Package ‘poorman’

May 10, 2020

Type Package
Title A Poor Man's Base R Copy of 'dplyr' Verbs
Version 0.2.0
Maintainer Nathan Eastwood <nathan.eastwood@icloud.com>
Description A simple replication of key 'dplyr' verbs using only base R.
URL https://github.com/nathaneastwood/poorman
BugReports https://github.com/nathaneastwood/poorman/issues
Depends R (>= 3.4)
Suggests knitr, roxygen2, tinytest
License MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.0.2
NeedsCompilation no
Author Nathan Eastwood [aut, cre]
Repository CRAN
Date/Publication 2020-05-10 16:30:03 UTC

R topics documented:

arrange ................................................................. 2
between ............................................................... 3
count ................................................................. 3
desc ................................................................. 5
filter ................................................................. 5
filter_joins ......................................................... 6
get_groups ....................................................... 7
groups ........................................................... 8
if_else ............................................................. 9
joins .............................................................. 9
**Description**

Order rows of a `data.frame` by an expression involving its variables.

**Usage**

```r
arrange(.data, ...)
```

**Arguments**

- `.data` A `data.frame`
- `...` A comma separated vector of unquoted name(s) to order the data by.

**Value**

A `data.frame`.

**Examples**

```r
arrange(mtcars, mpg)
mtcars %>% arrange(mpg)
mtcars %>% arrange(cyl, mpg)
```
between

Do values in a numeric vector fall in specified range?

Description
This is a shortcut for \( x \geq \text{left} \& x \leq \text{right} \).

Usage
\[
\text{between}(x, \text{left}, \text{right})
\]

Arguments

\( x \)  
A numeric vector of values.

\( \text{left}, \text{right} \)  
Boundary values.

Value
A logical vector the same length as \( x \).

Examples
\[
\text{between}(1:12, 7, 9)
\]
\[
x \leftarrow \text{rnorm}(1e2)
x[\text{between}(x, -1, 1)]
\]

count

Count observations by group

Description
count() lets you quickly count the unique values of one or more variables: \( \text{df} \%\% \text{count}(a,b) \) is roughly equivalent to \( \text{df} \%\% \text{group_by}(a,b) \%\% \text{summarise}(n = n()) \). count() is paired with tally(), a lower-level helper that is equivalent to \( \text{df} \%\% \text{summarise}(n = n()) \). Supply wt to perform weighted counts, switching the summary from \( n = n() \) to \( n = \text{sum(wt)} \). add_count() and add_tally() are equivalent to count() and tally() but use mutate() instead of summarise() so that they add a new column with group-wise counts.
Usage

```
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)
```

Arguments

- `x` A `data.frame`.
- `...` Variables to group by.
- `wt` If omitted, will count the number of rows. If specified, will perform a "weighted" count by summing the (non-missing) values of variable `wt`. If omitted, and column `n` exists, it will automatically be used as a weighting variable, although you will have to specify `name` to provide a new name for the output.
- `sort` logical(1). If `TRUE`, will show the largest groups at the top.
- `name` character(1). The name of the new column in the output. If omitted, it will default to `n`. If there's already a column called `n`, it will error, and require you to specify the name.

Value

A `data.frame`. `count()` and `add_count()` have the same groups as the input.

Examples

```
# count() is a convenient way to get a sense of the distribution of
# values in a dataset
mtcars %>% count(cyl)
mtcars %>% count(cyl, sort = TRUE)
mtcars %>% count(cyl, am, sort = TRUE)
# Note that if the data are already grouped, count() adds an additional grouping variable
# which is removed afterwards
mtcars %>% group_by(gear) %>% count(cyl)

# tally() is a lower-level function that assumes you've done the grouping
mtcars %>% tally()
mtcars %>% group_by(cyl) %>% tally()

# both count() and tally() have add_ variants that work like mutate() instead of summarise
mtcars %>% add_count(cyl, wt = am)
mtcars %>% add_tally(wt = am)
```
**desc**

**Descending order**

Description

Transform a vector into a format that will be sorted in descending order. This is useful within `arrange()`.

