Package ‘mlr3misc’

April 15, 2020

Title Helper Functions for 'mlr3'
Version 0.2.0
Description Frequently used helper functions and assertions
used in 'mlr3' and its companion packages. Comes with helper functions
for functional programming, for printing, to work with 'data.table',
as well as some generally useful 'R6' classes. This package also
supersedes the package 'BBmisc'.
License LGPL-3
BugReports https://github.com/mlr-org/mlr3misc/issues
Depends R (>= 3.1.0)
Imports backports (>= 0.1.5), checkmate, data.table, R6
Suggests bibtex, callr, evaluate, paradox, testthat
Encoding UTF-8
NeedsCompilation yes
RoxygenNote 7.1.0
Author Michel Lang [cre, aut] (<https://orcid.org/0000-0001-9754-0393>),
Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>)
Maintainer Michel Lang <michellang@gmail.com>
Repository CRAN
Date/Publication 2020-04-15 05:50:12 UTC

R topics documented:

mlr3misc-package ........................................... 3
as_factor .............................................. 3
as_short_string ........................................ 4
check_packages_installed .............................. 5
chunk_vector ........................................... 6
cite_bib ............................................... 7
Description

Frequently used helper functions and assertions used in 'mlr3' and its companion packages. Comes with helper functions for functional programming, for printing, to work with 'data.table', as well as some generally useful 'R6' classes. This package also supersedes the package 'BBmisc'.

Author(s)

Maintainer: Michel Lang <michellang@gmail.com> (ORCID)

Authors:

• Patrick Schratz <patrick.schratz@gmail.com> (ORCID)

See Also

Useful links:

• https://mlr3misc.mlr-org.com
• https://github.com/mlr-org/mlr3misc
• Report bugs at https://github.com/mlr-org/mlr3misc/issues

as_factor

Convert to Factor

Description

Converts a vector to a factor() and ensures that levels are in the order of the provided levels.

Usage

as_factor(x, levels, ordered = is.ordered(x))

Arguments

x (atomic vector())
Vector to convert to factor.

levels (character())
Levels of the new factor.

ordered (logical(1))
If TRUE, create an ordered factor.
as_short_string

Value

(factor()).

Examples

```
x = factor(c("a", "b"))
y = factor(c("a", "b"), levels = c("b", "a"))

# x with the level order of y
as_factor(x, levels(y))

# y with the level order of x
as_factor(y, levels(x))
```

Description

This function is intended to be convert any R object to a short descriptive string, e.g. in `base::print()` functions.

The following rules apply:

- if `x` is `atomic()` with length 0 or 1: printed as-is.
- if `x` is `atomic()` with length greater than 1, `x` is collapsed with ",", and the resulting string is truncated to `trunc_width` characters.
- if `x` is an expression: converted to character.
- Otherwise: the class is printed.

If `x` is a list, the above rules are applied (non-recursively) to its elements.

Usage

```
as_short_string(x, width = 30L, num_format = "%4g")
```

Arguments

- `x` (any)
  Arbitrary object.
- `width` (integer(1))
  Truncate strings to width `width`.
- `num_format` (character(1))
  Used to format numerical scalars via `base::sprintf()`.

Value

(character(1)).
check_packages_installed

### Examples

```r
as_short_string(list(a = 1, b = NULL, "foo", c = 1:10))
```

---

**Description**

Calls `find.package()` to check if all packages are installed. Returns a

**Usage**

```r
check_packages_installed(
  pkgs,
  warn = TRUE,
  msg = "The following packages are required but not installed: %s"
)
```

**Arguments**

- **pkgs** (character())
  Packages to check.
- **warn** (logical(1))
  If TRUE, signals a warning of class "packageNotFoundWarning" about the missing packages.
- **msg** (character(1))
  Format of the warning message. Use "%s" as placeholder for the list of packages.

**Value**

(logical()) named with package names. TRUE if the respective package is installed, FALSE otherwise.

**Examples**

```r
check_packages_installed(c("mlr3misc", "foobar"), warn = FALSE)
```

# catch warning
```r
tryCatch(check_packages_installed(c("mlr3misc", "foobaaar")),
  packageNotFoundWarning = function(w) as.character(w))
```
**chunk_vector**

_CHUNK Vectors_

**Description**

Chunk atomic vectors into parts of roughly equal size. `chunk()` takes a vector length `n` and returns an integer with chunk numbers. `chunk_vector()` uses `base::split()` and `chunk()` to split an atomic vector into chunks.

**Usage**

```r
chunk_vector(x, n_chunks = NULL, chunk_size = NULL, shuffle = TRUE)
chunk(n, n_chunks = NULL, chunk_size = NULL, shuffle = TRUE)
```

**Arguments**

- `x` (vector())
  Vector to split into chunks.
- `n_chunks` (integer(1))
  Requested number of chunks. Mutually exclusive with `chunk_size` and `props`.
- `chunk_size` (integer(1))
  Requested number of elements in each chunk. Mutually exclusive with `n_chunks` and `props`.
- `shuffle` (logical(1))
  If `TRUE`, permutes the order of `x` before chunking.
- `n` (integer(1))
  Length of vector to split.

**Value**

`chunk()` returns a `integer()` of chunk indices, `chunk_vector()` a `list()` of integer vectors.

