropensci/assertr

BugReports https://github.com/ropensci/assertr/issues

License MIT + file LICENSE

ByteCompile TRUE

Depends R (>= 3.1.0)

Imports dplyr (>= 0.7.0), MASS, methods, stats, utils, rlang (>= 0.3.0)

Suggests knitr, testthat (>= 3.0.0), magrittr, rmarkdown, tibble, covr

VignetteBuilder knitr

RoxygenNote 7.2.3

Encoding UTF-8

Config/testthat/edition 3

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

RemoteUrl https://github.com/tonyfischetti/assertr

RemoteRef HEAD

RemoteSha e484665dba582582faa343d97a09d4a097db3bb1

Contents

assert . . . 2
assertr . . . 4
assert_rows . . . 5
chaining_functions . . . 7
col_concat . . . 8
duplicates_across_cols . . . 9
generate_id . . . 10
has_all_names . . . 10
has_class . . . 11
has_only_names . . . 12
insist . . . 13
insist_rows . . . 15
in_set . . . 17
is_uniq . . . 18
maha_dist . . . 19
not_na . . . 20
num_row_NAs . . . 21
print.assertr_assert_error . . . 22
print.assertr_defect . . . 22
print.assertr_success . . . 23
print.assertr_verify_error . . . 23
success_and_error_functions . . . 24
summary.assertr_assert_error . . . 26
summary.assertr_verify_error . . . 26
verify . . . 27
within_bounds . . . 29
within_n_mads . . . 30
within_n_sds . . . 31

Index **33**

`assert` *Raises error if predicate is FALSE in any columns selected*

Description

Meant for use in a data analysis pipeline, this function will just return the data it's supplied if there are no FALSEs when the predicate is applied to every element of the columns indicated. If any element in any of the columns, when applied to the predicate, is FALSE, then this function will raise an error, effectively terminating the pipeline early.

Usage

```
assert(
  data,
  predicate,
  ...,
  success_fun = success_continue,
  error_fun = error_stop,
  skip_chain_opts = FALSE,
  obligatory = FALSE,
  defect_fun = defect_append,
  description = NA
)
```

Arguments

`data`	A data frame
`predicate`	A function that returns FALSE when violated
`...`	Comma separated list of unquoted expressions. Uses dplyr's `select` to select columns from data.
`success_fun`	Function to call if assertion passes. Defaults to returning `data`.
`error_fun`	Function to call if assertion fails. Defaults to printing a summary of all errors.
`skip_chain_opts`	If TRUE, `success_fun` and `error_fun` are used even if assertion is called within a chain.
`obligatory`	If TRUE and assertion failed the data is marked as defective. For defective data, all the following rules are handled by `defect_fun` function.
`defect_fun`	Function to call when data is defective. Defaults to skipping assertion and storing info about it in special attribute.
`description`	Custom description of the rule. Is stored in result reports and data.

Details

For examples of possible choices for the `success_fun` and `error_fun` parameters, run `help("success_and_error_functio`

Value

By default, the `data` is returned if predicate assertion is TRUE and and error is thrown if not. If a non-default `success_fun` or `error_fun` is used, the return values of these function will be returned.

Note

See `vignette("assertr")` for how to use this in context

See Also

`verify` `insist` `assert_rows` `insist_rows`

Examples

```
# returns mtcars
assert(mtcars, not_na, vs)

# return mtcars
assert(mtcars, not_na, mpg:carb)

library(magrittr)                    # for piping operator

mtcars %>%
  assert(in_set(c(0,1)), vs)
  # anything here will run

## Not run:
mtcars %>%
  assert(in_set(c(1, 2, 3, 4, 6)), carb)
  # the assertion is untrue so
  # nothing here will run
## End(Not run)
```

`assertr` *assertr: Assertive programming for R analysis pipeline.*

Description

The assertr package supplies a suite of functions designed to verify assumptions about data early in an analysis pipeline. See the assertr vignette or the documentation for more information
> `vignette("assertr")`

Details

You may also want to read the documentation for the functions that `assertr` provides:

- `assert`
- `verify`
- `insist`
- `assert_rows`
- `insist_rows`
- `not_na`
- `in_set`
- `has_all_names`
- `is_uniq`
- `num_row_NAs`
- `maha_dist`

- col_concat
- within_bounds
- within_n_sds
- within_n_mads
- success_and_error_functions
- chaining_functions

Examples

```
library(magrittr)     # for the piping operator
library(dplyr)

# this confirms that
#   - that the dataset contains more than 10 observations
#   - that the column for 'miles per gallon' (mpg) is a positive number
#   - that the column for 'miles per gallon' (mpg) does not contain a datum
#     that is outside 4 standard deviations from its mean, and
#   - that the am and vs columns (automatic/manual and v/straight engine,
#     respectively) contain 0s and 1s only
#   - each row contains at most 2 NAs
#   - each row's mahalanobis distance is within 10 median absolute deviations of
#     all the distance (for outlier detection)

mtcars %>%
  verify(nrow(.) > 10) %>%
  verify(mpg > 0) %>%
  insist(within_n_sds(4), mpg) %>%
  assert(in_set(0,1), am, vs) %>%
  assert_rows(num_row_NAs, within_bounds(0,2), everything()) %>%
  insist_rows(maha_dist, within_n_mads(10), everything()) %>%
  group_by(cyl) %>%
  summarise(avg.mpg=mean(mpg))
```

