Package: fctutils (via r-universe)

November 8, 2024

Type Package

Title Advanced Factor Manipulation Utilities

Version 0.0.8

Maintainer Kai Guo <guokai8@gmail.com>

Description Provides a collection of utility functions for manipulating and analyzing factor vectors in R. It offers tools for filtering, splitting, combining, and reordering factor levels based on various criteria. The package is designed to simplify common tasks in categorical data analysis, making it easier to work with factors in a flexible and efficient manner.

License GPL-3

Imports stringr, stringdist, utils, stats

Encoding UTF-8

LazyData yes

LazyDataCompression xz

Suggests markdown,rmarkdown,knitr

VignetteBuilder knitr

RoxygenNote 7.3.1

Config/pak/sysreqs libicu-dev

Repository https://guokai8.r-universe.dev

RemoteUrl https://github.com/guokai8/fctutils

RemoteRef HEAD

RemoteSha 18ac7c6cf3c498ed61465476c1b89cc6827cf43a

Contents

ft_apply *Apply a Function to Factor Levels*

Description

Transforms factor levels by applying a function to each level.

Usage

```
ft_apply(factor_vec, apply_func)
```

Arguments

factor_vec	A factor vector to transform.
apply_func	A function to apply to each level.

Value

A factor vector with transformed levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry'))

# Append '_fruit' to each level
ft_apply(factor_vec, function(x) paste0(x, '_fruit'))
```

ft_char_freq *Reorder Factor Levels Based on Character Frequency at Positions*

Description

Reorders the levels of a factor vector based on the frequency of characters at specified positions within the data.

Usage

```
ft_char_freq(
  factor_vec,
  positions,
  case = FALSE,
  decreasing = TRUE,
  inplace = TRUE
)
```

Arguments

factor_vec	A factor vector whose levels are to be reordered.
positions	A vector of positive integers specifying the character positions to consider.
case	Logical. Should the character comparison be case-sensitive? Default is FALSE.
decreasing	Logical. Should the ordering be decreasing by frequency? Default is TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on the frequency of characters at specified positions.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apricot', 'cherry', 'banana', 'banana', 'date'))

# Reorder based on characters at positions 1 and 2
ft_char_freq(factor_vec, positions = 1:2)

# Reorder, case-sensitive, decreasing order
ft_char_freq(factor_vec, positions = c(1, 3), case = TRUE)
```

---

ft_collapse_lev *Collapse Factor Levels Based on Grouping*

Description

Collapses specified levels of a factor into new levels based on a grouping list.

Usage

```
ft_collapse_lev(factor_vec, groups)
```

Arguments

factor_vec	A factor vector to modify.
groups	A named list where each element contains levels to be collapsed into a new level named after the list element's name.

Value

A factor vector with collapsed levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'elderberry'))

# Define groups
groups <- list(
  'Group1' = c('apple', 'banana'),
  'Group2' = c('cherry', 'date')
)

# Collapse levels
ft_collapse_lev(factor_vec, groups)
```

---

ft_combine *Combine Two Vectors of Unequal Lengths and Sort Based on Specified Levels*

Description

Combines two vectors, which may be of unequal lengths, into a factor vector and sorts based on the levels of either the first or second vector.

Usage

```
ft_combine(vector1, vector2, sort_by = 1, decreasing = FALSE)
```

Arguments

vector1	The first vector to combine.
vector2	The second vector to combine.
sort_by	An integer (1 or 2) indicating which vector's levels to use for sorting. Default is 1.
decreasing	Logical. Should the sorting be in decreasing order? Default is FALSE.

Value

A factor vector combining both vectors and sorted based on specified levels.

Examples

```
# Example vectors of unequal lengths
vector1 <- c('apple', 'banana', 'cherry')
vector2 <- c('date', 'fig', 'grape', 'honeydew')

# Combine and sort based on vector1 levels
combined_factor1 <- ft_combine(vector1, vector2, sort_by = 1)
print(combined_factor1)

# Combine and sort based on vector2 levels
combined_factor2 <- ft_combine(vector1, vector2, sort_by = 2)
print(combined_factor2)

# Combine with decreasing order based on vector1
combined_factor3 <- ft_combine(vector1, vector2, sort_by = 1, decreasing = TRUE)
print(combined_factor3)
```

---

ft_concat *Concatenate Multiple Factor Vectors*

Description

Combines multiple factor vectors into a single factor, unifying the levels.

Usage

```
ft_concat(...)
```

Arguments

...	Factor vectors to concatenate.

Value

A single factor vector containing all elements and unified levels.

Author(s)

Kai Guo

Examples

```
# Example factor vectors
factor_vec1 <- factor(c('apple', 'banana'))
factor_vec2 <- factor(c('cherry', 'date'))

# Concatenate factors
concatenated_factor <- ft_concat(factor_vec1, factor_vec2)
levels(concatenated_factor)
```

---

ft_count *Reorder Factor Levels by Level Count*

Description

Reorders the levels of a factor vector based on the count of each level in the data.

Usage

```
ft_count(factor_vec, decreasing = TRUE, inplace = FALSE)
```

Arguments

factor_vec	A factor vector whose levels are to be reordered.
decreasing	Logical. Should the ordering be decreasing by count? Default is TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on their count.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apple', 'cherry', 'banana', 'banana', 'date'))

# Reorder levels by decreasing count
ft_count(factor_vec)

# Reorder levels by increasing count
ft_count(factor_vec, decreasing = FALSE)
```

---

ft_decode *Decode Numeric Codes into Factor Levels*

Description

Converts numeric codes back into factor levels using a provided mapping.

Usage

```
ft_decode(codes, mapping)
```

Arguments

codes	A numeric vector of codes to decode.
mapping	A named vector where names are levels and values are codes.

Value

A factor vector with decoded levels.

Author(s)

Kai Guo

Examples

```
# Numeric codes
codes <- c(1, 2, 3, 2)

# Mapping from levels to codes
mapping <- c('low' = 1, 'medium' = 2, 'high' = 3)

# Decode codes into factor levels
ft_decode(codes, mapping = mapping)
```

ft_dummy *Create Dummy Variables from Factor Levels*

Description

Generates a data frame of dummy variables (one-hot encoded) from a factor vector.

Usage

```
ft_dummy(factor_vec)
```

Arguments

factor_vec	A factor vector.

Value

A data frame where each column represents a level of the factor, containing 1s and 0s.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apple', 'cherry'))

# Create dummy variables
ft_dummy(factor_vec)
```

ft_duplicates *Flag Duplicate Factor Levels*

Description

Identifies duplicate levels in a factor vector and returns a logical vector indicating which elements are duplicates.

Usage

```
ft_duplicates(factor_vec)
```

Arguments

factor_vec	A factor vector.

Value

A logical vector where TRUE indicates a duplicate level.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apple', 'cherry', 'banana'))

# Flag duplicates
ft_duplicates(factor_vec)
```

---

ft_encode *Encode Factor Levels into Numeric Codes*

Description

Converts the levels of a factor vector into numeric codes, optionally using a provided mapping.