**Examples**

```r
x = 1:11

ch = chunk(length(x), n_chunks = 2)
table(ch)
split(x, ch)

chunk_vector(x, n_chunks = 2)
chunk_vector(x, n_chunks = 3, shuffle = TRUE)
```
**cite_bib**

*Cite using a Bibtex File*

**Description**

This function is called by the provided Rd macro `\cite{pkg}{key}`:

- Parses the bibtex file `references.bib` in the root directory of package `package` using `bibtex::read.bib()`.
- Extracts the entry with key `key`.
- Converts to Rd with `tools::toRd()`.

**Usage**

```r
cite_bib(package, key)
```

**Arguments**

- `package` (character(1))
  Package to read the bibtex file from.
- `key` (character(1))
  Entry of the bibtex file. If the key is "pkg::citation", the `citation()` information of the package is used instead. If the package provides multiple citation entries, a specific one can be selected by appending "::n" to the string `key` where `n` is the number of the citation entry (defaults to the first entry).

**Value**

(character(1)) Bibentry formated as Rd.

**Examples**

```r
# examplary bibtex file
path = system.file("references.bib", package = "mlr3misc")
cat(readLines(path), sep = "\n")

# bibtex entry as raw Rd
cite_bib("mlr3misc", "mlr")

# citation info as raw Rd
cite_bib("stats", "pkg::citation")
```
compat-map

Apply Functions in the spirit of 'purrr'

Description

map-like functions, similar to the ones implemented in purrr:

- map() returns the results of .f applied to .x as list. If .f is not a function, map will call [] on all elements of .x using the value of .f as index.
- imap() applies .f to each value of .x (passed as first argument) and its name (passed as second argument). If .x does not have names, a sequence along .x is passed as second argument instead.
- pmap() expects .x to be a list of vectors of equal length, and then applies .f to the first element of each vector of .x, then the second element of .x, and so on.
- map_if() applies .f to each element of .x where the predicate .p evaluates to TRUE.
- map_at() applies .f to each element of .x referenced by .at. All other elements remain unchanged.
- keep() keeps those elements of .x where predicate .p evaluates to TRUE.
- discard() discards those elements of .x where predicate .p evaluates to TRUE.
- every() is TRUE if predicate .p evaluates to TRUE for each .x.
- some() is TRUE if predicate .p evaluates to TRUE for at least one .x.
- detect() returns the first element where predicate .p evaluates to TRUE.

Additionally, the functions map(), imap() and pmap have type-safe variants with the following suffixes:

- *_lgl() returns a logical(length(.x)).
- *_int() returns a integer(length(.x)).
- *_dbl() returns a double(length(.x)).
- *_chr() returns a character(length(.x)).
- *_dtl() returns a data.table::data.table() where the results of .f are put together in an base::rbind() fashion.
- *_dtc() returns a data.table::data.table() where the results of .f are put together in an base::cbind() fashion.

Usage

map(.x, .f, ...)

map_lgl(.x, .f, ...)

map_int(.x, .f, ...)

map_dbl(.x, .f, ...)
map_chr(.x, .f, ...)
map_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
map_dtc(.x, .f, ...)
pmap(.x, .f, ...)
pmap_lgl(.x, .f, ...)
pmap_int(.x, .f, ...)
pmap_dbl(.x, .f, ...)
pmap_chr(.x, .f, ...)
pmap_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
pmap_dtc(.x, .f, ...)
imap(.x, .f, ...)
imap_lgl(.x, .f, ...)
imap_int(.x, .f, ...)
imap_dbl(.x, .f, ...)
imap_chr(.x, .f, ...)
imap_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
imap_dtc(.x, .f, ...)
keep(.x, .f, ...)
discard(.x, .p, ...)
map_if(.x, .p, .f, ...)
map_at(.x, .at, .f, ...)
every(.x, .p, ...)
some(.x, .p, ...)
detect(.x, .p, ...)
**compute_mode**

## Arguments

- **.x** (list() | atomic vector()):
- **.f** (function() | character() | integer()):
  Function to apply, or element to extract by name (if .f is character()) or position (if .f is integer()).
- **...** (any):
  Additional arguments passed down to .f or .p.
- **.fill** (logical(1)):
  Passed down to `data.table::rbindlist()`.
- **.idcol** (logical(1)):
  Passed down to `data.table::rbindlist()`.
- **.p** (function() | logical()):
  Predicate function.
- **.at** (character() | integer() | logical()):
  Index vector.

## Description

Computes the mode (most frequent value) of an atomic vector.

## Usage

```r
compute_mode(x, ties_method = "random", na_rm = TRUE)
```

## Arguments

- **x** (vector()):
- **ties_method** (character(1)):
  Handling of ties. One of "first", "last" or "random" to return the first tied value, the last tied value, or a randomly selected tied value, respectively.
- **na_rm** (logical(1)):
  If TRUE, remove missing values prior to computing the mode.

## Value

(vector(1)): mode value.

## Examples

```r
compute_mode(c(1, 1, 1, 2, 2, 3))
compute_mode(c(1, 1, 1, 2, 2, 3), ties_method = "last")
compute_mode(c(1, 1, 1, 2, 2, 3), ties_method = "random")
```
**cross_join**  
*Cross-Join for data.table*

**Description**
A safe version of `data.table::CJ()` in case a column is called sorted or unique.

**Usage**
cross_join(dots, sorted = TRUE, unique = FALSE)

**Arguments**
dots  
(named list())
Vectors to cross-join.
sorted  
(logical(1))
See `data.table::CJ()`.
unique  
(logical(1))
See `data.table::CJ()`.

