Package ‘Pedixplorer’

May 4, 2024

Version 1.0.0
Date 2023-11-15
Title Pedigree Functions
Depends R (>= 4.3.0)
Imports graphics, stats, methods, ggplot2, utils, grDevices, stringr, plyr, dplyr, tidyr, quadprog, Matrix, S4Vectors, testthat
Description Routines to handle family data with a Pedigree object. The initial purpose was to create correlation structures that describe family relationships such as kinship and identity-by-descent, which can be used to model family data in mixed effects models, such as in the coxme function. Also includes a tool for Pedigree drawing which is focused on producing compact layouts without intervention. Recent additions include utilities to trim the Pedigree object with various criteria, and kinship for the X chromosome.
License Artistic-2.0
Encoding UTF-8
RoxygenNote 7.2.3
Roxygen list(markdown = TRUE)
VignetteBuilder knitr
Suggests diffviewer, vdiffr, rmarkdown, BiocStyle, knitr, withr, magick
Config/testthat/edition 3
biocViews Software, DataRepresentation, Genetics, Alignment
BugReports https://github.com/LouisLeLezet/Pedixplorer/issues
URL https://github.com/LouisLeNezet/Pedixplorer
BiocType Software
Collate 'AllValidity.R' 'AllClass.R' 'kindepth.R' 'kinship.R'
  'utils.R' 'AllConstructor.R' 'AllAccessors.R' 'AllGeneric.R'
  'Pedixplorer-package.R' 'alignped4.R' 'alignped3.R'
  'alignped2.R' 'alignped1.R' 'auto_hint.R' 'align.R'
  'best_hint.R' 'bit_size.R' 'data.R' 'descendants.R'
'family_check.R' 'find_unavailable.R' 'find_avail_affected.R'
'find_avail_noninform.R' 'fix_parents.R' 'generate_aff_inds.R'
'generate_colors.R' 'ibd_matrix.R' 'is_informative.R'
'make_famid.R' 'min_dist_inf.R' 'norm_data.R' 'num_child.R'
'ped_to_legdf.R' 'ped_to_plotdf.R' 'plot_fct.R' 'plot_fromdf.R'
'plot.R' 'shrink.R' 'unrelated.R' 'useful_inds.R'

LazyData false

git_url https://git.bioconductor.org/packages/Pedixplorer

git_branch RELEASE_3_19

git_last_commit 20c3c27

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-03

Author Louis Le Nézet [aut, cre] (<https://orcid.org/0009-0000-0202-2703>),
       Jason Sinnwell [aut],
       Terry Therneau [aut],
       Daniel Schaid [ctb],
       Elizabeth Atkinson [ctb],
       Louis Le Nezet [ctb]

Maintainer Louis Le Nézet <louislenezet@gmail.com>

Contents

Pedixplorer-package .......................................................................................... 4
align ................................................................................................................... 5
alignped1 ........................................................................................................ 7
alignped2 ........................................................................................................ 9
alignped3 ....................................................................................................... 10
alignped4 ....................................................................................................... 12
ancestors ....................................................................................................... 13
anchor_to_factor ......................................................................................... 14
auto_hint ...................................................................................................... 15
best_hint ...................................................................................................... 16
bit_size .......................................................................................................... 17
check_columns ............................................................................................ 18
check_num_na ............................................................................................. 20
check_slot_fd .............................................................................................. 20
check_values ............................................................................................... 21
circfun ......................................................................................................... 21
descendants .................................................................................................. 22
draw_arc ....................................................................................................... 22
draw_polygon ............................................................................................ 23
draw_segment ............................................................................................. 24
draw_text .................................................................................................... 25
duporder ...................................................................................................... 26
The Pedixplorer package for pedigree data

Description

The Pedixplorer package for pedigree data an updated package of the kinship2 package. The
kinship2 package was originally written by Terry Therneau and Jason Sinnwell. The Pedixplorer
package is a fork of the kinship2 package with additional functionality and bug fixes.

Details

The package download, NEWS, and README are available on CRAN: \url{https://cran.r-project.org/package=kinship2}
for the previous version of the package.

Functions

Below are listed some of the most widely used functions available in arsenal:

- **Pedigree()**: Constructor of the Pedigree class, given identifiers, sex, affection status(es), and
  special relationships
- **kinship()**: Calculates the kinship matrix, the probability having an allele sampled from two individuals be the same via IBD.
- **plot()**: Method to transform a Pedigree object into a graphical plot. Allows extra information to be included in the id under the plot symbol. This method use the `plot_fromdf()` function to transform the Pedigree object into a data frame of graphical elements, the same is done for the legend with the `ped_to_legdf()` function. When done, the data frames are plotted with the `plot_fromdf()` function.
- **shrink()**: Shrink a Pedigree to a specific bit size, removing non-informative members first.
- **bit_size()**: Approximate the output from SAS’s PROC FREQ procedure when using the /list option of the TABLE statement.

Data

- **sampleped()**: Pedigree example data sets with two pedigrees
- **minnbreast()**: Larger cohort of pedigrees from MN breast cancer study
align

Author(s)

Maintainer: Louis Le Nézet <louislenezet@gmail.com> (ORCID)

Authors:

• Jason Sinnwell <sinnwell.jason@mayo.edu>
• Terry Therneau

Other contributors:

• Daniel Schaid [contributor]
• Elizabeth Atkinson [contributor]
• Louis Le Nezet [contributor]

See Also

Useful links:

• https://github.com/LouisLeNezet/Pedixplorer
• Report bugs at https://github.com/LouisLeZet/Pedixplorer/issues

Examples

library(Pedixplorer)

align

Align a Pedigree object

Description

Given a Pedigree, this function creates helper matrices that describe the layout of a plot of the Pedigree.

Usage

## S4 method for signature 'Pedigree'
align(
  obj,
  packed = TRUE,
  width = 10,
  align = TRUE,
  hints = NULL,
  missid = "NA_character_"
)
Arguments

- **obj**
  A Pedigree object

- **packed**
  Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)

- **width**
  For a packed output, the minimum width of the plot, in inches.

- **align**
  For a packed Pedigree, align children under parents **TRUE**, to the extent possible given the page width, or align to the left margin **FALSE**. This argument can be a two element vector, giving the alignment parameters, or a logical value. If **TRUE**, the default is c(1.5, 2), or if numeric the routine `alignped4()` will be called.

- **hints**
  A Hints object or a named list containing **horder** and **spouse**. If **NULL** then the Hints stored in **obj** will be used.

- **missid**
  A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.

Details

This is an internal routine, used almost exclusively by `ped_to_plotdf()`.

The subservient functions `auto_hint()`, `alignped1()`, `alignped2()`, `alignped3()`, and `alignped4()` contain the bulk of the computation.

If the **hints** are missing the `auto_hint()` routine is called to supply an initial guess.

If multiple families are present in the **obj** Pedigree, this routine is called once for each family, and the results are combined in the list returned.

For more information you can read the associated vignette: `vignette("pedigree_alignment")`.

Value

A list with components

- **n**: A vector giving the number of subjects on each horizontal level of the plot
- **nid**: A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- **pos**: A matrix giving the horizontal position of each plot point
- **fam**: A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject’s parents.
- **spouse**: A matrix with values
  - 0 = not a spouse
  - 1 = subject plotted to the immediate right is a spouse
  - 2 = subject plotted to the immediate right is an inbred spouse
- **twins**: Optional matrix which will only be present if the Pedigree contains twins:
  - 0 = not a twin
  - 1 = sibling to the right is a monozygotic twin
  - 2 = sibling to the right is a dizygotic twin
  - 3 = sibling to the right is a twin of unknown zygosity
**alignped1**

**See Also**

alignped1(), alignped2(), alignped3(), alignped4(), auto_hint()

**Examples**

data(sampleped)
ped <- Pedigree(sampleped)
align(ped)

---

alignped1  
*Alignment first routine*

**Description**

First alignment routine which create the subtree founded on a single subject as though it were the only tree.

**Usage**

alignped1(idx, dadx, momx, level, horder, packed, spouselist)

**Arguments**

- **idx**  
  Indexes of the subjects

- **dadx**  
  Indexes of the fathers

- **momx**  
  Indexes of the mothers

- **level**  
  Vector of the level of each subject

- **horder**  
  A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.

- **packed**  
  Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)

- **spouselist**  
  Matrix of spouses with 4 columns:
  - 1: husband index
  - 2: wife index
  - 3: husband anchor
  - 4: wife anchor
Details

In this routine the \text{nid} array consists of the final \text{nid} array + 1/2 of the final spouse array. Note that the \text{spouselist} matrix will only contain spouse pairs that are not yet processed. The logic for anchoring is slightly tricky.

1. Anchoring::
First, if col 4 of the spouselist matrix is 0, we anchor at the first opportunity. Also note that if \text{spouselist[, 3]} == \text{spouselist[, 4]} it is the husband who is the anchor (just write out the possibilities).

2. Return values initialization::
Create the set of 3 return structures, which will be matrices with 1 + \text{nspouse} columns. If there are children then other routines will widen the result.

3. Create \text{lspouse} and \text{rspouse}::
This two complimentary lists denote the spouses plotted on the left and on the right. For someone with lots of spouses we try to split them evenly. If the number of spouses is odd, then men should have more on the right than on the left, women more on the right. Any hints in the spouselist matrix override. We put the undecided marriages closest to \text{idx}, then add predetermined ones to the left and right. The majority of marriages will be undetermined singletons, for which \text{nleft} will be 1 for female (put my husband to the left) and 0 for male. In one bug found by plotting canine data, lspouse could initially be empty but \text{length(rspouse)} > 1. This caused \text{nleft} > \text{length(idx)}. A fix was to not let \text{indx} to be indexed beyond its length, fix by JPS 5/2013.

4. List the children::
For each spouse get the list of children. If there are any we call \text{alignedped2()} to generate their tree and then mark the connection to their parent. If multiple marriages have children we need to join the trees.

