

Package ‘fastplyr’

October 4, 2024

Title Fast Alternatives to 'tidyverse' Functions

Version 0.2.0

Description A full set of fast data manipulation tools with a tidy front-end and a fast back-end using 'collapse' and 'cheapr'.

License MIT + file LICENSE

BugReports <https://github.com/NicChr/fastplyr/issues>

Depends R (>= 3.6.1)

Imports cheapr (>= 0.9.8), collapse (>= 2.0.0), dplyr (>= 1.1.0), magrittr, rlang, stringr, tidyselect, vctrs (>= 0.6.0)

Suggests nycflights13, testthat (>= 3.0.0), tidyr

LinkingTo cpp11

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Nick Christofides [aut, cre] (<<https://orcid.org/0000-0002-9743-7342>>)

Maintainer Nick Christofides <nick.christofides.r@gmail.com>

Repository CRAN

Date/Publication 2024-10-04 09:40:05 UTC

Contents

fastplyr-package	2
desc	3
f_arrange	3
f_bind_rows	4
f_consecutive_id	4
f_count	5
f_distinct	6
f_duplicates	7

f_expand	8
f_filter	9
f_group_by	10
f_left_join	11
f_select	13
f_slice	14
f_summarise	16
group_by_order_default	18
group_id	18
new_tbl	20
tidy_quantiles	21
Index	23

fastplyr-package *fastplyr: Fast Alternatives to 'tidyverse' Functions*

Description

fastplyr is a tidy front-end using a faster and more efficient back-end based on two packages, collapse and cheapr.

fastplyr includes dplyr and tidyr alternatives that behave like their tidyverse equivalents but are more efficient.

Similar in spirit to the excellent tidytable package, fastplyr also offers a tidy front-end that is fast and easy to use. Unlike tidytable, fastplyr verbs are interchangeable with dplyr verbs.

You can learn more about the tidyverse, collapse and cheapr using the links below.

[tidyverse](#)

[collapse](#)

[cheapr](#)

Author(s)

Maintainer: Nick Christofides <nick.christofides.r@gmail.com> ([ORCID](#))

See Also

Useful links:

- Report bugs at <https://github.com/NicChr/fastplyr/issues>

desc *Helpers to sort variables in ascending or descending order*

Description

An alternative to `dplyr::desc()` which is much faster for character vectors and factors.

Usage

```
desc(x)
```

Arguments

x Vector.

Value

A numeric vector that can be ordered in ascending or descending order.
Useful in `dplyr::arrange()` or `f_arrange()`.

f_arrange *A collapse version of dplyr::arrange()*

Description

This is a fast and near-identical alternative to `dplyr::arrange()` using the `collapse` package.
`desc()` is like `dplyr::desc()` but works faster when called directly on vectors.

Usage

```
f_arrange(data, ..., .by = NULL, .by_group = FALSE, .cols = NULL)
```

Arguments

data A data frame.
... Variables to arrange by.
.by (Optional). A selection of columns to group by for this operation. Columns are specified using `tidyselect`.
.by_group If TRUE the sorting will be first done by the group variables.
.cols (Optional) alternative to ... that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.

Value

A sorted `data.frame`.

f_bind_rows	<i>Bind data frame rows and columns</i>
-------------	---

Description

Faster bind rows and columns.

Usage

```
f_bind_rows(..., .fill = TRUE)
```

```
f_bind_cols(..., .repair_names = TRUE, .recycle = TRUE, .sep = "...")
```

Arguments

...	Data frames to bind.
.fill	Should missing columns be filled with NA? Default is TRUE.
.repair_names	Should duplicate column names be made unique? Default is TRUE.
.recycle	Should inputs be recycled to a common row size? Default is TRUE.
.sep	Separator to use for creating unique column names.

Value

f_bind_rows() performs a union of the data frames specified via ... and joins the rows of all the data frames, without removing duplicates.

f_bind_cols() joins the columns, creating unique column names if there are any duplicates by default.

f_consecutive_id	<i>Consecutive IDs</i>
------------------	------------------------

Description

f_consecutive_id(), an alternative to dplyr::consecutive_id() creates an integer vector with values in the range [1, n + 1) where n is the length of the vector or number of rows of the data frame. The ID increments every time $x[i] \neq x[i - 1]$ thus giving information on when there is a change in value.