**Value**
`data.table()`.

**Examples**
cross_join(dots = list(sorted = 1:3, b = letters[1:2]))

**Dictionary**  
*Key-Value Storage*

**Description**
A key-value store for `R6::R6` objects. On retrieval of an object, the following applies:

- If the object is a `R6ClassGenerator`, it is initialized with `new()`.
- If the object is a function, it is called and must return an instance of a `R6::R6` object.
- If the object is an instance of a R6 class, it is returned as-is.

Default argument required for construction can be stored alongside their constructors by passing them to `$add()`.

**S3 methods**
- `as.data.table(d)`
  Dictionary -> `data.table::data.table()`
  Converts the dictionary to a `data.table::data.table()`.
Public fields

items (environment())
Stores the items of the dictionary

Methods

Public methods:

- Dictionary$new()
- Dictionary$format()
- Dictionary$print()
- Dictionary$keys()
- Dictionary$has()
- Dictionary$get()
- Dictionary$mget()
- Dictionary$add()
- Dictionary$remove()
- Dictionary$required_args()
- Dictionary$clone()

Method new(): Construct a new Dictionary.

Usage:
Dictionary$new()

Method format(): Format object as simple string.

Usage:
Dictionary$format()

Method print(): Print object.

Usage:
Dictionary$print()

Method keys(): Returns all keys which comply to the regular expression pattern. If pattern is NULL (default), all keys are returned.

Usage:
Dictionary$keys(pattern = NULL)

Arguments:
pattern (character(1)).

Returns: character() of keys.

Method has(): Returns a logical vector with TRUE at its i-th position if the i-th key exists.

Usage:
Dictionary$has(keys)

Arguments:
keys (character()).

Returns: logical().

**Method get()**: Retrieves object with key `key` from the dictionary. Additional arguments must be named and are passed to the constructor of the stored object.

Usage:
Dictionary$get(key, ...)

Arguments:
key (character(1)).
... (any)
    Passed down to constructor.

Returns: Object with corresponding key.

**Method mget()**: Returns objects with keys `keys` in a list named with `keys`. Additional arguments must be named and are passed to the constructors of the stored objects.

Usage:
Dictionary$mget(keys, ...)

Arguments:
keys (character()).
... (any)
    Passed down to constructor.

Returns: Named list() of objects with corresponding keys.

**Method add()**: Adds object `value` to the dictionary with key `key`, potentially overwriting a previously stored item. Additional arguments in `...` must be named and are passed as default arguments to `value` during construction. The names of all additional arguments which are mandatory for construction and missing in `...` should be listed in `required_args`.

Usage:
Dictionary$add(key, value, ..., required_args = character())

Arguments:
key (character(1)).
value (any).
... (any)
    Passed down to constructor.
required_args (character()).

Returns: Dictionary.

**Method remove()**: Removes objects with from the dictionary.

Usage:
Dictionary$remove(keys)

Arguments:
keys (character())
    Keys of objects to remove.
Returns: Dictionary.

Method required_args(): Returns the names of arguments required to construct the object.

Usage:
Dictionary$required_args(key)

Arguments:
key (character(1))
  Key of object to query for required arguments.

Returns: character() of names of required arguments.

Method clone(): The objects of this class are cloneable with this method.

Usage:
Dictionary$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples

library(R6)
item1 = R6Class("Item", public = list(x = 1))
item2 = R6Class("Item", public = list(x = 2))
d = Dictionary$new()
d$add("a", item1)
d$add("b", item2)
d$add("c", item1$new())
d$keys()
d$get("a")
d$mget(c("a", "b"))

dictionary_sugar_get  A Quick Way to Initialize Objects from Dictionaries

Description

Given a Dictionary, retrieve objects with provided keys.

- dictionary_sugar_get() to retrieve a single object with key .key.
- dictionary_sugar_mget() to retrieve a list of objects with keys .keys.
- dictionary_sugar() is deprecated in favor of dictionary_sugar_get().
- If .key or .keys is missing, the dictionary itself is returned.

Arguments in ... must be named and are consumed in the following order:

1. All arguments whose names match the name of an argument of the constructor are passed to the $get() method of the Dictionary for construction.
2. All arguments whose names match the name of a parameter of the paradox::ParamSet of the constructed object are set as parameters. If there is no paradox::ParamSet in obj$param_set, this step is skipped.

3. All remaining arguments are assumed to be regular fields of the constructed R6 instance, and are assigned via <-.

Usage

dictionary_sugar_get(dict, .key, ...)
dictionary_sugar(dict, .key, ...)
dictionary_sugar_mget(dict, .keys, ...)

Arguments

dict (Dictionary).
.key (character(1))
  Key of the object to construct.
...
  (any)
  See description.
[keys (character())
  Keys of the objects to construct.

Value

R6::R6Class()

Examples

library(R6)
item = R6Class("Item", public = list(x = 0))
d = Dictionary$new()
d$add("key", item)
dictionary_sugar_get(d, "key", x = 2)

did_you_mean Suggest Alternatives

Description

Helps to suggest alternatives from a list of strings, based on the string similarity in utils::adist().