Usage

```r
desc(x)
```

Arguments

- `x` A vector to transform.

Value

A vector of the same length as `x`.

Examples

```r
desc(1:10)
desc(factor(letters))
```

```r
first_day <- seq(as.Date("1910/1/1"), as.Date("1920/1/1"), "years")
desc(first_day)
```

```r
mtcars %>% arrange(desc(mpg))
```

**filter**

**Return rows with matching conditions**

Description

Use `filter()` to choose rows/cases where conditions are `TRUE`.

Usage

```r
filter(.data, ...)
```

Arguments

- `.data` A `data.frame`.
- `...` Logical predicates defined in terms of the variables in `.data`. Multiple conditions are combined with `&`. Arguments within `...` are automatically quoted and evaluated within the context of the `data.frame`. 

Value

A data.frame.

Useful filter functions

- `==`, `>`, `>=`, etc.
- `&`, `|`, `!`, `xor()`
- `is.na()`

Examples

```r
filter(mtcars, am == 1)
mtcars %>% filter(cyl == 4)
mtcars %>% filter(cyl <= 5 & am > 0)
mtcars %>% filter(cyl == 4 | cyl == 8)
mtcars %>% filter(!(cyl %in% c(4, 6)), am != 0)
```

filter_joins

Filtering joins filter rows from x based on the presence or absence of matches in y:

Description

- `semi_join()` return all rows from x with a match in y.
- `anti_join()` return all rows from x without a match in y.

Usage

```r
anti_join(x, y, by = NULL)
semi_join(x, y, by = NULL)
```

Arguments

- `x`, `y` The data.frames to join.
- `by` A character vector of variables to join by. If NULL, the default, * _join() will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're right (to suppress the message, simply explicitly list the variables that you want to join).
**Examples**

```r
table1 <- data.frame(
  pupil = rep(1:3, each = 2),
  test = rep(c("A", "B"), 3),
  score = c(60, 70, 65, 80, 85, 70),
  stringsAsFactors = FALSE
)
table2 <- table1[c(1, 3, 4), ]

table1 %>% anti_join(table2, by = c("pupil", "test"))
table1 %>% semi_join(table2, by = c("pupil", "test"))
```

---

**Description**

Return grouping variables

**Usage**

```r
get_groups(x)
```

**Arguments**

- `x` A data.frame.

**Value**

A character vector of group names.

**Examples**

```r
df <- mtcars %>% group_by(am, cyl)
get_groups(df)
```
groups

Group by one or more variables

Description

Determine the groups within a data.frame to perform operations on. ungroup() removes the grouping levels.

Usage

```r
group_by(.data, ..., .add = FALSE)
```

```r
ungroup(x, ...)
```

Arguments

<table>
<thead>
<tr>
<th>.data</th>
<th>data.frame. The data to group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>One or more unquoted column names to group/ungroup the data by.</td>
</tr>
<tr>
<td>.add</td>
<td>logical(1). When FALSE (the default) group_by() will override existing groups. To add to existing groups, use .add = TRUE.</td>
</tr>
<tr>
<td>x</td>
<td>A data.frame.</td>
</tr>
</tbody>
</table>

Value

When using `group_by()`, a data.frame, grouped by the grouping variables.

When using `ungroup()`, data.frame.

Examples

```r
group_by(mtcars, am, cyl)
ungroup(mutate(group_by(mtcars, am, cyl), sumMpg = sum(mpg)))
mtcars %>%
  group_by(am, cyl) %>%
  mutate(sumMpg = sum(mpg)) %>%
  ungroup()
mtcars %>%
group_by(carb) %>%
filter(any(gear == 5))
```
if_else

Vectorised if

Description

This is a wrapper around ifelse() which checks that true and false are of the same type, making the output more predictable.