Usage

```
f_consecutive_id(x)
```

Arguments

x	A vector or data frame.
---	-------------------------

Details

'ALTREP' compact sequences are supported as well.

To mimic `dplyr::consecutive_id()` where multiple variables are selected, just use `dplyr::pick()`.

`f_consecutive_id` has a smaller overhead and thus should be faster when called many times, e.g. when using a `grouped_df` with many groups.

Value

An integer vector of consecutive run IDs in the range $[1, n + 1)$.

f_count	<i>A fast replacement to dplyr::count()</i>
---------	---

Description

Near-identical alternative to `dplyr::count()`.

Usage

```
f_count(
  data,
  ...,
  wt = NULL,
  sort = FALSE,
  order = df_group_by_order_default(data),
  name = NULL,
  .by = NULL,
  .cols = NULL
)
```

```
f_add_count(
  data,
  ...,
  wt = NULL,
  sort = FALSE,
  order = df_group_by_order_default(data),
  name = NULL,
  .by = NULL,
  .cols = NULL
)
```

Arguments

data	A data frame.
...	Variables to group by.
wt	Frequency weights. Can be NULL or a variable:

- If NULL (the default), counts the number of rows in each group.
- If a variable, computes `sum(wt)` for each group.

<code>sort</code>	If TRUE, will show the largest groups at the top.
<code>order</code>	Should the groups be calculated as ordered groups? If FALSE, this will return the groups in order of first appearance, and in many cases is faster. If TRUE (the default), the groups are returned in sorted order, exactly the same way as <code>dplyr::count</code> .
<code>name</code>	The name of the new column in the output. If there's already a column called <code>n</code> , it will use <code>nn</code> . If there's a column called <code>n</code> and <code>nn</code> , it'll use <code>nnn</code> , and so on, adding <code>ns</code> until it gets a new name.
<code>.by</code>	(Optional). A selection of columns to group by for this operation. Columns are specified using <code>tidy-select</code> .
<code>.cols</code>	(Optional) alternative to <code>...</code> that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.

Details

This is a fast and near-identical alternative to `dplyr::count()` using the `collapse` package. Unlike `collapse::fcount()`, this works very similarly to `dplyr::count()`. The only main difference is that anything supplied to `wt` is recycled and added as a data variable. Other than that everything works exactly as the `dplyr` equivalent.

`f_count()` and `f_add_count()` can be up to >100x faster than the `dplyr` equivalents.

Value

A data frame of frequency counts by group.

<code>f_distinct</code>	<i>Find distinct rows</i>
-------------------------	---------------------------

Description

Like `dplyr::distinct()` but faster when lots of groups are involved.

Usage

```
f_distinct(
  data,
  ...,
  .keep_all = FALSE,
  sort = FALSE,
  order = sort,
  .by = NULL,
  .cols = NULL
)
```

Arguments

data	A data frame.
...	Variables used to find distinct rows.
.keep_all	If TRUE then all columns of data frame are kept, default is FALSE.
sort	Should result be sorted? Default is FALSE. When order = FALSE this option has no effect on the result.
order	Should the groups be calculated as ordered groups? Setting to TRUE may sometimes offer a speed benefit, but usually this is not the case. The default is FALSE.
.by	(Optional). A selection of columns to group by for this operation. Columns are specified using tidy-select.
.cols	(Optional) alternative to ... that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.

Value

A data.frame of distinct groups.

f_duplicates	<i>Find duplicate rows</i>
--------------	----------------------------

Description

Find duplicate rows

Usage

```
f_duplicates(
  data,
  ...,
  .keep_all = FALSE,
  .both_ways = FALSE,
  .add_count = FALSE,
  .drop_empty = FALSE,
  sort = FALSE,
  .by = NULL,
  .cols = NULL
)
```

Arguments

data	A data frame.
...	Variables used to find duplicate rows.
.keep_all	If TRUE then all columns of data frame are kept, default is FALSE.

