Package ‘poorman’
April 1, 2020

Type Package
Title A Poor Man's Base R Copy of 'dplyr' Verbs
Version 0.1.9
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]
Maintainer Nathan Eastwood <nathan.eastwood@icloud.com>
Repository CRAN
Date/Publication 2020-04-01 12:10:02 UTC

R topics documented:

arrange .................................................. 2
filter .................................................... 2
filter_joins .......................................... 3
groups .................................................. 4
joins .................................................... 5
mutate ................................................... 5
pipe ...................................................... 6
print.grouped_data ................................. 7
pull ..................................................... 8
relocate ............................................... 8
filter

Return rows with matching conditions

Description

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

Usage

filter(.data, ...)

filter

Return rows with matching conditions

Description

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

Usage

filter(.data, ...)

arrange

Arrange rows by variables

Description

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

Usage

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

arrange(mtcars, mpg)
mtcars %>% arrange(mpg)
mtcars %>% arrange(cyl, mpg)
**filter_joins**

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

---

**Arguments**

- \(.data\) A data.frame.
- \( \ldots \) Logical predicated defined in terms of the variables in \(.data\). Multiple conditions are combined with &. Arguments within \( \ldots \) are automatically quoted and evaluated within the context of the data.frame.

**Value**

A data.frame.

**Useful filter functions**

- \( ==, >, \geq, \text{etc.} \)
- \&, |, !, 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.
- \( \text{by} \) 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"))
```

---

**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, ...)

ungroup(x, ...)
```

**Arguments**

- `.data` `data.frame`. The data to group.
- `...` One or more unquoted column names to group/ungroup the data by.
- `x` A `data.frame`.

**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) %>%
```
joins

\begin{verbatim}
filter(any(gear == 5))
\end{verbatim}

\section*{joins

\textbf{Join two data.frames together}}

\subsection*{Description}
Join two data.frames together

\subsection*{Usage}

\begin{verbatim}
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"))
\end{verbatim}

\subsection*{Arguments}

\begin{itemize}
  \item \textbf{x, y} The data.frames to join.
  \item \textbf{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.
  \item \textbf{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.
\end{itemize}

\section*{mutate

\textbf{Create or transform variables}}

\subsection*{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.

\subsection*{Usage}

\begin{verbatim}
mutate(.data, ...)
transmute(.data, ...)
\end{verbatim}
Arguments

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)

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

description

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)
Description

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

Usage

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

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)
```

---

### relocate

**Change column order**

**Description**

Change the column positions of a `data.frame`.

**Usage**

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

**Arguments**

- `.data` A `data.frame`.
- `...` The columns to move.
- `.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.
rename

Value

A data.frame

Description

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

Usage

rename(.data, ...)

select(.data, ...)

Arguments

.data
A data.frame.

... The name(s) of the column(s) to select.

Value

A data.frame.

Examples

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

select(mtcars, mpg, cyl)
select(mtcars, MilesPerGallon = mpg, Cylinders = cyl)
mtcars %>% select(mpg)
mtcars %>% select(mpg, cyl)
Tools for working with row names

Description
Tools for working with row names

Usage
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
mtcars %>% rownames_to_column()

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.
summarise

Examples

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

---

summarise \hspace{1cm} \textit{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, ...)

Arguments

    .data \hspace{1cm} A data.frame.
    ... \hspace{1cm} 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).

Examples

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

%>% (pipe),  6
anti_join(filter_joins),  3
arrange,  2
filter,  2
filter_joins,  3
full_join(joins),  5
ggetOption,  7
group_by (groups),  4
group_by(),  4
groups,  4
inner_join(joins),  5
joins,  5
left_join(joins),  5
mutate,  5
pipe,  6
print(),  7
print.default,  7
print.grouped_data,  7
pull,  8
relocate,  8
rename,  9
right_join(joins),  5
rownames,  10
rownames_to_column (rownames),  10
select (rename),  9
semi_join(filter_joins),  3
slice,  10
summarise,  11
transmute(mutate),  5
ungroup (groups),  4
ungroup(),  4