Package ‘sanityTracker’

October 14, 2022

Type Package

Title Keeps Track of all Performed Sanity Checks

Version 0.1.0

Date 2020-04-14

Maintainer Marsel Scheer <scheer@freescience.de>

Description During the preparation of data set(s) one usually performs some sanity checks. The idea is that irrespective of where the checks are performed, they are centralized by this package in order to list all at once with examples if a check failed.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Imports data.table (>= 1.12.2), checkmate (>= 2.0.0)

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/MarselScheer/sanityTracker

BugReports https://github.com/MarselScheer/sanityTracker/issues

NeedsCompilation no

Author Marsel Scheer [aut, cre]

Repository CRAN

Date/Publication 2020-04-22 16:12:07 UTC

R topics documented:

`.add_sanity_check` *Adds a sanity check to the list of already performed sanity checks*

Description

NOTE the also add_sanity_check calls this function, the parameters are documented in add_sanity_check because that function gets exported.

Usage

```
.add_sanity_check(
  fail_vec,
  description,
  counter_meas,
  data,
  data_name,
  example_size,
  param_name,
  call,
  fail_callback,
  .fail_vec_str,
  .generated_desc
)
```

Arguments

`fail_vec`	see add_sanity_check
`description`	see add_sanity_check
`counter_meas`	see add_sanity_check
`data`	see add_sanity_check
`data_name`	see add_sanity_check
`example_size`	see add_sanity_check

`param_name`	see add_sanity_check
`call`	see add_sanity_check
`fail_callback`	see add_sanity_check
`.fail_vec_str`	should capture what was used originally for `fail_vec`.
`.generated_desc`	for convenience functions like sc_col_elements to provide additional information about the check.

Value

see add_sanity_check

`add_sanity_check` *Adds a sanity check to the list of already performed sanity checks*

Description

Adds a sanity check to the list of already performed sanity checks

Usage

```
add_sanity_check(
  fail_vec,
  description = "-",
  counter_meas = "-",
  data,
  data_name = checkmate::vname(x = data),
  example_size = 3,
  param_name = "-",
  call = h_deparsed_sys_call(which = -3),
  fail_callback
)
```

Arguments

`fail_vec`	logical vector where `TRUE` indicates that a fail has happend
`description`	(optional) of the sanity check. default is "-".
`counter_meas`	(optional) description of the counter measures that were applied to correct the problems. default is "-".
`data`	(optional) where the fails were found. Is used to store examples of failures. default is "-".
`data_name`	(optional) name of the data set that was used. defaults is the name of the object passed to data.
`example_size`	(optional) number failures to be extracted from the object passed to `data`. By default 3 random examples are extracted.

`param_name`	(optional) name of the parameter(s) that is used. This may be helpful for filtering the table of all performed sanity checks.
`call`	(optional) by default tracks the function that called add_sanity_check.
`fail_callback`	(optional) user-defined function that is called if any element of `fail_vec` is `TRUE`. This is helpful if an additional warning or error should be thrown or maybe a log-entry should be created.

Value

a list with three elements

entry_sanity_table invisibly the sanity check that is stored internally with the other sanity checks

fail_vec `fail_vec` as passed over to this function

fail TRUE if any element of fail is TRUE. Otherwise FALSE.

All performed sanity checks can be fetched via get_sanity_checks

Examples

```
d <- data.frame(person_id = 1:4, bmi = c(18,23,-1,35), age = 31:34)
dummy_call <- function(x) {
  add_sanity_check(
    x$bmi < 15,
    description = "bmi above 15",
    counter_meas = "none",
    data = x,
    param_name = "bmi")
  add_sanity_check(
    x$bmi > 30,
    description = "bmi below 30",
    counter_meas = "none")
}
dummy_call(x = d)
get_sanity_checks()
add_sanity_check(
   d$bmi < 15,
   description = "bmi above 15",
   fail_callback = warning)
```

