Package 'groupdata2'

October 22, 2017

Title Creating Groups from Data

Version 1.0.0

Date 2017-10-22

Description Subsetting methods for balanced cross-validation, time series windowing, and general grouping and splitting of data.

Depends R (>= 3.3.1)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr (>= 0.7.0), plyr, utils, numbers

RoxygenNote 6.0.1

Suggests ggplot2, knitr, rmarkdown, tidyr, broom, testthat, lmerTest, hydroGOF

VignetteBuilder knitr

NeedsCompilation no

Author Ludvig Renbo Olsen [aut, cre]

Maintainer Ludvig Renbo Olsen <r-pkgs@ludvigolsen.dk>

Repository CRAN

Date/Publication 2017-10-22 03:14:18 UTC

R topics documented:

`find_missing_starts` *Find start positions that cannot be found in data.*

Description

Tells you which values and (optionally) skip_to numbers that are recursively removed when using the `l_starts` method with remove_missing_starts set to TRUE.

Usage

```
find_missing_starts(data, n, starts_col = NULL, return_skip_numbers = TRUE)
```

Arguments

`data`	Dataframe or Vector
`n`	List of starting positions. Skip values by c(value, skip_to_number) where skip_to_number is the nth appearance of the value in the vector. See group_factor for explanations and examples of using the `l_starts` method.
`starts_col`	Name of column with values to match when data is a dataframe. Pass 'index' to use row names. (Character)
`return_skip_numbers`	Return skip_to numbers along with values (Logical).

Value

List of start values and skip_to numbers or vector of the start values. Returns NULL if no values found.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other l_starts tools: `find_starts`, `group_factor`, `group`

Examples

```
# Attach packages
library(groupdata2)

# Create a dataframe
df <- data.frame('a' = c('a','a','b',
                         'b','c','c'))
```

```
# Create list of starts
starts <- c("a", "e", "b", "d", "c")

# Find missing starts with skip_to numbers
find_missing_starts(df, starts, starts_col = 'a')

# Find missing starts without skip_to numbers
find_missing_starts(df, starts, starts_col = 'a',
                    return_skip_numbers = FALSE)
```

`find_starts` *Find start positions of groups in data.*

Description

Find values or indices of values that are not the same as the previous value. E.g. use with the `l_starts` method.

Usage

```
find_starts(data, col = NULL, return_index = FALSE)
```

Arguments

`data`	Dataframe or Vector
`col`	Name of column to find starts in. Used when data is dataframe. (Character)
`return_index`	Return indices of starts. (Logical)

Value

Vector with either start values or indices of start values.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other l_starts tools: `find_missing_starts`, `group_factor`, `group`

Examples

```
# Attach packages
library(groupdata2)

# Create a dataframe
df <- data.frame('a' = c('a','a','b',
                         'b','c','c'))
```

```
# Get start values for new groups in column 'a'
find_starts(df, col = 'a')

# Get indices of start values for new groups
# in column 'a'
find_starts(df, col = 'a',
            return_index = TRUE)

## Use found starts with l_starts method
# Notice: This is equivalent to n = 'auto'
# with l_starts method

# Get start values for new groups in column 'a'
starts <- find_starts(df, col = 'a')

# Use starts in group() with 'l_starts' method
group(df, n = starts, method = 'l_starts',
      starts_col = 'a')

# Similar but with indices instead of values

# Get indices of start values for new groups
# in column 'a'
starts_ind <- find_starts(df, col = 'a',
                          return_index = TRUE)

# Use starts in group() with 'l_starts' method
group(df, n = starts_ind, method = 'l_starts',
      starts_col = 'index')
```

`fold` *Create balanced folds for cross-validation.*

Description

Divides data into groups by a range of methods. Balances a given categorical variable between folds and keeps (if possible) all data points with a shared ID (e.g. participant_id) in the same fold.

Usage

```
fold(data, k = 5, cat_col = NULL, id_col = NULL, starts_col = NULL,
  method = "n_dist", remove_missing_starts = FALSE)
```

Arguments

`data`	Dataframe or Vector.

k — *Dependent on method.*

Number of folds (default), fold size, with more (see `method`).

Given as whole number(s) and/or percentage(s) (0 < n < 1).

`cat_col` — Categorical variable to balance between folds.

E.g. when predicting a binary variable (a or b), it is necessary to have both represented in every fold

N.B. If also passing an id_col, cat_col should be constant within each ID.