5. Splice the tree::
To finish up we need to splice together the tree made up from all the kids, which only has data from \text{lev} + 1 down, with the data here. There are 3 cases:
  1. No children were found.
  2. The tree below is wider than the tree here, in which case we add the data from this level onto theirs.
  3. The tree below is narrower, for instance an only child.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the spouselist matrix. The latter has marriages removed as they are processed.

- \text{n}: A vector giving the number of subjects on each horizontal level of the plot
- \text{nid}: A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- \text{pos}: A matrix giving the horizontal position of each plot point
- \text{fam}: A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject’s parents.
- \text{spouselist}: Spouse matrix with anchors informations
alignped2

See Also

align()

Examples

data(sampleped)
ped <- Pedigree(sampleped)
align(ped)

alignped2  Alignment second routine

Description

Second of the four co-routines which takes a collection of siblings, grows the tree for each, and appends them side by side into a single tree.

Usage

alignped2(idx, dadx, momx, level, horder, packed, spouselist)

Arguments

idx  Indexes of the subjects

dadx  Indexes of the fathers

momx  Indexes of the mothers

level  Vector of the level of each subject

horder  A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.

packed  Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)

spouselist  Matrix of spouses with 4 columns:

  • 1: husband index
  • 2: wife index
  • 3: husband anchor
  • 4: wife anchor
Details

The input arguments are the same as those to `alignped1()` with the exception that `idx` will be a vector. This routine does nothing to the spouselist matrix, but needs to pass it down the tree and back since one of the routines called by `alignped2()` might change the matrix.

The code below has one non-obvious special case. Suppose that two sibs marry. When the first sib is processed by `alignped1` then both partners (and any children) will be added to the rval structure below. When the second sib is processed they will come back as a 1 element tree (the marriage will no longer be on the spouselist), which should be added onto rval. The rule thus is to not add any 1 element tree whose value (which must be `idx[i]`) is already in the rval structure for this level.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the spouselist matrix. The latter has marriages removed as they are processed.

- `n`: A vector giving the number of subjects on each horizontal level of the plot
- `nid`: A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- `pos`: A matrix giving the horizontal position of each plot point
- `fam`: A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject’s parents.
- `spouselist`: Spouse matrix with anchors informations

See Also

`align()`

Examples

data(sampleped)
ped <- Pedigree(sampleped)
align(ped)

alignped3

Alignment third routine

Description

Third of the four co-routines to merges two pedigree trees which are side by side into a single object.

Usage

`alignped3(alt1, alt2, packed, space = 1)`
Arguments

alt1  Alignment of the first tree
alt2  Alignment of the second tree
packed Should the Pedigree be compressed. (i.e. allow diagonal lines connecting par-
ents to children in order to have a smaller overall width for the plot.)
space  Space between two subjects

Details

The primary special case is when the rightmost person in the left tree is the same as the leftmost
person in the right tree; we need not plot two copies of the same person side by side. (When
initializing the output structures do not worry about this, there is no harm if they are a column
bigger than finally needed.) Beyond that the work is simple book keeping.

1. Slide::
   For the unpacked case, which is the traditional way to draw a Pedigree when we can assume the
   paper is infinitely wide, all parents are centered over their children. In this case we think if the
   two trees to be merged as solid blocks. On input they both have a left margin of 0. Compute how
   far over we have to slide the right tree.

2. Merge::
   Now merge the two trees. Start at the top level and work down.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the
spouselist matrix. The latter has marriages removed as they are processed.

• n : A vector giving the number of subjects on each horizontal level of the plot
• nid : A matrix with one row for each level, giving the numeric id of each subject plotted. (A
  value of 17 means the 17th subject in the Pedigree).
• pos : A matrix giving the horizontal position of each plot point
• fam : A matrix giving the family id of each plot point. A value of 3 would mean that the two
  subjects in positions 3 and 4, in the row above, are this subject’s parents.
• spouselist : Spouse matrix with anchors informations

See Also

align()

Examples

data(sampleped)
ped <- Pedigree(sampleped)
align(ped)
alignped4  

Alignment fourth routine

Description

Last routines which attempts to line up children under parents and put spouses and siblings "close" to each other, to the extent possible within the constraints of page width.

Usage

alignped4(rval, spouse, level, width, align)

Arguments

- **rval**: A list with components `n`, `nid`, `pos`, and `fam`.
- **spouse**: A boolean matrix with one row per level representing if the subject is a spouse or not.
- **level**: Vector of the level of each subject.
- **width**: For a packed output, the minimum width of the plot, in inches.
- **align**: For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is `c(1.5, 2)`, or if numeric the routine `alignped4()` will be called.

Details

The `alignped4()` routine is the final step of alignment. The current code does necessary setup and then calls the `quadprog::solve.QP()` function.

There are two important parameters for the function:

1. The maximum width specified. The smallest possible width is the maximum number of subjects on a line, if the user suggestion is too low it is increased to that 1 + that amount (to give just a little wiggle room).

2. The align vector of 2 alignment parameters `a` and `b`. For each set of siblings `x` with parents at `p_1` and `p_2` the alignment penalty is:

\[
\frac{1}{k^a} \sum_{i=1}^{k} (x_i - (p_1 + p_2))^2
\]

where `k` is the number of siblings in the set.

Using the fact that when `a = 1`:

\[
\sum (x_i - c)^2 = \sum (x_i - \mu)^2 + k(c - \mu)^2
\]
then moving a sibship with k sibs one unit to the left or right of optimal will incur the same cost as moving one with only 1 or two sibs out of place.

If \( a = 0 \) then large sibships are harder to move than small ones, with the default value \( a = 1.5 \) they are slightly easier to move than small ones. The rationale for the default is as long as the parents are somewhere between the first and last siblings the result looks fairly good, so we are more flexible with the spacing of a large family. By tethering all the sibs to a single spot they tend are kept close to each other.

The alignment penalty for spouses is \( b(x_1 - x_2)^2 \), which tends to keep them together. The size of \( b \) controls the relative importance of sib-parent and spouse-spouse closeness.

1. We start by adding in these penalties. The total number of parameters in the alignment problem (what we hand to quadprog) is the set of \( \sum(n) \) positions. A work array myid keeps track of the parameter number for each position so that it is easy to find. There is one extra penalty added at the end. Because the penalty amount would be the same if all the final positions were shifted by a constant, the penalty matrix will not be positive definite; \texttt{solve.QP()} does not like this. We add a tiny amount of leftward pull to the widest line.

2. If there are k subjects on a line there will be \( k+1 \) constraints for that line. The first point must be \( \geq 0 \), each subsequent one must be at least 1 unit to the right, and the final point must be \( \leq \) the max width.

\section*{Value}

The updated position matrix

\section*{See Also}

\texttt{align()}

\section*{Examples}

\begin{verbatim}
data(sampleped)
ped <- Pedigree(sampleped)
align(ped)
\end{verbatim}

\section*{ancestors}

\textit{Ancestors indexes of a subject}

\section*{Description}

Given the index of one or multiple individual(s), this function iterate through the mom and dad indexes to list out all the ancestors of the said individual(s). This function is use in the \texttt{align()} function to identify which spouse pairs has a common ancestor and therefore if they need to be connected with a double line (i.e. inbred).

\section*{Usage}

\begin{verbatim}
ancestors(idx, momx, dadx)
\end{verbatim}
anchor_to_factor

Arguments

- idx: Indexes of the subjects
- momx: Indexes of the mothers
- dadx: Indexes of the fathers

Value

A vector of ancestor indexes

See Also

align()

Examples

ancestors(c(1), c(3, 4, 5, 6), c(7, 8, 9, 10))
ancestors(c(1, 2), c(3, 4, 5, 6), c(7, 8, 9, 10))

anchor_to_factor

Anchor variable to ordered factor

Description

Anchor variable to ordered factor

Usage

anchor_to_factor(anchor)

Arguments

- anchor: A character, factor or numeric vector corresponding to the anchor of the individuals. The following values are recognized:
  - character() or factor(): "0", "1", "2", "left", "right", "either"
  - numeric(): 1 = "left", 2 = "right", 0 = "either"

Value

An ordered factor vector containing the transformed variable "either" < "left" < "right"

Examples

anchor_to_factor(c(1, 2, 0, "left", "right", "either"))
auto_hint

Initial hint for a Pedigree alignment

Description

Compute an initial guess for the alignment of a Pedigree

Usage

```r
## S4 method for signature 'Pedigree'
auto_hint(obj, hints = NULL, packed = TRUE, align = FALSE, reset = FALSE)
```

Arguments

- `obj`: A Pedigree object
- `hints`: A Hints object or a named list containing `horder` and `spouse`. If `NULL` then the Hints stored in `obj` will be used.
- `packed`: Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
- `align`: For a packed Pedigree, align children under parents `TRUE`, to the extent possible given the page width, or align to to the left margin `FALSE`. This argument can be a two element vector, giving the alignment parameters, or a logical value. If `TRUE`, the default is `c(1.5, 2)`, or if numeric the routine `alignped4()` will be called.
- `reset`: If `TRUE`, then even if the Ped object has Hints, reset them to the initial values.

Details

A Pedigree structure can contain a `Hints` object which helps to reorder the Pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine is used to create an initial version of the hints. They can then be modified if desired.

This routine would not normally be called by a user. It moves children within families, so that marriages are on the "edge" of a set children, closest to the spouse. For pedigrees that have only a single connection between two families this simple-minded approach works surprisingly well. For more complex structures hand-tuning of the hints may be required.

When `auto_hint()` is called with a a vector of numbers as the `hints` argument, the values for the founder females are used to order the founder families left to right across the plot. The values within a sibship are used as the preliminary order of siblings within a family; this may be changed to move one of them to the edge so as to match up with a spouse. The actual values in the vector are not important, only their order.

Value

The initial `Hints` object.
See Also

align(), best_hint(), Hints

Examples

data(sampleped)
ped <- Pedigree(sampleped[sampleped$famid == 1, ])
auto_hint(ped)

best_hint
Best hint for a Pedigree alignment

Description

When computer time is cheap, use this routine to get a best Pedigree alignment. This routine will try all possible founder orders, and return the one with the least stress.

