

# Package ‘accumulate’

September 22, 2023

**Type** Package

**Title** Split-Apply-Combine with Dynamic Groups

**Version** 0.9.3

**Maintainer** Mark van der Loo <mark.vanderloo@gmail.com>

**Description** Estimate group aggregates, where one can set user-defined conditions that each group of records must satisfy to be suitable for aggregation. If a group of records is not suitable, it is expanded using a collapsing scheme defined by the user.

**License** EUPL

**URL** <https://github.com/markvanderloo/accumulate>

**LazyData** TRUE

**VignetteBuilder** simplermardown

**Depends** R (>= 3.5.0)

**Suggests** tinytest, simplermardown, validate

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**NeedsCompilation** no

**Author** Mark van der Loo [aut, cre] (<<https://orcid.org/0000-0002-9807-4686>>)

**Repository** CRAN

**Date/Publication** 2023-09-22 12:50:02 UTC

## R topics documented:

accumulate . . . . .	2
csh_from_digits . . . . .	4
frac_complete . . . . .	5
from_validator . . . . .	6
min_complete . . . . .	7
min_records . . . . .	8
producers . . . . .	8
smoke_test . . . . .	9

---

accumulate	<i>Split-Apply-Combine with Collapsing Groups</i>
------------	---

---

### Description

Compute grouped aggregates. If a group does not satisfy certain user-defined conditions (such as too many missings, or not enough records) then the group is expanded according to a user-defined 'collapsing' scheme. This happens recursively until either the group satisfies all conditions and the aggregate is computed, or we run out of collapsing possibilities and the NA is returned for that group.

- accumulate aggregates over all non-grouping variables defined in collapse
- cumulate uses a syntax akin to `dplyr::summarise`

### Usage

```
accumulate(data, collapse, test, fun, ...)
```

```
cumulate(data, collapse, test, ...)
```

### Arguments

data	[data.frame] The data to aggregate by (collapsing) groups.
collapse	[formula data.frame] representing a group collapsing sequence. See below for details on how to specify each option.
test	[function] A function that takes a subset of data and returns TRUE if it is suitable for computing the desired aggregates and FALSE if a collapsing step is necessary.
fun	[function] A scalar function that will be applied to all columns of data.
...	For accumulate, extra arguments to be passed to fun. For cumulate, a comma-separated list of name=expression, where expression defines the aggregating operation.

### Value

A data frame where each row represents a (multivariate) group. The first columns contain the grouping variables. The next column is called `level` and indicates to what level collapsing was necessary to compute a value, where 0 means that no collapsing was necessary. The following columns contain the aggregates defined in the `...` argument. If no amount of collapsing yields a data set that is satisfactory according to `test`, then for that row, the `level` and subsequent columns are NA.

### Using a formula to define the collapsing sequence

If all combinations of collapsing options are stored as columns in data, the formula interface can be used. An example is the easiest way to see how it works. Suppose that  $\text{collapse} = A \times B \sim A1 \times B + B$ . This means:

- Compute output for groups defined by variables A and B
- If for a certain combination (a, b) in  $A \times B$  the data does not pass the test, use (a1, b) in  $A1 \times B$  as alternative combination to compute a value for (a, b) ( $A1 \times B$  must yield larger groups than  $A \times B$ ).
- If that does not work, use only B as a grouping variable to compute a value for (a, b).
- If that does not work, return NA for that particular combination (a, b).

Generally, the formula must be of the form  $X0 \sim X1 + X2 + \dots + Xn$  where each  $Xi$  is a (product of) grouping variable(s) in the data set.

### Using a data frame to define the collapsing scheme

In this case collapse is a data frame with columns  $[A0, A1, \dots, An]$ . The variable  $A0$  represents the most fine-grained grouping and must also be present in data. Aggregation works as follows.

- Compute output for groups defined by variable  $A0$
- If for a certain  $a0$  in  $A0$  the corresponding selected data does not pass the test, use the larger dataset corresponding to  $a1$  in  $A1$  to compute output for  $a1$ .
- Repeat the second step until either the test is passed or no more collapsing is possible. In the latter case, return NA for that particular value of  $a0$ .