`clear_sanity_checks` *Removes all tracked sanity checks*

Description

Removes all tracked sanity checks

Usage

```
clear_sanity_checks()
```

`get_sanity_checks` *Returns all performed sanity checks*

Description

Returns all performed sanity checks

Usage

```
get_sanity_checks()
```

Value

all sanity checks, i.e. a data.table with the following column

description character that was provided by the user through the parameter `description`

additional_desc character that provides additional information about the check that was generated by the convenience functions

data_name name of the data set that passed to the function that performed the sanity check. This can also be specified by the user

n a logical vector is the basis of all sanity checks. This is length of the logical vector that was used, which in general is the number of rows of the table that was checked

n_fail how often the logical vector was TRUE

n_na how often the logical vector was NA

counter_meas character provided by the user about how a fail will be addressed. Note that this never happens inside a function of `sanityTracker` but is realized by the user after the check was performed. It is only for documentation when the results of the checks are displayed.

fail_vec_str this captures how the actual logical vector of fails was build

param_name usually generated by the convenience functions and it also may be a composition of more than one parameter name. However this parameter could also have been provided by the user

call character of the call where the sanity check happend

example if a check failed and the table is available, then some examples of rows that lead to the fail are stored here

See Also

add_sanity_check

h_add_sanity_check *Wrapper for add_sanity_check for internal use*

Description

The convenience function usually provide some defaults like description that can be overwritten by the user through the ... argument of the convenience function. This function manages to set those values that were NOT overwritten by the user through the ... argument and then call add_sanity_check.

Usage

```
h_add_sanity_check(
  ellipsis,
  fail_vec,
  .generated_desc,
  data,
  data_name = "",
  param_name = "",
  call = h_deparsed_sys_call(which = -2),
  .fail_vec_str = checkmate::vname(x = fail_vec)
)
```

Arguments

`ellipsis`	usually list(...) of the function that calls this function. It contains the parameters defined by the user for add_sanity_check.
`fail_vec`	logical vector where `TRUE` indicates that a fail has happend
`.generated_desc`	will be passed to .add_sanity_check if ellipsis does not contain a element with name 'description'
`data`	will be passed to .add_sanity_check if ellipsis does not contain a element with name 'data'
`data_name`	will be passed to .add_sanity_check if ellipsis does not contain a element with name 'data_name'
`param_name`	will be passed to .add_sanity_check if ellipsis does not contain a element with name 'param_name'
`call`	will be passed to .add_sanity_check if ellipsis does not contain a element with name 'call'
`.fail_vec_str`	usually not used by the user. Captures what was passed to `fail_vec`.

Value

see return value of add_sanity_check

Examples

```
d <- data.frame(type = letters[1:4], nmb = 1:4)
# h_add_sanity_check is used on sc_col_elements()
sc_col_elements(object = d, col = "type", feasible_elements = letters[2:4])
get_sanity_checks()
```

`h_collapse_char_vec` *Collapse a vector of characters to a string with separators*

Description

Collapse a vector of characters to a string with separators

Usage

```
h_collapse_char_vec(v, collapse = ", ", qoute = "'")
```

Arguments

`v`	vector of chars to be collapsed
`collapse`	character that separates the elements in the returned object
`qoute`	character that surronds every element in `v` in the returned object

Value

collapsed version of `v`

Examples

```
cat(sanityTracker:::h_collapse_char_vec(v = letters[1:4]))
```

`h_complete_list` *Extends a list with an named element if the element does not exist*

Description

Extends a list with an named element if the element does not exist

Usage

```
h_complete_list(ell, name, value)
```

Arguments

`ell`	list to be extended (usually an ellipsis as list(...))
`name`	character with the name for the element to be added
`value`	value that will be stored in `ell[[el_name]]`

Value

if `ell` already contained the element `name`, then `ell` is returned without being modified. Otherwise, `ell` is returned extended by a new element with name `name` and value `value`.