Usage

if_else(condition, true, false, missing = NULL)

Arguments

condition A logical(n) vector.
true, false Values to use for TRUE and FALSE in condition. They must either be the same length as condition or be length 1. They must also be the same type.
missing If not NULL (the default), this will replace any missing values.

Value

A vector the same length as condition with values for TRUE and FALSE replaced by those specified in true and false, respectively.

Examples

```r
x <- c(-5:5, NA)
if_else(x < 0, NA_integer_, x)
if_else(x < 0, "negative", "positive", "missing")
```

# Unlike ifelse, if_else preserves types
```r
x <- factor(sample(letters[1:5], 10, replace = TRUE))
ifelse(x %in% c("a", "b", "c"), x, factor(NA))
# Attributes are taken from the 'true' vector
ifelse(x %in% c("a", "b", "c"), x, factor(NA))
```

joins

Join two data.frames together

Description

Join two data.frames together
Usage

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

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

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

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

Arguments

x, y  The data.frames to join.

by     A character vector of variables to join by. If NULL, the default, *_join() will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're right (to suppress the message, simply explicitly list the variables that you want to join).

   To join by different variables on x and y use a named vector. For example, by = c("a" = "b") will match x.a to y.b.

suffix     If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

lag

Compute lagged or leading values

Description

Find the "previous" \texttt{lag()} or "next" \texttt{lead()} values in a vector. Useful for comparing values behind of or ahead of the current values.

Usage

\texttt{lag(x, n = 1L, default = NA)}

\texttt{lead(x, n = 1L, default = NA)}

Arguments

\texttt{x}     A vector of values

\texttt{n}     A positive integer (1), giving the number of positions to lead or lag by.

\texttt{default}     The value used for non-existent rows (default: NA).
Examples

```r
lag(1:5)
lead(1:5)

x <- 1:5
data.frame(behind = lag(x), x, ahead = lead(x))

# If you want to look more rows behind or ahead, use `n`
lag(1:5, n = 1)
lag(1:5, n = 2)

lead(1:5, n = 1)
lead(1:5, n = 2)

# If you want to define a value for non-existing rows, use `default`
lag(1:5)
lag(1:5, default = 0)

lead(1:5)
lead(1:5, default = 6)
```

---

**mutate**

Create or transform variables

Description

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. Both functions preserve the number of rows of the input. New variables overwrite existing variables of the same name.

Usage

```r
mutate(.data, ...)
transmute(.data, ...)
```

Arguments

- `.data` A data.frame.
- `...` Name-value pairs of expressions, each with length 1L. The name of each argument will be the name of a new column and the value will be its corresponding value. Use a NULL value in `mutate` to drop a variable. New variables overwrite existing variables of the same name.
Examples

mutate(mtcars, mpg2 = mpg * 2)
mtcars %>% mutate(mpg2 = mpg * 2)
mtcars %>% mutate(mpg2 = mpg * 2, cyl2 = cyl * 2)

# Newly created variables are available immediately
mtcars %>% mutate(mpg2 = mpg * 2, mpg4 = mpg2 * 2)

# You can also use mutate() to remove variables and modify existing variables
mtcars %>% mutate(  
  mpg = NULL,
  disp = disp * 0.0163871 # convert to litres
)

# mutate() vs transmute --------------------------
# mutate() keeps all existing variables
mtcars %>% mutate(displ_l = disp / 61.0237)

# transmute keeps only the variables you create
mtcars %>% transmute(displ_l = disp / 61.0237)

---

n

*The number of observations in the current group*

Description

This function can be used within the context of summarise(), mutate() and filter().

Usage

n()

Value

An integer.

Examples

mt_gears <- mtcars %>% group_by(gear)
mt_gears %>% mutate(n = n())
mt_gears %>% filter(n < 10)
mt_gears %>% summarise(n = n())
n_distinct

Count the number of unique values in a set of vectors

Description

This is the equivalent of `length(unique(x))` for multiple vectors.