Usage

```
ft_encode(factor_vec, mapping = NULL)
```

Arguments

factor_vec	A factor vector to encode.
mapping	An optional named vector providing the numeric code for each level.

Value

A numeric vector with encoded values.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('low', 'medium', 'high', 'medium'))

# Encode without mapping
ft_encode(factor_vec)

# Encode with custom mapping
custom_mapping <- c('low' = 1, 'medium' = 2, 'high' = 3)
ft_encode(factor_vec, mapping = custom_mapping)
```

`ft_extract` *Extract Substrings from Factor Levels*

Description

Extracts substrings from the levels of a factor vector based on a regular expression pattern and creates a new factor.

Usage

```
ft_extract(factor_vec, pattern, capture_group = 0)
```

Arguments

`factor_vec`	A factor vector from which substrings will be extracted.
`pattern`	A regular expression pattern to match.
`capture_group`	An integer specifying which capture group to extract if using capturing groups in the pattern. Default is `0`, which extracts the entire match.

Value

A new factor vector containing the extracted substrings.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('item123', 'item456', 'item789'))

# Extract numeric part
ft_extract(factor_vec, pattern = '\\d+')

# Extract with capturing group
factor_vec <- factor(c('apple: red', 'banana: yellow', 'cherry: red'))
ft_extract(factor_vec, pattern = '^(\\w+):', capture_group = 1)
```

ft_factorize *Factorize Character Vectors with Consistent Levels*

Description

Converts one or more character vectors into factors, ensuring that they share the same levels.

Usage

```
ft_factorize(..., levels = NULL)
```

Arguments

...	Character vectors to factorize.
levels	An optional character vector specifying the levels. If NULL, levels are determined from the combined unique values of all vectors.

Value

A list of factor vectors with consistent levels.

Author(s)

Kai Guo

Examples

```
# Example character vectors
vec1 <- c('apple', 'banana', 'cherry')
vec2 <- c('banana', 'date', 'apple')

# Factorize with consistent levels
factors <- ft_factorize(vec1, vec2)
levels(factors[[1]])
levels(factors[[2]])
```

ft_filter_freq *Filter Factor Levels by Frequency and Recalculate Character Frequencies*

Description

Filters out factor levels that occur less than a specified frequency threshold and recalculates character frequencies excluding the removed levels. Offers options to handle NA values and returns additional information.

Usage

```
ft_filter_freq(
  factor_vec,
  min_freq = 1,
  na.rm = FALSE,
  case = FALSE,
  decreasing = TRUE,
  return_info = FALSE
)
```

Arguments

factor_vec	A factor vector to be filtered.
min_freq	A positive integer specifying the minimum frequency threshold. Factor levels occurring less than this number will be dropped.
na.rm	Logical. Should NA values be removed before filtering and frequency calculation? Default is FALSE.
case	Logical. Should the character frequency count be case-sensitive? Default is FALSE.
decreasing	Logical. Should the ordering of levels be decreasing by total character frequency? Default is TRUE.
return_info	Logical. Should the function return additional information such as removed levels and character frequencies? Default is FALSE.

Value

If return_info is FALSE, returns a factor vector with levels filtered by the specified frequency threshold and reordered based on recalculated total character frequency. If return_info is TRUE, returns a list containing the filtered factor vector, removed levels, and character frequency table.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'banana', 'apple', 'fig', NA))

# Filter levels occurring less than 2 times and reorder by character frequency
ft_filter_freq(factor_vec, min_freq = 2)

# Filter levels, remove NA values, and return additional information
result <- ft_filter_freq(factor_vec, min_freq = 2, na.rm = TRUE, return_info = TRUE)
result$filtered_factor
result$removed_levels
result$char_freq_table
```

ft_filter_func *Filter Factor Levels Using a Function*

Description

Removes levels from a factor vector based on a user-defined function.

Usage

```
ft_filter_func(factor_vec, func)
```

Arguments

factor_vec	A factor vector to filter.
func	A function that takes a character vector of levels and returns a logical vector.

Value

A factor vector with levels filtered according to the function.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date'))

# Remove levels that start with 'b'
ft_filter_func(factor_vec, function(x) !grepl('^b', x))
```

ft_filter_pos *Remove Factor Levels with Specific Characters at Specified Positions*

Description

Removes factor levels where a specified character appears at specified positions within the levels.

Usage

```
ft_filter_pos(
  factor_vec,
  positions = NULL,
  char,
  case = FALSE,
  remove_na = TRUE,
  invert = FALSE,
  .return = FALSE
)
```

Arguments

factor_vec	A factor vector from which levels will be removed.
positions	A vector of positive integers indicating the character positions to check.
char	A single character string specifying the character to look for.
case	Logical. Should the character matching be case-sensitive? Default is FALSE.
remove_na	remove NA from the output? Default is TRUE.
invert	logical. If TRUE return indices or values for elements that do not match.
.return	logical. If TRUE return TRUE or FALSE instead of element.

Value

A factor vector with levels removed where the specified character appears at the specified positions.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apricot', 'cherry', 'date', 'fig', 'grape'))

# Remove levels where 'a' appears at position 1
ft_filter_pos(factor_vec, positions = 1, char = 'a')

# Remove levels where 'e' appears at positions 2 or 3
ft_filter_pos(factor_vec, positions = c(2, 3), char = 'e')

# Case-sensitive removal
factor_vec_case <- factor(c('Apple', 'banana', 'Apricot', 'Cherry', 'Date', 'Fig', 'grape'))
ft_filter_pos(factor_vec_case, positions = 1, char = 'A', case = TRUE)
```

ft_freq *Reorder Factor Levels Based on Character Frequency*

Description

Reorders the levels of a factor vector based on the frequency of characters in each level's name. Supports case sensitivity, descending order, and optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_freq(factor_vec, case = FALSE, decreasing = TRUE, inplace = FALSE)
```

Arguments

factor_vec	A factor vector whose levels will be reordered.
case	Logical. If TRUE, case is considered during frequency calculation. If FALSE, all characters are converted to lowercase before frequency calculation. Defaults to FALSE.
decreasing	Logical. If TRUE, the levels are ordered in decreasing order based on character frequency. Defaults to TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A new factor vector with reordered levels. Depending on the inplace parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example 1: Reorder levels based on character frequency without reordering data elements
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig', 'grape'))
new <- ft_freq(
  factor_vec,
  case = FALSE,
  decreasing = TRUE,
  inplace = FALSE
)
print(new)
# [1] apple  banana cherry date   fig    grape
```