Usage

did_you_mean(str, candidates)
distinct_values

Arguments

- **str**  (character(1))
  String.

- **candidates**  (character())
  Candidate strings.

Value

(character(1)). Either a phrase suggesting one or more candidates from candidates, or an empty string if no close match is found.

Examples

```r
did_you_mean("yep", c("yes", "no"))
```

---

**distinct_values**  
Get Distinct Values

Description

Extracts the distinct values of an atomic vector, with the possibility to drop levels and remove missing values.

Usage

```r
distinct_values(x, drop = TRUE, na_rm = TRUE)
```

Arguments

- **x**  (atomic vector()).
- **drop**  :: logical(1)
  If TRUE, only returns values which are present in x. If FALSE, returns all levels for factor() and ordered(), as well as TRUE and FALSE for logical().

- **na_rm**  :: logical(1)
  If TRUE, missing values are removed from the vector of distinct values.

Value

(atomic vector()) with distinct values in no particular order.
Examples

```r
# for factors:
x = factor(c(letters[1:2], NA), levels = letters[1:3])
distinct_values(x)
distinct_values(x, na_rm = FALSE)
distinct_values(x, drop = FALSE)
distinct_values(x, drop = FALSE, na_rm = FALSE)

# for logicals:
distinct_values(TRUE, drop = FALSE)

# for numerics:
distinct_values(sample(1:3, 10, replace = TRUE))
```

---

**encapsulate**  
*Encapsulate Function Calls for Logging*

**Description**

Evaluates a function while both recording an output log and measuring the elapsed time. There are currently three different modes implemented to encapsulate a function call:

- "none": Just runs the call in the current session and measures the elapsed time. Does not keep a log, output is printed directly to the console. Works well together with `traceback()`.
- "evaluate": Uses the package `evaluate` to call the function, measure time and do the logging.
- "callr": Uses the package `callr` to call the function, measure time and do the logging. This encapsulation spawns a separate R session in which the function is called. While this comes with a considerable overhead, it also guards your session from being teared down by segfaults.

**Usage**

```r
encapsulate(
  method,  # (character(1))
  .f,     # (function())
  .args = list(),
  .opts = list(),
  .pkgs = character(),
  .seed = NA_integer_
)
```

**Arguments**

- `method` (character(1))  
  One of "none", "evaluate" or "callr".
- `f` (function())  
  Function to call.
.args (list())
Arguments passed to .f.

.opts (named list())
Options to set for the function call. Options get reset on exit.

.pkgs (character())
Packages to load (not attach).

.seed (integer(1))
Random seed to set before invoking the function call. Gets reset to the previous seed on exit.

Value
(named list()) with three fields:

- "result": the return value of .f
- "elapsed": elapsed time in seconds. Measured as proc.time() difference before/after the function call.
- "log": data.table() with columns "class" (ordered factor with levels "output", "warning" and "error") and "message" (character()).

Examples

f = function(n) {
  message("hi from f")
  if (n > 5) {
    stop("n must be <= 5")
  }
  runif(n)
}

encapsulate("none", f, list(n = 1), .seed = 1)

if (requireNamespace("evaluate", quietly = TRUE)) {
  encapsulate("evaluate", f, list(n = 1), .seed = 1)
}

if (requireNamespace("callr", quietly = TRUE)) {
  encapsulate("callr", f, list(n = 1), .seed = 1)
}

---

tenframe

Convert a Named Vector Into A data.table

Description

Returns a data.table::data.table() with two columns: The names of x (or seq_along(x) if unnamed) and the values of x.
Usage

enframe(x, name = "name", value = "value")

Arguments

x (vector())
   Vector to convert to a data.table::data.table().
name (character(1))
   Name for the first column with names.
value (character(1))
   Name for the second column with values.

Value

data.table::data.table().

Examples

x = 1:3
enframe(x)

x = set_names(1:3, letters[1:3])
enframe(x, value = "x_values")

extract_vars

Extract Variables from a Formula

Description

Given a formula() f, returns all variables used on the left-hand side and right-hand side of the formula.

Usage

extract_vars(f)

Arguments

f (formula()).

Value

(list()) with elements "lhs" and "rhs", both character().

Examples

extract_vars(Species ~ Sepal.Width + Sepal.Length)
extract_vars(Species ~ .)
formulate  

Create Formulas

Description
Given the left-hand side and right-hand side as character vectors, generates a new `stats::formula()`.

Usage
`formulate(lhs = NULL, rhs = NULL, env = NULL)`

Arguments
- `lhs` (character(1))
  - Left-hand side of formula.
- `rhs` (character())
  - Right-hand side of formula. Multiple elements will be collapsed with " + ".
- `env` (environment())
  - Environment for the new formula. Defaults to NULL.

Value
`stats::formula()`.

Examples
```r
formulate("Species", c("Sepal.Length", "Sepal.Width"))
formulate(rhs = c("Sepal.Length", "Sepal.Width"))
```

get_seed  

Get the Random Seed

Description
Retrieves the current random seed (.Random.seed in the global environment), and initializes the RNG first, if necessary.

Usage
`get_seed()`

Value
integer(). Depends on the `base::RNGkind()`.