`assert_rows` *Raises error if predicate is FALSE for any row after applying row reduction function*

Description

Meant for use in a data analysis pipeline, this function applies a function to a data frame that reduces each row to a single value. Then, a predicate function is applied to each of the row reduction values. If any of these predicate applications yield FALSE, this function will raise an error, effectively terminating the pipeline early. If there are no FALSEs, this function will just return the data that it was supplied for further use in later parts of the pipeline.

Usage

```
assert_rows(
  data,
  row_reduction_fn,
  predicate,
  ...,
  success_fun = success_continue,
  error_fun = error_stop,
  skip_chain_opts = FALSE,
  obligatory = FALSE,
  defect_fun = defect_append,
  description = NA
)
```

Arguments

data	A data frame
row_reduction_fn	A function that returns a value for each row of the provided data frame
predicate	A function that returns FALSE when violated
...	Comma separated list of unquoted expressions. Uses dplyr's select to select columns from data.
success_fun	Function to call if assertion passes. Defaults to returning data.
error_fun	Function to call if assertion fails. Defaults to printing a summary of all errors.
skip_chain_opts	If TRUE, success_fun and error_fun are used even if assertion is called within a chain.
obligatory	If TRUE and assertion failed the data is marked as defective. For defective data, all the following rules are handled by defect_fun function.
defect_fun	Function to call when data is defective. Defaults to skipping assertion and storing info about it in special attribute.
description	Custom description of the rule. Is stored in result reports and data.

Details

For examples of possible choices for the success_fun and error_fun parameters, run help("success_and_error_functio

Value

By default, the data is returned if predicate assertion is TRUE and and error is thrown if not. If a non-default success_fun or error_fun is used, the return values of these function will be returned.

Note

See vignette("assertr") for how to use this in context

See Also

insist_rows assert verify insist

Examples

```
# returns mtcars
assert_rows(mtcars, num_row_NAs, within_bounds(0,2), mpg:carb)

library(magrittr)                   # for piping operator

mtcars %>%
  assert_rows(rowSums, within_bounds(0,2), vs:am)
  # anything here will run

## Not run:
mtcars %>%
  assert_rows(rowSums, within_bounds(0,1), vs:am)
  # the assertion is untrue so
  # nothing here will run
## End(Not run)
```

---

chaining_functions *Chaining functions*

Description

These functions are for starting and ending a sequence of assertr assertions and overriding the default behavior of assertr halting execution on the first error.

Usage

```
chain_start(data, store_success = FALSE)

chain_end(data, success_fun = success_continue, error_fun = error_report)
```

Arguments

data	A data frame
store_success	If TRUE each successful assertion is stored in chain.
success_fun	Function to call if assertion passes. Defaults to returning data.
error_fun	Function to call if assertion fails. Defaults to printing a summary of all errors.

Details

For more information, read the relevant section in this package's vignette using, vignette("assertr")

For examples of possible choices for the success_fun and error_fun parameters, run help("success_and_error_functio

Examples

```
library(magrittr)

mtcars %>%
  chain_start() %>%
  verify(nrow(mtcars) > 10) %>%
  verify(mpg > 0) %>%
  insist(within_n_sds(4), mpg) %>%
  assert(in_set(0,1), am, vs) %>%
  chain_end()
```

`col_concat` *Concatenate all columns of each row in data frame into a string*

Description

This function will return a vector, with the same length as the number of rows of the provided data frame. Each element of the vector will be it's corresponding row with all of its values (one for each column) "pasted" together in a string.

Usage

```
col_concat(data, sep = "")
```

Arguments

`data`	A data frame
`sep`	A string to separate the columns with (default: "")

Value

A vector of rows concatenated into strings

See Also

`paste`

Examples

```
col_concat(mtcars)

library(magrittr)            # for piping operator

# you can use "assert_rows", "is_uniq", and this function to
# check if joint duplicates (across different columns) appear
# in a data frame
## Not run:
mtcars %>%
```

```
  assert_rows(col_concat, is_uniq, mpg, hp)
  # fails because the first two rows are jointly duplicates
  # on these two columns

## End(Not run)

mtcars %>%
  assert_rows(col_concat, is_uniq, mpg, hp, wt) # ok
```

`duplicates_across_cols` *Checks if row contains at least one value duplicated in its column*

Description

This function will return a vector, with the same length as the number of rows of the provided data frame. Each element of the vector will be logical value that states if any value from the row was duplicated in its column.

Usage

```
duplicates_across_cols(data, allow.na = FALSE)
```

Arguments

`data`	A data frame
`allow.na`	TRUE if we allow NAs in data. Default FALSE.