`id_col` — Factor with IDs. This will be used to keep all rows that share an ID in the same fold (if possible).

E.g. If we have measured a participant multiple times and want to see the effect of time, we want to have all observations of this participant in the same fold.

`starts_col` — Name of column with values to match in method `l_starts` when data is a dataframe. Pass 'index' to use row names. (Character)

`method` — `greedy`, `n_dist`, `n_fill`, `n_last`, `n_rand`, `l_sizes`, `l_starts`, `staircase`, or `primes`.

Notice: examples are sizes of the generated groups based on a vector with 57 elements.

greedy: Divides up the data greedily given a specified group size $(e.g. 10, 10, 10, 10, 10, 7)$.
`n` is group size

n_dist (default): Divides the data into a specified number of groups and distributes excess data points across groups $(e.g. 11, 11, 12, 11, 12)$.
`n` is number of groups

n_fill: Divides the data into a specified number of groups and fills up groups with excess data points from the beginning $(e.g. 12, 12, 11, 11, 11)$.
`n` is number of groups

n_last: Divides the data into a specified number of groups. It finds the most equal group sizes possible, using all data points. Only the last group is able to differ in size $(e.g. 11, 11, 11, 11, 13)$.
`n` is number of groups

n_rand: Divides the data into a specified number of groups. Excess data points are placed randomly in groups (only 1 per group) $(e.g. 12, 11, 11, 11, 12)$.
`n` is number of groups

l_sizes: Divides up the data by a list of group sizes. Excess data points are placed in an extra group at the end. $(e.g. n = list(0.2, 0.3) outputsgroupswithsizes(11, 17, 29))$.
`n` is a list of group sizes

l_starts: Starts new groups at specified values of vector.
`n` is a list of starting positions. Skip values by c(value, skip_to_number) where skip_to_number is the nth appearance of the value in the vector. Groups automatically start from first data point.
$E.g. n = c(1, 3, 7, 25, 50) outputsgroupswithsizes(2, 4, 18, 25, 8)$.
To skip: $givenvectorc("a", "e", "o", "a", "e", "o"), n = list("a", "e", c("o", 2)) outputsgroupswit$

If passing $n =' auto'$ the starting positions are automatically found with `find_starts()`.

staircase: Uses step size to divide up the data. Group size increases with 1 step for every group, until there is no more data ($e.g. 5, 10, 15, 20, 7$).
`n` is step size

primes: Uses prime numbers as group sizes. Group size increases to the next prime number until there is no more data. ($e.g. 5, 7, 11, 13, 17, 4$).
`n` is the prime number to start at

`remove_missing_starts`
: Recursively remove elements from the list of starts that are not found. For method `l_starts` only. (Logical)

Details

`cat_col`: data is first subset by `cat_col`. Subsets are folded/grouped and merged.

`id_col`: folds are created from unique IDs.

`cat_col` AND `id_col`: data is subset by `cat_col` and folds are created from unique IDs in each subset. Subsets are merged.

Value

Dataframe with grouping factor for subsetting in cross-validation.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Examples

```
# Attach packages
library(groupdata2)
library(dplyr)

# Create dataframe
df <- data.frame(
 "participant" = factor(rep(c('1','2', '3', '4', '5', '6'), 3)),
 "age" = rep(sample(c(1:100), 6), 3),
 "diagnosis" = rep(c('a', 'b', 'a', 'a', 'b', 'b'), 3),
 "score" = sample(c(1:100), 3*6))
df <- df[order(df$participant),]
df$session <- rep(c('1','2', '3'), 6)

# Using fold()
# Without cat_col and id_col
df_folded <- fold(df, 3, method = 'n_dist')

# With cat_col
df_folded <- fold(df, 3, cat_col = 'diagnosis',
 method = 'n_dist')

# With id_col
df_folded <- fold(df, 3, id_col = 'participant',
```

```
 method = 'n_dist')

# With cat_col and id_col
df_folded <- fold(df, 3, cat_col = 'diagnosis',
 id_col = 'participant', method = 'n_dist')

# Order by folds
df_folded <- df_folded[order(df_folded$.folds),]
```

---

group *Create groups from your data.*

Description

Divides data into groups by a range of methods. Creates a grouping factor with 1s for group 1, 2s for group 2, etc. Returns a dataframe grouped by the grouping factor for easy use in dplyr pipelines.

