

Package ‘dat’

January 30, 2026

Type Package

Title Tools for Data Manipulation

Version 0.7.0

BugReports <https://github.com/wahani/dat/issues>

Description An implementation of common higher order functions with syntactic sugar for anonymous function. Provides also a link to 'dplyr' and 'data.table' for common transformations on data frames to work around non standard evaluation by default.

License MIT + file LICENSE

Depends methods

Imports data.table, Formula, magrittr, progress, aaos

Suggests dplyr, knitr, rbenchmark, nycflights13, rlang, rmarkdown, testthat, tibble, tidyselect

VignetteBuilder knitr

Encoding UTF-8

ByteCompile TRUE

RoxygenNote 7.1.0

Collate 'NAMESPACE.R' 'FormulaList.R' 'helper.R' 'DataFrame.R'
'as.function.R' 'bindRows.R' 'dataTableBackend.R' 'deparse.R'
'extract.R' 'map.R' 'mutar.R' 'pipeExport.R' 'replace.R'
'useDplyr.R' 'verboseApply.R'

NeedsCompilation no

Author Sebastian Warnholz [aut, cre]

Maintainer Sebastian Warnholz <wahani@gmail.com>

Repository CRAN

Date/Publication 2026-01-30 11:00:19 UTC

Contents

as.function.formula	2
bindRows	3
DataFrame	3
extract	4
FL	6
map	7
mutar	9
replace	11
verboseApply	12
Index	13

as.function.formula	<i>Coerce a formula into a function</i>
---------------------	---

Description

Convert a formula into a function. See [map](#) and [extract](#) for examples.

Usage

```
## S3 method for class 'formula'
as.function(x, ...)
```

Arguments

- x (formula) see examples
- ... not used

Value

An object inheriting from class function.

Examples

```
as.function(~.)(1)
as.function(x ~ x)(1)
as.function(f(x, y) ~ c(x, y))(1, 2)
as.function(numeric : x ~ x)(1) # check for class
as.function(numeric(1) : x ~ x)(1) # check for class + length
```

bindRows

*Bind rows***Description**

This is a wrapper around [rbindlist](#) to preserve the input class.

Usage

```
bindRows(x, id = NULL, useNames = TRUE, fill = TRUE)
```

Arguments

`x` (list) a list of data frames
`id, useNames, fill`
 passed to [rbindlist](#)

Value

If the first element of `x` inherits from `data.frame` the type that first element.
`x` else.

DataFrame

*DataFrame and methods***Description**

This is a 'data.table' like implementation of a data.frame. Either `dplyr` or `data.table` is used as backend. The only purpose is to have R CMD check friendly syntax.

Usage

```
DataFrame(...)

as.DataFrame(x, ...)

## Default S3 method:
as.DataFrame(x, ...)

## S3 method for class 'data.frame'
as.DataFrame(x, ...)

## S3 method for class 'DataFrame'
x[i, j, ..., by, sby, drop]
```

Arguments

<code>...</code>	arbitrary number of args in [(TwoSidedFormulas) in constructor see tibble
<code>x</code>	(DataFrame data.frame)
<code>i</code>	(logical numeric integer OneSidedFormula TwoSidedFormula FormulaList) see the examples.
<code>j</code>	(logical character TwoSidedFormula FormulaList function) character beginning with '^' are interpreted as regular expression
<code>by, sby</code>	(character) variables to group by. <code>by</code> will be used to do transformations within groups. <code>sby</code> will collapse each group to one row.
<code>drop</code>	(ignored) never drops the class.

Details

OneSidedFormula is always used for subsetting rows.

TwoSidedFormula is used instead of name-value expressions in `summarise` and `mutate`.

See Also

[mutar](#), [FL](#)

Examples

```
data("airquality")
dat <- as.DataFrame(airquality)
dat[~ Month > 4, ][meanWind ~ mean(Wind), sby = "Month"][meanWind]
dat[FL(.n ~ mean(.n), .n = c("Wind", "Temp")), sby = "Month"]
```

extract

Extract elements from a vector

Description

Extract elements from an object as S4 generic function. See the examples.