Examples

```
ell <- list(a = 1, b = 2)
sanityTracker:::h_complete_list(ell = ell, name = "a", value = 100)
sanityTracker:::h_complete_list(ell = ell, name = "d", value = Inf)
```

`h_deparsed_sys_call` *Simply converts a call into a character*

Description

Simply converts a call into a character

Usage

```
h_deparsed_sys_call(which)
```

Arguments

`which`	see sys.call. However the function bounds it by the number of encolsing environments.

Value

the call of the corresponding environment as character

sc_cols_bounded — *Checks that all elements from the specified columns are in a certain range*

Description

Checks that all elements from the specified columns are in a certain range

Usage

```
sc_cols_bounded(object, cols, rule = "(-Inf, Inf)", ...)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters of columns that are checked against the specified range
`rule`	check as two numbers separated by a comma, enclosed by square brackets (endpoint included) or parentheses (endpoint excluded). For example, "[0, 3)" results in all(x >= 0 & x < 3). The lower and upper bound may be omitted which is the equivalent of a negative or positive infinite bound, respectively. By definition [0,] contains Inf, while [0,) does not. The same holds for the left (lower) boundary and -Inf. This explanation was copied from `checkmate::qtest`. That function is also the backbone of the this function.
`...`	further parameters that are passed to add_sanity_check.

Value

list of logical vectors where TRUE indicates where the check failed. Every list entry represents one of the columns specified in cols. This might be helpful if one wants to apply a counter-measure

Examples

```
dummy_call <- function(x) {
  sc_cols_bounded(object = iris, cols = c("Sepal.Length", "Petal.Length"),
    rule = "[1, 7.9)")
}
dummy_call(x = d)
get_sanity_checks()
```

sc_cols_bounded_above *Checks that all elements from the given columns are below a certain number*

Description

Checks that all elements from the given columns are below a certain number

Usage

```
sc_cols_bounded_above(
  object,
  cols,
  upper_bound,
  include_upper_bound = TRUE,
  ...
)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters of columns that are checked against the specified range
`upper_bound`	elements of the specified columns must be below this bound
`include_upper_bound`	if TRUE (default), elements are allowed to be equal to the `upper_bound`
`...`	further parameters that are passed to add_sanity_check.

Value

list of logical vectors where TRUE indicates where the check failed. Every list entry represents one of the columns specified in cols. This might be helpful if one wants to apply a counter-measure

sc_cols_bounded_below *Checks that all elements from the given columns are above a certain number*

Description

Checks that all elements from the given columns are above a certain number

Usage

```
sc_cols_bounded_below(
  object,
  cols,
  lower_bound,
  include_lower_bound = TRUE,
  ...
)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters of columns that are checked against the specified range
`lower_bound`	elements of the specified columns must be above this bound
`include_lower_bound`	if TRUE (default), elements are allowed to be equal to the `lower_bound`
`...`	further parameters that are passed to add_sanity_check.

Value

list of logical vectors where TRUE indicates where the check failed. Every list entry represents one of the columns specified in cols. This might be helpful if one wants to apply a counter-measure

Examples

```
d <- data.frame(a = c(0, 0.2, 3, Inf), b = c(1:4))
dummy_call <- function(x) {
  sc_cols_bounded_below(
    object = d, cols = c("a", "b"),
    lower_bound = 0.2,
    include_lower_bound = FALSE,
    description = "Measurements are expected to be bounded from below")
}
dummy_call(x = d)
get_sanity_checks()
```

`sc_cols_non_NA` *Checks that all elements from the specified columns are not NA*

Description

Checks that all elements from the specified columns are not NA

Usage

```
sc_cols_non_NA(object, cols = names(object), ..., unk_cols_callback = stop)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters of columns that are checked for NAs
`...`	further parameters that are passed to add_sanity_check.
`unk_cols_callback`	user-defined function that is called if some of the `cols` are not contained in the `object`. This is helpful if an additional warning or error should be thrown or maybe a log-entry should be created. Default is the function `stop`