Usage

```
n_distinct(..., na.rm = FALSE)
```

Arguments

- `...`: A vectors of values.
- `na.rm`: logical(1). If TRUE missing values don’t count.

Examples

```
x <- sample(1:10, 1e5, rep = TRUE)
length(unique(x))
n_distinct(x)
```

peek_vars

Peek at variables in the selection context

Description

Return the vector of column names of the data currently available for selection.

Usage

```
peek_vars()
```

Value

A vector of column names.
pipe  

Forward-pipe operator

Description

Pipe an object forward into a function or call expression.

Usage

lhs %>% rhs

Arguments

lhs The result you are piping.
rhs Where you are piping the result to.

Details

Unlike the magrittr pipe, you must supply an actual function instead of just a function name. For example mtcars %>% head will not work, but mtcars %>% head() will.

Examples

mtcars %>% head()
mtcars %>% select(mpg)

print.grouped_data  

Print a grouped data.frame

Description

A print method for grouped data.frames. Uses the standard print.data.frame() method but also reports the groups.

Usage

## S3 method for class 'grouped_data'
print(
x,  
...,  
digits = NULL,  
quote = FALSE,  
right = TRUE,  
row.names = TRUE,  
max = NULL
)
pull

Arguments

- **x**: An object of class `grouped_data`.
- **...**: Additional arguments to `print()`.
- **digits**: the minimum number of significant digits to be used: see `print.default`.
- **quote**: logical, indicating whether or not entries should be printed with surrounding quotes.
- **right**: logical, indicating whether or not strings should be right-aligned. The default is right-alignment.
- **row.names**: logical (or character vector), indicating whether (or what) row names should be printed.
- **max**: numeric or `NULL`, specifying the maximal number of entries to be printed. By default, when `NULL`, `getOption("max.print")` used.

Examples

```r
mtcars %>% group_by(cyl, am) %>% print()
```

---

pull

Pull out a single variable

Description

This is a direct replacement for `[[.data.frame`.

Usage

```r
pull(.data, var = -1)
```

Arguments

- **.data**: A `data.frame`.
- **var**: A variable specified as:
  - a literal variable name
  - a positive integer, giving the position counting from the left
  - a negative integer, giving the position counting from the right

The default returns the last column (on the assumption that’s the column you’ve created most recently).

Examples

```r
mtcars %>% pull(-1)  
mtcars %>% pull(1)    
mtcars %>% pull(cyl)  
mtcars %>% pull("cyl")
```
relocate  
*Select/relocate variables by name*

**Description**

Choose or relocate variables from a data.frame. `select()` keeps only the variables you mention; `relocate()` keeps all the variables.

**Usage**

```r
relocate(.data, ..., .before = NULL, .after = NULL)
```

```r
select(.data, ...)
```

**Arguments**

- `.data` A data.frame.
- `...` The name(s) of the column(s) to select.
- `.before, .after` Destination of the columns selected by `...`. Supplying neither will move the columns to the left-hand side whereas supplying both will result in an error.

**Value**

A data.frame.

**Useful functions**

There are a number of special functions which are designed to work in `select()` and `relocate()`:

- `starts_with()`, `ends_with()`, `contains()`
- `matches()`
- `num_range()`
- `everything()`

**Examples**

```r
select(mtcars, mpg:cyl)
select(mtcars, MilesPerGallon = mpg, Cylinders = cyl)
mtcars %>% select(mpg)
mtcars %>% select(!mpg, !cyl)
iris %>% select(contains("Petal"))

df <- as.data.frame(matrix(runif(100), nrow = 10))
df <- as.data.frame(df[c(3, 4, 7, 1, 9, 5, 2, 6, 10)])
df %>% select(num_range("V", 4:6))

mtcars %>% relocate(ends_with("p"), .before = mpg)
```
## rename

### Description

rename() changes the names of individual variables using new_name = old_name syntax.