Usage

```
extract(x, ind, ...)

## S4 method for signature 'list,`function`'
extract(x, ind, ...)

## S4 method for signature 'atomic,`function`'
extract(x, ind, ...)
```

```
## S4 method for signature 'ANY,formula'
extract(x, ind, ...)

## S4 method for signature 'atomicORlist,numericORintegerORlogical'
extract(x, ind, ...)

## S4 method for signature 'ANY,character'
extract(x, ind, ...)

## S4 method for signature 'data.frame,character'
extract(x, ind, ...)

extract2(x, ind, ...)

## S4 method for signature 'atomicORlist,numericORinteger'
extract2(x, ind, ...)

## S4 method for signature 'ANY,formula'
extract2(x, ind, ...)

## S4 method for signature 'atomicORlist,`function`'
extract2(x, ind, ...)

## S4 method for signature 'ANY,character'
extract2(x, ind, ...)
```

Arguments

x	(atomic list) a vector.
ind	(function formula character numeric integer logical) a formula is coerced into a function. For lists the function is applied to each element (and has to return a logical of length 1). For atomics a vectorized function is expected. If you supply an atomic it is used for subsetting. A character of length 1 beginning with "^" is interpreted as regular expression.
...	arguments passed to ind.

Examples

```
extract(1:15, ~ 15 %% . == 0)
extract(list(xy = 1, zy = 2), "^z")
extract(list(x = 1, z = 2), 1)
extract(list(x = 1, y = ""), is.character)

# Example: even numbers:
is.even <- function(x) (x %% 2) == 0
sum((1:10)[is.even(1:10)])
extract(1:10, ~ . %% 2 == 0) %>% sum
extract(1:10, is.even) %>% sum
```

```

# Example: factors of 15
extract(1:15, ~ 15 %% . == 0)

# Example: relative prime numbers
gcd <- function(a, b) {
  .gcd <- function(a, b) if (b == 0) a else Recall(b, a %% b)
  flatmap(a ~ b, .gcd)
}

extract(1:10, x ~ gcd(x, 10) == 1)

# Example: real prime numbers
isPrime <- function(n) {
  .isPrime <- function(n) {
    iter <- function(i) {
      if (i * i > n) TRUE
      else if (n %% i == 0 || n %% (i + 2) == 0) FALSE
      else Recall(i + 6)
    }
    if (n <= 1) FALSE
    else if (n <= 3) TRUE
    else if (n %% 2 == 0 || n %% 3 == 0) FALSE
    else iter(5)
  }
  flatmap(n, x ~ .isPrime(x))
}

extract(1:10, isPrime)

```

FL

Dynamically generate formulas

Description

Function to dynamically generate formulas - (F)ormula (L)ist - to be used in [mutar](#).

Usage

```

FL(..., .n = NULL, pattern = "\\\\.n")

makeFormulas(..., .n, pattern = "\\\\.n")

## S3 method for class 'FormulaList'
update(object, data, ...)

```

Arguments

... (formulas)

.n	names to be used in formulas. Can be any object which can be used by extract to select columns. NULL is interpreted to use the formulas without change.
pattern	(character) pattern to be replaced in formulas
object	(FormulaList)
data	(data.frame)

See Also[mutar](#)**Examples**

```
FL(.n ~ mean(.n), .n = "variable")
as(makeFormulas(.n ~ mean(.n), .n = "variable"), "FormulaList")
```

map	<i>An implementation of map</i>
-----	---------------------------------

Description

An implementation of map and flatmap. They support the use of formulas as syntactic sugar for anonymous functions.

Usage

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

S4 method for signature 'ANY,formula'

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

S4 method for signature 'atomic,function'

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

S4 method for signature 'list,function'

```
map(x, f, p = function(x) TRUE, ...)
```

S4 method for signature 'list,numericORcharacterORlogical'

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

S4 method for signature 'MList,function'

```
map(x, f, ..., simplify = FALSE)
```

S4 method for signature 'formula,function'

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

```
flatmap(x, f, ..., flatten = unlist)
```

```
## S4 method for signature 'ANY,formula'
flatmap(x, f, ..., flatten = unlist)

sac(x, f, by, ..., combine = bindRows)

## S4 method for signature 'data.frame,`function`'
sac(x, f, by, ..., combine = bindRows)

## S4 method for signature 'ANY,formula'
sac(x, f, by, ..., combine = bindRows)

vmap(x, f, ..., .mc = min(length(x), detectCores()), .bar = "bar")
```

Arguments

x	(vector data.frame formula) if x inherits from data.frame, a data.frame is returned. Use as.list if this is not what you want. When x is a formula it is interpreted to trigger a multivariate map.
f	(function formula character logical numeric) something which can be interpreted as a function. formula objects are coerced to a function. atomics are used for subsetting in each element of x. See the examples.
...	further arguments passed to the apply function.
p	(function formula) a predicate function indicating which columns in a data.frame to use in map. This is a filter for the map operation, the full data.frame is returned.
simplify	see SIMPLIFY in mapply
flatten	(function formula) a function used to flatten the results.
by	(e.g. character) argument is passed to extract to select columns.
combine	(function formula) a function which knows how to combine the list of results. bindRows is the default.
.mc	(integer) the number of cores. Passed down to mclapply or mcmapapply .
.bar	(character) see verboseApply .