Examples
```r
str(get_seed())
```
has_element

Check if an Object is Element of a List

Description

Simply checks if a list contains a given object.

- NB1: Objects are compared with identity.
- NB2: Only use this on lists with complex objects, for simpler structures there are faster operations.
- NB3: Clones of R6 objects are not detected.

Usage

has_element(.x, .y)

Arguments

.x (list() | atomic vector()).
.y (any)
   Object to test for.

Examples

has_element(list(1, 2, 3), 1)

ids

Extract ids from a List of Objects

Description

None.

Usage

ids(xs)

Arguments

xs (list())
   Every element must have a slot 'id'.

Value

(character()).
Examples

```r
xs = list(a = list(id = "foo", a = 1), bar = list(id = "bar", a = 2))
ids(xs)
```

---

**insert_named**  
*Insert or Remove Named Elements*

### Description

Insert elements from `y` into `x` by name, or remove elements from `x` by name. Works for vectors, lists, environments and data frames and data tables. Objects with reference semantic (environment()) and data.table::data.table() might be modified in-place.

### Usage

```r
insert_named(x, y)
```

- **Default S3 method:**
  ```r
  insert_named(x, y)
  ```

- **S3 method for class 'environment'**
  ```r
  insert_named(x, y)
  ```

- **S3 method for class 'data.frame'**
  ```r
  insert_named(x, y)
  ```

- **S3 method for class 'data.table'**
  ```r
  insert_named(x, y)
  ```

```r
remove_named(x, nn)
```

- **S3 method for class 'environment'**
  ```r
  remove_named(x, nn)
  ```

- **S3 method for class 'data.frame'**
  ```r
  remove_named(x, nn)
  ```

- **S3 method for class 'data.table'**
  ```r
  remove_named(x, nn)
  ```

### Arguments

- **x**  
  (vector() | list() | environment() | data.table::data.table())  
  Object to insert elements into, or remove elements from. Changes are by-reference for environments and data tables.

- **y**  
  (list())  
  List of elements to insert into `x`.
invoke

nn (character())
Character vector of elements to remove.

Value
Modified object.

Examples
x = list(a = 1, b = 2)
insert_named(x, list(b = 3, c = 4))
remove_named(x, "b")

invoke
Invoke a Function Call

Description
An alternative interface for do.call(), similar to the deprecated function in purrr. This function tries hard to not evaluate the passed arguments too eagerly which is important when working with large R objects.
It is recommended to pass all arguments named in order not to rely on on positional argument matching.

Usage
invoke(.f, ..., .args = list(), .opts = list(), .seed = NA_integer_)

Arguments
.f (function())
Function to call.
... (any)
Additional function arguments passed to .f.
.args (list())
Additional function arguments passed to .f, as (named) list(). These arguments will be concatenated to the arguments provided via ....
.opts (named list())
List of options which are set before the .f is called. Options are reset to their previous state afterwards.
.seed (integer(1))
Random seed to set before invoking the function call. Gets reset to the previous seed on exit.

Examples
invoke(mean, .args = list(x = 1:10))
invoke(mean, na.rm = TRUE, .args = list(1:10))
### `is_scalar_na`  
*Check for a Single Scalar Value*

**Description**
Check for a Single Scalar Value

**Usage**

```r
is_scalar_na(x)
```

**Arguments**

- `x` *(any)*  
  Argument to check.

**Value**

`(logical(1)).`

### `keep_in_bounds`  
*Remove All Elements Out Of Bounds*

**Description**
Filters vector `x` to only keep elements which are in bounds `[lower, upper]`. This is equivalent to the following, but tries to avoid unnecessary allocations:

```r
x[!is.na(x) & x >= lower & x <= upper]
```

Currently only works for integer `x`.

**Usage**

```r
keep_in_bounds(x, lower, upper)
```

**Arguments**

- `x` *(integer())*  
  Vector to filter.
- `lower` *(integer(1))*  
  Lower bound.
- `upper` *(integer(1))*  
  Upper bound.
load_dataset

Value
(integer()) with only values in [lower, upper].

Examples

keep_in_bounds(sample(20), 5, 10)

---
load_dataset

Retrieve a Single Data Set

Description
Loads a data set with name id from package package and returns it. If the package is not installed, an error with condition "packageNotFoundError" is raised. The name of the missing packages is stored in the condition as packages.

Usage

load_dataset(id, package, keep_rownames = FALSE)

Arguments

id (character(1))
Name of the data set.
package (character(1))
Package to load the data set from.
keep_rownames (logical(1))
Keep possible row names (default: FALSE).

Examples

head(load_dataset("iris", "datasets"))

---
map_values

Replace Elements of Vectors with New Values

Description
Replaces all values in x which match old with values in new. Values are matched with base::match().

Usage

map_values(x, old, new)
Modify elements of a vector selectively, similar to the functions in purrr.

modify_if() applies a predicate function .p to all elements of .x and applies .f to those elements of .x where .p evaluates to TRUE.
modify_at() applies .f to those elements of .x selected via .at.

