Package 'rank'

December 1, 2025

Title Customisable Ranking of Numerical and Categorical Data

Version 0.2.0

Description Provides a flexible alternative to the built-in rank() function called smartrank(). Optionally rank categorical variables by frequency (instead of in alphabetical order), and control whether ranking is based on descending/ascending order.
smartrank() is suitable for both numerical and categorical data.

License MIT + file LICENSE

Suggests covr, dplyr, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 2

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/selkamand/rank, https://selkamand.github.io/rank/

BugReports https://github.com/selkamand/rank/issues

VignetteBuilder knitr

NeedsCompilation no

Author Sam El-Kamand [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-2270-8088>)

Maintainer Sam El-Kamand <sam.elkamand@gmail.com>

Repository CRAN

Date/Publication 2025-12-01 09:40:02 UTC

Contents

rank_by_priority *Rank a character vector based on supplied priority values*

Description

Rank a character vector based on supplied priority values

Usage

```
rank_by_priority(x, priority_values, ties.method = "average")
```

Arguments

x	A character vector.
priority_values	A character vector descibing "priority" values. Elements of x matching priority_values will be ranked based on their order of appearance in priority_values
ties.method	a character string specifying how ties are treated, see 'Details'; can be abbreviated.

Value

A vector of ranks describing x such that x[order(ranks)] will move priority_values to the front of the vector

Examples

```
x <- c("A", "B", "C", "D", "E")
rank_by_priority(x, c("C", "A"))
#> "2" "4" "1" "4" "4"

rank_by_priority(1:6, c(4, 2, 7))
#>  4 2 1 3 5 6
```

rank_stratified *Stratified hierarchical ranking across multiple variables*

Description

rank_stratified() computes a single, combined rank for each row of a data frame using **stratified hierarchical ranking**. The first variable is ranked globally; each subsequent variable is then ranked **within strata defined by all previous variables**.

Usage

```
rank_stratified(
  data,
  cols = NULL,
  sort_by = "frequency",
  desc = FALSE,
  ties.method = "average",
  na.last = TRUE,
  freq_tiebreak = "match_desc",
  verbose = TRUE
)
```

Arguments

data	A data frame. Each selected column represents one level of the stratified hierarchy, in the order given by cols.
cols	Optional column specification indicating which variables in data to use for ranking, and in what order. Can be: • NULL (default): use all columns of data in their existing order. • A character vector of column names. • An integer vector of column positions.
sort_by	Character scalar or vector specifying how to rank each non-numeric column. Each element must be either "alphabetical" or "frequency", matching the behaviour of smartrank(). If a single value is supplied it is recycled for all columns. For numeric columns, sort_by is ignored and ranking is always based on numeric order.
desc	Logical scalar or vector indicating whether to rank each column in descending order. If a single value is supplied it is recycled for all columns.
ties.method	Passed to base::rank() when resolving ties at each level; must be one of "average", "first", "last", "random", "max", or "min". See base::rank() for details.
na.last	Logical, controlling the treatment of missing values, as in base::rank(). If TRUE, NAs are given the largest ranks; if FALSE, the smallest. Unlike base::rank() or smartrank(), na.last cannot be set to NA in rank_stratified(), because dropping rows would change group membership and break stratified ranking.
freq_tiebreak	Character scalar or vector controlling how alphabetical tie-breaking works when sort_by = "frequency" and the column is character/factor/logical. Each element must be one of: • "match_desc" (default): alphabetical tie-breaking follows desc for that column (ascending when desc = FALSE, descending when desc = TRUE). • "asc": ties are always broken by ascending alphabetical order. • "desc": ties are always broken by descending alphabetical order. If a single value is supplied, it is recycled for all columns.
verbose	Logical; if TRUE, emit messages when sort_by is ignored (e.g. for numeric columns), mirroring the behaviour of smartrank().

Details

This is useful when you want a "truly hierarchical" ordering where, for example, rows are first grouped and ordered by the frequency of `gender`, and then within each `gender` group, ordered by the frequency of `pet` **within that gender**, rather than globally.

The result is a single rank vector that can be passed directly to `base::order()` to obtain a stratified, multi-level ordering.

Stratified ranking proceeds level by level:

1. The first selected column is ranked globally, using `sort_by[1]` (for non-numeric) and `desc[1]`.
2. For the second column, ranks are computed **separately within each distinct combination of values of all previous columns**. Within each stratum, the second column is ranked using `sort_by[2]` / `desc[2]`.
3. This process continues for each subsequent column: at level *k*, ranking is done within strata defined by columns 1, 2, ..., *k-1*.

This yields a single composite rank per row that reflects a "true" hierarchical (i.e. stratified) ordering: earlier variables define strata, and later variables are only compared **within** those strata (for example, by within-stratum frequency).

Value

A numeric vector of length `nrow(data)`, containing stratified ranks. Smaller values indicate "earlier" rows in the stratified hierarchy.