Details

map will dispatch to [lapply](#). When x is a formula this is interpreted as a multivariate map; this is implemented using [mapply](#). When x is a data.frame map will iterate over columns, however the return value is a data.frame. p can be used to map over a subset of x.

flatmap will dispatch to map. The result is then wrapped by flatten which is [unlist](#) by default.

sac is a naive implementation of split-apply-combine and implemented using flatmap.

vmap is a 'verbose' version of map and provides a progress bar and a link to parallel map ([mclapply](#)).

map, flatmap, and sac can be extended; they are S4 generic functions. You don't and should not implement a new method for formulas. This method will coerce a formula into a function and pass it down to your map(newtype, function) method.

Examples

```
# Sugar for anonymous functions
map(data.frame(y = 1:10, z = 2), x ~ x + 1)
map(data.frame(y = 1:10, z = 2), x ~ x + 1, is.numeric)
map(data.frame(y = 1:10, z = 2), x ~ x + 1, x ~ all(x == 2))
sac(data.frame(y = 1:10, z = 1:2), df ~ data.frame(my = mean(df$y)), "z")

# Trigger a multivariate map with a formula
map(1:2 ~ 3:4, f(x, y) ~ x + y)
map(1:2 ~ 3:4, f(x, y) ~ x + y, simplify = TRUE)
map(1:2 ~ 3:4, f(x, y, z) ~ x + y + z, z = 1)

# Extracting values from lists
map(list(1:2, 3:4), 2)
map(list(1:3, 2:5), 2:3)
map(list(1:3, 2:5), c(TRUE, FALSE, TRUE))

# Some type checking along the way
map(as.numeric(1:2), numeric : x ~ x)
map(1:2, integer(1) : x ~ x)
map(1:2, numeric(1) : x ~ x + 0.5)
```

mutar

Tools for Data Frames

Description

mutar is literally the same function as `[.DataFrame` and can be used as interface to dplyr or data.table. Other functions here listed are a convenience to mimic dplyr's syntax in a R CMD check friendly way. These functions can also be used with S4 data.frame(s) / data_frame(s) / data.table(s). They will always try to preserve the input class.

Usage

```
mutar(x, i, j, ..., by, sby, drop)
```

```
filter(x, i)
```

```
summar(x, ..., by)
```

```
withReference(expr)
```

Arguments

x	(DataFrame data.frame)
i	(logical numeric integer OneSidedFormula TwoSidedFormula FormulaList) see the examples.

j	(logical character TwoSidedFormula FormulaList function) character beginning with '^' are interpreted as regular expression
...	arbitrary number of args in [(TwoSidedFormulas) in constructor see tibble
by	(character) variables to group by. by will be used to do transformations within groups. sby will collapse each group to one row.
sby	(character) variables to group by. by will be used to do transformations within groups. sby will collapse each group to one row.
drop	(ignored) never drops the class.
expr	(expression) any R expression that should be evaluated using data tables reference semantics on data transformations.

Details

The real workhorse of this interface is `mutar`. All other functions exist to ease the transition from `dplyr`.

`OneSidedFormula` is always used for subsetting rows.

`TwoSidedFormula` is used instead of name-value expressions. Instead of writing `x = 1` you simply write `x ~ 1`.

`FormulaList` can be used to repeat the same operation on different columns. See more details in [FL](#).