### Usage

```r
rename(.data, ...)
```

### Arguments

- `.data` A data.frame
- `...` Comma separated key-value pairs in the form of new_name = old_name to rename selected variables.

### Value

A data.frame

### Examples

```r
rename(mtcars, MilesPerGallon = mpg)
rename(mtcars, Cylinders = cyl, Gears = gear)
mtcars %>% rename(MilesPerGallon = mpg)
```

---

## rownames

### Description

Tools for working with row names

### Usage

```r
rownames_to_column(.data, var = "rowname")
```

### Arguments

- `.data` A data.frame.
- `var` character(1). The name of the column to use for row names.

### Value

A data.frame
Examples

```r
mtcars %>% rownames_to_column()
```

## Description

These functions allow you to select variables based on their names.

- `starts_with()`: Starts with a prefix.
- `ends_with()`: Ends with a prefix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `all_of()`: Matches variable names in a character vector. All names must be present, otherwise an error is thrown.
- `any_of()`: The same as `all_of()` except it doesn’t throw an error.
- `everything()`: Matches all variables.
- `last_col()`: Select the last variable, possibly with an offset.

## Usage

```r
starts_with(match, ignore.case = TRUE, vars = peek_vars())
ends_with(match, ignore.case = TRUE, vars = peek_vars())
contains(match, ignore.case = TRUE, vars = peek_vars())
moves(match, ignore.case = TRUE, perl = FALSE, vars = peek_vars())
um_range(prefix, range, width = NULL, vars = peek_vars())
all_of(x, vars = peek_vars())
any_of(x, vars = peek_vars())
everything(vars = peek_vars())
last_col(offset = 0L, vars = peek_vars())
```
Arguments

match character(n). If length > 1, the union of the matches is taken.
ignore.case logical(1). If TRUE, the default, ignores case when matching names.
vars character(n). A character vector of variable names. When called from inside selecting functions such as select(), these are automatically set to the names of the table.
perl logical(1). Should Perl-compatible regexps be used?
prefix A prefix which starts the numeric range.
range integer(n). A sequence of integers, e.g. 1:5.
width numeric(1). Optionally, the "width" of the numeric range. For example, a range of 2 gives "01", a range of three "001", etc.
x character(n). A vector of column names.
offset integer(1). Select the nth variable from the end of the data.frame.

Value

An integer vector giving the position of the matched variables.

See Also

select(), relocate()

Examples

mtcars %>% select(starts_with("c"))
mtcars %>% select(starts_with(c("c", "h")))
mtcars %>% select(ends_with("b"))
mtcars %>% relocate(contains("a"), .before = mpg)
iris %>% select(matches(".t."))
mtcars %>% select(last_col())

# `all_of()` selects the variables in a character vector:
iris %>% select(all_of(c("Petal.Length", "Petal.Width")))
# `all_of()` is strict and will throw an error if the column name isn't found
try(iris %>% select(all_of(c("Species", "Genres"))))
# However `any_of()` allows missing variables
iris %>% select(any_of(c("Species", "Genres")))
**slice**  
*Choose rows by position*

**Description**

Choose rows by their original position in the data.frame. Grouped data.frames use the position within each group.

**Usage**

slice(.data, ...)

**Arguments**

- **.data**  
  A data.frame.
- **...**  
  Integer row values. Provide either positive values to keep, or negative values to drop. The values provided must be either all positive or negative. Indices beyond the number of rows in the input are silently ignored.

**Examples**

slice(mtcars, c(1, 2, 3))
mtcars %>% slice(1:3)

**summarise**  
*Reduce multiple values down to a single value*

**Description**

Create one or more scalar variables summarising the variables of an existing data.frame. Grouped data.frames will result in one row in the output for each group.

**Usage**

summarise(.data, ...)

summarize(.data, ...)