<code>.both_ways</code>	If TRUE then duplicates and non-duplicate first instances are retained. The default is FALSE which returns only duplicate rows. Setting this to TRUE can be particularly useful when examining the differences between duplicate rows.
<code>.add_count</code>	If TRUE then a count column is added to denote the number of duplicates (including first non-duplicate instance). The naming convention of this column follows <code>dplyr::add_count()</code> .
<code>.drop_empty</code>	If TRUE then empty rows with all NA values are removed. The default is FALSE.
<code>sort</code>	Should result be sorted? If FALSE (the default), then rows are returned in the exact same order as they appear in the data. If TRUE then the duplicate rows are sorted.
<code>.by</code>	(Optional). A selection of columns to group by for this operation. Columns are specified using <code>tidy-select</code> .
<code>.cols</code>	(Optional) alternative to <code>...</code> that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.

Details

This function works like `dplyr::distinct()` in its handling of arguments and data-masking but returns duplicate rows. In certain situations it can be much faster than `data %>% group_by() %>% filter(n() > 1)` when there are many groups.

Value

A data.frame of duplicate rows.

See Also

[f_count](#) [f_distinct](#)

f_expand

Fast versions of `tidyr::expand()` and `tidyr::complete()`.

Description

Fast versions of `tidyr::expand()` and `tidyr::complete()`.

Usage

```
f_expand(data, ..., sort = FALSE, .by = NULL, .cols = NULL)
```

```
f_complete(data, ..., sort = FALSE, .by = NULL, .cols = NULL, fill = NA)
```

```
crossing(..., sort = FALSE)
```

```
nesting(..., sort = FALSE)
```


Arguments

data	A data frame
...	Variables to expand
sort	Logical. If TRUE expanded/completed variables are sorted. The default is FALSE.
.by	(Optional). A selection of columns to group by for this operation. Columns are specified using tidy-select.
.cols	(Optional) alternative to ... that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.
fill	A named list containing value-name pairs to fill the named implicit missing values.

Details

crossing and nesting are helpers that are basically identical to tidyr's crossing and nesting.

Value

A data.frame of expanded groups.

f_filter	<i>Alternative to dplyr::filter()</i>
----------	---------------------------------------

Description

Alternative to dplyr::filter()

Usage

```
f_filter(data, ..., .by = NULL)
```

Arguments

data	A data frame.
...	Expressions used to filter the data frame with.
.by	(Optional). A selection of columns to group by for this operation. Columns are specified using tidy-select.

Value

A filtered data frame.

f_group_by	<i>'collapse' version of dplyr::group_by()</i>
------------	--

Description

This works the exact same as `dplyr::group_by()` and typically performs around the same speed but uses slightly less memory.

Usage

```
f_group_by(
  data,
  ...,
  .add = FALSE,
  order = df_group_by_order_default(data),
  .by = NULL,
  .cols = NULL,
  .drop = df_group_by_drop_default(data)
)

group_ordered(data)
```

Arguments

<code>data</code>	data frame.
<code>...</code>	Variables to group by.
<code>.add</code>	Should groups be added to existing groups? Default is FALSE.
<code>order</code>	Should groups be ordered? If FALSE groups will be ordered based on first-appearance. Typically, setting order to FALSE is faster.
<code>.by</code>	(Optional). A selection of columns to group by for this operation. Columns are specified using <code>tidyselect</code> .
<code>.cols</code>	(Optional) alternative to <code>...</code> that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.
<code>.drop</code>	Should unused factor levels be dropped? Default is TRUE.

Details

`f_group_by()` works almost exactly like the `'dplyr'` equivalent. An attribute "ordered" (TRUE or FALSE) is added to the group data to signify if the groups are sorted or not.

Ordered vs Sorted:

The distinction between ordered and sorted is somewhat subtle. Functions in `fastplyr` that use a `sort` argument generally refer to the top-level dataset being sorted in some way, either by sorting the group columns like in `f_expand()` or `f_distinct()`, or some other columns, like the count column in `f_count()`.

The `order` argument, when set to `TRUE` (the default), is used to mean that the group data will be calculated using a sort-based algorithm, leading to sorted group data. When `order` is `FALSE`, the group data will be returned based on the order-of-first appearance of the groups in the data. This order-of-first appearance may still naturally be sorted depending on the data. For example, `group_id(1:3, order = T)` results in the same group IDs as `group_id(1:3, order = F)` because 1, 2, and 3 appear in the data in ascending sequence whereas `group_id(3:1, order = T)` does not equal `group_id(3:1, order = F)`.