Usage

```
group(data, n, method = "n_dist", starts_col = NULL, force_equal = FALSE,
  allow_zero = FALSE, return_factor = FALSE, descending = FALSE,
  randomize = FALSE, col_name = ".groups", remove_missing_starts = FALSE)
```

Arguments

`data`	Dataframe or Vector.
`n`	*Dependent on method.* Number of groups (default), group size, list of group sizes, list of group starts, step size or prime number to start at. See `method`. Passed as whole number(s) and/or percentage(s) (`0 < n < 1`) and/or character. Method `l_starts` allows `'auto'`.
`method`	`greedy`, `n_dist`, `n_fill`, `n_last`, `n_rand`, `l_sizes`, `l_starts`, `staircase`, or `primes`. **Notice**: examples are sizes of the generated groups based on a vector with 57 elements. **greedy:** Divides up the data greedily given a specified group size $(e.g. 10, 10, 10, 10, 10, 7)$. `n` is group size **n_dist (default):** Divides the data into a specified number of groups and distributes excess data points across groups $(e.g. 11, 11, 12, 11, 12)$. `n` is number of groups **n_fill:** Divides the data into a specified number of groups and fills up groups with excess data points from the beginning $(e.g. 12, 12, 11, 11, 11)$. `n` is number of groups

n_last: Divides the data into a specified number of groups. It finds the most equal group sizes possible, using all data points. Only the last group is able to differ in size $(e.g. 11, 11, 11, 11, 13)$.
n is number of groups

n_rand: Divides the data into a specified number of groups. Excess data points are placed randomly in groups (only 1 per group) $(e.g. 12, 11, 11, 11, 12)$.
n is number of groups

l_sizes: Divides up the data by a list of group sizes. Excess data points are placed in an extra group at the end. $(e.g. n = list(0.2, 0.3) outputsgroupswithsizes(11, 17, 29))$.
n is a list of group sizes

l_starts: Starts new groups at specified values of vector.
n is a list of starting positions. Skip values by c(value, skip_to_number) where skip_to_number is the nth appearance of the value in the vector. Groups automatically start from first data point.
$E.g. n = c(1, 3, 7, 25, 50) outputsgroupswithsizes(2, 4, 18, 25, 8)$.
To skip: $givenvectorc("a", "e", "o", "a", "e", "o"), n = list("a", "e", c("o", 2)) outputsgroupswit$

If passing $n =' auto'$ the starting positions are automatically found with `find_starts()`.

staircase: Uses step size to divide up the data. Group size increases with 1 step for every group, until there is no more data $(e.g. 5, 10, 15, 20, 7)$.
n is step size

primes: Uses prime numbers as group sizes. Group size increases to the next prime number until there is no more data. $(e.g. 5, 7, 11, 13, 17, 4)$.
n is the prime number to start at

`starts_col` Name of column with values to match in method `l_starts` when data is a dataframe. Pass 'index' to use row names. (Character)

`force_equal` Create equal groups by discarding excess data points. Implementation varies between methods. (Logical)

`allow_zero` Whether n can be passed as `0`. (Logical)

`return_factor` Return only grouping factor (Logical)

`descending` Change direction of method. (Not fully implemented) (Logical)

`randomize` Randomize the grouping factor (Logical)

`col_name` Name of added grouping factor

`remove_missing_starts`
Recursively remove elements from the list of starts that are not found. For method `l_starts` only. (Logical)

Value

Dataframe grouped by new grouping factor

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other grouping functions: group_factor, splt

Other staircase tools: %primes%, %staircase%, group_factor

Other l_starts tools: find_missing_starts, find_starts, group_factor

Examples

```
# Attach packages
library(groupdata2)
library(dplyr)

# Create dataframe
df <- data.frame("x"=c(1:12),
 "species" = rep(c('cat','pig', 'human'), 4),
 "age" = sample(c(1:100), 12))

# Using group()
df_grouped <- group(df, 5, method = 'n_dist')

# Using group() with dplyr pipeline to get mean age
df_means <- df %>%
 group(5, method = 'n_dist') %>%
 dplyr::summarise(mean_age = mean(age))

# Using group_factor() with l_starts
# "c('pig',2)" skips to the second appearance of
# "pig" after the first appearance of "cat"
df_grouped <- group(df,
                    list('cat', c('pig',2), 'human'),
                    method = 'l_starts',
                    starts_col = 'species')
```

---

groupdata2 *groupdata2: A package for creating groups from data*

Description

Subsetting methods for balanced cross-validation, time series windowing, and general grouping and splitting of data.