**Usage**

```r
modify_if(.x, .p, .f, ...)
modify_at(.x, .at, .f, ...)
```

**Arguments**

- `.x` (vector())
- `.p` (function())
- `.f` (function())
- `...` (any)
- `.at` ((integer()) | character()))

**Examples**

```r
x = letters[1:5]

# replace all "b" with "_b_", and all "c" with "_c_
old = c("b", "c")
new = c("_b_", "_c_")
map_values(x, old, new)
```
named_list

Examples
x = modify_if(iris, is.factor, as.character)
str(x)

x = modify_at(iris, 5, as.character)
x = modify_at(iris, "Sepal.Length", sqrt)
str(x)

named_list

Create a Named List

Description
Create a Named List

Usage
named_list(nn = character(0L), init = NULL)

Arguments

nn
(character())
Names of new list.

init
(any)
All list elements are initialized to this value.

Value
(named list()).

Examples
named_list(c("a", "b"))
named_list(c("a", "b"), init = 1)

named_vector

Create a Named Vector

Description
Creates a simple atomic vector with init as values.

Usage
named_vector(nn = character(0L), init = NA)
Arguments

nn (character())
Names of new vector

init (atomic)
All vector elements are initialized to this value.

Value

(named vector()).

Examples

named_vector(c("a", "b"), NA)

named_vector(character())

names2 A Type-Stable names() Replacement

Description

A simple wrapper around base::names(). Returns a character vector even if no names attribute is set. Values NA and "" are treated as missing and replaced with the value provided in missing_val.

Usage

names2(x, missing_val = NA_character_)

Arguments

x (any)
Object.

missing_val (atomic(1))
Value to set for missing names. Default is NA_character_.

Value

(character(length(x))).

Examples

x = 1:3
names(x)
names2(x)

names(x)[1:2] = letters[1:2]
names(x)
names2(x, missing_val = "")
open_help

Description

Simply opens a manual page specified in "package::topic" syntax.

Usage

open_help(man)

Arguments

man (character(1))

Manual page to open in "package::topic" syntax.

Value

Nothing.

printf

Functions for Formatted Output and Conditions

Description

catf(), messagef(), warningf() and stopf() are wrappers around base::cat(), base::message(), base::warning() and base::stop(), respectively. The call is not included for warnings and errors.

Usage

catf(msg, ..., file = "", wrap = FALSE)

messagef(msg, ..., wrap = FALSE)

warningf(msg, ..., wrap = FALSE)

stopf(msg, ..., wrap = FALSE)
Arguments

**msg**
(character(1))
Format string passed to `base::sprintf()`.

**...**
(any)
Arguments passed down to `base::sprintf()`.

**file**
(character(1))
Passed to `base::cat()`.

**wrap**
(integer(1) | logical(1))
If set to a positive integer, `base::strwrap()` is used to wrap the string to the provided width. If set to TRUE, the width defaults to \(0.9 \times \text{getOption("width")}\). If set to FALSE, wrapping is disabled (default). If wrapping is enabled, all whitespace characters ([[:space:]]) are converted to spaces, and consecutive spaces are converted to a single space.

Examples

```r
messagef("This is a rather long %s on multiple lines which will get wrapped.
", "string", wrap = 15)
```

rcbind

**rcbind**  
**Bind Columns by Reference**

Description

Performs `base::cbind()` on `data.tables`, possibly by reference.

Usage

```r
rcbind(x, y)
```

Arguments

**x**    
(data.table::data.table())
`data.table::data.table()` to add columns to.

**y**    
(data.table::data.table())
`data.table::data.table()` to take columns from.

Value

(data.table::data.table()): Updated `x`.

Examples

```r
x = data.table::data.table(a = 1:3, b = 3:1)
y = data.table::data.table(c = runif(3))
rcbind(x, y)
```

---

**require_namespaces**

**Require Multiple Namespaces**

### Description

Packages are loaded (not attached) via `base::requireNamespace()`. If at least one package cannot be loaded, an exception of class "packageNotFoundError" is raised. The character vector of missing packages is stored in the condition as `.packages`.

### Usage

```r
require_namespaces(
  pkgs,
  msg = "The following packages could not be loaded: %s"
)
```

### Arguments

- **pkgs** *(character())*
  Packages to load.

- **msg** *(character(1))*
  Message to print on error. Use "%s" as placeholder for the list of packages.

### Value

(named character()) of loaded packages (invisibly).

### Examples

```r
require_namespaces("mlr3misc")
```

# catch condition, return missing packages
tryCatch(require_namespaces(c("mlr3misc", "foobaaar")),
  packageNotFoundError = function(e) e$packages)
Description

Similar to the tibble function tribble(), this function allows to construct tabular data in a row-wise fashion.

The first arguments passed as formula will be interpreted as column names. The remaining arguments will be put into the resulting table.

Usage

rowwise_table(..., .key = NULL)

Arguments

... (any)
Arguments: Column names in first rows as formulas (with empty left hand side), then the tabular data in the following rows.

.key (character(1))
If not NULL, set the key via data.table::setkeyv() after constructing the table.

Value

data.table::data.table().

Examples

rowwise_table(~ a, ~ b,
             1, "a",
             2, "b")

sequence_helpers  Sequence Construction Helpers

Description

seq_row() creates a sequence along the number of rows of x, seq_col() a sequence along the number of columns of x. seq_len0() and seq_along0() are the 0-based counterparts to base::seq_len() and base::seq_along().
set_class

Usage

seq_row(x)
seq_col(x)
seq_len0(n)
seq_along0(x)

Arguments

x (any)
   Arbitrary object. Used to query its rows, cols or length.
n (integer(1))
   Length of the sequence.