Part of the reason for the distinction is that internally `fastplyr` can in theory calculate group data using the sort-based algorithm and still return unsorted groups, though this combination is only available to the user in limited places like `f_distinct(order = TRUE, sort = FALSE)`.

The other reason is to prevent confusion in the meaning of `sort` and `order` so that `order` always refers to the algorithm specified, resulting in sorted groups, and `sort` implies a physical sorting of the returned data. It's also worth mentioning that in most functions, `sort` will implicitly utilise the sort-based algorithm specified via `order = TRUE`.

Using the order-of-first appearance algorithm for speed:

In many situations (not all) it can be faster to use the order-of-first appearance algorithm, specified via `order = FALSE`.

This can generally be accessed by first calling `f_group_by(data, ..., order = FALSE)` and then performing your calculations.

To utilise this algorithm more globally and package-wide, set the `'fastplyr.order.groups'` option to `FALSE` using the code: `options(fastplyr.order.groups = FALSE)`.

Value

`f_group_by()` returns a `grouped_df` that can be used for further for grouped calculations.

`group_ordered()` returns `TRUE` if the group data are sorted, i.e if `attr(attr(data, "groups"), "ordered") == TRUE`. If sorted, which is usually the default, this leads to summary calculations like `f_summarise()` or `dplyr::summarise()` producing sorted groups. If `FALSE` they are returned based on order-of-first appearance in the data.

f_left_join

Fast SQL joins

Description

Mostly a wrapper around `collapse::join()` that behaves more like `dplyr`'s joins. List columns, lubridate intervals and `vctrs` `rcrds` work here too.

Usage

```
f_left_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
```

```
    multiple = TRUE,  
    keep = FALSE,  
    ...  
  )  
  
  f_right_join(  
    x,  
    y,  
    by = NULL,  
    suffix = c(".x", ".y"),  
    multiple = TRUE,  
    keep = FALSE,  
    ...  
  )  
  
  f_inner_join(  
    x,  
    y,  
    by = NULL,  
    suffix = c(".x", ".y"),  
    multiple = TRUE,  
    keep = FALSE,  
    ...  
  )  
  
  f_full_join(  
    x,  
    y,  
    by = NULL,  
    suffix = c(".x", ".y"),  
    multiple = TRUE,  
    keep = FALSE,  
    ...  
  )  
  
  f_anti_join(  
    x,  
    y,  
    by = NULL,  
    suffix = c(".x", ".y"),  
    multiple = TRUE,  
    keep = FALSE,  
    ...  
  )  
  
  f_semi_join(  
    x,  
    y,
```

```

    by = NULL,
    suffix = c(".x", ".y"),
    multiple = TRUE,
    keep = FALSE,
    ...
)

f_cross_join(x, y, suffix = c(".x", ".y"), ...)

f_union_all(x, y, ...)

f_union(x, y, ...)

```

Arguments

x	Left data frame.
y	Right data frame.
by	character(1) - Columns to join on.
suffix	character(2) - Suffix to paste onto common cols between x and y in the joined output.
multiple	logical(1) - Should multiple matches be returned? If FALSE the first match in y is used. Default is TRUE.
keep	logical(1) - Should join columns from both data frames be kept? Default is FALSE.
...	Additional arguments passed to collapse::join().

Value

A joined data frame, joined on the columns specified with by, using an equality join.

f_cross_join() returns all possible combinations between the two data frames.

f_select	<i>Fast</i> dplyr::select()/dplyr::rename()
----------	---

Description

f_select() operates the exact same way as dplyr::select() and can be used naturally with tidy-select helpers. It uses collapse to perform the actual selecting of variables and is considerably faster than dplyr for selecting exact columns, and even more so when supplying the .cols argument.

Usage

```
f_select(data, ..., .cols = NULL)
```

```
f_rename(data, ..., .cols = NULL)
```

Arguments

data	A data frame.
...	Variables to select using tidy-select. See <code>?dplyr::select</code> for more info.
.cols	(Optional) faster alternative to ... that accepts a named character vector or numeric vector. No checks on duplicates column names are done when using .cols. If speed is an expensive resource, it is recommended to use this.