Details

The groupdata2 package provides five main functions: group, group_factor, splt, partition and fold

group

Create groups from your data.

Divides data into groups by a range of methods. Creates a grouping factor with 1s for group 1, 2s for group 2, etc. Returns a dataframe grouped by the grouping factor for easy use in dplyr pipelines.

Go to `group`

group_factor

Create grouping factor for subsetting your data.

Divides data into groups by a range of methods. Creates and returns a grouping factor with 1s for group 1, 2s for group 2, etc.

Go to `group_factor`

splt

Split data by a range of methods.

Divides data into groups by a range of methods. Splits data by these groups.

Go to `splt`

partition

Create balanced partitions (e.g. training/test sets).

Splits data into partitions. Balances a given categorical variable between partitions and keeps (if possible) all data points with a shared ID (e.g. participant_id) in the same partition.

Go to `partition`

fold

Create balanced folds for cross-validation.

Divides data into groups (folds) by a range of methods. Balances a given categorical variable between folds and keeps (if possible) all data points with the same ID (e.g. participant_id) in the same fold.

Go to `fold`

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

group_factor *Create grouping factor for subsetting your data.*

Description

Divides data into groups by a range of methods. Creates and returns a grouping factor with 1s for group 1, 2s for group 2, etc.

Usage

```
group_factor(data, n, method = "n_dist", starts_col = NULL,
  force_equal = FALSE, allow_zero = FALSE, descending = FALSE,
  randomize = FALSE, remove_missing_starts = FALSE)
```

Arguments

data	Dataframe or Vector.
n	*Dependent on method.* Number of groups (default), group size, list of group sizes, list of group starts, step size or prime number to start at. See method. Passed as whole number(s) and/or percentage(s) (0 < n < 1) and/or character. Method l_starts allows 'auto'.
method	greedy, n_dist, n_fill, n_last, n_rand, l_sizes, l_starts, staircase, or primes. **Notice**: examples are sizes of the generated groups based on a vector with 57 elements. **greedy:** Divides up the data greedily given a specified group size $(e.g. 10, 10, 10, 10, 10, 7)$. n is group size **n_dist (default):** Divides the data into a specified number of groups and distributes excess data points across groups $(e.g. 11, 11, 12, 11, 12)$. n is number of groups **n_fill:** Divides the data into a specified number of groups and fills up groups with excess data points from the beginning $(e.g. 12, 12, 11, 11, 11)$. n is number of groups **n_last:** Divides the data into a specified number of groups. It finds the most equal group sizes possible, using all data points. Only the last group is able to differ in size $(e.g. 11, 11, 11, 11, 13)$. n is number of groups **n_rand:** Divides the data into a specified number of groups. Excess data points are placed randomly in groups (only 1 per group) $(e.g. 12, 11, 11, 11, 12)$. n is number of groups **l_sizes:** Divides up the data by a list of group sizes. Excess data points are placed in an extra group at the end. $(e.g. n = list(0.2, 0.3) outputsgroupswithsizes(11, 17, 29))$. n is a list of group sizes

l_starts: Starts new groups at specified values of vector.
n is a list of starting positions. Skip values by c(value, skip_to_number) where skip_to_number is the nth appearance of the value in the vector. Groups automatically start from first data point.
$E.g. n = c(1, 3, 7, 25, 50) outputs groups with sizes (2, 4, 18, 25, 8)$.
To skip: $given vector c("a", "e", "o", "a", "e", "o"), n = list("a", "e", c("o", 2)) outputs groups wit$

If passing $n =' auto'$ the starting positions are automatically found with `find_starts()`.

staircase: Uses step size to divide up the data. Group size increases with 1 step for every group, until there is no more data $(e.g. 5, 10, 15, 20, 7)$.
n is step size

primes: Uses prime numbers as group sizes. Group size increases to the next prime number until there is no more data. $(e.g. 5, 7, 11, 13, 17, 4)$.
n is the prime number to start at

`starts_col` Name of column with values to match in method `l_starts` when data is a dataframe. Pass 'index' to use row names. (Character)

`force_equal` Create equal groups by discarding excess data points. Implementation varies between methods. (Logical)

`allow_zero` Whether n can be passed as `0`. (Logical)

`descending` Change direction of method. (Not fully implemented) (Logical)

`randomize` Randomize the grouping factor (Logical)

`remove_missing_starts`
Recursively remove elements from the list of starts that are not found. For method `l_starts` only. (Logical)

Value

Grouping factor with 1s for group 1, 2s for group 2, etc.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other grouping functions: `group`, `splt`