Value

A logical vector.

See Also

`paste`

Examples

```
df <- data.frame(v1 = c(1, 1, 2, 3), v2 = c(4, 5, 5, 6))
duplicates_across_cols(df)

library(magrittr)            # for piping operator

# you can use "assert_rows", "in_set", and this function to
# check if specified variables set and all subsets are keys for the data.

correct_df <- data.frame(id = 1:5, sub_id = letters[1:5], work_id = LETTERS[1:5])
correct_df %>%
```

```
  assert_rows(duplicates_across_cols, in_set(FALSE), id, sub_id, work_id)
  # passes because each subset of correct_df variables is key

## Not run:
incorrect_df <- data.frame(id = 1:5, sub_id = letters[1:5], age = c(10, 20, 20, 15, 30))
incorrect_df %>%
  assert_rows(duplicates_across_cols, in_set(FALSE), id, sub_id, age)
  # fails because age is not key of the data (age == 20 is placed twice)

## End(Not run)
```

`generate_id` *Generates random ID string*

Description

This is used to generate id for each assertion error.

Usage

```
generate_id()
```

Details

For single assertion that checks multiple columns, each error log is stored as a separate element. We provide the ID to allow detecting which errors come from the same assertion.

`has_all_names` *Returns TRUE if data.frame or list has specified names*

Description

This function checks parent frame environment for existence of names. This is meant to be used with ‘assertr‘’s ‘verify‘ function to check for the existence of specific column names in a ‘data.frame‘ that is piped to ‘verify‘. It can also work on a non-‘data.frame‘ list.

Usage

```
has_all_names(...)
```

Arguments

`...`	A arbitrary amount of quoted names to check for

Value

TRUE if all names exist, FALSE if not

See Also

exists

Other Name verification: has_only_names()

Examples

```
verify(mtcars, has_all_names("mpg", "wt", "qsec"))

library(magrittr)   # for pipe operator

## Not run:
mtcars %>%
  verify(has_all_names("mpgg"))  # fails

## End(Not run)

mpgg <- "something"

mtcars %>%
  verify(exists("mpgg"))   # passes but big mistake

## Not run:
mtcars %>%
  verify(has_all_names("mpgg")) # correctly fails

## End(Not run)
```

---

has_class *Returns TRUE if data.frame columns have a specified class*

Description

This is meant to be used with 'assertr''s 'verify' function to check for the existence of a specific column class in a 'data.frame' that is piped to 'verify'.

Usage

```
has_class(..., class)
```

Arguments

...	An arbitrary amount of quoted column names to check for
class	Expected class for chosen columns.

Value

TRUE if all classes are correct, FALSE if not

Examples

```
verify(mtcars, has_class("mpg", "wt", class = "numeric"))

library(magrittr)   # for pipe operator

## Not run:
mtcars %>%
  verify(has_class("mpg", class = "character"))  # fails

## End(Not run)
```

---

has_only_names *Returns TRUE if data.frame or list has only the specified names*

Description

This function checks parent frame environment for a specific set of names; if more columns are present than those specified, an error is raised.

Usage

```
has_only_names(...)
```

Arguments

...	A arbitrary amount of quoted names to check for

Details

This is meant to be used with 'assertr''s 'verify' function to check for the existence of specific column names in a 'data.frame' that is piped to 'verify'. It can also work on a non-'data.frame' list.

Value

TRUE is all names exist, FALSE if not

See Also

Other Name verification: has_all_names()

Examples

```
# The last two columns names are switched in order, but all column names are
# present, so it passes.
verify(
  mtcars,
  has_only_names(c(
    "mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am",
```

```
    "carb", "gear"
  ))
)

# More than one set of character strings can be provided.
verify(
  mtcars,
  has_only_names(
    c("mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am"),
    c("carb", "gear")
  )
)

## Not run:
# The some columns are missing, so it fails.
verify(mtcars, has_only_names("mpg"))

## End(Not run)
```

`insist` *Raises error if dynamically created predicate is FALSE in any columns selected*

Description

Meant for use in a data analysis pipeline, this function applies a predicate generating function to each of the columns indicated. It will then use these predicates to check every element of those columns. If any of these predicate applications yield FALSE, this function will raise an error, effectively terminating the pipeline early. If there are no FALSES, this function will just return the data that it was supplied for further use in later parts of the pipeline.

Usage

```
insist(
  data,
  predicate_generator,
  ...,
  success_fun = success_continue,
  error_fun = error_stop,
  skip_chain_opts = FALSE,
  obligatory = FALSE,
  defect_fun = defect_append,
  description = NA
)
```

Arguments

`data`	A data frame

predicate_generator
: A function that is applied to each of the column vectors selected. This will produce, for every column, a true predicate function to be applied to every element in the column vectors selected

...
: Comma separated list of unquoted expressions. Uses dplyr's `select` to select columns from data.

success_fun
: Function to call if assertion passes. Defaults to returning `data`.

error_fun
: Function to call if assertion fails. Defaults to printing a summary of all errors.

skip_chain_opts
: If TRUE, `success_fun` and `error_fun` are used even if assertion is called within a chain.

obligatory
: If TRUE and assertion failed the data is marked as defective. For defective data, all the following rules are handled by `defect_fun` function.

defect_fun
: Function to call when data is defective. Defaults to skipping assertion and storing info about it in special attribute.

description
: Custom description of the rule. Is stored in result reports and data.