Value

A data.frame of selected columns.

f_slice	<i>Faster</i> <code>dplyr::slice()</code>
---------	---

Description

When there are lots of groups, the `f_slice()` functions are much faster.

Usage

```
f_slice(data, i = 0L, ..., .by = NULL, keep_order = FALSE)
f_slice_head(data, n, prop, .by = NULL, keep_order = FALSE)
f_slice_tail(data, n, prop, .by = NULL, keep_order = FALSE)

f_slice_min(
  data,
  order_by,
  n,
  prop,
  .by = NULL,
  with_ties = TRUE,
  na_rm = FALSE,
  keep_order = FALSE
)

f_slice_max(
  data,
  order_by,
  n,
  prop,
  .by = NULL,
  with_ties = TRUE,
  na_rm = FALSE,
```

```

    keep_order = FALSE
  )

  f_slice_sample(
    data,
    n,
    replace = FALSE,
    prop,
    .by = NULL,
    keep_order = FALSE,
    weights = NULL,
    seed = NULL
  )

```

Arguments

data	A data frame.
i	An integer vector of slice locations. Please see the details below on how <code>i</code> works as it only accepts simple integer vectors.
...	A temporary argument to give the user an error if dots are used.
.by	(Optional). A selection of columns to group by for this operation. Columns are specified using tidy-select.
keep_order	Should the sliced data frame be returned in its original order? The default is FALSE.
n	Number of rows.
prop	Proportion of rows.
order_by	Variables to order by.
with_ties	Should ties be kept together? The default is TRUE.
na_rm	Should missing values in <code>f_slice_max()</code> and <code>f_slice_min()</code> be removed? The default is FALSE.
replace	Should <code>f_slice_sample()</code> sample with or without replacement? Default is FALSE, without replacement.
weights	Probability weights used in <code>f_slice_sample()</code> .
seed	Seed number defining RNG state. If supplied, this is only applied locally within the function and the seed state isn't retained after sampling. To clarify, whatever seed state was in place before the function call, is restored to ensure seed continuity. If left NULL (the default), then the seed is never modified.

Details

Important note about the `i` argument in `f_slice`:

`i` is first evaluated on an un-grouped basis and then searches for those locations in each group. Thus if you supply an expression of slice locations that vary by-group, this will not be respected nor checked. For example,

```
do f_slice(data, 10:20, .by = group)
not f_slice(data, sample(1:10), .by = group).
```

The former results in slice locations that do not vary by group but the latter will result in different within-group slice locations which `f_slice` cannot correctly compute.

To do the the latter type of by-group slicing, use `f_filter`, e.g.
`f_filter(data, row_number() %in% slices, .by = groups)` or even faster:
`library(cheapr)`
`f_filter(data, row_number() %in_% slices, .by = groups)`

`f_slice_sample`:

The arguments of `f_slice_sample()` align more closely with `base::sample()` and thus by default re-samples each entire group without replacement.

Value

A data.frame filtered on the specified row indices.

f_summarise	<i>Summarise each group down to one row</i>
-------------	---

Description

Like `dplyr::summarise()` but with some internal optimisations for common statistical functions.

Usage

```
f_summarise(data, ..., .by = NULL, .optimise = TRUE)
```

```
f_summarize(data, ..., .by = NULL, .optimise = TRUE)
```

Arguments

data	A data frame.
...	Name-value pairs of summary functions. Expressions with <code>across()</code> are also accepted.
.by	(Optional). A selection of columns to group by for this operation. Columns are specified using <code>tidy-select</code> .
.optimise	(Optionally) turn off optimisations for common statistical functions by setting to <code>FALSE</code> . Default is <code>TRUE</code> which uses optimisations.

Details

f_summarise behaves mostly like `dplyr::summarise` except that expressions supplied to `...` are evaluated independently.

Optimised statistical functions:

Some functions are internally optimised using 'collapse' fast statistical functions. This makes execution on many groups very fast.

For fast quantiles (percentiles) by group, see [tidy_quantiles](#)

List of currently optimised functions and their equivalent 'collapse' function

```
base::sum -> collapse::fsum
base::prod -> collapse::fprod
base::min -> collapse::fmin
base::max -> collapse::fmax
stats::mean -> collapse::fmean
stats::median -> collapse::fmedian
stats::sd -> collapse::fsd
stats::var -> collapse::fvar
dplyr::first -> collapse::ffirst
dplyr::last -> collapse::flast
dplyr::n_distinct -> collapse::fndistinct
```

Value

An un-grouped data frame of summaries by group.