Other staircase tools: `%primes%`, `%staircase%`, `group`

Other l_starts tools: `find_missing_starts`, `find_starts`, `group`

Examples

```
# Attach packages
library(groupdata2)
library(dplyr)

# Create a dataframe
df <- data.frame("x"=c(1:12),
```

```
 "species" = rep(c('cat','pig', 'human'), 4),
 "age" = sample(c(1:100), 12))

# Using group_factor() with n_dist
groups <- group_factor(df, 5, method = 'n_dist')
df$groups <- groups

# Using group_factor() with greedy
groups <- group_factor(df, 5, method = 'greedy')
df$groups <- groups

# Using group_factor() with l_sizes
groups <- group_factor(df, list(0.2, 0.3), method = 'l_sizes')
df$groups <- groups

# Using group_factor() with l_starts
groups <- group_factor(df, list('cat', c('pig',2), 'human'),
                       method = 'l_starts', starts_col = 'species')
df$groups <- groups
```

`partition` *Create balanced partitions.*

Description

Splits data into partitions. Balances a given categorical variable between partitions and keeps (if possible) all data points with a shared ID (e.g. participant_id) in the same partition.

Usage

```
partition(data, p = 0.2, cat_col = NULL, id_col = NULL,
  force_equal = FALSE, list_out = TRUE)
```

Arguments

`data`	Dataframe or Vector.
`p`	List / vector of partition sizes. Given as whole number(s) and/or percentage(s) ($0 < n < 1$). E.g. $c(0.2, 3, 0.1)$.
`cat_col`	Categorical variable to balance between partitions. E.g. when training/testing a model for predicting a binary variable (a or b), it is necessary to have both represented in both the training set and the test set. N.B. If also passing an id_col, cat_col should be constant within each ID.
`id_col`	Factor with IDs. Used to keep all rows that share an ID in the same partition (if possible). E.g. If we have measured a participant multiple times and want to see the effect of time, we want to have all observations of this participant in the same partition.
`force_equal`	Discard excess data. (Logical)
`list_out`	Return partitions in a list. (Logical)

Details

`cat_col`: data is first subset by `cat_col`. Subsets are grouped and merged.

`id_col`: groups are created from unique IDs.

`cat_col` AND `id_col`: data is subset by `cat_col` and groups are created from unique IDs in each subset. Subsets are merged.

Value

If `list_out is TRUE`:

A list of partitions where partitions are dataframes.

If `list_out is FALSE`:

A dataframe with grouping factor for subsetting.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Examples

```
# Attach packages
library(groupdata2)
library(dplyr)

# Create dataframe
df <- data.frame(
 "participant" = factor(rep(c('1','2', '3', '4', '5', '6'), 3)),
 "age" = rep(sample(c(1:100), 6), 3),
 "diagnosis" = rep(c('a', 'b', 'a', 'a', 'b', 'b'), 3),
 "score" = sample(c(1:100), 3*6))
df <- df[order(df$participant),]
df$session <- rep(c('1','2', '3'), 6)

# Using partition()
# Without cat_col and id_col
partitions <- partition(df, c(0.2,0.3))

# With cat_col
partitions <- partition(df, c(0.5), cat_col = 'diagnosis')

# With id_col
partitions <- partition(df, c(0.5), id_col = 'participant')

# With cat_col and id_col
partitions <- partition(df, c(0.5), cat_col = 'diagnosis',
                        id_col = 'participant')

# Return dataframe with grouping factor
# with list_out = FALSE
partitions <- partition(df, c(0.5), list_out = FALSE)
```

splt *Split data by a range of methods.*

Description

Divides data into groups by a range of methods. Splits data by these groups.

Usage

```
splt(data, n, method = "n_dist", starts_col = NULL, force_equal = FALSE,
  allow_zero = FALSE, descending = FALSE, randomize = FALSE,
  remove_missing_starts = FALSE)
```