Details

For examples of possible choices for the `success_fun` and `error_fun` parameters, run `help("success_and_error_functio`

Value

By default, the `data` is returned if dynamically created predicate assertion is TRUE and and error is thrown if not. If a non-default `success_fun` or `error_fun` is used, the return values of these function will be returned.

Note

See `vignette("assertr")` for how to use this in context

See Also

`assert` `verify` `insist_rows` `assert_rows`

Examples

```
insist(iris, within_n_sds(3), Sepal.Length)   # returns iris

library(magrittr)

iris %>%
  insist(within_n_sds(4), Sepal.Length:Petal.Width)
  # anything here will run

## Not run:
iris %>%
  insist(within_n_sds(3), Sepal.Length:Petal.Width)
  # datum at index 16 of 'Sepal.Width' vector is (4.4)
```

```
  # is outside 3 standard deviations from the mean of Sepal.Width.
  # The check fails, raises a fatal error, and the pipeline
  # is terminated so nothing after this statement will run
## End(Not run)
```

`insist_rows` *Raises error if dynamically created predicate is FALSE for any row after applying row reduction function*

Description

Meant for use in a data analysis pipeline, this function applies a function to a data frame that reduces each row to a single value. Then, a predicate generating function is applied to row reduction values. It will then use these predicates to check each of the row reduction values. If any of these predicate applications yield FALSE, this function will raise an error, effectively terminating the pipeline early. If there are no FALSEs, this function will just return the data that it was supplied for further use in later parts of the pipeline.

Usage

```
insist_rows(
  data,
  row_reduction_fn,
  predicate_generator,
  ...,
  success_fun = success_continue,
  error_fun = error_stop,
  skip_chain_opts = FALSE,
  obligatory = FALSE,
  defect_fun = defect_append,
  description = NA
)
```

Arguments

`data`	A data frame
`row_reduction_fn`	A function that returns a value for each row of the provided data frame
`predicate_generator`	A function that is applied to the results of the row reduction function. This will produce, a true predicate function to be applied to every element in the vector that the row reduction function returns.
`...`	Comma separated list of unquoted expressions. Uses dplyr's `select` to select columns from data.
`success_fun`	Function to call if assertion passes. Defaults to returning `data`.

error_fun	Function to call if assertion fails. Defaults to printing a summary of all errors.
skip_chain_opts	If TRUE, success_fun and error_fun are used even if assertion is called within a chain.
obligatory	If TRUE and assertion failed the data is marked as defective. For defective data, all the following rules are handled by defect_fun function.
defect_fun	Function to call when data is defective. Defaults to skipping assertion and storing info about it in special attribute.
description	Custom description of the rule. Is stored in result reports and data.

Details

For examples of possible choices for the success_fun and error_fun parameters, run help("success_and_error_functio

Value

By default, the data is returned if dynamically created predicate assertion is TRUE and and error is thrown if not. If a non-default success_fun or error_fun is used, the return values of these function will be returned.

Note

See vignette("assertr") for how to use this in context

See Also

insist assert_rows assert verify

Examples

```
# returns mtcars
insist_rows(mtcars, maha_dist, within_n_mads(30), mpg:carb)

library(magrittr)                   # for piping operator

mtcars %>%
  insist_rows(maha_dist, within_n_mads(10), vs:am)
  # anything here will run

## Not run:
mtcars %>%
  insist_rows(maha_dist, within_n_mads(1), everything())
  # the assertion is untrue so
  # nothing here will run
## End(Not run)
```

in_set *Returns TRUE if value in set*

Description

This function returns a predicate function that will take a single value and return TRUE if the value is a member of the set of objects supplied. This doesn't actually check the membership of anything–it only returns a function that actually does the checking when called with a value. This is a convenience function meant to return a predicate function to be used in an assertr assertion. You can use the 'inverse' flag (default FALSE) to check if the arguments are NOT in the set.

