Package: countmaskr (via r-universe)

October 12, 2026

Title Small Cell Masking Tool for One- & Two-Way Tabular Reports

Version 0.1.1

Description Provides automated small-cell suppression for one- and two-way frequency tables. Cells falling below a user-defined frequency threshold are masked, with suppression propagated to secondary cells to prevent indirect disclosure. Designed for clinical and health administrative data, the package supports a range of tabular structures and fits into reproducible reporting pipelines, reducing manual review while applying consistent suppression rules across data sharing workflows.

License MPL-2.0

Depends R (>= 2.10), dplyr, tibble, tidyr

Imports lifecycle

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://query-fulfillment.github.io/countmaskr/

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev

Repository https://query-fulfillment.r-universe.dev

Date/Publication 2026-04-15 13:37:57 UTC

RemoteUrl https://github.com/query-fulfillment/countmaskr

RemoteRef HEAD

RemoteSha c284c53db988e6bf7a13378b84898030d3122cce

Contents

countmaskr_data 2
mask_counts 2
mask_counts_2 4
mask_table 6
perturb_counts 8

Index **11**

`countmaskr_data` *countmaskr dataset*

Description

A synthetic dataset of 1500 subjects with demographic information ascertaining at least one category from the demographics has counts less than 11.

Usage

```
data(countmaskr_data)
```

Format

`countmaskr_data`**:**

A data frame with 1500 rows and 6 columns:

id Subject id

age Age in years as integer

gender Administrative sex

race Race

ethnicity Ethnicity

age_group Age as categorical variable ...

`mask_counts` *Perform threshold-based cell masking with primary and secondary masking (Algorithm 1 - A1)*

Description

Identifies primary and secondary cells in a numeric vector and masks them according to the specified threshold.

Usage

```
mask_counts(x, threshold = 11, zero_masking = FALSE, secondary_cell = "min")
```

Arguments

x	A numeric vector.
threshold	A positive numeric value specifying the threshold below which values must be suppressed. Default is 11.
zero_masking	Logical; if TRUE, zeros can be masked as secondary cells when present. Default is FALSE.
secondary_cell	Character string specifying the method for selecting secondary cells when necessary. Options are "min", "max", or "random". Default is "min".

Details

The function operates in two main steps: **primary masking** and **secondary masking**.

Primary Masking: Values greater than 0 and less than the specified threshold are considered primary cells. These values are masked by replacing them with <threshold.

Secondary Masking: Secondary masking is applied to prevent the deduction of masked primary cells from the totals. The logic for identifying the need for secondary masking is based on the following conditions:

- **Condition A**: Only one primary masked cell exists, and there are other counts greater than or equal to the threshold.
- **Condition B**: Two or more counts of 1 are masked, and there are other counts greater than or equal to the threshold.
- **Condition C**: The threshold is 11, and two or more counts of 10 are masked, and there are other counts greater than or equal to the threshold.

If any of these conditions are met, secondary masking is performed as follows:

- If zero_masking is TRUE and zeros are present in the data, one zero is randomly selected and masked as <threshold.
- If zeros are not to be masked or not present, a non-zero cell is selected for masking based on the secondary_cell parameter:
  - "min": The smallest unmasked count greater than zero is selected.
  - "max": The largest unmasked count is selected.
  - "random": A random unmasked count is selected.

The selected secondary cell is then masked by calculating a new masking threshold using the formula:

$$mask_value = 5 \times \lceil (selected_value + 1)/5 \rceil$$

The formula calculates the masking threshold by first adding 1 to the selected value, then dividing by 5, and rounding up to the nearest whole number. This result is then multiplied by 5 to get the final mask_value. Essentially, it rounds the selected value up to the next multiple of 5 after incrementing it by 1.

The cell is then replaced with <mask_value.

Value

A character vector with primary and/or secondary masked cells.

Examples

```
x1 <- c(5, 11, 43, 55, 65, 121, 1213, 0, NA)
x2 <- c(1, 1, 1, 55, 65, 121, 1213, 0, NA)
x3 <- c(11, 10, 10, 55, 65, 121, 1213, 0, NA)

mask_counts(x1)
mask_counts(x2)
mask_counts(x3)

if (requireNamespace("dplyr", quietly = TRUE) && requireNamespace("tidyr", quietly = TRUE)) {
  data("countmaskr_data")

  aggregate_table <- countmaskr_data %>%
    dplyr::select(-c(id, age)) %>%
    tidyr::gather(block, Characteristics) %>%
    dplyr::group_by(block, Characteristics) %>%
    dplyr::summarise(N = dplyr::n()) %>%
    dplyr::ungroup()

  aggregate_table %>%
    dplyr::group_by(block) %>%
    dplyr::mutate(N_masked = mask_counts(N))
}
```

`mask_counts_2` *Perform threshold-based cell masking with primary and secondary masking (Algorithm 2 - A2)*

Description

This function masks values in a numeric vector based on a specified threshold, using primary and secondary masking to ensure data privacy.