```
# Levels: apple banana date cherry fig grape

# Example 2: Reorder levels based on character frequency and reorder data elements
new_inplace <- ft_freq(
  factor_vec,
  case = FALSE,
  decreasing = TRUE,
  inplace = TRUE
)
print(new_inplace)
# [1] apple  banana date   cherry fig    grape
# Levels: apple banana date cherry fig grape

# Example 3: Reorder levels in decreasing order based on character frequency
# without reordering data elements
new_dec <- ft_freq(
  factor_vec,
  case = FALSE,
  decreasing = TRUE,
  inplace = FALSE
)
print(new_dec)
# [1] apple  banana cherry date   fig    grape
# Levels: apple banana date cherry fig grape

# Example 4: Reorder levels with case sensitivity and reorder data elements
factor_vec_case <- factor(c('Apple', 'banana', 'Cherry', 'date', 'Fig', 'grape'))
new_case <- ft_freq(
  factor_vec_case,
  case = TRUE,
  decreasing = TRUE,
  inplace = TRUE
)
print(new_case)
# [1] Apple   banana  Cherry date    Fig     grape
# Levels: cherry Apple banana grape Fig date

# Example 5: Reorder levels based on character frequency, allowing insertion beyond string length
factor_vec_short <- factor(c('go', 'dog', 'cat', 'bird'))
new_short <- ft_freq(
  factor_vec_short,
  case = FALSE,
  decreasing = TRUE,
  inplace = FALSE
)
print(new_short)
# [1] go   dog  cat  bird
# Levels: cat dog bird go
```

---

ft_group_by_prefix *Group Factor Levels by Common Prefix*

Description

Groups factor levels by a common prefix of specified length.

Usage

```
ft_group_by_prefix(factor_vec, prefix_length)
```

Arguments

factor_vec	A factor vector to be grouped.
prefix_length	An integer specifying the number of characters in the prefix.

Value

A factor vector with levels grouped by the common prefix.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple_red', 'apple_green', 'banana_yellow', 'banana_green', 'cherry_red'))

# Group by first 5 characters (common prefix)
ft_group_by_prefix(factor_vec, prefix_length = 5)
```

---

ft_impute *Impute Missing Values in Factor Vector*

Description

Replaces NA values in a factor vector using specified imputation methods.

Usage

```
ft_impute(factor_vec, method = "mode")
```

Arguments

factor_vec	A factor vector with potential NA values.
method	The imputation method: 'mode', 'random', or a user-defined function.

Value

A factor vector with NA values imputed.

Author(s)

Kai Guo

Examples

```
# Example factor vector with NAs
factor_vec <- factor(c('apple', NA, 'banana', 'apple', NA))

# Impute using mode
ft_impute(factor_vec, method = 'mode')

# Impute using random selection
ft_impute(factor_vec, method = 'random')
```

---

ft_insert *Insert New Levels into a Factor Vector at Specified Positions*

Description

Inserts one or more new levels into a factor vector immediately before or after specified target levels or positions. Each new level corresponds to its respective target level or position. If `positions` or `target` has a single element, it will be repeated to match the length of `insert`. Supports exact matches, position-based targeting, and pattern-based matching with optional case sensitivity. Can handle multiple insertions, manage duplicates, and optionally reorder the data vector's elements to align with the new levels.

If any position in `positions` exceeds the number of levels in `factor_vec`, the new level(s) will be appended at the end.

Usage

```
ft_insert(
  factor_vec,
  insert,
  target = NULL,
  positions = NULL,
  pattern = NULL,
  case = FALSE,
  insert_after_na = FALSE,
  allow_duplicates = FALSE,
  position = "after",
  inplace = FALSE
)
```

Arguments

factor_vec	A factor vector into which new levels will be inserted.
insert	A character vector of new levels to insert. Each new level corresponds to the respective target level or position.
target	A character vector specifying the levels before or after which the new levels will be inserted. If length 1 and length(insert) > 1, it will be repeated to match the length of insert. Overrides positions and pattern if provided.
positions	An integer vector specifying the positions of levels before or after which the new levels will be inserted. If length 1 and length(insert) > 1, it will be repeated to match the length of insert. If any position exceeds the number of levels, the new level(s) will be appended at the end. Overrides target and pattern if both are provided.
pattern	A regular expression pattern to identify target levels for insertion. Overrides both target and positions if provided.
case	Logical. Should pattern matching be case-sensitive? Defaults to FALSE.
insert_after_na	Logical. Should NA be considered as a target level for insertion? Defaults to FALSE.
allow_duplicates	Logical. If TRUE, allows insertion of new levels that already exist in the factor by making them unique (appending suffixes). Defaults to FALSE.
position	Character. Where to insert the new levels relative to the target: "after" or "before". Defaults to "after".
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A new factor vector with the new levels inserted at the specified positions. If inplace = TRUE, the data vector's elements are reordered to match the new levels' order. If inplace = FALSE, only the levels' order is adjusted without changing the data vector's elements' order.

Author(s)