Usage

```
in_set(..., allow.na = TRUE, inverse = FALSE)
```

Arguments

...	objects that make up the set
allow.na	A logical indicating whether NAs (including NaNs) should be permitted (default TRUE)
inverse	A logical indicating whether it should test if arguments are NOT in the set

Value

A function that takes one value and returns TRUE if the value is in the set defined by the arguments supplied by in_set and FALSE otherwise

See Also

%in%

Examples

```
predicate <- in_set(3,4)
predicate(4)

## is equivalent to

in_set(3,4)(3)

# inverting the function works thusly...
in_set(3, 4, inverse=TRUE)(c(5, 2, 3))
# TRUE TRUE FALSE

# the remainder of division by 2 is always 0 or 1
rem <- 10 %% 2
in_set(0,1)(rem)
```

```
## this is meant to be used as a predicate in an assert statement
assert(mtcars, in_set(3,4,5), gear)

## or in a pipeline, like this was meant for

library(magrittr)

mtcars %>%
  assert(in_set(3,4,5), gear) %>%
  assert(in_set(0,1), vs, am)
```

`is_uniq` *Returns TRUE where no elements appear more than once*

Description

This function is meant to take only a vector. It relies heavily on the `duplicated` function where it can be thought of as the inverse. Where this function differs, though–besides being only meant for one vector or column–is that it marks the first occurrence of a duplicated value as "non unique", as well.

Usage

```
is_uniq(..., allow.na = FALSE)
```

Arguments

`...`	One or more vectors to check for unique combinations of elements
`allow.na`	A logical indicating whether NAs should be preserved as missing values in the return value (FALSE) or if they should be treated just like any other value (TRUE) (default is FALSE)

Value

A vector of the same length where the corresponding element is TRUE if the element only appears once in the vector and FALSE otherwise

See Also

`duplicated`

Examples

```
is_uniq(1:10)
is_uniq(c(1,1,2,3), c(1,2,2,3))

## Not run:
# returns FALSE where a "5" appears
```

```
is_uniq(c(1:10, 5))

## End(Not run)

library(magrittr)

## Not run:
# this fails 4 times
mtcars %>% assert(is_uniq, qsec)

## End(Not run)

# to use the version of this function that allows NAs in `assert`,
# you can use a lambda/anonymous function like so:

mtcars %>%
  assert(function(x){is_uniq(x, allow.na=TRUE)}, qsec)
```

---

maha_dist *Computes mahalanobis distance for each row of data frame*

Description

This function will return a vector, with the same length as the number of rows of the provided data frame, corresponding to the average mahalanobis distances of each row from the whole data set.

Usage

```
maha_dist(data, keep.NA = TRUE, robust = FALSE, stringsAsFactors = FALSE)
```

Arguments

data	A data frame
keep.NA	Ensure that every row with missing data remains NA in the output? TRUE by default.
robust	Attempt to compute mahalanobis distance based on robust covariance matrix? FALSE by default
stringsAsFactors	Convert non-factor string columns into factors? FALSE by default

Details

This is useful for finding anomalous observations, row-wise.

It will convert any categorical variables in the data frame into numerics as long as they are factors. For example, in order for a character column to be used as a component in the distance calculations, it must either be a factor, or converted to a factor by using the `stringsAsFactors` parameter.

Value

A vector of observation-wise mahalanobis distances.

See Also

insist_rows

Examples

```
maha_dist(mtcars)

maha_dist(iris, robust=TRUE)


library(magrittr)            # for piping operator
library(dplyr)               # for "everything()" function

# using every column from mtcars, compute mahalanobis distance
# for each observation, and ensure that each distance is within 10
# median absolute deviations from the median
mtcars %>%
  insist_rows(maha_dist, within_n_mads(10), everything())
  ## anything here will run
```

---

not_na *Returns TRUE if value is not NA*

Description

This is the inverse of is.na. This is a convenience function meant to be used as a predicate in an assertr assertion.

Usage

```
not_na(x, allow.NaN = FALSE)
```

Arguments

x	A R object that supports is.na an is.nan
allow.NaN	A logical indicating whether NaNs should be allowed (default FALSE)

Value

A vector of the same length that is TRUE when the element is not NA and FALSE otherwise

See Also

is.na is.nan

Examples

```
not_na(NA)
not_na(2.8)
not_na("tree")
not_na(c(1, 2, NA, 4))
```

`num_row_NAs` *Counts number of NAs in each row*

Description

This function will return a vector, with the same length as the number of rows of the provided data frame, corresponding to the number of missing values in each row

Usage

```
num_row_NAs(data, allow.NaN = FALSE)
```

Arguments

`data`	A data frame
`allow.NaN`	Treat NaN like NA (by counting it). FALSE by default

Value

A vector of number of missing values in each row

See Also

`is.na` `is.nan` `not_na`

Examples

```
num_row_NAs(mtcars)


library(magrittr)            # for piping operator
library(dplyr)               # for "everything()" function

# using every column from mtcars, make sure there are at most
# 2 NAs in each row. If there are any more than two, error out
mtcars %>%
  assert_rows(num_row_NAs, within_bounds(0,2), everything())
  ## anything here will run
```

print.assertr_assert_error *Printing assertr's assert errors*

Description

'print' method for class "assertr_assert_error" This prints the error message and the entire two-column 'data.frame' holding the indexes and values of the offending data.

Usage

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

Arguments

x	An assertr_assert_error object
...	Further arguments passed to or from other methods

See Also

summary.assertr_assert_error

print.assertr_defect *Printing assertr's defect*

Description

'print' method for class "assertr_defect" This prints the defect message along with columns that were checked.