Usage

```
mask_counts_2(x, threshold = 11, zero_masking = FALSE)
```

Arguments

`x`	Numeric vector to mask.
`threshold`	Positive numeric value for the threshold below which cells are masked. Default is 11.
`zero_masking`	Logical; if `TRUE`, zeros may be masked as secondary cells if present. Default is `FALSE`.

Details

The function operates in two main steps:

- **Primary Masking**: Values greater than 0 but less than the threshold are masked by replacing them with `<threshold`.
- **Secondary Masking**: Applied when additional masking is required to prevent deduction of masked cells from totals. Secondary masking is triggered under the following conditions:
  - **Condition A**: A single primary masked cell exists, and there are other values that meet or exceed the threshold.
  - **Condition B**: Two or more counts of 1 are masked, with other values meeting or exceeding the threshold.
  - **Condition C**: The threshold is set to 11, with two or more counts of 10 masked and other counts meeting or exceeding the threshold.

If any of these conditions are met:

- When `zero_masking = TRUE` and zeros are present, one zero is randomly selected and masked as `<threshold`.
- When `zero_masking = FALSE` (or zeros are absent), the function masks the largest unmasked count (i.e., the maximum non-zero value).

Formula for Mask Value Calculation: To calculate the `mask_value` for the secondary cell, the following formula is used:

$$mask_value = selected_value - (threshold - totals_of_small_cells)$$

In words, this formula subtracts the difference between the threshold and the sum of all small cells (those masked in the primary masking step) from the selected maximum unmasked value. This adjusted `mask_value` helps ensure privacy while retaining consistent totals.

Value

A character vector with masked cells, retaining `NA` as `NA_character_`.

Examples

```
x1 <- c(5, 11, 43, 55, 65, 121, 1213, 0, NA)

mask_counts_2(x1)

if (requireNamespace("dplyr", quietly = TRUE) && requireNamespace("tidyr", quietly = TRUE)) {
  data("countmaskr_data")
  countmaskr_data %>%
    dplyr::select(-c(id, age)) %>%
    tidyr::gather(block, Characteristics) %>%
    dplyr::group_by(block, Characteristics) %>%
    dplyr::summarise(N = dplyr::n()) %>%
    dplyr::ungroup() %>%
    dplyr::mutate(N_masked = mask_counts_2(N))
}
```

mask_table *Apply Threshold-Based Masking to a Data Frame*

Description

The mask_table function applies threshold-based masking to specified columns in a data frame. It uses the mask_counts function to mask counts that are below a certain threshold, adhering to data privacy requirements. The function can handle grouped data and calculate percentages if required. It ensures convergence by checking specific criteria after each iteration.

Usage

```
mask_table(
  data,
  threshold = 11,
  col_groups,
  group_by = NULL,
  overwrite_columns = TRUE,
  percentages = FALSE,
  perc_decimal = 0,
  zero_masking = FALSE,
  secondary_cell = "min",
  .verbose = FALSE
)
```

Arguments

data	A data frame containing the counts to be masked. Must be a data frame.
threshold	A positive numeric value specifying the threshold below which values must be suppressed. Default is 11.
col_groups	A character vector or a list of character vectors, where each character vector specifies columns in data to which masking should be applied.
group_by	An optional character string specifying a column name in data to group the data by before masking.
overwrite_columns	Logical; if TRUE, the original columns are overwritten with masked counts. If FALSE, new columns are added with masked counts. Default is TRUE.
percentages	Logical; if TRUE, percentages are calculated and masked accordingly. Default is FALSE.
perc_decimal	= A positive numeric value specifying the decimals for percentages. Default is 0.
zero_masking	Logical; if TRUE, zeros can be masked as secondary cells when present. Passed to mask_counts. Default is FALSE.

secondary_cell	Character string specifying the method for selecting secondary cells when necessary. Options are "min", "max", or "random". Passed to mask_counts. Default is "min".
.verbose	Logical; if TRUE, progress messages are printed during masking. Default is FALSE.

Value

A data frame with masked counts in specified columns. If percentages = TRUE, additional columns with percentages are added. The structure of the returned data frame depends on the overwrite_columns parameter.