Value

a list where every element is an object returned by add_sanity_check for each column specified in `cols` that exists in `object`

Examples

```
iris[c(1,3,5,7,9), 1] <- NA
dummy_call <- function(x) {
  sc_cols_non_NA(object = iris, description = "No NAs expected in iris")
}
dummy_call(x = iris)
get_sanity_checks()
```

`sc_cols_positive` *Checks that all elements from the specified columns are positive*

Description

Checks that all elements from the specified columns are positive

Usage

```
sc_cols_positive(object, cols, zero_feasible = TRUE, ...)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters of columns that are checked against the specified range
`zero_feasible`	if zero is in the range or not
`...`	further parameters that are passed to add_sanity_check.

Value

list of logical vectors where TRUE indicates where the check failed. Every list entry represents one of the columns specified in cols. This might be helpful if one wants to apply a counter-measure.

Examples

```
d <- data.frame(a = c(0, 0.2, 3, Inf), b = c(1:4))
dummy_call <- function(x) {
  sc_cols_positive(d, cols = c("a", "b"), zero_feasible = FALSE,
    description = "Measurements are expected to be positive")
}
dummy_call(x = d)
get_sanity_checks()
```

---

sc_cols_unique — *Checks that the combination of the specified columns is unique*

Description

Checks that the combination of the specified columns is unique

Usage

```
sc_cols_unique(object, cols = names(object), ...)
```

Arguments

`object`	table with a columns specified by `cols`
`cols`	vector of characters which combination is checked to be unique
`...`	further parameters that are passed to add_sanity_check.

Value

see return object of add_sanity_check. Note that if a combination appears 3 times, then n_fail will increased by 3.

Examples

```
dummy_call <- function(x) {
  sc_cols_unique(
    object = x,
    cols = c("Species", "Sepal.Length",
             "Sepal.Width", "Petal.Length"))
}
dummy_call(x = iris)
get_sanity_checks()
get_sanity_checks()[["example"]]
```

`sc_col_elements` *Checks that the elements of a column belong to a certain set*

Description

Checks that the elements of a column belong to a certain set

Usage

```
sc_col_elements(object, col, feasible_elements, ...)
```

Arguments

`object`	table with a column specified by `col`
`col`	name as a character of the column which is checked
`feasible_elements`	vector with characters that are feasible for `col`. Note that an element that is NA it is always counted as a fail if `feasible_elements` does not explicitly contains NA.
`...`	further parameters that are passed to add_sanity_check.

Value

see return object of add_sanity_check

Examples

```
d <- data.frame(type = letters[1:4], nmb = 1:4)
dummy_call <- function(x) {
  sc_col_elements(object = d, col = "type", feasible_elements = letters[2:4])
}
dummy_call(x = d)
get_sanity_checks()
```

`sc_left_join` *Performs various checks after a left-join was performed*

Description

One check is that no rows were duplicated during merge and the other check is that no columns were duplicated during merge.

Usage

```
sc_left_join(joined, left, right, by, ..., find_nonunique_key = TRUE)
```

Arguments

`joined`	the result of the left-join
`left`	the left table used in the left-join
`right`	the right table used in the left-join
`by`	the variables used for the left-join
`...`	further parameters that are passed to add_sanity_check.
`find_nonunique_key`	if TRUE a sanity-check is performed that finds keys (defined by `by`) that are non-unique. However this can be a time-consuming step.

Value

list with two elements for the two sanity checks performed by this function. The structure of each element is as the return object of add_sanity_check.

Examples

```
ab <- data.table::data.table(a = 1:4, b = letters[1:4])
abc <- data.table::data.table(a = c(1:4, 2), b = letters[1:5], c = rnorm(5))
j <- merge(x = ab, y = abc, by = "a")
dummy_call <- function() {
  sc_left_join(joined = j, left = ab, right = abc, by = "a",
    description = "Left join outcome to main population")
}
dummy_call()
get_sanity_checks()
```

Index