See Also

[tidy_quantiles](#)

Examples

```
library(fastplyr)
library(nycflights13)

# Number of flights per month, including first and last day
flights %>%
  f_group_by(year, month) %>%
  f_summarise(first_day = first(day),
              last_day = last(day),
              num_flights = n())

## Fast mean summary using `across()`

flights %>%
  f_summarise(
    across(where(is.double), mean),
    .by = tailnum
  )
```

```
# To ignore or keep NAs, use collapse::set_collapse(na.rm)
collapse::set_collapse(na.rm = FALSE)
flights %>%
  summarise(
    across(where(is.double), mean),
    .by = origin
  )
collapse::set_collapse(na.rm = TRUE)
```

group_by_order_default

Default value for ordering of groups

Description

A default value, TRUE or FALSE that controls which algorithm to use for calculating groups. See [f_group_by](#) for more details.

Usage

```
group_by_order_default(x)
```

Arguments

x A data frame.

Value

A logical of length 1, either TRUE or FALSE.

group_id

Fast group and row IDs

Description

These are tidy-based functions for calculating group IDs and row IDs.

- `group_id()` returns an integer vector of group IDs the same size as the data.
- `row_id()` returns an integer vector of row IDs.

The `add_` variants add a column of group IDs/row IDs.

Usage

```

group_id(data, order = TRUE, ascending = TRUE, as_qg = FALSE)

add_group_id(
  data,
  ...,
  order = TRUE,
  ascending = TRUE,
  .by = NULL,
  .cols = NULL,
  .name = NULL,
  as_qg = FALSE
)

## S3 method for class 'data.frame'
add_group_id(
  data,
  ...,
  order = df_group_by_order_default(data),
  ascending = TRUE,
  .by = NULL,
  .cols = NULL,
  .name = NULL,
  as_qg = FALSE
)

row_id(data, ascending = TRUE)

## S3 method for class 'GRP'
row_id(data, ascending = TRUE)

add_row_id(data, ..., ascending = TRUE, .by = NULL, .cols = NULL, .name = NULL)

## S3 method for class 'data.frame'
add_row_id(data, ..., ascending = TRUE, .by = NULL, .cols = NULL, .name = NULL)

```

Arguments

data	A data frame or vector.
order	Should the groups be ordered? THE PHYSICAL ORDER OF THE DATA IS NOT CHANGED. When order is TRUE (the default) the group IDs will be ordered but not sorted.
ascending	If FALSE the order of the group IDs will be based on first appearance. Should the group order be ascending or descending? The default is TRUE. For row_id() this determines if the row IDs are increasing or decreasing. NOTE - When order = FALSE, the ascending argument is ignored. This is something that will be fixed in a later version.

as_qg	Should the group IDs be returned as a collapse "qG" class? The default (FALSE) always returns an integer vector.
...	Additional groups using tidy data-masking rules. To specify groups using tidycselect, simply use the .by argument.
.by	Alternative way of supplying groups using tidycselect notation.
.cols	(Optional) alternative to ... that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.
.name	Name of the added ID column which should be a character vector of length 1. If .name = NULL (the default), add_group_id() will add a column named "group_id", and if one already exists, a unique name will be used.

Value

An integer vector.

new_tbl	<i>Fast 'tibble' alternatives</i>
---------	-----------------------------------

Description

Fast 'tibble' alternatives

Usage

```
new_tbl(..., .nrows = NULL, .recycle = TRUE, .name_repair = FALSE)
```

```
f_enframe(x, name = "name", value = "value")
```

```
f_deframe(x)
```

```
as_tbl(x)
```

Arguments

...	Key-value pairs.
.nrows	integer(1) (Optional) number of rows. Commonly used to initialise a 0-column data frame with rows.
.recycle	logical(1) Should arguments be recycled? Default is FALSE.
.name_repair	logical(1) Should duplicate names be made unique? Default is TRUE.
x	A data frame or vector.
name	character(1) Name to use for column of names.
value	character(1) Name to use for column of values.