Examples

seq_len0(3)

---

set_class               Set the Class

Description

Simple wrapper for class(x) = classes.

Usage

set_class(x, classes)

Arguments

x (any).
classes (character(1))
   Vector of new class names.

Value

Object x, with updated class attribute.

Examples

set_class(list(), c("foo1", "foo2"))
**set_names**

**Set Names**

**Description**

Sets the names (or colnames) of `x` to `nm`. If `nm` is a function, it is used to transform the already existing names of `x`.

**Usage**

`set_names(x, nm = x, ...)`

`set_col_names(x, nm, ...)`

**Arguments**

- `x` *(any.)*  
  Object to set names for.

- `nm` *(character() | function())*  
  New names, or a function which transforms already existing names.

- `...` *(any)*  
  Passed down to `nm` if `nm` is a function.

**Value**

`x` with updated names.

**Examples**

```r
x = letters[1:3]

# name x with itself:
x = set_names(x)
print(x)

# convert names to uppercase
x = set_names(x, toupper)
print(x)
```
shuffle

Safe Version of Sample

Description
A version of sample() which does not treat positive scalar integer x differently. See example.

Usage
shuffle(x, n = length(x), ...)

Arguments
x (vector())
Vector to sample elements from.

n (integer())
Number of elements to sample.

...(any)
Arguments passed down to base::sample.int().

Examples
x = 2:3
sample(x)
shuffle(x)

x = 3
sample(x)
shuffle(x)

strCollapse

Collapse Strings

Description
Collapse multiple strings into a single string.

Usage
strCollapse(str, sep = "", quote = "", n = Inf, ellipsis = "[...]")
Arguments

str (character())
Vector of strings.
sep (character(1))
String used to collapse the elements of x.
quote (character(1))
Quotes to use around each element of x.
n (integer(1))
Number of elements to keep from x. See utils::head()
ellipsis (character(1))
If the string has to be shortened, this is signaled by appending ellipsis to str. Default is "[...]".

Value

(character(1)).

Examples

strCollapse(letters, quote = "/", n = 5)

---

str_indent

Indent Strings

Description

Formats a text block for printing.

Usage

str_indent(initial, str, width = 0.9 * getOption("width"), exdent = 2L, ...)

Arguments

initial (character(1))
Initial string, passed to strwrap().
str (character())
Vector of strings.
width (integer(1))
Width of the output.
exdent (integer(1))
Indentation of subsequent lines in paragraph.
... (any)
Additional parameters passed to strCollapse().
str_trunc

Value

(character()).

Examples

cat(str_indent("Letters:", strCollapse(letters), width = 25), sep = "\n")

---

| str_trunc | Truncate Strings |

Description

str_trunc() truncates a string to a given width.

Usage

str_trunc(str, width = 0.9 * getOption("width"), ellipsis = "[...]")

Arguments

str (character())
Vector of strings.

width (integer(1))
Width of the output.

ellipsis (character(1))
If the string has to be shortened, this is signaled by appending ellipsis to str. Default is "[...]".

Value

(character()).

Examples

str_trunc("This is a quite long string", 20)
**topo_sort**  
*Topological Sorting of Dependency Graphs*

**Description**
Topologically sort a graph, where we are passed node labels and a list of direct parents for each node, as labels, too. A node can be 'processed' if all its parents have been 'processed', and hence occur at previous indices in the resulting sorting. Returns a table, in topological row order for IDs, and an entry depth, which encodes the topological layer, starting at 0. So nodes with depth == 0 are the ones with no dependencies, and the one with maximal depth are the ones on which nothing else depends on.

**Usage**
```r
topo_sort(nodes)
```

**Arguments**
- **nodes** *(data.table::data.table())*
  Has 2 columns:
  - id of type character, contains all node labels.
  - parents of type list of character, contains all direct parents label of id.

**Value**
*(data.table::data.table())* with columns id, depth, sorted topologically for IDs.

**Examples**
```r
nodes = rowwise_table(~id, ~parents,  
  "a", "b",  
  "b", "c",  
  "c", character()  
)
topo_sort(nodes)
```

**transpose_list**  
*Transpose lists of lists*

**Description**
Transposes a list of list, and turns it inside out, similar to the function `transpose()` in package `purrr`.

```r
```
Usage

transpose_list(.l)

Arguments

.l (list() of list()).

Value

list().

Examples

x = list(list(a = 2, b = 3), list(a = 5, b = 10))
str(x)
str(transpose_list(x))

# list of data frame rows:
transpose_list(iris[1:2, ])

unnest

Unnest List Data Table Columns

Description

Transforms list columns to separate columns, possibly by reference. The original columns are removed from the returned table. All non-atomic objects in the list columns are expand to new list column.

Usage

unnest(x, cols, prefix = NULL)

Arguments

x (data.table::data.table())
data.table::data.table() with columns to unnest.

cols (character())
Column names of list columns to operate on.

prefix (character(1))
String to prefix the new column names with.

Value

(data.table::data.table()).
Examples

```r
data.table::data.table(
  id = 1:2,
  value = list(list(a = 1, b = 2), list(a = 2, b = 2))
)
print(x)
unnest(x, "value")
```

which_min

Index of the Minimum/Maximum Value, with ties correction

Description

Works similar to `base::which.min()`/`base::which.max()`, but corrects for ties. Missing values are set to Inf for which_min and to -Inf for which_max().

