Package: svBase (via r-universe)

July 5, 2024

Type Package

Version 1.4.0

Title Base Objects like Data Frames for 'SciViews::R'

Description Functions to manipulated the three main classes of ``data frames'' for 'SciViews::R': data.frame, data.table and tibble. Allow to select the preferred one, and to convert more carefully between the three, taking care of correct presentation of row names and data.table's keys. More homogeneous way of creating these three data frames and of printing them on the R console.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 4.2.0)

Imports collapse (>= 2.0.12), data.table (>= 1.15.4), dplyr (>= 1.1.4), rlang (>= 1.1.1), tibble (>= 3.2.1), tidyr (>= 1.3.0), zeallot (>= 0.1.0)

Suggests dtplyr (>= 1.3.1), svMisc (>= 1.4.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

Remotes SciViews/svMisc

License MIT + file LICENSE

URL https://github.com/SciViews/svBase, https://www.sciviews.org/svBase/

BugReports https://github.com/SciViews/svBase/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

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

RemoteUrl https://github.com/SciViews/svBase

RemoteRef HEAD

RemoteSha 47bf016b6227600af8175c906d59bb1d7c41f061

Contents

`svBase-package` *Base Objects like Data Frames for 'SciViews::R'*

Description

The {svBase} package sets up the way data frames (with objects like R base's **data.frame**, **data.table** and tibble **tbl_df**) are managed in SciViews::R. The user can select the class of object it uses by default and many other SciViews::R functions return that format. Conversion from one to the other is made easier, including for the management of **data.frame**'s row names or **data.table**'s keys. Also homogeneous ways to create a data frame or to print it are also provided.

Important functions

- `dtx()` creates a data frame in the preferred format, with the following functions `dtbl()`, `dtf()` and `dtt()` that force respectively the creation of a data frame in one of the specified three formats. Use `getOption("SciViews.as_dtx", default = as_dtt)` to specify which function to use to convert into the preferred format.

`alt_assign` *Alternate assignment (multiple and/or collect results from dplyr)*

Description

These alternate assignment operators can be used to perform multiple assignment (also known as destructuring assignment). These are imported from the {zeallot} package (see the corresponding help page at zeallot::operator for complete description). They also performs a `dplyr::collect()` allowing to get results from dplyr extensions like {dtplyr} for data.tables, or {dbplyr} for databases. Finally these two assignment operators also make sure that the preferred data frame object is returned by using `default_dtx()`.

Usage

```
value %->% x

x %<-% value

## Default S3 method:
collect(x, ...)
```

Arguments

`value`	The object to be assigned.
`x`	A name, or a name structure for multiple (deconstructing) assignment, or any object that does not have a specific [dplyr::collect[]) method for `collect.default()`.
`...`	further arguments passed to the method (not used for the default one)

Details

These assignation operator are overloaded to get interesting properties in the context of {tidyverse} pipelines and to make sure to always return our preferred data frame object (data.frame, data.table, or tibble). Thus, before being assigned, `value` is modified by calling `dplyr::collect()` on it and by applying `default_dtx()`.

Value

These operators invisibly return `value`. `collect.default()` simply return `x`.

Examples

```
# The alternate assignment operator performs three steps:
# 1) Collect results from dbplyr or dtplyr
library(dplyr)
library(data.table)
library(dtplyr)
library(svBase)
```

```
dtt <- data.table(x = 1:5, y = rnorm(5))
dtt |>
  mutate(x2 = x^2) |>
  select(x2, y) ->
  res

print(res)
class(res) # This is a data frame

dtt |>
  lazy_dt() |>
  mutate(x2 = x^2) |>
  select(x2, y) ->
  res

print(res)
class(res) # This is NOT a data frame

# Same pipeline, but assigning with %->%
dtt |>
  lazy_dt() |>
  mutate(x2 = x^2) |>
  select(x2, y) %->%
  res

print(res)
class(res) # res is the preferred data frame (data.table by default)

# 2) Convert data frame in the chosen format using default_dtx()
dtf <- data.frame(x = 1:5, y = rnorm(5))
class(dtf)
res %<-% dtf
class(res) # A data.table by default
# but it can be changed with options("SciViews.as_dtx)

# 3) If the zeallot syntax is used, make multiple assignment
c(X, Y) %<-% dtf # Variables of dtf assigned to different names
X
Y

# The %->% is meant to be used in pipelines, otherwise it does the same
```

---

as_dtx *Coerce objects into data.frames, data.tables, tibbles or matrices*

Description

Objects are coerced into the desired class. For `as_dtx()`, the desired class is obtained from `getOption("SciViews.as_dtx")`, with a default value producing a data.table object. If the data are grouped with `dplyr::group_by()`, the resulting data frame is also `dplyr::ungroup()`ed in the process.