Details

`new_ttbl` and `as_ttbl` are alternatives to `tibble` and `as_tibble` respectively. One of the main reasons that these do not share the same name prefixed with `f_` is because they don't always return the same result. For example `new_ttbl()` does not support 'quosures' and tidy injection.

`f_enframe(x)` where `x` is a `data.frame` converts `x` into a tibble of column names and list-values.

Value

A tibble or vector.

tidy_quantiles	<i>Fast grouped sample quantiles</i>
----------------	--------------------------------------

Description

Fast grouped sample quantiles

Usage

```
tidy_quantiles(
  data,
  ...,
  probs = seq(0, 1, 0.25),
  type = 7,
  pivot = c("wide", "long"),
  na.rm = TRUE,
  .by = NULL,
  .cols = NULL,
  .drop_groups = TRUE
)
```

Arguments

<code>data</code>	A data frame.
<code>...</code>	<data-masking> Variables to calculate quantiles for.
<code>probs</code>	<code>numeric(n)</code> - Quantile probabilities.
<code>type</code>	<code>integer(1)</code> - Quantile type, see <code>?collapse::fquantile</code>
<code>pivot</code>	<code>character(1)</code> - Pivot result wide or long? Default is "wide".
<code>na.rm</code>	<code>logical(1)</code> Should NA values be ignored? Default is TRUE.
<code>.by</code>	(Optional). A selection of columns to group by for this operation. Columns are specified using tidy-select.
<code>.cols</code>	(Optional) alternative to <code>...</code> that accepts a named character vector or numeric vector. If speed is an expensive resource, it is recommended to use this.
<code>.drop_groups</code>	<code>logical(1)</code> Should groups be dropped after calculation? Default is TRUE.

Value

A data frame of sample quantiles.

Examples

```
library(fastplyr)
library(dplyr)
groups <- 1 * 2^(0:10)

# Normal distributed samples by group using the group value as the mean
# and sqrt(groups) as the sd

samples <- tibble(groups) %>%
  reframe(x = rnorm(100, mean = groups, sd = sqrt(groups)), .by = groups) %>%
  f_group_by(groups)

# Fast means and quantiles by group

quantiles <- samples %>%
  tidy_quantiles(x)

means <- samples %>%
  f_summarise(mean = mean(x))

means %>%
  f_left_join(quantiles)
```

Index

`add_group_id` (`group_id`), 18
`add_row_id` (`group_id`), 18
`as_tbl` (`new_tbl`), 20

`crossing` (`f_expand`), 8

`desc`, 3

`f_add_count` (`f_count`), 5
`f_anti_join` (`f_left_join`), 11
`f_arrange`, 3
`f_bind_cols` (`f_bind_rows`), 4
`f_bind_rows`, 4
`f_complete` (`f_expand`), 8
`f_consecutive_id`, 4
`f_count`, 5, 8
`f_cross_join` (`f_left_join`), 11
`f_deframe` (`new_tbl`), 20
`f_distinct`, 6, 8
`f_duplicates`, 7
`f_enframe` (`new_tbl`), 20
`f_expand`, 8
`f_filter`, 9
`f_full_join` (`f_left_join`), 11
`f_group_by`, 10, 18
`f_inner_join` (`f_left_join`), 11
`f_left_join`, 11
`f_rename` (`f_select`), 13
`f_right_join` (`f_left_join`), 11
`f_select`, 13
`f_semi_join` (`f_left_join`), 11
`f_slice`, 14
`f_slice_head` (`f_slice`), 14
`f_slice_max` (`f_slice`), 14
`f_slice_min` (`f_slice`), 14
`f_slice_sample` (`f_slice`), 14
`f_slice_tail` (`f_slice`), 14
`f_summarise`, 16
`f_summarize` (`f_summarise`), 16
`f_union` (`f_left_join`), 11
`f_union_all` (`f_left_join`), 11
`fastplyr` (`fastplyr-package`), 2
`fastplyr-package`, 2

`group_by_order_default`, 18
`group_id`, 18
`group_ordered` (`f_group_by`), 10

`integer`, 15

`nesting` (`f_expand`), 8
`new_tbl`, 20

`row_id` (`group_id`), 18

`tidy_quantiles`, 17, 21