Kai Guo

Examples

```
# Example 1: Insert 'date' after position 2 and 'grape' after position 4
# without allowing duplicates, returning a new factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig', 'grape'))
new_factor <- ft_insert(
  factor_vec,
  insert = c('date', 'grape'),
  positions = c(2, 4),
```

```
  inplace = FALSE
)
print(new_factor)
# [1] apple  banana date   cherry fig    grape
# Levels: apple banana date cherry fig grape

# Example 2: Insert 'date' after position 2 and 'grape' after position 4,
# allowing duplicates, returning a new factor vector
new_factor_dup <- ft_insert(
  factor_vec,
  insert = c('date', 'grape'),
  positions = c(2, 4),
  allow_duplicates = TRUE,
  inplace = FALSE
)
print(new_factor_dup)
# [1] apple  banana date   cherry fig    grape.1
# Levels: apple banana date cherry fig grape.1

# Example 3: Insert 'date' after position 2 and 'grape' after position 4,
# and reorder data elements
new_factor_inplace <- ft_insert(
  factor_vec,
  insert = c('date', 'grape'),
  positions = c(2, 4),
  inplace = TRUE
)
print(new_factor_inplace)
# [1] apple  banana date   cherry fig    grape
# Levels: apple banana date cherry fig grape

# Example 4: Insert 'kiwi' after 'banana' and 'grape', case-sensitive,
# allowing duplicates, returning a new factor vector
factor_vec_case <- factor(c('Apple', 'banana', 'Cherry', 'date', 'Fig', 'grape'))
new_factor_case <- ft_insert(
  factor_vec_case,
  insert = c('kiwi', 'kiwi'),
  target = c('banana', 'grape'),
  case = TRUE,
  allow_duplicates = TRUE,
  inplace = FALSE
)
print(new_factor_case)
# [1] Apple   banana  Cherry  date    Fig     grape   kiwi    kiwi.1

# Example 5: Insert 'date' and 'elderberry' after position 2
factor_vec <- factor(c('apple', 'banana', 'cherry', 'fig', 'grape'))
new_factor <- ft_insert(
  factor_vec,
  insert = c('date', 'elderberry'),
  positions = 2,
  position = "after",
  inplace = FALSE
```

```
)
print(levels(new_factor))
# [1] "apple"     "banana"    "date"      "elderberry" "cherry"    "fig"       "grape"

# Example 6: Insert 'kiwi' at position exceeding the number of levels
new_factor_exceed <- ft_insert(
  factor_vec,
  insert = 'kiwi',
  positions = 10,  # Position exceeds number of levels
  position = "after",
  inplace = FALSE
)
print(levels(new_factor_exceed))
# [1] "apple" "banana" "cherry" "fig" "grape" "kiwi"

# Example 7: Insert multiple levels with positions exceeding the number of levels
new_factor_multi_exceed <- ft_insert(
  factor_vec,
  insert = c('lemon', 'mango'),
  positions = c(5, 10),  # Second position exceeds number of levels
  position = "after",
  inplace = FALSE
)
print(levels(new_factor_multi_exceed))
# [1] "apple" "banana" "cherry" "fig" "grape" "lemon" "mango"
# Example 8: Insert multiple levels after a single position (positions repeated)
new_factor_repeat <- ft_insert(
  factor_vec,
  insert = c('kiwi', 'lemon', 'mango'),
  positions = 2,
  position = "after",
  inplace = FALSE
)
print(levels(new_factor_repeat))
# [1] "apple" "banana" "kiwi" "lemon" "mango" "cherry" "fig" "grape"

# Example 9: Insert multiple levels before a single target (target repeated)
new_factor_target_repeat <- ft_insert(
  factor_vec,
  insert = c('kiwi', 'lemon', 'mango'),
  target = 'cherry',
  position = "before",
  inplace = FALSE
)
print(levels(new_factor_target_repeat))
# [1] "apple" "banana" "kiwi" "lemon" "mango" "cherry" "fig" "grape"
```

`ft_intersect` *Get Intersection of Factor Levels from Multiple Vectors*

Description

Combines multiple factor vectors and returns a factor vector containing only the levels common to all.

Usage

```
ft_intersect(...)
```

Arguments

...	Factor vectors to be intersected.

Value

A factor vector containing the intersection of levels from all provided factors.

Author(s)

Kai Guo

Examples

```
# Example factor vectors
factor_vec1 <- factor(c('apple', 'banana', 'cherry'))
factor_vec2 <- factor(c('banana', 'date', 'cherry'))
factor_vec3 <- factor(c('banana', 'cherry', 'fig'))

# Get intersection of levels
ft_intersect(factor_vec1, factor_vec2, factor_vec3)
```

---

ft_len *Sort Factor Levels Based on Their Length*

Description

Reorders the levels of a factor vector based on the character length of each level. Optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_len(factor_vec, decreasing = FALSE, inplace = FALSE)
```

Arguments

factor_vec	A factor vector to be sorted.
decreasing	Logical. Should the ordering be decreasing by length? Default is FALSE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with levels reordered based on their length without changing the data vector's elements' order. Defaults to FALSE.

Value

A factor vector with levels reordered based on their length. Depending on the `inplace` parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date'))

# Sort levels by length without reordering data elements
sorted_factor <- ft_len(factor_vec)
print(sorted_factor)
# [1] apple  banana cherry date
# Levels: apple date banana cherry

# Sort levels by length and reorder data elements
sorted_factor_inplace <- ft_len(factor_vec, inplace = TRUE)
print(sorted_factor_inplace)
# [1] date   apple  banana cherry
# Levels: apple date banana cherry
```

`ft_level_lengths` *Get Character Lengths of Factor Levels*

Description

Calculates the number of characters in each level of a factor vector.

Usage

```
ft_level_lengths(factor_vec)
```

Arguments

`factor_vec`	A factor vector.

Value

A named numeric vector with the length of each level.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry'))

# Get level lengths
ft_level_lengths(factor_vec)
```

`ft_level_order` *Get Order of Factor Levels in Data*

Description

Returns a vector indicating the order in which factor levels appear in the data.

Usage

```
ft_level_order(factor_vec)
```

Arguments

`factor_vec`	A factor vector.

Value

A numeric vector representing the order of levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('banana', 'apple', 'cherry', 'apple', 'banana'))

# Get level order
ft_level_order(factor_vec)
```

ft_level_stats *Calculate Statistics for Each Factor Level*

Description

Computes statistical summaries for each level of a factor vector based on associated numeric data.

Usage

```
ft_level_stats(factor_vec, numeric_vec, stat_func)
```

Arguments

factor_vec	A factor vector.
numeric_vec	A numeric vector of the same length as factor_vec.
stat_func	A function to compute the statistic (e.g., mean, median).

Value

A data frame with factor levels and their corresponding statistics.

Author(s)

Kai Guo

Examples

```
# Example data
factor_vec <- factor(c('A', 'B', 'A', 'B', 'C'))
numeric_vec <- c(10, 20, 15, 25, 30)

# Calculate mean for each level
ft_level_stats(factor_vec, numeric_vec, stat_func = mean)
```

ft_mapping *Create a Mapping Table of Original and Modified Factor Levels*

Description

Creates a data frame mapping the original factor levels to the modified levels.

Usage

```
ft_mapping(original_factor, modified_factor)
```

Arguments

original_factor	The original factor vector before modification.
modified_factor	The modified factor vector after modification.

Value

A data frame containing the mapping of original to modified levels.

Author(s)

Kai Guo

Examples

```
# Original and modified factor vectors
original_factor <- factor(c('apple', 'banana', 'cherry'))
modified_factor <- factor(c('apple_fruit', 'banana_fruit', 'cherry_fruit'))

# Create mapping table
ft_mapping(original_factor, modified_factor)
```

---

ft_map_func *Map Factor Levels Using a Function*

Description

Transforms factor levels by applying a function that can include complex logic.

Usage

```
ft_map_func(factor_vec, map_func)
```

Arguments

factor_vec	A factor vector to map.
map_func	A function that takes a character vector of levels and returns a character vector of new levels.