### Examples

```
## Example of data frame defining collapsing scheme, using accumulate

input <- data.frame(Y1 = 2^(0:8), Y2 = 2^(0:8))
input$Y2[c(1,4,7)] <- NA
# make sure that the input data also has the most fine-grained (target)
# grouping variable
input$A0 <- c(123,123,123,135,136,137,212,213,225)

# define collapsing sequence
collapse <- data.frame(
  A0 = c(123, 135, 136, 137, 212, 213, 225)
  , A1 = c(12 , 13 , 13 , 13 , 21 , 21 , 22 )
  , A2 = c(1 , 1 , 1 , 1 , 2 , 2 , 2 )
)

accumulate(input
  , collapse
  , test = function(d) nrow(d)>=3
  , fun = sum, na.rm=TRUE)
```

```

## Example of formula defining collapsing scheme, using cumulate
input <- data.frame(
  A = c(1,1,1,2,2,2,3,3,3)
, B = c(11,11,11,12,12,13,21,22,12)
, B1 = c(1,1,1,1,1,1,2,2,1)
, Y = 2^(0:8)
)
cumulate(input, collapse=A*B ~ A*B1 + A
, test = function(d) nrow(d) >= 3
, tY = sum(Y))

## Example with formula defining collapsing scheme, using accumulate
# The collapsing scheme must be represented by variables in the
# data. All columns not part of the collapsing scheme will be aggregated
# over.

input <- data.frame(
  A = c(1,1,1,2,2,2,3,3,3)
, B = c(11,11,11,12,12,13,21,22,12)
, B1 = c(1,1,1,1,1,1,2,2,1)
, Y1 = 2^(0:8)
, Y2 = 2^(0:8)
)

input$Y2[c(1,4,7)] <- NA

accumulate(input
, collapse = A*B ~ A*B1 + A
, test=function(a) nrow(a)>=3
, fun = sum, na.rm=TRUE)

## Example with data.frame defining collapsing scheme, using cumulate
dat <- data.frame(A0 = c("11","12","11","22"), Y = c(2,4,6,8))
# collapsing scheme
csh <- data.frame(
  A0 = c("11","12","22")
, A1 = c("1" ,"1" , "2")
)
cumulate(data = dat
, collapse = csh
, test = function(d) if (nrow(d)<2) FALSE else TRUE
, mn = mean(Y, na.rm=TRUE)
, md = median(Y, na.rm=TRUE)
)

```

**Description**

Derive a collapsing scheme where group labels collapse to their parents in the hierarchy.

**Usage**

```
csh_from_digits(x, levels = max(nchar(x)) - 1)
```

**Arguments**

**x** [character|integer] labels in a hierarchical classification (lowest level)  
**levels** [integer >=0] how many collapsing levels to include. Zero means only include the original labels.

**Value**

A data frame where each consecutive pair of columns represents one collapsing step induced by the hierarchical classification encoded by the digits in x.

**Examples**

```
# balanced hierarchical classification
csh_from_digits(c("111", "112", "121", "122", "123"))
csh_from_digits(c("111", "112", "121", "122", "123"), levels=1)

# unbalanced hierarchical classification
csh_from_digits(c("111", "112", "121", "122", "1221", "1222"))
csh_from_digits(c("111", "112", "121", "122", "1221", "1222"), levels=2)
```

---

frac_complete	<i>Demand minimal fraction of complete records</i>
---------------	--

---

**Description**

Demand minimal fraction of complete records

**Usage**

```
frac_complete(r, vars = TRUE)
```

**Arguments**

**r** Minimal fraction of records that must be complete.  
**vars** [TRUE|column index] Column index into the data to be tested (e.g. a character vectod with variable names or a numeric vector with column positions). The indexed columns will be teststed for completeness (absence of NA). Be default vars=TRUE meaning that all columns are taken into account.

**Value**

a function that accepts a data frame and returns TRUE when the fraction of complete records is larger than or equal to n and otherwise FALSE.

**See Also**

Other helpers: `min_complete()`, `min_records()`

**Examples**

```
f <- frac_complete(0.1)
f(mtcars) # TRUE (all complete)
mt <- mtcars
mt[1:5,1] <- NA
f(mt)     # FALSE (5/32 incomplete)
```

---

from\_validator      *Use a validate::validator object to define a test*

---

**Description**

Create a test function that accepts a data.frame, and returns TRUE when the data passes all checks defined in the validator object, and otherwise FALSE.

**Usage**

```
from_validator(v, ...)
```

**Arguments**

v                    [validator] a validator object from the validate package.  
...                   options passed to `validate::confront`

**Value**

a function that accepts a data fram and returns TRUE when the data passes all checks in v and otherwise FALSE.