Usage

```
as_dtx(x, ..., rownames = NULL, keep.key = TRUE, byref = FALSE)

as_dtf(x, ..., rownames = NULL, keep.key = TRUE, byref = NULL)

as_dtt(x, ..., rownames = NULL, keep.key = TRUE, byref = FALSE)

as_dtbl(x, ..., rownames = NULL, keep.key = TRUE, byref = NULL)

default_dtx(x, ..., rownames = NULL, keep.key = TRUE, byref = FALSE)

## S3 method for class 'tbl_df'
as.matrix(x, row.names = NULL, optional = FALSE, ...)

as_matrix(x, rownames = NULL, ...)
```

Arguments

x	An object.
...	Further arguments passed to the methods (not used yet).
rownames	The name of the column with row names. If NULL, it is assessed from getOptions("SciViews.dtx.rowna
keep.key	Do we keep the data.table key into a "key" attribute or do we restore data.tablekey from the attribute?
byref	If TRUE, the object is modified by reference when converted into a data.table (faster, but not conventional). This is FALSE by default, or NULL if the argument does not apply in the context.
row.names	Same as rownames, but for base R functions.
optional	logical, If TRUE, setting row names and converting column names to syntactically correct names is optional.

Value

The coerced object. For as_dtx(), the coercion is determined from getOption("SciViews.as_dtx") which must return one of the three other as_dt...() functions (as_dtt by default). The default_dtx() does the same as as_dtx() if the object is a data.frame, a data.table, or a tibble, but it return the unmodified object for any other class (including subclassed data frames). This is a convenient function to force conversion only between those three objects classes.

Note

Use as_matrix() instead of base::as.matrix(): it has different default arguments to better account for rownames in data.table and tibble!

Examples

```
# A data.frame
dtf <- dtf(
```

```
  x = 1:5,
  y = rnorm(5),
  f = letters[1:5],
  l = sample(c(TRUE, FALSE), 5, replace = TRUE))

# Convert into a tibble
(dtbl <- as_dtbl(dtf))
# Since row names are trivial (1 -> 5), a .rownames column is not added

dtf2 <- dtf
rownames(dtf2) <- letters[1:5]
dtf2

# Now, the conversion into a tibble adds .rownames
(dtbl2 <- as_dtbl(dtf2))
# and data frame row names are set again when converted bock to dtf
as_dtf(dtbl2)

# It also work for conversions data.frame <-> data.table
(dtt2 <- as_dtt(dtf2))
as_dtf(dtt2)

# It does not work when converting a tibble or a data.table into a matrix
# with as.matrix()
as.matrix(dtbl2)
# ... but as_matrix() does the job!
as_matrix(dtbl2)

# The name for row in dtt and dtbl is in:
# (data.frame's row names are converted into a column with this name)
getOption("SciViews.dtx.rownames", default = ".rownames")

# Convert into the preferred data frame object (data.table by default)
(dtx2 <- as_dtx(dtf2))
class(dtx2)

# The default data frame object used:
getOption("SciViews.as_dtx", default = as_dtt)

# default_dtx() does the same as as_dtx(),
# but it also does not change other objects
# So, it is safe to use whaterver the object you pass to it
(dtx2 <- default_dtx(dtf2))
class(dtx2)
# Any other object than data.frame, data.table or tbl_df is not converted
res <- default_dtx(1:5)
class(res)
# No conversion if the data frame is subclassed
dtf3 <- dtf2
class(dtf3) <- c("subclassed", "data.frame")
class(default_dtx(dtf3))

# data.table keys are converted into a 'key' attribute and back
```

```
library(data.table)
setkey(dtt2, 'x')
haskey(dtt2)
key(dtt2)

(dtf3 <- as_dtf(dtt2))
attributes(dtf3)
# Key is restored when converted back into a data.table (also from a tibble)
(dtt3 <- as_dtt(dtf3))
haskey(dtt3)
key(dtt3)

# Grouped tibbles are ungrouped with as_dtbl() or as_dtx()/default_dtx()!
mtcars |> dplyr::group_by(cyl) -> mtcars_grouped
class(mtcars_grouped)
mtcars2 <- as_dtbl(mtcars_grouped)
class(mtcars2)
```