Usage

## S4 method for signature 'Pedigree'
best_hint(obj, wt = c(1000, 10, 1), tolerance = 0)

Arguments

obj A Pedigree object

wt A vector of three weights for the three error measures. Default is c(1000, 10, 1).

1. The number of duplicate individuals in the plot
2. The sum of the absolute values of the differences in the positions of duplicate individuals
3. The sum of the absolute values of the differences between the center of the children and the parents.

tolerance The maximum stress level to accept. Default is 0

Details

The auto_hint() routine will rearrange sibling order, but not founder order. This calls auto_hint() with every possible founder order, and finds that plot with the least "stress". The stress is computed as a weighted sum of three error measures:

- nbArcs The number of duplicate individuals in the plot
- lgArcs The sum of the absolute values of the differences in the positions of duplicate individuals
- lgParentsChilds The sum of the absolute values of the differences between the center of the children and the parents
bit_size

\[
\text{stress} = \text{wt}[1] \times \text{nbArcs} + \text{wt}[2] \times \text{lgArcs} + \text{wt}[3] \times \text{lgParentsChilds}
\]

If during the search, a plot is found with a stress level less than \textbf{tolerance}, the search is terminated.

**Value**

The best Hints object out of all the permutations

**See Also**

\texttt{auto_hint()}, \texttt{align()}

**Examples**

```r
data(sampleped)
ped <- Pedigree(sampleped[sampleped$famid == 1,])
best_hint(ped)
```

---

**bit_size  
Bit size of a Pedigree**

**Description**

Utility function used in the \texttt{shrink()} function to calculate the bit size of a Pedigree.

**Usage**

```r
## S4 method for signature 'character.OR.integer'
bit_size(obj, momid, missid = NA_character_)

## S4 method for signature 'Pedigree'
bit_size(obj)

## S4 method for signature 'Ped'
bit_size(obj)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A Ped or Pedigree object or a vector of fathers identifiers</td>
</tr>
<tr>
<td>momid</td>
<td>A vector containing for each subject, the identifiers of the biologicals mothers.</td>
</tr>
<tr>
<td>missid</td>
<td>A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to \texttt{NA_character_}.</td>
</tr>
</tbody>
</table>
check_columns

Details

The bit size of a Pedigree is defined as:

\[
2 \times \text{NbNonFounders} - \text{NbFounders}
\]

Where \text{NbNonFounders} is the number of non founders in the Pedigree (i.e. individuals with identified parents) and \text{NbFounders} is the number of founders in the Pedigree (i.e. individuals without identified parents).

Value

A list with the following components:

- \text{bit_size} The bit size of the Pedigree
- \text{nFounder} The number of founders in the Pedigree
- \text{nNonFounder} The number of non founders in the Pedigree

See Also

\text{shrink()}

Examples

data(sampleped)
ped <- Pedigree(sampleped)
bit_size(ped)

check_columns

Check columns presence in a dataframe

Description

Check for presence / absence of columns names depending on their need

Usage

ccheck_columns(
  df,
  cols_needed = NULL,
  cols_used = NULL,
  cols_to_use = NULL,
  others_cols = FALSE,
  cols_used_init = FALSE,
  cols_to_use_init = FALSE,
  cols_used_del = FALSE,
  verbose = FALSE
)

check_columns

Check columns presence in a dataframe

Description

Check for presence / absence of columns names depending on their need

Usage

ccheck_columns(
  df,
  cols_needed = NULL,
  cols_used = NULL,
  cols_to_use = NULL,
  others_cols = FALSE,
  cols_used_init = FALSE,
  cols_to_use_init = FALSE,
  cols_used_del = FALSE,
  verbose = FALSE
)
check_columns

Arguments

- **df**: The dataframe to use
- **cols_needed**: A vector of columns needed
- **cols_used**: A vector of columns that are used by the script and that will be overwritten.
- **cols_to_use**: A vector of optional columns that are authorized.
- **others_cols**: Boolean defining if non defined columns should be allowed.
- **cols_used_init**: Boolean defining if the columns that will be used should be initialised to NA.
- **cols_to_use_init**: Boolean defining if the optional columns should be initialised to NA.
- **cols_used_del**: Boolean defining if the columns that will be used should be deleted.
- **verbose**: Should message be prompted to the user

Details

3 types of columns are here checked:

- **cols_needed**: those columns need to be present if any is missing an error will be prompted and the script will stop
- **cols_used**: those columns will be used in the script and will be overwritten to NA.
- **cols_to_use**: those columns are optional and will be recognise if present. The last two types of columns can be initialised to NA if needed.

Value

Dataframe with only the column allowed and all the column correctly initialised.

Examples

```r
data.frame
df <- data.frame(ColN1 = c(1, 2), ColN2 = 4,
                  ColU1 = 'B', ColU2 = '1',
                  ColTU1 = 'A', ColTU2 = 3,
                  ColNR1 = 4, ColNR2 = 5)
tryCatch(
  check_columns(df,
                c('ColN1', 'ColN2'), c('ColU1', 'ColU2'),
                c('ColTU1', 'ColTU2'))
  , error = function(e) print(e))
```
check_num_na

Description
Check if a variable given is numeric or NA.

Usage
check_num_na(var, na_as_num = TRUE)

Arguments
- var: Vector of value to test.
- na_as_num: Boolean defining if the NA string should be considered as numerical values.

Details
Check if the values in var are numeric or if they are NA in the case that na_as_num is set to TRUE.

Value
A vector of boolean of the same size as var.

check_slot_fd

Description
Check if the fields are present in an object slot.

Usage
check_slot_fd(obj, slot = NULL, fields = character())

Arguments
- obj: An object.
- slot: A slot of object.
- fields: A character vector with the fields to check.

Value
A character vector with the errors if any.
check_values  

Check values in a slot

Description

Check if the all the values in a slot are in a vector of values.

Usage

check_values(val, ref, name = NULL, present = TRUE)

Arguments

- **val**: A vector of values to check.
- **ref**: A vector of reference values.
- **name**: A character vector with the name of the values to check.
- **present**: A logical value indicating if the values should be present or not

Value

A character vector with the errors if any.

circfun  

Circular element

Description

Create a list of x and y coordinates for a circle with a given number of slices.

Usage

circfun(nslice, n = 50)

Arguments

- **nslice**: Number of slices in the circle
- **n**: Total number of points in the circle

Value

A list of x and y coordinates per slice.

Examples

circfun(1)
circfun(1, 10)
circfun(4, 50)
descendants

*Descendants of individuals*

### Description

Find all the descendants of a particular list of individuals given a Pedigree object.

### Usage

```r
## S4 method for signature 'character_OR_integer,character_OR_integer'
descendants(idlist, obj, dadid, momid)
## S4 method for signature 'character_OR_integer,Pedigree'
descendants(idlist, obj)
## S4 method for signature 'character_OR_integer,Ped'
descendants(idlist, obj)
```

### Arguments

- **idlist** List of individuals identifiers to be considered
- **obj** A Ped or Pedigree object or a vector of the individuals identifiers.
- **dadid** A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid** A vector containing for each subject, the identifiers of the biologicals mothers.

### Value

Vector of all descendants of the individuals in idlist. The list is not ordered.

### Examples

```r
data("sampleped")
ped <- Pedigree(sampleped)
descendants(c("1_101", "2_208"), ped)
```

draw_arc

*Draw arcs*

### Description

Draw arcs

### Usage

```r
draw_arc(x0, y0, x1, y1, p, ggplot_gen = FALSE, lwd = 1, col = "black")
```
draw_polygon

**Arguments**

- `x0` x coordinate of the first point
- `y0` y coordinate of the first point
- `x1` x coordinate of the second point
- `y1` y coordinate of the second point
- `p` ggplot object
- `ggplot_gen` If TRUE add the segments to the ggplot object
- `lwd` Line width
- `col` Line color

**Value**

Plot the arcs to the current device or add it to a ggplot object

---

draw_polygon | Draw a polygon

**Description**

Draw a polygon

**Usage**

```r
draw_polygon(
  x, 
  y, 
  p, 
  ggplot_gen = FALSE, 
  fill = "grey", 
  border = NULL, 
  density = NULL, 
  angle = 45 
)
```

**Arguments**

- `x` x coordinates
- `y` y coordinates
- `p` ggplot object
- `ggplot_gen` If TRUE add the segments to the ggplot object
- `fill` Fill color
- `border` Border color
- `density` Density of shading
- `angle` Angle of shading
draw_segment

Description

Draw segments

Usage

draw_segment(
  x0,
  y0,
  x1,
  y1,
  p,
  ggplot_gen,
  col = par("fg"),
  lwd = par("lwd"),
  lty = par("lty")
)

Arguments

  x0    x coordinate of the first point
  y0    y coordinate of the first point
  x1    x coordinate of the second point
  y1    y coordinate of the second point
  p     ggplot object
  ggplot_gen  If TRUE add the segments to the ggplot object
  col  Line color
  lwd  Line width
  lty  Line type

Value

Plot the segments to the current device or add it to a ggplot object
draw_text  

**Draw texts**

### Description

Draw texts

### Usage