**Note**

Requires the validate package to be installed.

**References**

Mark P. J. van der Loo, Edwin de Jonge (2021). Data Validation Infrastructure for R. Journal of Statistical Software, 97(10), 1-31. doi:10.18637/jss.v097.i10

## Examples

```
if (requireNamespace("validate", quietly=TRUE)){
  v <- validate::validator(height >= 0, weight >= 0)
  f <- from_validator(v)
  f(women) # TRUE (all heights and weights are nonnegative)
}
```

---

min\_complete

*Demand minimal number of complete records*

---

## Description

Demand minimal number of complete records

## Usage

```
min_complete(n, vars = TRUE)
```

## Arguments

n	Minimal number of records that must be complete
vars	[TRUE   column index] Column index into the data to be tested (e.g. a character vected with variable names or a numeric vector with column positions). The indexed columns will be testsed for completeness (absence of NA). Be default vars=TRUE meaning that all columns are taken into account.

## Value

a function that accepts a data frame and returns TRUE when the number of complete records is larger than or equal to n and otherwise FALSE.

## See Also

Other helpers: [frac\\_complete\(\)](#), [min\\_records\(\)](#)

## Examples

```
f <- min_complete(20)
f(women) # FALSE (15 records)
f(mtcars) # TRUE (32 records)
```

---

min_records	<i>Demand minimal number of records</i>
-------------	---

---

### Description

Demand minimal number of records

### Usage

```
min_records(n)
```

### Arguments

n Minimal number of records in a group.

### Value

a function that accepts a data frame and returns TRUE when the number of records is larger than or equal to n and otherwise FALSE.

### See Also

Other helpers: [frac\\_complete\(\)](#), [min\\_complete\(\)](#)

### Examples

```
min_records(5)(women)
min_records(200)(women)
```

---

producers	<i>Synthetic data on producers</i>
-----------	------------------------------------

---

### Description

A synthetic dataset listing several sources of turnover and other income for producers. The producers are classified in size classes and SBI (a refinement of NACE). Load with `data(producers)`.

- `sbi`: Classification of economic activity (refinement of NACE2008)
- `size`: Size class in 0 (smallest) to 9.
- `industrial`: Turnover from industrial activities.
- `trade`: Turnover from trade
- `other`: Turnover from other activities
- `other_income`: Income not from turnover (e.g. from financial transactions)
- `total`: Rowwise sum of industrial, trade, and other turnover and other income.



**Format**

A .rda file, one producer per row.

---

 smoke\_test

*Check your testing function against common edge cases*


---

**Description**

Writing a testing function that works on any subset of records of a dataframe can be quite subtle. This function tries the testing function on a number of common (edge) cases that are easily overlooked. It is *not* a unit test: a smoke test will not tell you whether your output is correct. It only checks the output data type (must be TRUE or FALSE and reports if errors, warnings, or messages occur).

**Usage**

```
smoke_test(dat, test, verbose = FALSE, halt = TRUE)
```

**Arguments**

dat	an example dataset. For example the full dataset to be fed into <a href="#">accumulate</a> or <a href="#">cumulate</a> .
test	A testing function to be passed as argument to <a href="#">accumulate</a> or <a href="#">cumulate</a> .
verbose	[logical] If TRUE, all results (including passed tests) are printed. If FALSE only failed tests are printed.
halt	[logical] toggle stopping when an error is thrown

**Value**

NULL, invisibly. This function has as side-effect that test results are printed to screen.

**Examples**

```
dat <- data.frame(x = 1:5, y=(-2):2)
smoke_test(dat, function(d) y > 0) #error: Y not found
smoke_test(dat, function(d) d$y > 0) # issue: output too long, not robust against NA
smoke_test(dat, function(d) sum(d$y > 0) > 2) # issue: not robust against NA
smoke_test(dat, function(d) sum(d$y > 0, na.rm=TRUE) > 2) # OK
```

# Index

- \* **datasets**

- producers, 8

- \* **helpers**

- frac\_complete, 5

- min\_complete, 7

- min\_records, 8

accumulate, 2, 9

csh\_from\_digits, 4

cumulate, 9

cumulate (accumulate), 2

frac\_complete, 5, 7, 8

from\_validator, 6

min\_complete, 6, 7, 8

min\_records, 6, 7, 8

producers, 8

smoke\_test, 9