Usage

```r
which_min(x, ties_method = "random", na_rm = FALSE)
which_max(x, ties_method = "random", na_rm = FALSE)
```

Arguments

- **x** (numeric()): Numeric vector.
- **ties_method** (character(1)): Handling of ties. One of "first", "last" or "random" (default) to return the first index, the last index, or a random index of the minimum/maximum values.
- **na_rm** (logical(1)): Remove NAs before computation?

Value

`(integer())`: Index of the minimum/maximum value. Returns an empty integer vector for empty input vectors and vectors with no non-missing values (if na_rm is TRUE). Returns NA if na_rm is FALSE and at least one NA is found in x.

Examples

```r
x = c(2, 3, 1, 3, 5, 1, 1)
which_min(x, ties_method = "first")
which_min(x, ties_method = "last")
which_min(x, ties_method = "random")
which_max(x)
which_max(integer(0))
which_max(NA)
which_max(c(NA, 1))
```
with_package

Description
Attaches a package to the search path (if not already attached), executes code and eventually removes the package from the search path again, restoring the previous state.

Usage
with_package(package, code, ...)

Arguments
package (character(1))
Name of the package to attach.

code (expression)
Code to run.

... (any)
Additional arguments passed to library().

Value
Result of the evaluation of code.

See Also
withr package.

%nin%

Description
This operator is equivalent to !(x %in% y).

Usage
x %nin% y

Arguments
x (vector())
Values that should not be in y.

y (vector())
Values to match against.
Index

%nin%, 41
as_factor, 3
as_short_string, 4
base::cat(), 29, 30
base::cbind(), 8, 30
base::match(), 25
base::message(), 29
base::names(), 28
base::print(), 4
base::rbind(), 8
base::rep_len(), 26
base::requireNamespace(), 31
base::RNGkind(), 20
base::sample.int(), 35
base::seq_along(), 32
base::seq_len(), 32
base::split(), 6
base::sprintf(), 4, 30
base::stop(), 29
base::strwrap(), 30
base::warning(), 29
base::which.max(), 40
base::which.min(), 40
bibtex::read.bib(), 7
catf(printf), 29
check_packages_installed, 5
chunk(chunk_vector), 6
chunk_vector, 6
citation(), 7
cite_bib, 7
compat-map, 8
compute_mode, 10
cross_join, 11
data.table(), 11
data.table::CJ(), 11
data.table::data.table(), 8, 11, 18, 19, 22, 30, 32, 38, 39
data.table::rbindlist(), 10
data.table::setkeyv(), 32
data.tables, 30
detect(compat-map), 8
dictionary, 11, 11, 14, 15
dictionary_sugar
dictionary_sugar_get, 14
dictionary_sugar_mget, 14
did_you_mean, 15
discard(compat-map), 8
distinct_values, 16
do.call(), 23
encapsulate, 17
enframe, 18
every(compat-map), 8
extract_vars, 19
factor(), 3, 16
find.package(), 5
formula(), 19
formulate, 20
get_seed, 20
has_element, 21
ids, 21
imap(compat-map), 8
imap_chr (compat-map), 8
imap_dbl (compat-map), 8
imap_dtc (compat-map), 8
imap_dtr (compat-map), 8
imap_int (compat-map), 8
imap_lgl (compat-map), 8
insert_lgl (compat-map), 8
invoke, 23
is_scalar_na, 24
INDEX

keep (compat-map), 8
keep_in_bounds, 24

library(), 41
load_dataset, 25
logical(), 16

map (compat-map), 8
map_at (compat-map), 8
map_chr (compat-map), 8
map_dbl (compat-map), 8
map_dtc (compat-map), 8
map_dtr (compat-map), 8
map_if (compat-map), 8
map_int (compat-map), 8
map_lgl (compat-map), 8
map_values, 25
messagef (printf), 29
mlr3misc (mlr3misc-package), 3
mlr3misc-package, 3
modify_at (modify_if), 26
modify_if, 26

named_list, 27
named_vector, 27
names2, 28

open_help, 29
ordered(), 16

paradox::ParamSet, 15
pmap (compat-map), 8
pmap_chr (compat-map), 8
pmap_dbl (compat-map), 8
pmap_dtc (compat-map), 8
pmap_dtr (compat-map), 8
pmap_int (compat-map), 8
pmap_lgl (compat-map), 8
printf, 29
proc.time(), 18
R6::R6, 11
R6::R6Class(), 15
rcbind, 30
remove_named (insert_named), 22
require_namespaces, 31
rowwise_table, 32

seq_along0 (sequence_helpers), 32
seq_row (sequence_helpers), 32
sequence_helpers, 32
set_class, 33
set_col_names (set_names), 34
set_names, 34
shuffle, 35
some (compat-map), 8
stats::formula(), 20
stopf (printf), 29
strCollapse, 35
strCollapse(), 36
str_indent, 36
str_trunc, 37
strwrap(), 36
tools::toRd(), 7
topo_sort, 38
traceback(), 17
transpose_list, 38

unnest, 39
utils::adist(), 15
utils::head(), 36

warningf (printf), 29
which_max (which_min), 40
which_min, 40
with_package, 41