Arguments

`data`	Dataframe or Vector.
`n`	*Dependent on method.* Number of groups (default), group size, list of group sizes, list of group starts, step size or prime number to start at. See `method`. Passed as whole number(s) and/or percentage(s) (`0 < n < 1`) and/or character. Method `l_starts` allows `'auto'`.
`method`	`greedy`, `n_dist`, `n_fill`, `n_last`, `n_rand`, `l_sizes`, `l_starts`, `staircase`, or `primes`. **Notice**: examples are sizes of the generated groups based on a vector with 57 elements. **greedy:** Divides up the data greedily given a specified group size $(e.g. 10, 10, 10, 10, 10, 7)$. `n` is group size **n_dist (default):** Divides the data into a specified number of groups and distributes excess data points across groups $(e.g. 11, 11, 12, 11, 12)$. `n` is number of groups **n_fill:** Divides the data into a specified number of groups and fills up groups with excess data points from the beginning $(e.g. 12, 12, 11, 11, 11)$. `n` is number of groups **n_last:** Divides the data into a specified number of groups. It finds the most equal group sizes possible, using all data points. Only the last group is able to differ in size $(e.g. 11, 11, 11, 11, 13)$. `n` is number of groups **n_rand:** Divides the data into a specified number of groups. Excess data points are placed randomly in groups (only 1 per group) $(e.g. 12, 11, 11, 11, 12)$. `n` is number of groups **l_sizes:** Divides up the data by a list of group sizes. Excess data points are placed in an extra group at the end. $(e.g. n = list(0.2, 0.3) outputsgroupswithsizes(11, 17, 29))$. `n` is a list of group sizes

l_starts: Starts new groups at specified values of vector.
n is a list of starting positions. Skip values by c(value, skip_to_number) where skip_to_number is the nth appearance of the value in the vector. Groups automatically start from first data point.
$E.g. n = c(1, 3, 7, 25, 50) outputs groups with sizes (2, 4, 18, 25, 8)$.
To skip: $given vector c("a", "e", "o", "a", "e", "o"), n = list("a", "e", c("o", 2)) outputs groups wit$

If passing $n =' auto'$ the starting positions are automatically found with `find_starts()`.

staircase: Uses step size to divide up the data. Group size increases with 1 step for every group, until there is no more data $(e.g. 5, 10, 15, 20, 7)$.
n is step size

primes: Uses prime numbers as group sizes. Group size increases to the next prime number until there is no more data. $(e.g. 5, 7, 11, 13, 17, 4)$.
n is the prime number to start at

`starts_col` Name of column with values to match in method `l_starts` when data is a dataframe. Pass 'index' to use row names. (Character)

`force_equal` Create equal groups by discarding excess data points. Implementation varies between methods. (Logical)

`allow_zero` Whether n can be passed as `0`. (Logical)

`descending` Change direction of method. (Not fully implemented) (Logical)

`randomize` Randomize the grouping factor (Logical)

`remove_missing_starts`
Recursively remove elements from the list of starts that are not found. For method `l_starts` only. (Logical)

Value

List of splitted data

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

See Also

Other grouping functions: `group_factor`, `group`

Examples

```
# Attach packages
library(groupdata2)
library(dplyr)

# Create dataframe
df <- data.frame("x"=c(1:12),
 "species" = rep(c('cat','pig', 'human'), 4),
 "age" = sample(c(1:100), 12))
```

```
# Using splt()
df_list <- splt(df, 5, method = 'n_dist')
```

`%primes%` *Find remainder from primes method.*

Description

When using the primes method, the last group might not have the size of the associated prime number if there are not enough elements left. Use %primes% to find this remainder.

Usage

```
size %primes% start_at
```

Arguments

`size`	Size to group (Integer)
`start_at`	Prime to start at (Integer)

Value

Remainder (Integer). Returns 0 if the last group has the size of the associated prime number.

Author(s)

Ludvig Renbo Olsen, <mail@ludvigolsen.dk>

See Also

Other staircase tools: `%staircase%`, `group_factor`, `group`

Other remainder tools: `%staircase%`

Examples

```
# Attach packages
library(groupdata2)

100 %primes% 2
```

`%staircase%` *Find remainder from staircase method.*

Description

When using the staircase method, the last group might not have the size of the second last group + step size. Use %staircase% to find this remainder.

Usage

```
size %staircase% step_size
```

Arguments

`size`	Size to staircase (Integer)
`step_size`	Step size (Integer)

Value

Remainder (Integer). Returns 0 if the last group has the size of the second last group + step size.

Author(s)

Ludvig Renbo Olsen, <mail@ludvigolsen.dk>

See Also

Other staircase tools: `%primes%`, `group_factor`, `group`

Other remainder tools: `%primes%`

Examples

```
# Attach packages
library(groupdata2)

100 %staircase% 2

# Finding remainder with value 0
size = 150
for (step_size in c(1:30)){
 if(size %staircase% step_size == 0){
   print(step_size)
 }}
```

Index