See Also

mask_counts

Examples

```
data("countmaskr_data")

aggregate_table <- countmaskr_data %>%
  select(-c(id, age)) %>%
  gather(block, Characteristics) %>%
  group_by(block, Characteristics) %>%
  summarise(N = n()) %>%
  ungroup()

mask_table(aggregate_table,
  group_by = "block",
  col_groups = list("N")
)

mask_table(aggregate_table,
  group_by = "block",
  col_groups = list("N"),
  overwrite_columns = FALSE,
  percentages = TRUE
)

countmaskr_data %>%
  count(race, gender) %>%
  pivot_wider(names_from = gender, values_from = n) %>%
  mutate(across(all_of(c("Male", "Other")), ~ ifelse(is.na(.), 0, .)),
    Overall = Female + Male + Other, .after = 1
  ) %>%
  countmaskr::mask_table(.,
    col_groups = list(c("Overall", "Female", "Male", "Other")),
    overwrite_columns = TRUE,
    percentages = FALSE
  )
```

`perturb_counts` *Perturb Counts in a Vector with Small Cells*

Description

The `perturb_counts` function perturbs counts in a numeric vector containing small cells, specifically when only one primary cell is present and secondary cells need to be masked, following Algorithm 3 (A3). The function adjusts the counts by distributing noise to non-primary cells while preserving the overall distribution as much as possible.

Usage

```
perturb_counts(x, threshold = 10)
```

Arguments

`x`	Numeric vector of length N containing counts.
`threshold`	Numeric value specifying the threshold for small cells (primary cells). Defaults to 10.

Details

Perturbation Process Overview:

The function performs perturbation through the following steps:

1. **Identification of Small Cells**: Cells with counts greater than 0 and less than the specified `threshold` are identified as small cells (primary cells).

$$\text{Small Cells} = \{i \mid 0 < x_i < \text{threshold}\}$$

2. **Adjustment of Small Cells**: The counts of small cells are set to the `threshold` value.

$$x_i' = \begin{cases} \text{threshold} & \text{if } x_i \text{ is a small cell} \\ x_i & \text{otherwise} \end{cases}$$

3. **Calculation of Total Noise**: The total noise to be distributed is calculated as the difference between the original total sum and the adjusted sum.

$$\text{Total Noise} = \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i'$$

4. **Distribution of Noise to Non-Small Cells**: The total noise is proportionally distributed to the non-small cells based on their original counts.
   - **Weights Calculation**:

$$w_i = \frac{x_i}{\sum_{j \in \text{Non-Small Cells}} x_j}$$

- **Noise Allocation**:

$$\text{Noise}_i = w_i \times \text{Total Noise}$$

- **Adjusted Counts**:

$$x_i'' = x_i' + \text{Noise}_i$$

5. **Rounding Adjusted Counts**: The adjusted counts are rounded to the nearest integer.

$$x_i''' = \text{round}(x_i'')$$

6. **Adjustment for Rounding Discrepancies**: Any remaining noise due to rounding discrepancies is adjusted by iteratively adding or subtracting 1 from the largest counts until the total counts are balanced, ensuring that no count falls below the `threshold`.
7. **Verification of Proportions**: The function checks if the proportions of the non-small cells remain consistent before and after perturbation. If the proportions differ, the function coerces to mask counts using the `mask_counts()` function.

Coercion to Mask Counts:

The function coerces to mask counts in the following scenarios:

- **Multiple Small Cells Detected**: If more than one small cell is identified, perturbation may not be necessary unless intended to use. The function will still proceed with perturbation but recommends using threshold-based suppression.
- **Insufficient Available Counts**: If the non-small cells do not have enough counts to absorb the total noise without any count falling below the `threshold`, the operation will lead to information loss.
- **Proportions Changed After Perturbation**: If perturbation alters the original proportions of the non-small cells, the operation will lead to information loss.

#' - **All Counts Below Threshold**: If all counts in the vector are below the specified `threshold`, there is no meaningful perturbation possible. In this case, the function coerces to `mask_counts()` as a more secure alternative.

In these cases, the function calls `mask_counts()` to apply threshold-based cell suppression as a more secure alternative.

Value

A character vector with perturbed counts formatted with digit precision and thousands separator. If perturbation is not feasible, the function returns counts masked using `mask_counts()`.

Examples

```
# Example vectors
x1 <- c(5, 11, 43, 55, 65, 121, 1213, 0, NA)
x2 <- c(1, 1, 1, 55, 65, 121, 1213, 0, NA)
x3 <- c(11, 10, 10, 55, 65, 121, 1213, 0, NA)

# Apply the function
lapply(list(x1, x2, x3), perturb_counts)
```

```
# Using the function within a data frame
data("countmaskr_data")
aggregate_table <- countmaskr_data %>%
  select(-c(id, age)) %>%
  tidyr::gather(block, Characteristics) %>%
  group_by(block, Characteristics) %>%
  summarise(N = n()) %>%
  ungroup()

aggregate_table %>%
  group_by(block) %>%
  mutate(N_masked = perturb_counts(N))
```

Index

∗ **datasets**
countmaskr_data, 2

countmaskr_data, 2

mask_counts, 2, 7
mask_counts(), 9
mask_counts_2, 4
mask_table, 6

perturb_counts, 8