Value

A factor vector with levels mapped according to the function.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry'))

# Map levels to uppercase if they start with 'a'
ft_map_func(factor_vec, function(x) {
  ifelse(grepl('^a', x), toupper(x), x)
})
```

`ft_merge` *Merge Two Factors and Retain Unique Levels*

Description

Merges two factor vectors into one, retaining unique levels from both factors.

Usage

```
ft_merge(factor_vec1, factor_vec2, level_order = NULL)
```

Arguments

`factor_vec1`	The first factor vector.
`factor_vec2`	The second factor vector.
`level_order`	A character vector specifying the desired order of levels. If NULL, levels are ordered by their first appearance.

Value

A factor vector containing the combined data from both factors with unique levels.

Author(s)

Kai Guo

Examples

```
# Example factor vectors
factor_vec1 <- factor(c('apple', 'banana', 'cherry'))
factor_vec2 <- factor(c('banana', 'date', 'fig', 'grape'))

# Merge factors and retain unique levels
ft_merge(factor_vec1, factor_vec2)
```

ft_merge_similar *Merge Similar Factor Levels*

Description

Merges levels of a factor that are similar based on string distance.

Usage

```
ft_merge_similar(factor_vec, max_distance = 1, method = "lv")
```

Arguments

factor_vec	A factor vector to modify.
max_distance	A numeric value specifying the maximum string distance for merging levels.
method	The method for computing string distance (default is 'lv' for Levenshtein distance).

Value

A factor vector with similar levels merged.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'appel', 'banana', 'bananna', 'cherry'))

# Merge similar levels
ft_merge_similar(factor_vec, max_distance = 1)
```

ft_na *Handle NA Values in Factor Vectors*

Description

Handles NA values in a factor vector by either keeping NA as a level or removing levels and characters corresponding to NA values.

Usage

```
ft_na(factor_vec, keep_na = TRUE)
```

Arguments

factor_vec	A factor vector to be processed.
keep_na	Logical. Should NA values be kept as a level in the factor? Default is TRUE.

Value

A factor vector with NA values handled as specified.

Author(s)

Kai Guo

Examples

```
# Example factor vector with NA values
factor_vec <- factor(c('apple', NA, 'banana', 'cherry', NA, 'date'))

# Keep NA as a level
ft_na(factor_vec, keep_na = TRUE)

# Remove NA values
ft_na(factor_vec, keep_na = FALSE)
```

---

ft_pad_levels *Pad Factor Levels with Leading Characters*

Description

Pads each level of a factor vector with leading characters to reach a specified width.

Usage

```
ft_pad_levels(factor_vec, width, pad_char)
```

Arguments

factor_vec	A factor vector whose levels will be padded.
width	An integer specifying the desired total width for each level after padding.
pad_char	A character string used for padding. Can be of length one or more characters.

Value

A factor vector with padded levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('A', 'B', 'C', 'D'))

# Pad levels to width 4 using '0' as padding character
padded_factor <- ft_pad_levels(factor_vec, width = 4, pad_char = '0')
print(levels(padded_factor))
# Output: "000A" "000B" "000C" "000D"

# Pad levels to width 6 using '%A' as padding string
padded_factor <- ft_pad_levels(factor_vec, width = 6, pad_char = '%A')
print(levels(padded_factor))
# Output: "%%A%A" "%%A%B" "%%A%C" "%%A%D"
```

---

ft_pairs *Generate Pairwise Comparisons Between Elements in factor*

Description

Creates all unique pairwise combinations between factor_vec of a vector, with options for references, symmetry, NA handling, custom filtering, and output formats. Automatically handles factors by converting them to vectors and removes extra spaces from factor_vec before processing.

Usage

```
ft_pairs(
  factor_vec,
  ref = NULL,
  symmetric = TRUE,
  include_na = FALSE,
  include_self = FALSE,
  filter_fn = NULL,
  pre_fn = trimws,
  sort_by = "none",
  output_format = "data.frame"
)
```

Arguments

factor_vec	A vector containing the factor_vec to compare. Can be of any type (character, numeric, factor, etc.).
ref	Optional. A vector containing the reference factor_vec. If NULL (default), comparisons are made within the factor_vec vector.
symmetric	Logical. If TRUE (default), unique unordered pairs are returned. If FALSE, all ordered pairs are returned.
include_na	Logical. If FALSE (default), NA values are excluded from comparisons. If TRUE, NA values are included.

include_self	Logical. If FALSE (default), pairs where Var1 == Var2 are excluded. If TRUE, they are included.
filter_fn	Optional. A custom function to filter the pairs. Should accept a data frame and return a logical vector.
pre_fn	Optional. A function to preprocess the factor_vec before comparison (e.g., tolower, trimws). Default is trimws.
sort_by	Character string specifying how to sort the output. Options are "Var1", "Var2", "both", or "none" (default).
output_format	Character string specifying the output format. Options are "data.frame" (default), "list", or "matrix".

Value

A data frame, list, matrix, or tibble containing pairwise comparisons between factor_vec.

Examples

```
# Example vector with extra spaces and mixed case
vec <- c(' A', 'b ', ' C ', 'D')

# Generate pairwise comparisons within vec
ft_pairs(vec)

# Use a custom preprocessing function to convert factor_vec to lower case
ft_pairs(vec, pre_fn = function(x) tolower(trimws(x)))
ft_pairs(vec, ref = c("A","C"))
```

---

ft_pos *Reorder Factor Levels Based on Characters at Specified Positions*

Description

Reorders the levels of a factor vector based on characters extracted from specified positions within each level's name. Supports case sensitivity, descending order, and optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_pos(
  factor_vec,
  positions,
  case = FALSE,
  decreasing = FALSE,
  inplace = FALSE
)
```

Arguments

factor_vec	A factor vector whose levels will be reordered.
positions	An integer vector specifying the character positions to extract from each level's name for ordering.
case	Logical. If TRUE, case is considered during ordering. If FALSE, all characters are converted to lowercase before ordering. Defaults to FALSE.
decreasing	Logical. If TRUE, the levels are ordered in decreasing order based on the extracted characters. Defaults to FALSE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A new factor vector with reordered levels. Depending on the inplace parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example 1: Reorder levels based on characters at positions 2 and 4
# without reordering data elements
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig', 'grape'))
new <- ft_pos(
  factor_vec,
  positions = c(2, 4),
  case = FALSE,
  decreasing = FALSE,
  inplace = FALSE
)
print(new)
# [1] apple  banana cherry date   fig    grape
# Levels: apple banana date cherry fig grape

# Example 2: Reorder levels based on characters at positions 2 and 4
# and reorder data elements
new_inplace <- ft_pos(
  factor_vec,
  positions = c(2, 4),
  case = FALSE,
  decreasing = FALSE,
  inplace = TRUE
)
print(new_inplace)
# [1] apple  banana date   cherry fig    grape
# Levels: apple banana date cherry fig grape
```

```
# Example 3: Reorder levels in decreasing order based on characters at
# positions 1 and 3 without reordering data elements
new_dec <- ft_pos(
  factor_vec,
  positions = c(1, 3),
  case = FALSE,
  decreasing = TRUE,
  inplace = FALSE
)
print(new_dec)
# [1] apple  banana cherry date   fig    grape
# Levels: grape fig date cherry banana apple