`collect_dtx` *Force computation of a lazy tidyverse object*

Description

When {dplyr} or {tidyr} verbs are applied to a **data.table** or a database connection, they do not output data frames but objects like **dtplyr_step** or **tbl_sql** that are called lazy data frames. The actual process is triggered by using `as_dtx()`, or more explicitly with `dplyr::collect()` which coerces the result to a **tibble**. If you want the default {svBase} data frame object instead, use `collect_dtx()`, or if you want a specific object, use one of the other variants.

Usage

```
collect_dtx(x, ...)

collect_dtf(x, ...)

collect_dtt(x, ...)

collect_dtbl(x, ...)
```

Arguments

`x`	A data.frame, data.table, tibble or a lazy data frame (dtplyr_step, tbl_sql...).
`...`	Arguments passed on to methods for `dplyr::collect()`.

Value

A data frame (data.frame, data.table or tibble's tbl_df), the default version for `collect_dtx()`.

Examples

```
# Assuming the default data frame for svBase is a data.table
mtcars_dtt <- as_dtt(mtcars)
library(dplyr)
library(dtplyr)
# A lazy data frame, not a "real" data frame!
mtcars_dtt |> lazy_dt() |> select(mpg:disp) |> class()
# A data frame
mtcars |> select(mpg:disp) |> class()
# A data table
mtcars_dtt |> select(mpg:disp) |> class()
# A tibble, always!
mtcars_dtt |> lazy_dt() |> select(mpg:disp) |> collect() |> class()
# The data frame object you want, default one specified for svBase
mtcars_dtt |> lazy_dt() |> select(mpg:disp) |> collect_dtx() |> class()
```

---

dtx *Create a data frame (base's data.frame, data.table or tibble's tbl_df)*

Description

Create a data frame (base's data.frame, data.table or tibble's tbl_df)

Usage

```
dtx(..., .name_repair = c("check_unique", "unique", "universal", "minimal"))

dtbl(..., .name_repair = c("check_unique", "unique", "universal", "minimal"))

dtf(..., .name_repair = c("check_unique", "unique", "universal", "minimal"))

dtt(..., .name_repair = c("check_unique", "unique", "universal", "minimal"))
```

Arguments

`...`	A set of name-value pairs. The content of the data frame. See `tibble()` for more details on the way dynamic-dots are processed.
`.name_repair`	The way problematic column names are treated, see also `tibble()` for details.

Value

A data frame as a **tbl_df** object for `dtbl()`, a **data.frame** for `dtf()` and a **data.table** for `dtt()`.

Note

data.table and tibble's tbl_df do no use row names. However, you can add a column named `.rownames`(by default), or the name that is in `getOption("SciViews.dtx.rownames")` and it will be automatically set as row names when the object is converted into a data.frame with `as_dtf()`. For `dtf()`, just create a column of this name and it is directly used as row names for the resulting data.frame object.

Examples

```
dtbl1 <- dtbl(
  x = 1:5,
  y = rnorm(5),
  f = letters[1:5],
  l = sample(c(TRUE, FALSE), 5, replace = TRUE)
)
class(dtbl1)

dtf1 <- dtf(
  x = 1:5,
  y = rnorm(5),
  f = letters[1:5],
  l = sample(c(TRUE, FALSE), 5, replace = TRUE)
)
class(dtf1)

dtt1 <- dtt(
  x = 1:5,
  y = rnorm(5),
  f = letters[1:5],
  l = sample(c(TRUE, FALSE), 5, replace = TRUE))
class(dtt1)

# Using dtx(), one construct the preferred data frame object
# (a data.table by default, can be changed with options(SciViews.as_dtx = ...))
dtx1 <- dtx(
  x = 1:5,
  y = rnorm(5),
  f = letters[1:5],
  l = sample(c(TRUE, FALSE), 5, replace = TRUE))
class(dtx1) # data.table by default

# With svBase data.table and data.frame objects have the same nice print as tibbles
dtbl1
dtf1
dtt1

# Use tribble() inside dtx() to easily create a data frame:
library(tibble)
dtx2 <- dtx(tribble(
  ~x, ~y, ~f,
   1,  3, 'a',
   2,  4, 'b'
))
dtx2
class(dtx2)

# This is how you specify row names for dtf (data.frame)
dtf(x = 1:3, y = 4:6, .rownames = letters[1:3])
```

dtx_rows *Row-wise creation of a data frame*

Description

The presentation of the data (see examples) is easier to read than with the traditional column-wise entry in `dtx()`. This could be used to enter small tables in R, but do not abuse of it!