Usage

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

Arguments

x	An assertr_defect object
...	Further arguments passed to or from other methods

print.assertr_success *Printing assertr's success*

Description

'print' method for class "assertr_success" This prints the success message along with columns that were checked.

Usage

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

Arguments

x	An assertr_success object
...	Further arguments passed to or from other methods

print.assertr_verify_error *Printing assertr's verify errors*

Description

'summary' method for class "assertr_verify_error"

Usage

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

Arguments

x	An assertr_verify_error object.
...	Further arguments passed to or from other methods

See Also

summary.assertr_verify_error

success_and_error_functions
Success and error functions

Description

The behavior of functions like `assert`, `assert_rows`, `insist`, `insist_rows`, `verify` when the assertion passes or fails is configurable via the `success_fun` and `error_fun` parameters, respectively. The `success_fun` parameter takes a function that takes the data passed to the assertion function as a parameter. You can write your own success handler function, but there are a few provided by this package:

- `success_continue` - just returns the data that was passed into the assertion function
- `success_logical` - returns TRUE
- `success_append` - returns the data that was passed into the assertion function but also stores basic information about verification result
- `success_report` - When success results are stored, and each verification ended up with success prints summary of all successful validations
- `success_df_return` - When success results are stored, and each verification ended up with success prints data.frame with verification results

The `error_fun` parameter takes a function that takes the data passed to the assertion function as a parameter. You can write your own error handler function, but there are a few provided by this package:

- `error_stop` - Prints a summary of the errors and halts execution.
- `error_report` - Prints all the information available about the errors in a "tidy" `data.frame` (including information such as the name of the predicate used, the offending value, etc...) and halts execution.
- `error_append` - Attaches the errors to a special attribute of `data` and returns the data. This is chiefly to allow assertr errors to be accumulated in a pipeline so that all assertions can have a chance to be checked and so that all the errors can be displayed at the end of the chain.
- `error_return` - Returns the raw object containing all the errors
- `error_df_return` - Returns a "tidy" `data.frame` containing all the errors, including informations such as the name of the predicate used, the offending value, etc...
- `error_logical` - returns FALSE
- `just_warn` - Prints a summary of the errors but does not halt execution, it just issues a warning.
- `warn_report` - Prints all the information available about the errors but does not halt execution, it just issues a warning.
- `defect_report` - For single rule and defective data it displays short info about skipping current assertion. For `chain_end` sums up all skipped rules for defective data.
- `defect_df_return` - For single rule and defective data it returns info data.frame about skipping current assertion. For `chain_end` returns all skipped rules info data.frame for defective data.

You may find the third type of data verification result. In a scenario when validation rule was obligatory (obligatory = TRUE) in order to execute the following ones we may want to skip them and register that fact. In order to do this there are three callbacks reacting to defective data:

- `defect_report` - For single rule and defective data it displays short info about skipping current assertion.
- `defect_df_return` - For single rule and defective data it returns info data.frame about skipping current assertion.
- `defect_append` - Appends info about skipped rule due to data defect into one of data attributes. Rules skipped on defective data, or its summary, can be returned with proper error_fun callback in `chain_end`.

Usage

```
success_logical(data, ...)

success_continue(data, ...)

success_append(data, ...)

success_report(data, ...)

success_df_return(data, ...)

error_stop(errors, data = NULL, warn = FALSE, ...)

just_warn(errors, data = NULL)

error_report(errors, data = NULL, warn = FALSE, ...)

warn_report(errors, data = NULL)

error_append(errors, data = NULL)

warning_append(errors, data = NULL)

error_return(errors, data = NULL)

error_df_return(errors, data = NULL)

error_logical(errors, data = NULL, ...)

defect_append(errors, data, ...)

defect_report(errors, data, ...)

defect_df_return(errors, data, ...)
```

Arguments

`data`	A data frame
`...`	Further arguments passed to or from other methods
`errors`	A list of objects of class `assertr_errors`
`warn`	If TRUE, assertr will issue a warning instead of an error

---

`summary.assertr_assert_error`

Summarizing assertr's assert errors

Description

'summary' method for class "assertr_assert_error" This prints the error message and the first five rows of the two-column 'data.frame' holding the indexes and values of the offending data.