Examples

```
library(rank)

data <- data.frame(
  gender = c("male", "male", "male", "male", "female", "female", "male", "female"),
  pet    = c("cat",  "cat",  "magpie", "magpie", "giraffe", "cat", "giraffe", "cat")
)

# Stratified ranking: first by gender frequency, then within each gender
# by pet frequency *within that gender*
r <- rank_stratified(
  data,
  cols = c("gender", "pet"),
  sort_by = c("frequency", "frequency"),
  desc = TRUE
)

data[order(r), ]
```

reorder_by_priority *Bring specified values in a vector to the front*

Description

Reorders a vector so that any elements matching the values in `values` appear first, in the order they appear in `values`. All remaining elements are returned afterward, preserving their original order.

Usage

```
reorder_by_priority(x, priority_values)
```

Arguments

`x`	A character or numeric vector to reorder.
`priority_values`	A vector of "priority" values. Elements of `x` that match entries in `priority_values` are moved to the front in the order they appear in `priority_values`. Values not found in `x` are ignored.

Value

A reordered vector with priority values first, followed by all remaining elements in their original order.

Examples

```
reorder_by_priority(c("A", "B", "C", "D", "E"), c("C", "A"))
reorder_by_priority(1:6, c(4, 2, 7))
```

smartrank *Rank a vector based on either alphabetical or frequency order*

Description

This function acts as a drop-in replacement for the base `rank()` function with the added option to:

1. Rank categorical factors based on frequency instead of alphabetically
2. Rank in descending or ascending order

Usage

```
smartrank(
  x,
  sort_by = c("alphabetical", "frequency"),
  desc = FALSE,
  ties.method = "average",
  na.last = TRUE,
  freq_tiebreak = c("match_desc", "asc", "desc"),
  verbose = TRUE
)
```

Arguments

x	A numeric, character, or factor vector
sort_by	Sort ranking either by "alphabetical" or "frequency" . Default is "alphabetical"
desc	A logical indicating whether the ranking should be in descending (TRUE) or ascending (FALSE) order. When input is numeric, ranking is always based on numeric order.
ties.method	a character string specifying how ties are treated, see 'Details'; can be abbreviated.
na.last	a logical or character string controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if "keep" they are kept with rank NA.
freq_tiebreak	Controls how alphabetical tie-breaking works when sort_by = "frequency" and x is character/factor/logical. Must be one of: • "match_desc" (default): alphabetical tie-breaking direction follows desc (ascending when desc = FALSE, descending when desc = TRUE). • "asc": ties are always broken by **ascending** alphabetical order, regardless of desc. • "desc": ties are always broken by **descending** alphabetical order, regardless of desc.
verbose	verbose (flag)

Details

If x includes 'ties' (equal values), the ties.method argument determines how the rank value is decided. Must be one of:

- **average**: replaces integer ranks of tied values with their average (default)
- **first**: first-occurring value is assumed to be the lower rank (closer to one)
- **last**: last-occurring value is assumed to be the lower rank (closer to one)
- **max** or **min**: integer ranks of tied values are replaced with their maximum and minimum respectively (latter is typical in sports-ranking)
- **random** which of the tied values are higher / lower rank is randomly decided.

NA values are never considered to be equal: for na.last = TRUE and na.last = FALSE they are given distinct ranks in the order in which they occur in x.

Value

The ranked vector

Note

When `sort_by = "frequency"`, ties based on frequency are broken by alphabetical order of the terms. Use `freq_tiebreak` to control whether that alphabetical tie-breaking is ascending, descending, or follows `desc`.

When `sort_by = "frequency"` and input is character, ties.method is ignored. Each distinct element level gets its own rank, and each rank is 1 unit away from the next element, irrespective of how many duplicates

Examples

```
# ------------------
## CATEGORICAL INPUT
# ------------------

fruits <- c("Apple", "Orange", "Apple", "Pear", "Orange")

# rank alphabetically
smartrank(fruits)
#> [1] 1.5 3.5 1.5 5.0 3.5

# rank based on frequency
smartrank(fruits, sort_by = "frequency")
#> [1] 2.5 4.5 2.5 1.0 4.5

# rank based on descending order of frequency
smartrank(fruits, sort_by = "frequency", desc = TRUE)
#> [1] 1.5 3.5 1.5 5.0 3.5

# sort fruits vector based on rank
ranks <- smartrank(fruits,sort_by = "frequency", desc = TRUE)
fruits[order(ranks)]
#> [1] "Apple"  "Apple"  "Orange" "Orange" "Pear"


# ------------------
## NUMERICAL INPUT
# ------------------

# rank numerically
smartrank(c(1, 3, 2))
#> [1] 1 3 2

# rank numerically based on descending order
smartrank(c(1, 3, 2), desc = TRUE)
#> [1] 3 1 2

# always rank numeric vectors based on values, irrespective of sort_by
```

```
smartrank(c(1, 3, 2), sort_by = "frequency")
#> smartrank: Sorting a non-categorical variable. Ignoring `sort_by` and sorting numerically
#> [1] 1 3 2
```

Index