# Example 4: Reorder levels with case sensitivity and reorder data elements
factor_vec_case <- factor(c('Apple', 'banana', 'Cherry', 'date', 'Fig', 'grape'))
new_case <- ft_pos(
  factor_vec_case,
  positions = c(1, 2),
  case = TRUE,
  decreasing = FALSE,
  inplace = TRUE
)
print(new_case)
# [1] Apple  banana Cherry date   Fig    grape
# Levels: Apple banana Cherry date Fig grape

# Example 5: Reorder levels based on characters at positions 3, allowing
# insertion at positions beyond string length
factor_vec_short <- factor(c('go', 'dog', 'cat', 'bird'))
new_short <- ft_pos(
  factor_vec_short,
  positions = c(3),
  case = FALSE,
  decreasing = FALSE,
  inplace = FALSE
)
print(new_short)
# [1] go   dog  cat  bird
# Levels: cat dog bird go
```

`ft_regex_freq` *Reorder Factor Levels Based on Regex Pattern Frequency*

Description

Reorders the levels of a factor vector based on the frequency of substrings matching a regular expression.

Usage

```
ft_regex_freq(
  factor_vec,
  pattern,
  case = FALSE,
  decreasing = TRUE,
  inplace = TRUE
)
```

Arguments

factor_vec	A factor vector whose levels are to be reordered.
pattern	A string representing the regular expression pattern to match.
case	Logical. Should the pattern matching be case-sensitive? Default is FALSE.
decreasing	Logical. Should the ordering be decreasing by frequency? Default is TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on the frequency of matched substrings.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'apricot', 'cherry', 'blueberry', 'blackberry', 'date'))

# Reorder based on pattern matching 'a'
ft_regex_freq(factor_vec, pattern = 'a')

# Reorder with case-sensitive matching
ft_regex_freq(factor_vec, pattern = '^[A-Z]', case = TRUE)
```

---

ft_remove_levels *Remove Specified Levels from a Factor*

Description

Removes specified levels from a factor vector, keeping the remaining levels and their order unchanged.

Usage

```
ft_remove_levels(factor_vec, levels_to_remove, remove_na = TRUE)
```

Arguments

factor_vec	A factor vector from which levels will be removed.
levels_to_remove	A character vector of levels to be removed from the factor.
remove_na	remove NA from the output? Default is TRUE.

Value

A factor vector with specified levels removed and remaining levels unchanged.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig', 'grape'))

# Remove levels 'banana' and 'date'
ft_remove_levels(factor_vec, levels_to_remove = c('banana', 'date'))
```

---

ft_rename_levels *Rename Factor Levels Using Data Frame Mapping*

Description

Renames the levels of a factor vector based on a mapping provided in a data frame.

Usage

```
ft_rename_levels(factor_vec, mapping_df)
```

Arguments

factor_vec	A factor vector to modify.
mapping_df	A data frame with two columns: 'old' and 'new', representing old and new level names.

Value

A factor vector with levels renamed.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('A', 'B', 'C'))

# Mapping data frame
mapping_df <- data.frame(old = c('A', 'B'), new = c('Alpha', 'Beta'))

# Rename levels
ft_rename_levels(factor_vec, mapping_df)
```

---

ft_reorder_within *Reorder Factor Levels Within Groups*

Description

Reorders the levels of a factor vector within groups defined by another factor vector.

Usage

```
ft_reorder_within(factor_vec, group_vec, by, fun = mean, decreasing = FALSE)
```

Arguments

factor_vec	A factor vector to be reordered.
group_vec	A factor vector defining the groups.
by	A numeric vector to order by.
fun	A function to summarize within groups (e.g., mean, median).
decreasing	Logical. Should the ordering be decreasing? Default is FALSE.

Value

A factor vector with levels reordered within groups.

Author(s)

Kai Guo

Examples

```
# Example data
data <- data.frame(
  item = factor(c('A', 'B', 'C', 'D', 'E', 'F')),
  group = factor(c('G1', 'G1', 'G1', 'G2', 'G2', 'G2')),
  value = c(10, 15, 5, 20, 25, 15)
)
data <- rbind(data, data)
# Reorder 'item' within 'group' by 'value'
data$item <- ft_reorder_within(data$item, data$group, data$value, mean)
```

`ft_replace` *Replace a Factor Level and Optionally Insert at Specified Position*

Description

Replaces a specified level in a factor vector with a new level. If a position is provided, the new level is inserted at the specified position among the levels; otherwise, the original level order is preserved.

Usage

```
ft_replace(factor_vec, old_level, new_level, position = NULL)
```

Arguments

`factor_vec`	A factor vector in which a level will be replaced.
`old_level`	A character string specifying the level to be replaced.
`new_level`	A character string specifying the new level to replace the old level.
`position`	Optional. A positive integer specifying the position to insert the new level in the levels vector. If `NULL`, the original level order is preserved. Default is `NULL`.

Value

A factor vector with the level replaced and the new level optionally inserted at the specified position.

Author(s)

Kai Guo

Examples

```
#
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig', 'grape'))

# replace 'banana' as 'blueberry', and keep original order
ft_replace(factor_vec, old_level = 'banana', new_level = 'blueberry')

# replace 'banana' as 'blueberry'
ft_replace(factor_vec, old_level = 'banana', new_level = 'blueberry', position = 2)
```

ft_replace_na *Replace NA Values in Factor Vector*

Description

Replaces NA values in a factor vector with a specified level.

Usage

```
ft_replace_na(factor_vec, replacement_level)
```

Arguments

factor_vec	A factor vector.
replacement_level	A string specifying the level to replace NA values with.

Value

A factor vector with NA values replaced.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', NA, 'banana', 'cherry', NA))

# Replace NAs with 'Unknown'
ft_replace_na(factor_vec, replacement_level = 'Unknown')
```

ft_replace_pattern *Replace Parts of Factor Levels Based on a Pattern*

Description

Replaces parts of the factor levels that match a specified pattern with a new string.

Usage

```
ft_replace_pattern(factor_vec, pattern, replacement, replace_all = TRUE)
```

Arguments

factor_vec	A factor vector to be modified.
pattern	A regular expression pattern to match.
replacement	A string to replace the matched parts.
replace_all	Logical. If TRUE (default), all occurrences of the pattern are replaced. If FALSE, only the first occurrence is replaced.

Value

A factor vector with levels modified.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple_pie', 'banana_bread', 'cherry_cake'))

# Replace '_pie', '_bread', '_cake' with '_dessert' (all occurrences)
ft_replace_pattern(factor_vec, pattern = '_.*', replacement = '_dessert')

# Replace only the first occurrence of '_' with '-'
ft_replace_pattern(factor_vec, pattern = '_', replacement = '-', replace_all = FALSE)
```

---

ft_reverse *Reverse Factor Levels*

Description

Reverses the order of the levels in a factor vector. Optionally reorders the data vector's elements to align with the reversed levels' order.