**Arguments**

- **.data**  
  A data.frame.
- **...**  
  Name-value pairs of summary functions. The name will be the name of the variable in the result. The value should be an expression that returns a single value, e.g. min(x).
Details

summarise() and summarize() are synonyms.

Examples

```r
summarise(mtcars, mean(mpg))
summarise(mtcars, meanMpg = mean(mpg), sumMpg = sum(mpg))
mtcars %>% summarise(mean(mpg))
```

---

**window_rank**  
***Windowed Rank Functions***

**Description**

Six variations on ranking functions, mimicking the ranking functions described in SQL2003. They are currently implemented using the built in `rank()` function. All ranking functions map smallest inputs to smallest outputs. Use `desc()` to reverse the direction.

**Usage**

```r
cume_dist(x)
dense_rank(x)
min_rank(x)
ntile(x = row_number(), n)
percent_rank(x)
row_number(x)
```

**Arguments**

- `x` A vector of values to rank. Missing values are left as is. If you want to treat them as the smallest or largest values, replace with `Inf` or `-Inf` before ranking.
- `n` integer(1). The number of groups to split up into.

**Details**

- `cume_dist()`: a cumulative distribution function. Proportion of all values less than or equal to the current rank.
- `dense_rank()`: like `min_rank()`, but with no gaps between ranks
- `min_rank()`: equivalent to `rank(ties.method = "min")`
- `ntile()`: a rough rank, which breaks the input vector into `n` buckets. The size of the buckets may differ by up to one, larger buckets have lower rank.
- `percent_rank()`: a number between 0 and 1 computed by rescaling `min_rank` to [0, 1]
- `row_number()`: equivalent to `rank(ties.method = "first")`

**Examples**

```r
x <- c(5, 1, 3, 2, 2, NA)
row_number(x)
min_rank(x)
dense_rank(x)
percent_rank(x)
cume_dist(x)

ntile(x, 2)
ntile(1:8, 3)

# row_number can be used with single table verbs without specifying x
# (for data frames and databases that support windowing)
mutate(mtcars, row_number() == 1L)
mtcars %>% filter(between(row_number(), 1, 10))
```
Index

%>% (pipe), 14
add_count (count), 3
add_tally (count), 3
all_of (select_helpers), 18
anti_join (filter_joins), 6
any_of (select_helpers), 18
arrange, 2
arrange(), 5
between, 3
contains (select_helpers), 18
contains(), 16
count, 3
cume_dist (window_rank), 21
dense_rank (window_rank), 21
desc, 5
ends_with (select Helpers), 18
ends_with(), 16
everything (select_helpers), 18
everything(), 16
filter, 5
filter(), 12
filter_joins, 6
full_join (joins), 9
get_groups, 7
getOption, 15
group_by (groups), 8
group_by(), 8
groups, 8
if_else, 9
inner_join (joins), 9
joins, 9
lag, 10
last_col (select_helpers), 18
lead (lag), 10
left_join (joins), 9
matches (select_helpers), 18
matches(), 16
min_rank (window_rank), 21
mutate, 11
mutate(), 12
n, 12
n_distinct, 13
ntile (window_rank), 21
num_range (select_helpers), 18
num_range(), 16
peek_vars, 13
percent_rank (window_rank), 21
pipe, 14
print(), 15
print.default, 15
print.grouped_data, 14
pull, 15
rank(), 21
relocate, 16
relocate(), 19
rename, 17
right_join (joins), 9
row_number (window_rank), 21
rownames, 17
rownames_to_column (rownames), 17
select (relocate), 16
select(), 19
select_helpers, 18
semi_join (filter_joins), 6
slice, 20
starts_with (select_helpers), 18
starts_with(), 16
summarise, 20
summarise(), 12
summarize(summarise), 20

tally(count), 3
transmute(mutate), 11

ungroup(groups), 8
ungroup(), 8

window_rank, 21