```r
draw_text(
  x,
  y,
  label,
  p,
  ggplot_gen = FALSE,
  cex = 1,
  col = NULL,
  adjx = 0,
  adjy = 0
)
```

### Arguments

- **x**: x coordinates
- **y**: y coordinates
- **label**: Text to be displayed
- **p**: ggplot object
- **ggplot_gen**: If TRUE add the segments to the ggplot object
- **cex**: Character expansion of the text
- **col**: Text color
- **adjx**: x adjustment
- **adjy**: y adjustment

### Value

Plot the text to the current device or add it to a ggplot object
**duporder**

*Find the duplicate pairs of a subject*

**Description**

Find the duplicate pairs of a subject

**Usage**

duporder(idlist, plist, lev, obj)

**Arguments**

- **idlist** List of individuals identifiers to be considered
- **plist** The alignment structure representing the Pedigree layout. See align() for details.
- **lev** The generation level of the subject
- **obj** A Pedigree object

**Details**

This routine is used by auto_hint(). It finds the duplicate pairs of a subject and returns them in the order they should be plotted.

**Value**

A matrix of duplicate pairs

**See Also**

- auto_hint()

**exclude_stray_marryin**

*Exclude stray marry-ins*

**Description**

Exclude any founders who are not parents.

**Usage**

exclude_stray_marryin(id, dadid, momid)
exclude_unavail_founders

Arguments

id A character vector with the identifiers of each individuals
dadid A vector containing for each subject, the identifiers of the biologicals fathers.
momid A vector containing for each subject, the identifiers of the biologicals mothers.

Value

Returns a data frame of subject identifiers and their parents. The data frame is trimmed of any founders who are not parents.

See Also

shrink()

Description

Exclude any unavailable founders.

Usage

exclude_unavail_founders(id, dadid, momid, avail, missid = NA_character_)

Arguments

id A character vector with the identifiers of each individuals
dadid A vector containing for each subject, the identifiers of the biologicals fathers.
momid A vector containing for each subject, the identifiers of the biologicals mothers.
avail A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
missid A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Value

Returns a list with the following components:

- n_trimmed Number of trimmed individuals
- id_trimmed Vector of IDs of trimmed individuals
- id Vector of subject identifiers
- dadid Vector of father identifiers
- momid Vector of mother identifiers
family_check

Description

Error check for a family classification

Usage

## S4 method for signature 'character_OR_integer'
family_check(obj, dadid, momid, famid, newfam)

## S4 method for signature 'Pedigree'
family_check(obj)

## S4 method for signature 'Ped'
family_check(obj)

Arguments

obj A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid A vector containing for each subject, the identifiers of the biologicals fathers.
momid A vector containing for each subject, the identifiers of the biologicals mothers.
famid A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
newfam The result of a call to make_famid(). If this has already been computed by the user, adding it as an argument shortens the running time somewhat.

Details

Given a family id vector, also compute the familial grouping from first principles using the parenting data, and compare the results.

The make_famid() function is used to create a de novo family id from the parentage data, and this is compared to the family id given in the data.

If there are any joins, then an attribute 'join' is attached. It will be a matrix with family as row labels, new-family-id as the columns, and the number of subjects as entries.

See Also

shrink()
Value

a data frame with one row for each unique family id in the famid argument or the one detected in the Pedigree object. Components of the output are:

- **famid**: The family id, as entered into the data set
- **n**: Number of subjects in the family
- **unrelated**: Number of them that appear to be unrelated to anyone else in the entire Pedigree. This is usually marry-ins with no children (in the Pedigree), and if so are not a problem.
- **split**: Number of unique 'new' family ids.
  - 0 = no one in this 'family' is related to anyone else (not good)
  - 1 = everythings is fine
  - 2 and + = the family appears to be a set of disjoint trees. Are you missing some of the people?
- **join**: Number of other families that had a unique family, but are actually joined to this one. 0 is the hope.

See Also

```r
make_famid()
```

Examples

```r
# use 2 samplepeds
data(sampleped)
pedAll <- Pedigree(sampleped)

## check them giving separate ped ids
fcheck.sep <- family_check(pedAll)
fcheck.sep

## check assigning them same ped id
fcheck.combined <- with(sampleped, family_check(id, dadid, momid,
rep(1, nrow(sampleped))))
fcheck.combined
```

### Description

Find the siblings of a subject

### Usage

```r
findsibs(idpos, plist, lev)
```
findspouse

Arguments

idpos  The position of the subject
plist  The alignment structure representing the Pedigree layout. See align() for details.
lev    The generation level of the subject

Details

This routine is used by auto_hint(). It finds the siblings of a subject.

Value

The positions of the siblings

See Also

auto_hint()

findspouse  Find the spouse of a subject

Description

Find the spouse of a subject

Usage

findspouse(idpos, plist, lev, obj)

Arguments

idpos  The position of the subject
plist  The alignment structure representing the Pedigree layout. See align() for details.
lev    The generation level of the subject
obj    A Pedigree object

Details

This routine is used by auto_hint(). It finds the spouse of a subject.

Value

The position of the spouse

See Also

auto_hint()
find_avail_affected

Find single affected and available individual from a Pedigree

Description
Finds one subject from among available non-parents with indicated affection status.

Usage
## S4 method for signature 'Ped'
find_avail_affected(obj, avail = NULL, affected = NULL, affstatus = NA)

## S4 method for signature 'Pedigree'
find_avail_affected(obj, avail = NULL, affected = NULL, affstatus = NA)

Arguments
- obj A Ped or Pedigree object.
- avail A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
- affected A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
- affstatus Affection status to search for.

Details
When used within shrink(), this function is called with the first affected indicator, if the affected item in the Pedigree is a matrix of multiple affected indicators.

If avail or affected is null, then the function will use the corresponding Ped accessor.

Value
A list is returned with the following components
- ped The new Ped object
- newAvail Vector of availability status of trimmed individuals
- idTrimmed Vector of IDs of trimmed individuals
- isTrimmed logical value indicating whether Ped object has been trimmed
- bit_size Bit size of the trimmed Ped

See Also
shrink()
findavailnoninform

**Examples**

```r
data(sampleped)
ped <- Pedigree(sampleped)
find_avail_affected(ped, affstatus = 1)
```

---

**findavailnoninform**  
*Find uninformative but available subject*

**Description**

Finds subjects from among available non-parents with all affection equal to 0.

**Usage**

```r
## S4 method for signature 'Ped'
findavailnoninform(obj, avail = NULL, affected = NULL)
## S4 method for signature 'Pedigree'
findavailnoninform(obj, avail = NULL, affected = NULL)
```

**Arguments**

- `obj`: A Ped or Pedigree object.
- `avail`: A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
- `affected`: A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).

**Details**

Identify subjects to remove from a Pedigree who are available but non-informative (unaffected). This is the second step to remove subjects in `shrink()` if the Pedigree does not meet the desired bit size.

If `avail` or `affected` is null, then the function will use the corresponding Ped accessor.

**Value**

Vector of subject ids who can be removed by having lowest informativeness.

**See Also**

`shrink()`

**Examples**

```r
data(sampleped)
ped <- Pedigree(sampleped)
findavailnoninform(ped)
```
find_unavailable

find_unavailable

Description

Find the identifiers of subjects in a Pedigree iteratively, as anyone who is not available and does not have an available descendant by successively removing unavailable terminal nodes.

Usage

## S4 method for signature 'Ped'
find_unavailable(obj, avail = NULL)

## S4 method for signature 'Pedigree'
find_unavailable(obj, avail = NULL)

Arguments

obj A Ped or Pedigree object.

avail A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).

Details

If avail is null, then the function will use the corresponding Ped accessor.

Originally written as pedTrim by Steve Iturria, modified by Dan Schaid 2007, and now split into the two separate functions: find_unavailable(), and trim() to do the tasks separately. find_unavailable() calls exclude_stray_marryin() to find stray available marry-ins who are isolated after trimming their unavailable offspring, and exclude_unavail_founders(). If the subject ids are character, make sure none of the characters in the ids is a colon (""), which is a special character used to concatenate and split subjects within the utility. The trim() functions is now replaced by the subset() function.

Value

Returns a vector of subject ids for who can be removed.

Side Effects

Relation matrix from subsetting is trimmed of any special relations that include the subjects to trim.

See Also

shrink()
Examples

```r
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
find_unavailable(ped1)
```

---

**fix_parents**

*Fix parents relationship and gender*

**Description**

Fix the sex of parents, add parents that are missing from the data. Can be used with a dataframe or a vector of the different individuals informations.

**Usage**

```r
## S4 method for signature 'character'
fix_parents(obj, dadid, momid, sex, famid = NULL, missid = NA_character_)

## S4 method for signature 'data.frame'
fix_parents(obj, delete = FALSE, filter = NULL, missid = NA_character_)
```

**Arguments**

- **obj** A data.frame or a vector of the individuals identifiers. If a dataframe is given it must contain the columns `id`, `dadid`, `momid`, `sex` and `famid` (optional).
- **dadid** A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid** A vector containing for each subject, the identifiers of the biologicals mothers.
- **sex** A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: `male` < `female` < `unknown` < `terminated` The following values are recognized:
  - character() or factor(): "f", "m", "woman", "man", "male", "female", "unknown", "terminated"
  - numeric(): 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"
- **famid** A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
- **missid** A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.
- **delete** Boolean defining if missing parents needs to be:
  - TRUE: added as a new row
  - FALSE: be deleted
- **filter** Filtering column containing 0 or 1 for the rows to kept before proceeding.
generate_aff_inds

Details

First look to add parents whose ids are given in momid/dadid. Second, fix sex of parents. Last look to add second parent for children for whom only one parent id is given. If a famid vector is given the family id will be added to the ids of all individuals (id, dadid, momid) separated by an underscore before proceeding.

**Special case for dataframe:**
Check for presence of both parents id in the id field. If not both presence behaviour depend of delete parameter

- If TRUE then use fix_parents function and merge back the other fields in the dataframe then set availability to O for non available parents.
- If FALSE then delete the id of missing parents

Value

A data.frame with id, dadid, momid, sex as columns with the relationships fixed.

Author(s)

Jason Sinnwell

Examples

test1char <- data.frame(
    id = paste('fam', 101:111, sep = ''),
    sex = c('male', 'female')[c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2)],
    father = c(0, 0, 'fam101', 'fam101', 'fam101', 0, 0,
               'fam106', 'fam106', 'fam106', 'fam109'),
    mother = c(0, 0, 'fam102', 'fam102', 'fam102', 0, 0,
               'fam107', 'fam107', 'fam107', 'fam112')
)
test1newmom <- with(test1char, fix_parents(id, father, mother, sex, missid = NA_character_))
Pedigree(test1newmom)

---

**generate_aff_inds**  
Process the affection informations

Description

Perform transformation uppon a vector given as the one containing the affection status to obtain an affected binary state.
Usage

generate_aff_inds(
  values,
  mods_aff = NULL,
  threshold = NULL,
  sup_thres_aff = NULL
)

Arguments

values  Vector containing the values of the column to process.
mods_aff  Vector of modality to consider as affected in the case where the values is a factor.
threshold  Numeric value separating the affected and healthy subject in the case where the values is numeric.
sup_thres_aff  Boolean defining if the affected individual are above the threshold or not. If TRUE, the individuals will be considered affected if the value of values is strictly above the threshold. If FALSE, the individuals will be considered affected if the value is strictly under the threshold.

Details

This function helps to configure a binary state from a character or numeric variable.

If the variable is a character or a factor::
In this case the affected state will depend on the modality provided as an affected status. All individuals with a value corresponding to one of the element in the vector mods_aff will be considered as affected.

If the variable is numeric::
In this case the affected state will be TRUE if the value of the individual is above the threshold if sup_thres_aff is TRUE and FALSE otherwise.

Value

A dataframe with the affected column processed accordingly. The different columns are:

- mods: The different modalities of the column
- labels: The labels of the different modalities
- affected: The column processed to have only TRUE/FALSE values

Author(s)

Louis Le Nézet

Examples

generate_aff_inds(c(1, 2, 3, 4, 5), threshold = 3, sup_thres_aff = TRUE)
generate_aff_inds(c("A", "B", "C", "A", "V", "B"), mods_aff = c("A", "B"))
**generate_border**  
*Process the border colors based on availability*

---

**Description**

Perform transformation upon a vector given as the one containing the availability status to compute the border color. The vector given will be transformed using the `vect_to_binary()` function.

**Usage**