Usage

```
## S3 method for class 'assertr_assert_error'
summary(object, ...)
```

Arguments

`object`	An assertr_assert_error object
`...`	Additional arguments affecting the summary produced

See Also

`print.assertr_assert_error`

`summary.assertr_verify_error`

Summarizing assertr's verify errors

Description

'summary' method for class "assertr_verify_error"

Usage

```
## S3 method for class 'assertr_verify_error'
summary(object, ...)
```

Arguments

object	An assertr_verify_error object
...	Additional arguments affecting the summary produced

See Also

print.assertr_verify_error

---

verify *Raises error if expression is FALSE anywhere*

Description

Meant for use in a data analysis pipeline, this function will just return the data it's supplied if all the logicals in the expression supplied are TRUE. If at least one is FALSE, this function will raise a error, effectively terminating the pipeline early

Usage

```
verify(
  data,
  expr,
  success_fun = success_continue,
  error_fun = error_stop,
  skip_chain_opts = FALSE,
  obligatory = FALSE,
  defect_fun = defect_append,
  description = NA
)
```

Arguments

data	A data frame, list, or environment
expr	A logical expression
success_fun	Function to call if assertion passes. Defaults to returning data.
error_fun	Function to call if assertion fails. Defaults to printing a summary of all errors.
skip_chain_opts	If TRUE, success_fun and error_fun are used even if assertion is called within a chain.
obligatory	If TRUE and assertion failed the data is marked as defective. For defective data, all the following rules are handled by defect_fun function.
defect_fun	Function to call when data is defective. Defaults to skipping assertion and storing info about it in special attribute.
description	Custom description of the rule. Is stored in result reports and data.

Details

For examples of possible choices for the `success_fun` and `error_fun` parameters, run `help("success_and_error_functio`

Value

By default, the `data` is returned if predicate assertion is TRUE and and error is thrown if not. If a non-default `success_fun` or `error_fun` is used, the return values of these function will be returned.

Note

See `vignette("assertr")` for how to use this in context

See Also

`assert` `insist`

Examples

```
verify(mtcars, drat > 2)     # returns mtcars
## Not run:
verify(mtcars, drat > 3)     # produces error
## End(Not run)


library(magrittr)            # for piping operator

## Not run:
mtcars %>%
  verify(drat > 3) %>%
  # anything here will not run
## End(Not run)

mtcars %>%
  verify(nrow(mtcars) > 2)
  # anything here will run

alist <- list(a=c(1,2,3), b=c(4,5,6))
verify(alist, length(a) > 2)
verify(alist, length(a) > 2 && length(b) > 2)
verify(alist, a > 0 & b > 2)

## Not run:
alist %>%
  verify(alist, length(a) > 5)
  # nothing here will run
## End(Not run)
```

within_bounds *Creates bounds checking predicate*

Description

This function returns a predicate function that will take a numeric value or vector and return TRUE if the value(s) is/are within the bounds set. This does not actually check the bounds of anything– it only returns a function that actually does the checking when called with a number. This is a convenience function meant to return a predicate function to be used in an `assertr` assertion.

Usage

```
within_bounds(
  lower.bound,
  upper.bound,
  include.lower = TRUE,
  include.upper = TRUE,
  allow.na = TRUE,
  check.class = TRUE
)
```

Arguments

`lower.bound`	The lowest permitted value
`upper.bound`	The upper permitted value
`include.lower`	A logical indicating whether lower bound should be inclusive (default TRUE)
`include.upper`	A logical indicating whether upprt bound should be inclusive (default TRUE)
`allow.na`	A logical indicating whether NAs (including NaNs) should be permitted (default TRUE)
`check.class`	Should the class of the `lower.bound`, `upper_bound`, and the input to the returned function be checked to be numeric or of the same class? If `FALSE`, the comparison may have unexpected results.

Value

A function that takes numeric value or numeric vactor and returns TRUE if the value(s) is/are within the bounds defined by the arguments supplied by `within_bounds` and FALSE otherwise

Examples

```
predicate <- within_bounds(3,4)
predicate(pi)

## is equivalent to

within_bounds(3,4)(pi)
```

```
# a correlation coefficient must always be between 0 and 1
coeff <- cor.test(c(1,2,3), c(.5, 2.4, 4))[["estimate"]]
within_bounds(0,1)(coeff)

## check for positive number
positivep <- within_bounds(0, Inf, include.lower=FALSE)

## this is meant to be used as a predicate in an assert statement
assert(mtcars, within_bounds(4,8), cyl)

## or in a pipeline

library(magrittr)

mtcars %>%
  assert(within_bounds(4,8), cyl)
```

`within_n_mads` *Return a function to create robust z-score checking predicate*

Description

This function takes one argument, the number of median absolute deviations within which to accept a particular data point. This is generally more useful than its sister function `within_n_sds` because it is more robust to the presence of outliers. It is therefore better suited to identify potentially erroneous data points.

Usage

```
within_n_mads(n, ...)
```

Arguments

`n`	The number of median absolute deviations from the median within which to accept a datum
`...`	Additional arguments to be passed to `within_bounds`

Details

As an example, if '2' is passed into this function, this will return a function that takes a vector and figures out the bounds of two median absolute deviations (MADs) from the median. That function will then return a `within_bounds` function that can then be applied to a single datum. If the datum is within two MADs of the median of the vector given to the function returned by this function, it will return TRUE. If not, FALSE.

This function isn't meant to be used on its own, although it can. Rather, this function is meant to be used with the `insist` function to search for potentially erroneous data points in a data set.

Value

A function that takes a vector and returns a within_bounds predicate based on the MAD of that vector.