Usage

```
dtx_rows(...)

dtf_rows(...)

dtt_rows(...)

dtbl_rows(...)
```

Arguments

`...`	Specify the structure of the data frame by using formulas for variable names like `~x` for variable `x`. Then, use one argument per value in the data frame. It is possible to unquote with `!!` and to unquote-splice with `!!!`.

Value

A data frame of class **data.frame** for `dtf_rows()`, **data.table** for `dtt_rows()`, tibble **tbl_df** for `dtbl_rows()` and the default object with `dtx_rows()`.

Examples

```
df <- dtx_rows(
  ~x, ~y, ~group,
   1,  3,    "A",
   6,  2,    "A",
   10, 4,    "B"
)
df
```

`fstat_functions` *Fast (flexible and friendly) statistical functions (mainly from collapse) for matrix-like and data frame objects*

Description

The fast statistical function, or fast-flexible-friendly statistical functions are prefixed with "f". These vectorized functions supersede the no-f functions, bringing the capacity to work smoothly on matrix-like and data frame objects. Most of them are defined in the {collapse} package For instance, base `mean()` operates on a vector, but not on a data frame. A matrix is recognized as a vector and a single mean is returned. On, the contrary, `fmean()` calculates one mean per column. It does the same for a data frame, and it does so usually quicker than base functions. No need for `colMeans()`, a separate function to do so. Fast statistical functions also recognize grouping with `fgroup_by()`, `sgroup_by()` or `group_by()` and calculate the mean by group in this case. Again, no need for a different function like `stats::ave()`. Finally, these functions also have a `TRA=` argument that computes, for instance, if `TRA = "-"`, `(x  f(x))` very efficiently (for instance to calculate residuals by subtracting the mean). Another particularity is the `na.rm=` argument that is `TRUE` by default, while it is `FALSE` by default for `mean()`. These are generic functions with methods for **matrix**, **data.frame**, **grouped_df** and a **default** method used for simple numeric vectors. Most of them are defined in the {collapse} package, but there are a couple more here, together with an alternate syntax to replace `TRA=` with `%_f%`.

Usage

```
list_fstat_functions()

fn(x, ...)

fna(x, ...)

x %replacef% expr

x %replace_fillf% expr

x %-f% expr

x %+f% expr

x %-+f% expr

x %/f% expr

x %/*100f% expr

x %*f% expr

x %modf% expr
```

```
x %-modf% expr
```

Arguments

x	A numeric vector, matrix, data frame or grouped data frame (class 'grouped_df').
...	Further arguments passed to the method, like w=, a numeric vector of (non-negative) weights that may contain missing values, or TRA=, a quoted operator indicating the transformation to perform: "replace" to get a vector of same size of x with results, "replace_fill" idem but also replace missing data, "-" to subtract, "+" to add, "-+" to subtract and add the global statistic, "/" to divide, "%" to divide and multiply by 100 (percent), "*" to multiply, "%%" to take the modulus (remainder from division by the statistic) and "-%%" to subtract modulus ('i.e., to floor the data by the statistic), see collapse::TRA(). Also na.rm=, a logical indicating if we skip missing values in x if TRUE(by default). If FALSE for any missing data in x, NAis returned. For details and other arguments, see the corresponding help page in the collapse package.
expr	The expression to evaluate as RHS of the %__f% operators.

Value

The number of all observations for fn() or the number of missing observations for fna(). list_fstat_functions() returns a list of all the known fast statistical functions.

Note

The page collapse::fast-statistical-functions gives more details. fn() count all observations, including NAs, fna() counts only NAs, where fnobs() counts non-missing observations. Instead of TRA= one can use the %__f% functions where __ is replace, replace_fill, -, +, -+, /, /*100 for TRA="%", *, mod for TRA="%%", or -mod for TRA="-%%". See example.