```r
generate_border(values, colors_avail = c("green", "black"))
```

**Arguments**

- **values**: The vector containing the values to process as available.
- **colors_avail**: Set of 2 colors to use for the box’s border of an individual. The first color will be used for available individual (`avail == 1`) and the second for the unavailable individual (`avail == 0`).

**Value**

A list of three elements

- `mods`: The processed values column as a numeric factor
- `avail`: A logical vector indicating if the individual is available
- `sc_bord`: A dataframe containing the description of each modality of the scale

**Examples**

```r
generate_border(c(1, 0, 1, 0, NA, 1, 0, 1, 0, NA))
```

---

**generate_colors**  
*Process the filling and border colors based on affection and availability*

---

**Description**

Perform transformation upon a dataframe given to compute the colors for the filling and the border of the individuals based on the affection and availability status.
Usage

## S4 method for signature 'character'
generate_colors(
  obj,
  avail,
  mods_aff = NULL,
  keep_full_scale = FALSE,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black")
)

## S4 method for signature 'numeric'
generate_colors(
  obj,
  avail,
  threshold = 0.5,
  sup_thres_aff = TRUE,
  keep_full_scale = FALSE,
  breaks = 3,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black")
)

## S4 method for signature 'Pedigree'
generate_colors(
  obj,
  col_aff = "affected",
  add_to_scale = TRUE,
  col_avail = "avail",
  mods_aff = NULL,
  threshold = 0.5,
  sup_thres_aff = TRUE,
  keep_full_scale = FALSE,
  breaks = 3,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black"),
  reset = TRUE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A Pedigree object or a vector containing the affection status for each individual. The affection status can be numeric or a character.</td>
</tr>
<tr>
<td>avail</td>
<td>A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).</td>
</tr>
</tbody>
</table>
generate_colors

mods_aff Vector of modality to consider as affected in the case where the values is a factor.

keep_full_scale Boolean defining if the affection values need to be set as a scale. If values is numeric the filling scale will be calculated based on the values and the number of breaks given. If values isn’t numeric then each levels will get it’s own color.

colors_aff Set of increasing colors to use for the filling of the affected individuals.

colors_unaff Set of increasing colors to use for the filling of the unaffected individuals.

colors_avail Set of 2 colors to use for the box’s border of an individual. The first color will be used for available individual (avail == 1) and the second for the unavailable individual (avail == 0).

threshold Numeric value separating the affected and healthy subject in the case where the values is numeric.

sup_thres_aff Boolean defining if the affected individual are above the threshold or not. If TRUE, the individuals will be considered affected if the value of values is strictly above the threshold. If FALSE, the individuals will be considered affected if the value is strictly under the threshold.

breaks Number of breaks to use when using full scale with numeric values. The same number of breaks will be done for values from affected individuals and unaffected individuals.

col_aff A character vector with the name of the column to be used for the affection status.

add_to_scale Boolean defining if the scales need to be added to the existing scales or if they need to replace the existing scales.

col_avail A character vector with the name of the column to be used for the availability status.

reset If TRUE the scale of the specified column will be reset if already present.

Details

The colors will be set using the generate_fill() and the generate_border() functions respectively for the filling and the border.

Value

When used with a vector:
A list of two elements
• The list containing the filling colors processed and their description
• The list containing the border colors processed and their description

When used with a Pedigree object:
The Pedigree object with the affected and avail columns processed accordingly as well as the scales slot updated.
Examples

generate_colors(
  c(1, 0, 1, 0, NA, 1, 0, 1, 0, NA),
  mods_aff = "A",
)

generate_colors(
  c(10, 0, 5, 7, NA, 6, 2, 1, 3, NA),
  c(1, 0, 1, 0, NA, 1, 0, 1, 0, NA),
  threshold = 3, keep_full_scale = TRUE
)
data("sampleped")
ped <- Pedigree(sampleped)
ped <- generate_colors(ped, "affected", add_to_scale=FALSE)
scales(ped)

---

`generate_fill`  
Process the filling colors based on affection

Description

Perform transformation upon a column given as the one containing affection status to compute the filling color.

Usage

```r
generate_fill(
  values,
  affected,
  labels,
  keep_full_scale = FALSE,
  breaks = 3,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4")
)
```

Arguments

- **values**: The vector containing the values to process as affection.
- **affected**: A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
- **labels**: The vector containing the labels to use for the affection.
- **keep_full_scale**: Boolean defining if the affection values need to be set as a scale. If values is numeric the filling scale will be calculated based on the values and the number of breaks given. If values isn’t numeric then each levels will get it’s own color.
get_twin_rel

breaks Number of breaks to use when using full scale with numeric values. The same number of breaks will be done for values from affected individuals and unaffected individuals.

colors_aff Set of increasing colors to use for the filling of the affected individuals.

colors_unaff Set of increasing colors to use for the filling of the unaffected individuals.

Details

The colors will be set using the `grDevices::colorRampPalette()` function with the colors given as parameters.

The colors will be set as follow:

- If `keep_full_scale` is FALSE: Then the affected individuals will get the first color of the `colors_aff` vector and the unaffected individuals will get the first color of the `colors_unaff` vector.
- If `keep_full_scale` is TRUE:
  - If `values` isn’t numeric: Each levels of the affected `values` vector will get it’s own color from the `colors_aff` vector using the `grDevices::colorRampPalette()` and the same will be done for the unaffected individuals using the `colors_unaff`.
  - If `values` is numeric: The mean of the affected individuals will be compared to the mean of the unaffected individuals and the colors will be set up such as the color gradient follow the direction of the affection.

Value

A list of three elements

- `mods`: The processed values column as a numeric factor
- `affected`: A logical vector indicating if the individual is affected
- `sc_fill`: A dataframe containing the description of each modality of the scale

Examples

```r
aff <- generate_aff_inds(seq_len(5), threshold = 3, sup_thres_aff = TRUE)
generate_fill(seq_len(5), aff$affected, aff$labels)
generate_fill(seq_len(5), aff$affected, aff$labels, keep_full_scale = TRUE)
```

---

**Description**

Get twin relationships

**Usage**

`get_twin_rel(obj)`
Hints-class

Arguments

obj A Pedigree object

Details

This routine function determine the twin relationships in a Pedigree. It determine the order of the twins in the Pedigree. It is used by auto_hint().

Value

A list containing components

1. twinset the set of twins
2. twinrel the twins relationships
3. twinord the order of the twins

See Also

auto_hint()

Hints-class

Hints object

Description

The hints are used to specify the order of the individuals in the pedigree and to specify the order of the spouses.

Constructor ::
You either need to provide horder or spouse in the dedicated parameters (together or separately), or inside a list.

Usage

Hints(horder, spouse)

# S4 method for signature 'list,missing.OR_NULL'
Hints(horder, spouse)

# S4 method for signature 'numeric,data.frame'
Hints(horder, spouse)

# S4 method for signature 'numeric,missing.OR_NULL'
Hints(horder, spouse)
Hints-class

Arguments

horder A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.

spouse A data.frame with one row per hinted marriage, usually only a few marriages in a pedigree will need an added hint, for instance reverse the plot order of a husband/wife pair. Each row contains the id of the left spouse (i.e. id1), the id of the right hand spouse (i.e. idr), and the anchor (i.e : anchor : 1 = left, 2 = right, 0 = either). Children will preferentially appear under the parents of the anchored spouse.

Value

A Hints object.

Slots

horder A numeric named vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters).

spouse A data.frame with one row per hinted marriage, usually only a few marriages in a Pedigree will need an added hint, for instance reverse the plot order of a husband/wife pair. Each row contains the identifiers of the left spouse, the right hand spouse, and the anchor (i.e : anchor : 1 = left, 2 = right, 0 = either).

Accessors

• horder(x) : Get the horder vector
• horder(x) <- value : Set the horder vector
• spouse(x) : Get the spouse data.frame
• spouse(x) <- value : Set the spouse data.frame

Generics

• as.list(x): Convert a Hints object to a list
• subset(x, i, keep = TRUE): Subset a Hints object based on the individuals identifiers given.
  – i : A vector of individuals identifiers to keep.
  – keep : A logical value indicating if the individuals should be kept or deleted.

See Also

Pedigree()
ibd_matrix

**Examples**

```
Hints(
  list(
    horder = c("1" = 1, "2" = 2, "3" = 3),
    spouse = data.frame(
      id1 = c("1", "2"),
      idr = c("2", "3"),
      anchor = c(1, 2)
    )
  )
)
Hints(
  horder = c("1" = 1, "2" = 2, "3" = 3),
  spouse = data.frame(
    id1 = c("1", "2"),
    idr = c("2", "3"),
    anchor = c(1, 2)
  )
)
Hints(
  horder = c("1" = 1, "2" = 2, "3" = 3)
)
```

---

**ibd_matrix**  
**IBD matrix**

**Description**

Transform identity by descent (IBD) matrix data from the form produced by external programs such as SOLAR into the compact form used by the coxme and lme4kin routines.

**Usage**