See Also

[extract](#), [DataFrame](#), [FL](#)

Examples

```
data("airquality")
airquality %>%
  filter(~Month > 4) %>%
  mutar(meanWind ~ mean(Wind), by = "Month") %>%
  sumar(meanWind ~ mean(Wind), by = "Month") %>%
  extract("meanWind")

airquality %>%
  sumar(
    .n ~ mean(.n) | c("Wind", "Temp"),
    by = "Month"
  )

# Enable data.tables reference semantics with:
withReference({
  x <- data.table::data.table(x = 1)
  mutar(x, y ~ 2)
})
```

```
## Not run:
# Use dplyr as back-end:
options(dat.use.dplyr = TRUE)
x <- data.frame(x = 1)
mutar(x, y ~ dplyr::n())

## End(Not run)
```

replace	<i>Replace elements in a vector</i>
---------	-------------------------------------

Description

This function replaces elements in a vector. It is a link to [replace](#) as a generic function.

Usage

```
replace(x, ind, values, ...)

## S4 method for signature 'ANY,`function`'
replace(x, ind, values, ...)

## S4 method for signature 'ANY,formula'
replace(x, ind, values, ...)

## S4 method for signature 'ANY,character'
replace(x, ind, values, ...)
```

Arguments

x	(atomic list) a vector.
ind	used as index for elements to be replaced. See details.
values	the values used for replacement.
...	arguments passed to ind if it can be interpreted as function. For a regex arguments are passed to grep .

Details

The idea is to provide a more flexible interface for the specification of the index. It can be a character, numeric, integer or logical which is then simply used in `base::replace`. It can be a regular expression in which case x should be named – a character of length 1 and a leading "^" is interpreted as regex. When ind is a function (or formula) and x is a list then it should be a predicate function – see the examples. When x is an atomic the function is applied on x and the result is used for subsetting.

Examples

```

replace(c(1, 2, NA), is.na, 0)
replace(c(1, 2, NA), rep(TRUE, 3), 0)
replace(c(1, 2, NA), 3, 0)
replace(list(x = 1, y = 2), "x", 0)
replace(list(x = 1, y = 2), "^x$", 0)
replace(list(x = 1, y = "a"), is.character, NULL)

```

verboseApply	<i>Verbose apply function</i>
--------------	-------------------------------

Description

This apply function has a progress bar and enables computations in parallel. By default it is not verbose. As an interactive version with proper 'verbose' output by default please use [vmap](#).

Usage

```
verboseApply(x, f, ..., .mc = 1, .mapper = mclapply, .bar = "none")
```

Arguments

x	(vector)
f	(function)
...	arguments passed to .mapper and hence f
.mc	(integer) the number of processes to start
.mapper	(function) the actual apply function used. Should have an argument mc.cores.
.bar	(character) one in 'none', '.' or 'bar'

Examples

```

## Not run:
verboseApply(
  1:4,
  function(...) Sys.sleep(1),
  .bar = "bar",
  .mc = 2
)

## End(Not run)

```

Index

[.DataFrame (DataFrame), 3

as.DataFrame (DataFrame), 3

as.function.formula, 2

as.list, 8

bindRows, 3, 8

data.frame, 8

DataFrame, 3, 10

extract, 2, 4, 7, 8, 10

extract, ANY, character-method (extract), 4

extract, ANY, formula-method (extract), 4

extract, atomic, function-method (extract), 4

extract, atomicORlist, numericORintegerORlogical-method (extract), 4

extract, data.frame, character-method (extract), 4

extract, list, function-method (extract), 4

extract2 (extract), 4

extract2, ANY, character-method (extract), 4

extract2, ANY, formula-method (extract), 4

extract2, atomicORlist, function-method (extract), 4

extract2, atomicORlist, numericORinteger-method (extract), 4

filter (mutar), 9

FL, 4, 6, 10

flatmap (map), 7

flatmap, ANY, formula-method (map), 7

formula, 8

function, 8

grep, 11

lapply, 8

makeFormulas (FL), 6

map, 2, 7

map, ANY, formula-method (map), 7

map, atomic, function-method (map), 7

map, formula, function-method (map), 7

map, list, function-method (map), 7

map, list, numericORcharacterORlogical-method (map), 7

map, MList, function-method (map), 7

mapply, 8

mclapply, 8

mcmapply, 8

mutar, 4, 6, 7, 9

rbindlist, 3

replace, 11, 11

replace, ANY, character-method (replace), 11

replace, ANY, formula-method (replace), 11

replace, ANY, function-method (replace), 11

sac (map), 7

sac, ANY, formula-method (map), 7

sac, data.frame, function-method (map), 7

sumar (mutar), 9

tibble, 4, 10

unlist, 8

update.FormulaList (FL), 6

vector, 8

verboseApply, 8, 12

vmap, 12

vmap (map), 7

withReference (mutar), 9