Usage

```
ft_reverse(factor_vec, inplace = FALSE)
```

Arguments

factor_vec	A factor vector whose levels will be reversed.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the reversed levels' order. If FALSE, returns a new factor vector with levels reversed without changing the data vector's elements' order. Defaults to FALSE.

Value

A factor vector with levels in reversed order. Depending on the `inplace` parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('low', 'medium', 'high'))

# Reverse the levels without reordering data elements
reversed_factor <- ft_reverse(factor_vec)
print(reversed_factor)
# [1] low    medium high
# Levels: high medium low

# Reverse the levels and reorder data elements
reversed_factor_inplace <- ft_reverse(factor_vec, inplace = TRUE)
print(reversed_factor_inplace)
# [1] high   medium low
# Levels: high medium low
```

---

ft_rollup *Aggregate Factor Levels Based on Grouping*

Description

Aggregates the levels of a factor vector based on another grouping vector.

Usage

```
ft_rollup(factor_vec, groups)
```

Arguments

`factor_vec`	A factor vector to aggregate.
`groups`	A vector of the same length as `factor_vec` indicating group assignments.

Value

A factor vector with aggregated levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector and groups
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date', 'fig'))
groups <- c('fruit', 'fruit', 'fruit', 'dry fruit', 'dry fruit')

# Aggregate levels based on groups
ft_rollup(factor_vec, groups)
```

---

ft_sample_levels *Sample Levels from a Factor Vector*

Description

Randomly selects a specified number of levels from a factor vector.

Usage

```
ft_sample_levels(factor_vec, size, seed = NULL)
```

Arguments

factor_vec	A factor vector.
size	An integer specifying the number of levels to sample.
seed	An optional integer for setting the random seed.

Value

A factor vector containing only the sampled levels.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(letters[1:10])

# Sample 5 levels
ft_sample_levels(factor_vec, size = 5, seed = 123)
```

ft_sort *Sort Factor Levels Based on Another Vector or Column*

Description

Sorts the levels of a factor vector based on the values of another vector or a column from a data frame. Handles cases where the sorting vector may contain 'NA's. Optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_sort(factor_vec, by, decreasing = FALSE, na_last = TRUE, inplace = FALSE)
```

Arguments

factor_vec	A factor vector whose levels are to be sorted.
by	A vector or data frame column used as the basis for sorting. Must be the same length as 'factor_vec'.
decreasing	Logical. Should the sorting be in decreasing order? Default is FALSE.
na_last	Logical. Should 'NA' values be put last? Default is TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels sorted based on 'by'. Depending on the inplace parameter, the data vector's elements may also be reordered.

Examples

```
# Example using a vector without reordering data elements
factor_vec <- factor(c('apple', 'banana', 'cherry', 'date'))
by_vec <- c(2, 3, 1, NA)
sorted_factor <- ft_sort(factor_vec, by = by_vec)
print(sorted_factor)
# [1] apple  banana cherry date
# Levels: cherry apple banana date

# Example using a vector and reordering data elements
sorted_factor_inplace <- ft_sort(factor_vec, by = by_vec, inplace = TRUE)
print(sorted_factor_inplace)
# [1] cherry apple banana date
# Levels: cherry apple banana date

# Example using a data frame column without reordering data elements
```

```
data <- data.frame(
  Category = factor(c('apple', 'banana', 'cherry', 'date')),
  Value = c(2, 3, 1, NA)
)
sorted_factor_df <- ft_sort(data$Category, by = data$Value)
print(sorted_factor_df)
# [1] apple  banana cherry date
# Levels: cherry apple banana date

# Example using a data frame column and reordering data elements
sorted_factor_df_inplace <- ft_sort(data$Category, by = data$Value, inplace = TRUE)
print(sorted_factor_df_inplace)
# [1] cherry apple banana date
# Levels: cherry apple banana date
```

`ft_sort_custom` *Sort Factor Levels Using a Custom Function*

Description

Reorders the levels of a factor vector based on a custom function applied to each level. Optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_sort_custom(factor_vec, sort_func, decreasing = FALSE, inplace = FALSE)
```

Arguments

`factor_vec`	A factor vector to sort.
`sort_func`	A function that takes a character vector (the levels) and returns a vector of the same length to sort by.
`decreasing`	Logical. Should the sort be decreasing? Default is `FALSE`.
`inplace`	Logical. If `TRUE`, returns a new factor vector with elements reordered to align with the new levels' order. If `FALSE`, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to `FALSE`.

Value

A factor vector with levels reordered according to the custom function. Depending on the `inplace` parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('apple', 'banana', 'cherry'))

# Sort levels by reverse alphabetical order without reordering data elements
sorted_custom <- ft_sort_custom(factor_vec, function(x) -rank(x))
print(sorted_custom)
# [1] apple  banana cherry
# Levels: cherry banana apple

# Sort levels by reverse alphabetical order and reorder data elements
sorted_custom_inplace <- ft_sort_custom(factor_vec, function(x) -rank(x), inplace = TRUE)
print(sorted_custom_inplace)
# [1] cherry banana apple
# Levels: cherry banana apple

# Sort levels by length of the level name without reordering data elements
sorted_custom_length <- ft_sort_custom(factor_vec, function(x) nchar(x))
print(sorted_custom_length)
# [1] apple  banana cherry
# Levels: apple cherry banana

# Sort levels by length of the level name and reorder data elements
sorted_custom_length_inplace <- ft_sort_custom(factor_vec, function(x) nchar(x), inplace = TRUE)
print(sorted_custom_length_inplace)
# [1] apple  cherry banana
# Levels: apple cherry banana
```

---

ft_split *Split Factor Levels and Reorder Based on Specified Criteria*

Description

Splits the levels of a factor vector using specified patterns or positions and reorders based on specified parts or criteria. Optionally reorders the data vector's elements to align with the new levels' order.

Usage

```
ft_split(
  factor_vec,
  split_pattern,
  use_pattern = NULL,
  part = 1,
  position = NULL,
  char_freq = FALSE,
  decreasing = FALSE,
  inplace = FALSE
)
```

Arguments

factor_vec	A factor vector to be processed.
split_pattern	A character vector specifying the pattern(s) or position(s) to use for splitting. Can be regular expressions or integer positions.
use_pattern	An integer specifying which pattern to use if multiple patterns are provided. Default is NULL by using all patterns.
part	An integer or integer vector specifying which part(s) to use after splitting (e.g., 1 for the first part). Can be a range or specific indices.
position	An integer or integer vector specifying the character positions within the part(s) to consider.
char_freq	Logical. Should the sorting be based on character frequencies within the specified part(s)? Default is FALSE.
decreasing	Logical. Should the ordering be decreasing? Default is FALSE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on the specified conditions. Depending on the inplace parameter, the data vector's elements may also be reordered.