```
ibd_matrix(id1, id2, ibd, idmap, diagonal)
```

**Arguments**

- `id1`: A character vector with the id of the first individuals of each pair
- `id2`: A character vector with the id of the second individuals of each pair
- `ibd`: the IBD value for that pair
- `idmap`: an optional 2 column matrix or data frame whose first element is the internal value (as found in `id1` and `id2`, and whose second element will be used for the dimnames of the result
- `diagonal`: optional value for the diagonal element. If present, any missing diagonal elements in the input data will be set to this value.
Details

The IBD matrix for a set of n subjects will be an n by n symmetric matrix whose i,j element is the contains, for some given genetic location, a 0/1 indicator of whether 0, 1/2 or 2/2 of the alleles for i and j are identical by descent. Fractional values occur if the IBD fraction must be imputed. The diagonal will be 1. Since a large fraction of the values will be zero, programs such as Solar return a data set containing only the non-zero elements. As well, Solar will have renumbered the subjects as seq_len(n) in such a way families are grouped together in the matrix; a separate index file contains the mapping between this new id and the original one. The final matrix should be labeled with the original identifiers.

Value

A sparse matrix of class dsCMatrix. This is the same form used for kinship matrices.

See Also

kinship()

---

**is_disconnected**

Are individuals disconnected

Description

Check which individuals are disconnected.

Usage

is_disconnected(id, dadid, momid)

Arguments

dadid A vector containing for each subject, the identifiers of the biologicals fathers.
momid A vector containing for each subject, the identifiers of the biologicals mothers.

Details

An individuals is considered disconnected if the kinship with all the other individuals is 0.

Value

A vector of boolean of the same size as id with TRUE if the individual is disconnected and FALSE otherwise.
Examples

```r
is_disconnected(
  c("1", "2", "3", "4", "5"),
  c("3", "3", NA, NA, NA),
  c("4", "4", NA, NA, NA)
)
```

---

**is_founder**

*Are individuals founders*

**Description**

Check which individuals are founders.

**Usage**

```r
is_founder(momid, dadid, missid = NA_character_)
```

**Arguments**

- **momid**: A vector containing for each subject, the identifiers of the biologicals mothers.
- **dadid**: A vector containing for each subject, the identifiers of the biologicals fathers.
- **missid**: A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.

**Value**

A vector of boolean of the same size as `dadid` and `momid` with `TRUE` if the individual has no parents (i.e is a founder) and `FALSE` otherwise.

**Examples**

```r
is_founder(c("3", "3", NA, NA), c("4", "4", NA, NA))
```

---

**is_informative**

*Find informative individuals*

**Description**

Select the ids of the informative individuals.
is_informative

Usage

## S4 method for signature 'character OR integer'

is_informative(obj, avail, affected, informative = "AvAf")

## S4 method for signature 'Pedigree'

is_informative(obj, col_aff = NULL, informative = "AvAf", reset = FALSE)

Arguments

obj A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.

avail A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).

affected A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).

informative Informative individuals selection can take 5 values:
- 'AvAf' (available and affected),
- 'AvOrAf' (available or affected),
- 'Av' (available only),
- 'Af' (affected only),
- 'All' (all individuals)
- A numeric/character vector of individuals id
- A boolean

col_aff A character vector with the name of the column to be used for the affection status.

reset If TRUE, the isinf slot is reset

Details

Depending on the informative parameter, the function will extract the ids of the informative individuals. In the case of a numeric vector, the function will return the same vector. In the case of a boolean, the function will return the ids of the individuals if TRUE, NA otherwise. In the case of a string, the function will return the ids of the corresponding informative individuals based on the avail and affected columns.

Value

When obj is a vector:
A vector of individuals informative identifiers.

When obj is a Pedigree:
The Pedigree object with its isinf slot updated.
Examples

```r
is_informative(c("A", "B", "C", "D", "E"), informative = c("A", "B"))
is_informative(c("A", "B", "C", "D", "E"), informative = c(1, 2))
is_informative(c("A", "B", "C", "D", "E"), informative = c("A", "B"))
is_informative(c("A", "B", "C", "D", "E"), avail = c(1, 0, 0, 1, 1),
              affected = c(0, 1, 0, 1, 1), informative = "AvAf")
is_informative(c("A", "B", "C", "D", "E"), avail = c(1, 0, 0, 1, 1),
              affected = c(0, 1, 0, 1, 1), informative = "AvOrAf")
is_informative(c("A", "B", "C", "D", "E"),
              informative = c(TRUE, FALSE, TRUE, FALSE, TRUE))
data("sampleped")
ped <- Pedigree(sampleped)
ped <- is_informative(ped, col_aff = "affection_mods")
isinf(ped(ped))
```

Description

Check which individuals are parents.

Usage

```r
## S4 method for signature 'character_OR_integer'
is_parent(obj, dadid, momid, missid = NA_character_)

## S4 method for signature 'Ped'
is_parent(obj, missid = NA_character_)
```

Arguments

- `obj` A vector of each subjects identifiers or a Ped object
- `dadid` A vector containing for each subject, the identifiers of the biologicals fathers.
- `momid` A vector containing for each subject, the identifiers of the biologicals mothers.
- `missid` A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Value

A vector of boolean of the same size as `obj` with TRUE if the individual is a parent and FALSE otherwise
is_valid_hints

Examples

is_parent(c("1", "2", "3", "4"), c("3", "3", NA, NA), c("4", "4", NA, NA))

data(sampleped)
ped <- Pedigree(sampleped)
is_parent(ped(ped))

---

is_valid_hints  Check if a Hints object is valid

Description

Check if horder and spouse slots are valid:

- horder is named numeric vector
- spouse is a data.frame
  - Has the three idr,idl, anchor columns
  - idr and idl are different and doesn't contains NA
  - idr and idl couple are unique
  - anchor column only have right, left, either values
- all ids in spouse needs to be in the names of the horder vector

Usage

is_valid_hints(object)

Arguments

object  A Hints object.

Value

A character vector with the errors or TRUE if no errors.
is_valid_ped

Check if a Ped object is valid

Description

Multiple checks are done here

Usage

is_valid_ped(object)

Arguments

object A Ped object.

Details

1. Check that the ped ids slots have the right values
2. Check that the sex, steril, status, avail and affected slots have the right values
3. Check that dad are male and mom are female
4. Check that individuals have both parents or none

Value

A character vector with the errors or TRUE if no errors.

is_valid_pedigree

Check if a Pedigree object is valid

Description

Multiple checks are done here

Usage

is_valid_pedigree(object)

Arguments

object A Ped object.
Details

1. Check that all Rel id are in the Ped object
2. Check that twins have same parents
3. Check that MZ twins have same sex
4. Check that all columns used in scales are in the Ped object column
5. Check that all id used in Hints object are in the Ped object
6. Check that all spouse in Hints object are male / female

Value

A character vector with the errors or TRUE if no errors.

---

**is_valid_rel**

*Check if a Rel object is valid*

Description

Multiple checks are done here

Usage

`is_valid_rel(object)`

Arguments

- **object**: A Ped object.

Details

1. Check that the "id1", "id2", "code", "famid" slots exist
2. Check that the "code" slots have the right values (i.e. "MZ twin", "DZ twin", "UZ twin", "Spouse")
3. Check that all "id1" are different to "id2"
4. Check that all "id1" are smaller than "id2"
5. Check that no duplicate relation are present

Value

A character vector with the errors or TRUE if no errors.
is_valid_scales Check if a Scales object is valid

Description

Check if the fill and border slots are valid:

- fill slot is a data.frame with "order", "column_values", "column_mods", "mods", "labels", "affected", "fill", "density", "angle" columns.
  - "affected" is logical.
  - "density", "angle", "order", "mods" are numeric.
  - "column_values", "column_mods", "labels", "fill" are character.
- border slot is a data.frame with "column_values", "column_mods", "mods", "labels", "border" columns.
  - "column_values", "column_mods", "labels", "border" are character.
  - "mods" is numeric.

Usage

is_valid_scales(object)

Arguments

object A Scales object.

Value

A character vector with the errors or TRUE if no errors.

---

kindepth Individual’s depth in a pedigree

Description

Computes the depth of each subject in the Pedigree.

Usage

kindepth(obj, ...)

## S4 method for signature 'character.OR_integer'
kindepth(obj, dadid, momid, align_parents = FALSE)

## S4 method for signature 'Pedigree'
kindepth(obj, align_parents = FALSE)
## S4 method for signature 'Ped'

kindepth(obj, align_parents = FALSE)

### Arguments

- **obj**: A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
- **...**: Additional arguments
- **dadid**: A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid**: A vector containing for each subject, the identifiers of the biologicals mothers.
- **align_parents**: If `align_parents = TRUE`, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that ‘join in the middle’ via a marriage.

### Details

Mark each person as to their depth in a Pedigree; 0 for a founder, otherwise :

$$ depth = 1 + \max(fatherDepth, motherDepth) $$

In the case of an inbred Pedigree a perfect alignment may not exist.

### Value

An integer vector containing the depth for each subject

### Author(s)

Terry Therneau, updated by Louis Le Nézet

### See Also

- `align()`

### Examples