Examples

```
library(collapse)
data(iris)
iris_num <- iris[, -5] # Only numerical variables
mean(iris$Sepal.Length) # OK, but mean(iris_num does not work)
colMeans(iris_num)
# Same
fmean(iris_num)
# Idem, but mean by group for all 4 numerical variables
iris |> fgroup_by(Species) |> fmean()
# Residuals (x - mean(x)) by group
iris |> fgroup_by(Species) |> fmean(TRA = "-")
# The same calculation, in a little bit more expressive way
iris |> fgroup_by(Species) %-f% fmean()
# or:
iris_num %-f% fmean(g = iris$Species)
```

is_dtx *Test if the object is a data frame (data.frame, data.table or tibble)*

Description

Test if the object is a data frame (data.frame, data.table or tibble)

Usage

```
is_dtx(x, strict = TRUE)

is_dtf(x, strict = TRUE)

is_dtt(x, strict = TRUE)

is_dtbl(x, strict = TRUE)
```

Arguments

x	An object
strict	Should this be strictly the corresponding class TRUE, by default, or could it be subclassed too (FALSE). With strict = TRUE, the **grouped_df** tibbles and **grouped_ts** tsibbles are also considered (tibbles or tsibbles where dplyr::group_by() was applied).

Value

These functions return TRUE if the object is of the correct class, otherwise they return FALSE. is_dtx() return TRUE if x is one of a data.frame, data.table or tibble.

Examples

```
# data(mtcars)
is_dtf(mtcars) # TRUE
is_dtx(mtcars) # Also TRUE
is_dtt(mtcars) # FALSE
is_dtbl(mtcars) # FALSE
# but...
is_dtt(as_dtt(mtcars)) # TRUE
is_dtx(as_dtt(mtcars)) # TRUE
is_dtbl(as_dtbl(mtcars)) # TRUE
is_dtx(as_dtbl(mtcars)) # TRUE
is_dtx(as_dtbl(mtcars) |> dplyr::group_by(cyl)) # TRUE (special case)

is_dtx("some string") # FALSE
```

speedy_functions *Speedy functions (mainly from collapse and data.table) to manipulate data frames*

Description

The Tidyverse defines a coherent set of tools to manipulate data frames that use a non-standard evaluation and sometimes require extra care. These functions, like `mutate()` or `summarise()` are defined in the {dplyr} and {tidyr} packages. The {collapse} package proposes a couple of functions with similar interface, but with different and much faster code. For instance, `fselect()` is similar to `select()`, or `fsummarise()` is similar to `summarise()`. Not all functions are implemented, arguments and argument names differ, and the behavior may be very different, like `frename()` which uses `old_name = new_name`, while `rename()` uses `new_name = old_name`! The speedy functions all are prefixed with an "s", like `smutate()`, and build on the work initiated in {collapse} to propose a series of paired functions with the tidy ones. So, `smutate()` and `mutate()` are "speedy" and 'tidy" counterparts and they are used in a very similar, if not identical way. This notation using a "s" prefix is there to draw the attention on their particularities. Their classes are **function** and **speedy_fn**. Avoid mixing tidy, speedy and non-tidy/speedy functions in the same pipeline. **This is a global page to present all the speedy functions in one place.** It is not meant to be a clear and detailed help page of all individual "s" functions. Please, refer to the corresponding help page of the non-"s" paired function for more details! You can use the {svMisc}'s `.?smutate` syntax to go to the help page of the non-"s" function with a message.