Author(s)

Kai Guo

Examples

```
# Example 1: Split by patterns '-', '_', or '|' and reorder based on the
# first part without reordering data elements
factor_vec <- factor(c('item1-sub1', 'item2_sub2', 'item3|sub3', 'item1-sub4'))
ft_split(factor_vec, split_pattern = c('-', '_', '\\|'), part = 1, inplace = FALSE)

# Example 2: Use the second pattern '_' for splitting and reorder
# data elements
ft_split(factor_vec, split_pattern = c('-', '_', '\\|'), use_pattern = 2, part = 2, inplace = TRUE)

# Example 3: Reorder based on character frequencies in the specified part
# without reordering data elements
ft_split(factor_vec, split_pattern = '-', part = 2, char_freq = TRUE, inplace = FALSE)

# Example 4: Split by pattern '-' and reorder both levels and data
# elements based on the first part
ft_split(factor_vec, split_pattern = '-', part = 1, inplace = TRUE)
```

ft_split_levels *Split Factor Levels into Multiple Factors*

Description

Splits the levels of a factor vector into multiple factors based on a specified delimiter.

Usage

```
ft_split_levels(factor_vec, delimiter, names = NULL)
```

Arguments

factor_vec	A factor vector to split.
delimiter	A character string used to split the factor levels.
names	A character vector specifying names for the resulting factors. Default is NULL, in which case factors are named 'Factor1', 'Factor2', etc.

Value

A data frame containing the resulting factors.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('red_large', 'blue_small', 'green_medium'))

# Split levels into two factors
ft_split_levels(factor_vec, delimiter = '_')
```

ft_sub *Reorder Factor Levels Based on Substrings*

Description

Reorders the levels of a factor vector based on substrings extracted from the factor levels.

Usage

```
ft_sub(
  factor_vec,
  start_pos = NULL,
  end_pos = NULL,
  case = FALSE,
  decreasing = FALSE,
  inplace = TRUE
)
```

Arguments

factor_vec	A factor vector whose levels are to be reordered.
start_pos	Positive integer. The starting position of the substring. If NULL, starts from the beginning.
end_pos	Positive integer. The ending position of the substring. If NULL, goes to the end of the string.
case	Logical. Should the substring comparison be case-sensitive? Default is FALSE.
decreasing	Logical. Should the ordering be decreasing? Default is FALSE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on the specified substring.

Author(s)

Kai Guo

Examples

```
# Example factor vector
factor_vec <- factor(c('Apple', 'banana', 'Cherry', 'date', 'Fig', 'grape'))

# Reorder based on substring from position 2 to 4
ft_sub(factor_vec, start_pos = 2, end_pos = 4)

# Reorder from position 3 to end, case-sensitive
ft_sub(factor_vec, start_pos = 3, case = TRUE)
```

ft_substr_freq *Reorder Factor Levels Based on Substring Frequency*

Description

Reorders the levels of a factor vector based on the frequency of substrings extracted from the data.

Usage

```
ft_substr_freq(
  factor_vec,
  start_pos = NULL,
  end_pos = NULL,
  case = FALSE,
  decreasing = TRUE,
  inplace = TRUE
)
```

Arguments

factor_vec	A factor vector whose levels are to be reordered.
start_pos	Positive integer. The starting position of the substring. If NULL, starts from the beginning.
end_pos	Positive integer. The ending position of the substring. If NULL, goes to the end of the string.
case	Logical. Should the substring comparison be case-sensitive? Default is FALSE.
decreasing	Logical. Should the ordering be decreasing by frequency? Default is TRUE.
inplace	Logical. If TRUE, returns a new factor vector with elements reordered to align with the new levels' order. If FALSE, returns a new factor vector with only the levels' order adjusted, leaving the data vector's elements' order unchanged. Defaults to FALSE.

Value

A factor vector with levels reordered based on the frequency of substrings.

Author(s)

Kai Guo

Examples

```
# Example factor vector with multi-byte characters
factor_vec <- factor(c('apple', 'banana', 'apricot', 'cherry', 'banana', 'banana', 'date'))
# Reorder from position 2 to end
ft_substr_freq(factor_vec, start_pos = 2)
factor_vec <- factor(c('apple', 'banana', 'apricot', 'cherry', 'banana', 'banana', 'date'))
ft_substr_freq(factor_vec, start_pos = 2, end_pos=3)
```

ft_table *Count Character Frequencies in Factor Levels (Including NA Handling)*

Description

Counts the frequency of each character appearing in the levels of a factor vector, optionally including NA values, and returns a table or vector.

Usage

```
ft_table(factor_vec, case = FALSE, include_na = FALSE, as_table = TRUE)
```

Arguments

factor_vec	A factor vector whose levels will be analyzed.
case	Logical. Should the character count be case-sensitive? Default is FALSE.
include_na	Logical. Should NA levels be included in the character count? Default is FALSE.
as_table	Logical. Should the result be returned as a table? If FALSE, a named vector is returned. Default is TRUE.

Value

A table or named vector of character frequencies.

Author(s)

Kai Guo

Examples

```
# Example factor vector with NA levels
factor_vec <- factor(c('apple', 'banana', NA, 'cherry', 'date', NA, 'fig', 'grape'), exclude = NULL)

# Get character frequencies (case-insensitive), excluding NA levels
ft_table(factor_vec)

# Include NA levels in the character frequencies
ft_table(factor_vec, include_na = TRUE)
```

ft_union *Get Union of Factor Levels from Multiple Vectors*

Description

Combines multiple factor vectors and returns a factor vector containing all unique levels.

Usage

```
ft_union(...)
```

Arguments

...	Factor vectors to be united.

Value

A factor vector containing all unique levels from all provided factors.

Author(s)

Kai Guo

Examples

```
# Example factor vectors
factor_vec1 <- factor(c('apple', 'banana'))
factor_vec2 <- factor(c('banana', 'cherry'))
factor_vec3 <- factor(c('date', 'fig'))

# Get union of levels
ft_union(factor_vec1, factor_vec2, factor_vec3)
```

ft_unique_comb *Create Factor of Unique Combinations from Multiple Factors*

Description

Generates a new factor where each level represents a unique combination of levels from the input factors.

Usage

```
ft_unique_comb(..., sep = "_")
```

Arguments

...	Factor vectors to combine.
sep	A string to separate levels in the combined factor. Default is '_'.

Value

A factor vector representing unique combinations.

Author(s)

Kai Guo

Examples

```
# Example factors
factor_vec1 <- factor(c('A', 'A', 'B', 'B'))
factor_vec2 <- factor(c('X', 'Y', 'X', 'Y'))

# Create unique combinations
combined_factor <- ft_unique_comb(factor_vec1, factor_vec2)
levels(combined_factor)
```

Index