```r
kindepth(
  c("A", "B", "C", "D", "E"),
  c("C", "D", "0", "0", "0"),
  c("E", "E", "0", "0", "0")
)
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
kindepth(ped1)
```
Kinship

Description

Compute the kinship matrix for a set of related autosomal subjects. The function is generic, and can accept a Pedigree, a Ped or a vector as the first argument.

Usage

## S4 method for signature 'Ped'
kinship(obj, chrtype = "autosome")

## S4 method for signature 'character'
kinship(obj, dadid, momid, sex, chrtype = "autosome")

## S4 method for signature 'Pedigree'
kinship(obj, chrtype = "autosome")

Arguments

- **obj**: A Pedigree or Ped object or a vector of subject identifiers.
- **chrtype**: chromosome type. The currently supported types are 'autosome' and 'X' or 'x'.
- **dadid**: A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid**: A vector containing for each subject, the identifiers of the biologicals mothers.
- **sex**: A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: male < female < unknown < 'terminated' The following values are recognized:
  - character() or factor(): "f", "m", "woman", "man", "male", "female", "unknown", "terminated"
  - numeric(): 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"

Details

The function will usually be called with a Pedigree. The call with a Ped or a vector is provided for backwards compatibility with an earlier release of the library that was less capable. Note that when using with a Ped or a vector, any information on twins is not available to the function.

When called with a Pedigree, the routine will create a block-diagonal-symmetric sparse matrix object of class dsCMatrix. Since the [i, j] value of the result is 0 for any two unrelated individuals i and j and a Matrix utilizes sparse representation, the resulting object is often orders of magnitude smaller than an ordinary matrix.

Two genes G1 and G2 are identical by descent (IBD) if they are both physical copies of the same ancestral gene; two genes are identical by state if they represent the same allele. So the brown eye gene that I inherited from my mother is IBD with hers; the same gene in an unrelated individual is not.
The kinship coefficient between two subjects is the probability that a randomly selected allele from a locus will be IBD between them. It is obviously 0 between unrelated individuals. For an autosomal site and no inbreeding it will be 0.5 for an individual with themselves, .25 between mother and child, .125 between an uncle and niece, etc.

The computation is based on a recursive algorithm described in Lange, which assumes that the founder alleles are all independent.

**Value**

**When obj is a vector:**
A matrix of kinship coefficients.

**When obj is a Pedigree:**
A matrix of kinship coefficients ordered by families present in the Pedigree object.

**References**

**See Also**
make_famid(), kindepth()

**Examples**

```r
kinship(c("A", "B", "C", "D", "E"), c("C", "D", "0", "0", "0"), 
c("E", "E", "0", "0", "0"), sex = c(1, 2, 1, 2, 1))
kinship(c("A", "B", "C", "D", "E"), c("C", "0", "0", "0", "0"), 
c("E", "E", "0", "0", "0"), sex = c(1, 2, 1, 2, 1), 
chrtype = "x")
```

data(sampleped)
ped <- Pedigree(sampleped)
kinship(ped)

---

**Description**

Construct a family identifier from pedigree information
Usage

```r
## S4 method for signature 'character'
make_famid(obj, dadid, momid)
## S4 method for signature 'Pedigree'
make_famid(obj)
```

Arguments

obj A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid A vector containing for each subject, the identifiers of the biologicals fathers.
momid A vector containing for each subject, the identifiers of the biologicals mothers.

Details

Create a vector of length n, giving the family 'tree' number of each subject. If the Pedigree is totally connected, then everyone will end up in tree 1, otherwise the tree numbers represent the disconnected subfamilies. Singleton subjects give a zero for family number.

Value

When used with a character vector:

An integer vector giving family groupings

When used with a Pedigree object:

An updated Pedigree object with the family id added and with all ids updated

See Also

kinship()

Examples

```r
make_famid(
  c("A", "B", "C", "D", "E", "F"),
  c("C", "D", "0", "0", "0", "0"),
  c("E", "E", "0", "0", "0", "0")
)
data(sampleped)
ped1 <- Pedigree(sampleped[, -1])
make_famid(ped1)
```
**Description**

Data from the Minnesota Breast Cancer Family Study. This contains extended pedigrees from 426 families, each identified by a single proband in 1945-1952, with follow up for incident breast cancer.

**Usage**

data(minnbreast)

**Format**

A data frame with 28081 observations, one line per subject, on the following 14 variables.

- **id**: Subject identifier
- **proband**: If 1, this subject is one of the original 426 probands
- **fatherid**: Identifier of the father, if the father is part of the data set; zero otherwise
- **motherid**: Identifier of the mother, if the mother is part of the data set; zero otherwise
- **famid**: Family identifier
- **endage**: Age at last follow-up or incident cancer
- **cancer**: 1 = breast cancer (females) or prostate cancer (males), 0 = censored
- **yob**: Year of birth
- **education**: Amount of education: 1-8 years, 9-12 years, high school graduate, vocational education beyond high school, some college but did not graduate, college graduate, postgraduate education, refused to answer on the questionnaire
- **marstat**: Marital status: married, living with someone in a marriage-like relationship, separated or divorced, widowed, never married, refused to answer the questionnaire
- **everpreg**: Ever pregnant at the time of baseline survey
- **parity**: Number of births
- **nbreast**: Number of breast biopsies
- **sex**: M or F
- **bcpc**: Part of one of the families in the breast / prostate cancer substudy: 0 = no, 1 = yes. Note that subjects who were recruited to the overall study after the date of the BP substudy are coded as zero.
Details

The original study was conducted by Dr. Elving Anderson at the Dight Institute for Human Genetics at the University of Minnesota. From 1944 to 1952, 544 sequential breast cancer cases seen at the University Hospital were enrolled, and information gathered on parents, siblings, offspring, aunts / uncles, and grandparents with the goal of understanding possible familial aspects of breast cancer. In 1991 the study was resurrected by Dr Tom Sellers.

Of the original 544 he excluded 58 prevalent cases, along with another 19 who had less than 2 living relatives at the time of Dr Anderson’s survey. Of the remaining 462 families 10 had no living members, 23 could not be located and 8 refused, leaving 426 families on whom updated pedigrees were obtained.

This gave a study with 13351 males and 12699 females (5183 marry-ins). Primary questions were the relationship of early life exposures, breast density, and pharmacogenomics on incident breast cancer risk. For a subset of the families data was gathered on prostate cancer risk for male subjects via questionnaires sent to men over 40. Other than this, data items other than parentage are limited to the female subjects. In 2003 a second phase of the study was instituted. The pedigrees were further extended to the numbers found in this data set, and further data gathered by questionnaire.

References


Examples

data(minnbreast)
breastped <- Pedigree(minnbreast, 
cols_ren_ped = list( 
  "indId" = "id", "fatherId" = "fatherid", 
  "motherId" = "motherid", "gender" = "sex", "family" = "famid" 
), missid = "0", col_aff = "cancer"
)
scales(breastped)
#plot family 8, proband is solid, slash for cancers
#plot(breastped[famid(breastped) == "8"])
**Description**

Compute the minimum distance between the informative individuals and all the others. This distance is a transformation of the maximum kinship degree between the informative individuals and all the others. This transformation is done by taking the log2 of the inverse of the maximum kinship degree.

\[
\text{minDist} = \log_2(1 / \max(\text{kinship}))
\]

Therefore, the minimum distance is 0 when the maximum kinship is 1 and is infinite when the maximum kinship is 0. For siblings, the kinship value is 0.5 and the minimum distance is 1. Each time the kinship degree is divided by 2, the minimum distance is increased by 1.

**Usage**

```r
## S4 method for signature 'character'
min_dist_inf(obj, dadid, momid, sex, id_inf)

## S4 method for signature 'Pedigree'
min_dist_inf(obj, col_aff = NULL, informative = "AvAf", reset = FALSE, ...)

## S4 method for signature 'Ped'
min_dist_inf(obj, informative = "AvAf", reset = FALSE)
```

**Arguments**

- `obj` A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
- `...` Additional arguments
- `dadid` A vector containing for each subject, the identifiers of the biologicals fathers.
- `momid` A vector containing for each subject, the identifiers of the biologicals mothers.
- `sex` A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: `male` < `female` < `unknown` < `terminated` The following values are recognized:
  - character() or factor() : "f", "m", "woman", "man", "male", "female", "unknown", "terminated"
  - numeric() : 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"
- `id_inf` An identifiers vector of informative individuals.
- `col_aff` A character vector with the name of the column to be used for the affection status.
- `informative` Informative individuals selection can take 5 values:
  - 'AvAf' (available and affected),
• 'AvOrAf' (available or affected),
• 'Av' (available only),
• 'Af' (affected only),
• 'All' (all individuals)
• A numeric/character vector of individuals id
• A boolean

reset If TRUE, the kin and if isinf columns is reset

Value

When obj is a vector:
A vector of the minimum distance between the informative individuals and all the others corresponding to the order of the individuals in the obj vector.

When obj is a Pedigree:
The Pedigree object with a new slot named 'kin' containing the minimum distance between each individuals and the informative individuals. The isinf slot is also updated with the informative individuals.

See Also

kinship()

Examples

min_dist_inf(
  c("A", "B", "C", "D", "E"),
  c("C", "D", "0", "0", "0"),
  c("E", "E", "0", "0", "0"),
  sex = c(1, 2, 1, 2, 1),
  id_inf = c("D", "E")
)

data(sampleped)
ped <- Pedigree(sampleped)
kin(ped(min_dist_inf(ped, col_aff = "affection_mods")))

---

na_to_length NA to specific length

Description

Check if all value in a vector is NA. If so set all of them to a new value matching the length of the template. If not check that the size of the vector is equal to the template.

Usage

na_to_length(x, temp, value)
Arguments

- **x**  The vector to check.
- **temp**  A template vector to use to determine the length.
- **value**  The value to use to fill the vector.

Value

A vector with the same length as temp.

Examples

```r
na_to_length(NA, rep(0, 4), "NewValue")
na_to_length(c(1, 2, 3, NA), rep(0, 4), "NewValue")
```

---

**norm_ped**  

*Normalise a Ped object dataframe*

---

Description

Normalise dataframe for a Ped object

Usage