Usage

```
list_speedy_functions()

sgroup_by(.data, ...)

sungroup(.data, ...)

srename(.data, ...)

srename_with(.data, .fn, .cols = everything(), ...)

sfilter(.data, ...)

sfilter_ungroup(.data, ...)

sselect(.data, ...)

smutate(.data, ..., .keep = "all")

smutate_ungroup(.data, ..., .keep = "all")

stransmute(.data, ...)
```

```
stransmute_ungroup(.data, ...)

ssummarise(.data, ...)

sfull_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)

sleft_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)

sright_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)

sinner_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)

sbind_rows(..., .id = NULL)

scount(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = dplyr::group_by_drop_default(x),
  sort_cat = TRUE,
  decreasing = FALSE
)

stally(
  x,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  sort_cat = TRUE,
  decreasing = FALSE
)

sadd_count(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = NULL,
  sort_cat = TRUE,
  decreasing = FALSE
)

sadd_tally(
  x,
  wt = NULL,
```

```
  sort = FALSE,
  name = NULL,
  sort_cat = TRUE,
  decreasing = FALSE
)

sbind_cols(
  ...,
  .name_repair = c("unique", "universal", "check_unique", "minimal")
)

sarrange(.data, ..., .by_group = FALSE)

spull(.data, var = -1, name = NULL, ...)

sdistinct(.data, ..., .keep_all = FALSE)

sdrop_na(data, ...)

sreplace_na(data, replace, ...)

spivot_longer(data, cols, names_to = "name", values_to = "value", ...)

spivot_wider(data, names_from = name, values_from = value, ...)

suncount(data, weights, .remove = TRUE, .id = NULL)

sunite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)

sseparate(
  data,
  col,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  ...
)

sseparate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

sfill(data, ..., .direction = c("down", "up", "downup", "updown"))

sextract(
  data,
  col,
  into,
  regex = "([[:alnum:]]+)",
```

```
  remove = TRUE,
  convert = FALSE,
  ...
)
```

Arguments

.data	A data frame (data.frame, data.table or tibble's tbl_df)
...	Arguments dependent to the context of the function and most of the time, not evaluated in a standard way (cf. the tidyverse approach).
.fn	A function to use.
.cols	The list of the column where to apply the transformation. For the moment, only all existing columns, which means .cols = everything() is implemented
.keep	Which columns to keep. The default is "all", possible values are "used", "unused", or "none" (see mutate()).
x	A data frame (data.frame, data.table or tibble's tbl_df).
y	A second data frame.
by	A list of names of the columns to use for joining the two data frames.
suffix	The suffix to the column names to use to differentiate the columns that come from the first or the second data frame. By default it is c(".x", ".y").
copy	This argument is there for compatibility with the "t" matching functions, but it is not used here.
.id	The name of the column for the origin id, either names if all other arguments are named, or numbers.
wt	Frequency weights. Can be NULL or a variable. Use data masking.
sort	If TRUE largest group will be shown on top.
name	The name of the new column in the output (n by default, and no existing column must have this name, or an error is generated).4
.drop	Are levels with no observations dropped (TRUE by default).
sort_cat	Are levels sorted (TRUE by default).
decreasing	Is sorting done in decreasing order (FALSE by default)?
.name_repair	How should the name be "repaired" to avoid duplicate column names? See dplyr::bind_cols() for more details.
.by_group	Logical. If TRUE rows are first arranger by the grouping variables in any. FALSE by default.
var	A variable specified as a name, a positive or a negative integer (counting from the end). The default is -1 and returns last variable.
.keep_all	If TRUE keep all variables in .data.
data	A data frame, or for replace_na() a vector or a data frame.
replace	If data is a vector, a unique value to replace NAs, otherwise, a list of values, one per column of the data frame.
cols	A selection of the columns using tidy-select syntax, seetidyr::pivot_longer().

names_to	A character vector with the name or names of the columns for the names.
values_to	A string with the name of the column that receives the values.
names_from	The column or columns containing the names (use tidy selection and do not quote the names).
values_from	Idem for the column or columns that contain the values.
weights	A vector of weight to use to "uncount" data.
.remove	If TRUE, and weights is the name of a column, that column is removed from data.
col	The name quoted or not of the new column with united variable.
sep	Separator to use between values for united or separated columns.
remove	If TRUE the initial columns that are separated are also removed from data.
na.rm	If TRUE, NAs are eliminated before uniting the values.
into	Name of the new column to put separated variables. Use NA for items to drop.
convert	If 'TRUE resulting values are converted into numeric, integer or logical.
.direction	Direction in which to fill missing data: "down" (by default), "up", or "downup" (first down, then up), "updown" (the opposite).
regex	A regular expression used to extract the desired values (use one group with (and) for each element of into).

Value

See corresponding "non-s" function for the full help page with indication of the return values.

Note

The ssummarise() function does not support n() as does dplyr::summarise(). You can use fn() instead, but then, you must give a variable name as argument. The fn() alternative can also be used in summarise() for homogeneous syntax between the two. From {dplyr}, the slice() and slice_xxx() functions are not added yet because they are not available for {dbplyr}. Also anti_join(), semi_join() and nest_join() are not implemented yet. From {tidyr} expand(), chop(), unchop(), nest(), unnest(), unnest_longer(), unnest_wider(), hoist(), pack() and unpack() are not implemented yet.