See Also

within_n_sds

Examples

```
test.vector <- rnorm(100, mean=100, sd=20)

within.one.mad <- within_n_mads(1)
custom.bounds.checker <- within.one.mad(test.vector)
custom.bounds.checker(105)     # returns TRUE
custom.bounds.checker(40)      # returns FALSE

# same as
within_n_mads(1)(test.vector)(40)    # returns FALSE

within_n_mads(2)(test.vector)(as.numeric(NA))  # returns TRUE
# because, by default, within_bounds() will accept
# NA values. If we want to reject NAs, we have to
# provide extra arguments to this function
within_n_mads(2, allow.na=FALSE)(test.vector)(as.numeric(NA))  # returns FALSE

# or in a pipeline, like this was meant for

library(magrittr)

iris %>%
  insist(within_n_mads(5), Sepal.Length)
```

---

within_n_sds *Return a function to create z-score checking predicate*

Description

This function takes one argument, the number of standard deviations within which to accept a particular data point.

Usage

```
within_n_sds(n, ...)
```

Arguments

n	The number of standard deviations from the mean within which to accept a datum
...	Additional arguments to be passed to within_bounds

Details

As an example, if '2' is passed into this function, this will return a function that takes a vector and figures out the bounds of two standard deviations from the mean. That function will then return a within_bounds function that can then be applied to a single datum. If the datum is within two standard deviations of the mean of the vector given to the function returned by this function, it will return TRUE. If not, FALSE.

This function isn't meant to be used on its own, although it can. Rather, this function is meant to be used with the insist function to search for potentially erroneous data points in a data set.

Value

A function that takes a vector and returns a within_bounds predicate based on the standard deviation of that vector.

See Also

within_n_mads

Examples

```
test.vector <- rnorm(100, mean=100, sd=20)

within.one.sd <- within_n_sds(1)
custom.bounds.checker <- within.one.sd(test.vector)
custom.bounds.checker(105)     # returns TRUE
custom.bounds.checker(40)      # returns FALSE

# same as
within_n_sds(1)(test.vector)(40)    # returns FALSE

within_n_sds(2)(test.vector)(as.numeric(NA))  # returns TRUE
# because, by default, within_bounds() will accept
# NA values. If we want to reject NAs, we have to
# provide extra arguments to this function
within_n_sds(2, allow.na=FALSE)(test.vector)(as.numeric(NA))  # returns FALSE

# or in a pipeline, like this was meant for

library(magrittr)

iris %>%
  insist(within_n_sds(5), Sepal.Length)
```

Index

∗ **Name verification**
    has_all_names, 10
    has_only_names, 12
%in%, *17*

assert, 2, *4*, *7*, *14*, *16*, *28*
assert_rows, *3, 4*, 5, *14*, *16*
assertr, 4, *17*, *20*, *29*
assertr-package *(*assertr*)*, 4

chain_end *(*chaining_functions*)*, 7
chain_start *(*chaining_functions*)*, 7
chaining_functions, *5*, 7
col_concat, *5*, 8

defect_append
        *(*success_and_error_functions*)*, 24
defect_df_return
        *(*success_and_error_functions*)*, 24
defect_report
        *(*success_and_error_functions*)*, 24
duplicated, *18*
duplicates_across_cols, 9

error_append
        *(*success_and_error_functions*)*, 24
error_df_return
        *(*success_and_error_functions*)*, 24
error_logical
        *(*success_and_error_functions*)*, 24
error_report
        *(*success_and_error_functions*)*, 24
error_return
        *(*success_and_error_functions*)*, 24
error_stop
        *(*success_and_error_functions*)*, 24
exists, *11*

generate_id, 10

has_all_names, *4*, 10, *12*
has_class, 11
has_only_names, *11*, 12

in_set, *4*, 17
insist, *3, 4*, *7*, 13, *16*, *28*, *30*, *32*
insist_rows, *3, 4*, *7*, *14*, 15, *20*
is.na, *20, 21*
is.nan, *20, 21*
is_uniq, *4*, 18

just_warn
        *(*success_and_error_functions*)*, 24

maha_dist, *4*, 19

not_na, *4*, 20, *21*
num_row_NAs, *4*, 21

paste, *8, 9*
print.assertr_assert_error, 22, *26*
print.assertr_defect, 22
print.assertr_success, 23
print.assertr_verify_error, 23, *27*

success_and_error_functions, *5*, 24
success_append
        *(*success_and_error_functions*)*, 24

success_continue
(success_and_error_functions), 24
success_df_return
(success_and_error_functions), 24
success_logical
(success_and_error_functions), 24
success_report
(success_and_error_functions), 24
summary.assertr_assert_error, *22*, 26
summary.assertr_verify_error, *23*, 26

verify, *3, 4*, *7*, *14*, *16*, 27

warn_report
(success_and_error_functions), 24
warning_append
(success_and_error_functions), 24
within_bounds, *5*, 29, *30–32*
within_n_mads, *5*, 30, *32*
within_n_sds, *5*, *30, 31*, 31