Examples

```
# TODO...
```

tidy_functions *Tidy functions (mainly from dplyr and tidyr) to manipulate data frames*

Description

The Tidyverse defines a coherent set of tools to manipulate data frames that use a non-standard evaluation and sometimes require extra care. These functions, like `mutate()` or `summarise()` are defined in the {dplyr} and {tidyr} packages. When using variants, like {dtplyr} for **data.frame** objects, or {dbplyr} to work with external databases, successive commands in a pipeline are pooled together but not computed. One has to `collect()` the result to get its final form. Most of the tidy functions that have their "speedy" counterpart prefixed with "s" are listed with`list_tidy_functions()`. Their main usages are (excluding less used arguments, or those that are not compatibles with the speedy "s" counterpart functions):

- `group_by(.data, ...)`
- `ungroup(.data)`
- `rename(.data, ...)`
- `rename_with(.data, .fn, .cols = everything(), ...)`
- `filter(.data, ...)`
- `select(.data, ...)`
- `mutate(.data, ..., .keep = "all")`
- `transmute(.data, ...)`
- `summarise(.data, ...)`
- `full_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)`
- `left_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)`
- `right_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)`
- `inner_join(x, y, by = NULL, suffix = c(".x", ".y"), copy = FALSE, ...)`
- `bind_rows(..., .id = NULL)`
- `bind_cols(..., .name_repair = c("unique", "universal", "check_unique", "minimal"))`
- `arrange(.data, ..., .by_group = FALSE)`
- `count(x, ..., wt = NULL, sort = FALSE, name = NULL)`
- `tally(x, wt = NULL, sort = FALSE, name = NULL)`
- `add_count(x, ..., wt = NULL, sort = FALSE, name = NULL)`
- `add_tally(x, wt = NULL, sort = FALSE, name = NULL)`
- `pull(.data, var = -1, name = NULL)`
- `distinct(.data, ..., .keep_all = FALSE)`
- `drop_na(data, ...)`
- `replace_na(data, replace)`
- `pivot_longer(data, cols, names_to = "name", values_to = "value")`

- `pivot_wider(data, names_from = name, values_from = value)`
- `uncount(data, weights, .remove = TRUE, .id = NULL)`
- `unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)`
- `separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE)`
- `separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)`
- `fill(data, ..., .direction = c("down", "up", "downup", "updown"))`
- `extract(data, col, into, regex = "([[:alnum:]]+)", remove = TRUE, convert = FALSE)`

plus the functions defined here under.

Usage

```
list_tidy_functions()

filter_ungroup(.data, ...)

mutate_ungroup(.data, ..., .keep = "all")

transmute_ungroup(.data, ...)
```

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See `mutate()` for more details.
`...`	Arguments dependent to the context of the function and most of the time, not evaluated in a standard way (cf. the tidyverse approach).
`.keep`	Which columns to keep. The default is `"all"`, possible values are `"used"`, `"unused"`, or `"none"` (see `mutate()`).

Value

See corresponding "non-t" function for the full help page with indication of the return values. `list_tidy_functions()` returns a list of all the tidy(verse) functions that have their speedy "s" counterpart, see speedy_functions.

Note

The help page here is very basic and it aims mainly to list all the tidy functions. For more complete help, see the {dplyr} or {tidyr} packages. From {dplyr}, the `slice()` and `slice_xxx()` functions are not added yet because they are not available for {dbplyr}. Also `anti_join()`, `semi_join()` and `nest_join()` are not implemented yet. From {dplyr}, the `slice()` and `slice_xxx()` functions are not added yet because they are not available for {dbplyr}. Also `anti_join()`, `semi_join()` and `nest_join()` are not implemented yet. From {tidyr} `expand()`, `chop()`, `unchop()`, `nest()`, `unnest()`, `unnest_longer()`, `unnest_wider()`, `hoist()`, `pack()` and `unpack()` are not implemented yet.

See Also

`collapse::num_vars()` to easily keep only numeric columns from a data frame, `collapse::fscale()` for scaling and centering matrix-like objects and data frames.

Examples

```
# TODO...
```

Index