```r
norm_ped(
  ped_df,
  na_strings = c("NA", ""),
  missid = NA_character_,
  try_num = FALSE
)
```

Arguments

- **ped_df**  A data.frame with the individuals informations. The minimum columns required are:
  - `indID`  individual identifiers  ->  `id`
  - `fatherId`  biological fathers identifiers  ->  `dadid`
  - `motherId`  biological mothers identifiers  ->  `momdid`
  - `gender`  sex of the individual  ->  `sex`
  - `family`  family identifiers  ->  `famid`

The family column, if provided, will be merged to the *ids* field separated by an underscore using the *upd_famid_id()* function.

The following columns are also recognize and will be transformed with the *vect_to_binary()* function:
- 'sterilisation' status -> 'steril'
- 'available' status -> 'avail'
- 'vitalStatus', is the individual dead -> 'status'
- 'affection' status -> 'affected'

The values recognized for those columns are 1 or 0, TRUE or FALSE.

na_strings Vector of strings to be considered as NA values.

missid A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

try_num Boolean defining if the function should try to convert all the columns to numeric.

Details

Normalise a dataframe and check for columns correspondance to be able to use it as an input to create a Ped object. Multiple test are done and errors are checked. Sex is calculated based on the gender column.

The steril column need to be a boolean either TRUE, FALSE or 'NA'. Will be considered available any individual with no 'NA' values in the available column. Duplicated indId will nullify the relationship of the individual. All individuals with errors will be remove from the dataframe and will be transfered to the error dataframe.

A number of checks are done to ensure the dataframe is correct:

On identifiers::
- All ids (id, dadid, momid, famid) are not empty (!= "")
- All id are unique (no duplicated)
- All dadid and momid are unique in the id column (no duplicated)
- id is not the same as dadid or momid
- Either have both parents or none

On sex:
- All sex code are either male, female, terminated or unknown.
- No parents are steril
- All fathers are male
- All mothers are female

Value

A dataframe with different variable correctly standardized and with the errors identified in the error column

See Also

Ped(), Ped, Pedigree()
Examples

```r
df <- data.frame(
  indId = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  fatherId = c("A", 0, 1, 3, 0, 4, 1, 0, 6, 6),
  motherId = c(0, 0, 2, 0, 5, 2, 0, 8, 8),
  gender = c(1, 2, "m", "f", "m", "m", "f", "f"),
  available = c("A", "1", 0, NA, 1, 0, 1, 0, 1, 0),
  famid = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2),
  sterilisation = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, 1, 0, 1, "TRUE"),
  vitalStatus = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, 1, 0, 1, 0),
  affection = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, 1, 0, 1, 0)
)

tryCatch(
  norm_ped(df),
  error = function(e) print(e)
)
```

---

**norm.rel**

Normalise a Rel object dataframe

**Description**

Normalise a dataframe and check for columns correspondance to be able to use it as an input to create a Ped object.

**Usage**

`norm.rel(rel_df, na_strings = c("NA", ""), missid = NA_character_)`

**Arguments**

- `rel_df`: A data.frame with the special relationships between individuals. See `Rel()` for more informations. The minimum columns required are `id1`, `id2` and `code`. The `famid` column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as `id1`, `id2`, `code` and `famid`. The code values are:
  - 1 = Monozygotic twin
  - 2 = Dizygotic twin
  - 3 = Twin of unknown zygosity
  - 4 = Spouse

  The value relation code recognized by the function are the one defined by the `rel_code_to_factor()` function.

- `na_strings`: Vector of strings to be considered as NA values.

- `missid`: A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`. 
Details

The `famid` column, if provided, will be merged to the `ids` field separated by an underscore using the `upd_famid_id()` function. The `code` column will be transformed with the `rel_code_to_factor()` function. Multiple test are done and errors are checked.

A number of checks are done to ensure the dataframe is correct:

**On identifiers::**
- All ids (id1, id2) are not empty (\(!= \ "\)"
- id1 and id2 are not the same

**On code:**
- All code are recognised as either "MZ twin", "DZ twin", "UZ twin" or "Spouse"

Value

A dataframe with the errors identified

Examples

df <- data.frame(  
id1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),  
id2 = c(2, 3, 4, 5, 6, 7, 8, 9, 10, 1),  
code = c("MZ twin", "DZ twin", "UZ twin", "Spouse", 1, 2, 3, 4, "MzTwin", "sp oUse"),  
famid = c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2)
)

norm_rel(df)

<table>
<thead>
<tr>
<th>num_child</th>
<th>Number of childs</th>
</tr>
</thead>
</table>

Description

Compute the number of childs per individual

Usage

```r
## S4 method for signature 'character.OR.integer'
num_child(obj, dadid, momid, rel_df = NULL, missid = NA_character_)

## S4 method for signature 'Pedigree'
num_child(obj, reset = FALSE)
```
Arguments

- **obj**: A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
- **dadid**: A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid**: A vector containing for each subject, the identifiers of the biologicals mothers.
- **rel_df**: A data.frame with the special relationships between individuals. See Rel() for more informations. The minimum columns required are id1, id2 and code. The famid column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as id1, id2, code and famid. The code values are:
  - 1 = Monozygotic twin
  - 2 = Dizygotic twin
  - 3 = twin of unknown zygosity
  - 4 = Spouse

  The value relation code recognized by the function are the one defined by the rel_code_to_factor() function.
- **missid**: A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.
- **reset**: If TRUE, the num_child_tot, num_child_ind and the num_child_dir columns are reset.

Details

Compute the number of direct child but also the number of indirect child given by the ones related with the linked spouses. If a relation ship dataframe is given, then even if no children is present between 2 spouses, the indirect childs will still be added.

Value

- **When obj is a vector**: A dataframe with the columns num_child_dir, num_child_ind and num_child_tot giving respectively the direct, indirect and total number of child.

- **When obj is a Pedigree object**: An updated Pedigree object with the columns num_child_dir, num_child_ind and num_child_tot added to the Pedigree ped slot.

Examples

```r
num_child(
  obj = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10"),
  dadid = c("3", "3", "6", "8", "0", "0", "0", "0", "0", "0"),
  momid = c("4", "5", "7", "9", "0", "0", "0", "0", "0", "0"),
  rel_df = data.frame(
    id1 = "10",
    id2 = "3",
```
code = "Spouse"

)

)

data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
ped1 <- num_child(ped1, reset = TRUE)
summary(ped(ped1))

-------------------
paste0max  Print0 to max
-------------------

Description
Print0 the elements inside a vector until a maximum is reached.

Usage
paste0max(x, max = 5, sep = "", ...)

Arguments
x  A vector.
max The maximum number of elements to print.
... Additional arguments passed to print0

Value
The character vector aggregated until the maximum is reached.

-------------------
Ped-class  Ped object
-------------------

Description
S4 class to represent the identity informations of the individuals in a pedigree.

Constructor ::
You either need to provide a vector of the same size for each slot or a data.frame with the corresponding columns.
The metadata will correspond to the columns that do not correspond to the Ped slots.
Usage

## S4 method for signature 'data.frame'
Ped(obj, cols_used_init = FALSE, cols_used_del = FALSE)

## S4 method for signature 'character_OR_integer'
Ped(
  obj,
  sex,
  dadid,
  momid,
  famid = NA,
  steril = NA,
  status = NA,
  avail = NA,
  affected = NA,
  missid = NA_character_
)

Arguments

obj A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.

cols_used_init Boolean defining if the columns that will be used should be initialised to NA.

cols_used_del Boolean defining if the columns that will be used should be deleted.

sex A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: male < female < unknown < ‘terminated’.

• character() or factor(): "f", "m", "woman", "man", "male", "female", "unknown", "terminated"

• numeric(): 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"

dadid A vector containing for each subject, the identifiers of the biologicals fathers.

momid A vector containing for each subject, the identifiers of the biologicals mothers.

famid A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.

steril A logical vector with the sterilisation status of the individuals (i.e. FALSE = not sterilised, TRUE = sterilised, NA = unknown).

status A logical vector with the affection status of the individuals (i.e. FALSE = alive, TRUE = dead, NA = unknown).

avail A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).

affected A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).

missid A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Details

The minimal needed informations are id, dadid, momid and sex. The other slots are used to store recognized informations. Additional columns can be added to the Ped object and will be stored in the elementMetadata slot of the Ped object.

Value

A Ped object.

Slots

- **id**: A character vector with the id of the individuals.
- **dadid**: A character vector with the id of the father of the individuals.
- **momid**: A character vector with the id of the mother of the individuals.
- **sex**: An ordered factor vector for the sex of the individuals (i.e. male < female < unknown < terminated).
- **famid**: A character vector with the family identifiers of the individuals (optional).
- **steril**: A logical vector with the sterilisation status of the individuals (i.e. FALSE = not sterilised, TRUE = sterilised, NA = unknown).
- **status**: A logical vector with the affection status of the individuals (i.e. FALSE = alive, TRUE = dead, NA = unknown).
- **avail**: A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
- **affected**: A logical vector with the affection status of the individuals (i.e. FALSE = not affected, TRUE = affected, NA = unknown).
- **useful**: A logical vector with the usefulness status of the individuals (i.e. FALSE = not useful, TRUE = useful).
- **isinf**: A logical vector indicating if the individual is informative or not (i.e. FALSE = not informative, TRUE = informative).
- **kin**: A numeric vector with minimal kinship value between the individuals and the useful individuals.
- **num_child_tot**: A numeric vector with the total number of children of the individuals.
- **num_child_dir**: A numeric vector with the number of children of the individuals.
- **num_child_ind**: A numeric vector with the number of children of the individuals.
- **elementMetadata**: A DataFrame with the additional metadata columns of the Ped object.
- **metadata**: Meta informations about the pedigree.

Accessors

For all the following accessors, the x parameters is a Ped object. Each getters return a vector of the same length as x with the values of the corresponding slot. For each getter, you have a setter with the same name, to be use as slot(x) <- value. The value parameter is a vector of the same length as x, except for the mcols() accessors where value is a list or a data.frame with each elements with the same length as x.
• id(x): Individuals identifiers
• dadid(x): Individuals’ father identifiers
• momid(x): Individuals’ mother identifiers
• famid(x): Individuals’ family identifiers
• sex(x): Individuals’ gender
• affected(x): Individuals’ affection status
• avail(x): Individuals’ availability status
• status(x): Individuals’ death status
• isinf(x): Individuals’ informativeness status
• kin(x): Individuals’ kinship distance to the informative individuals
• useful(x): Individuals’ usefullness status
• mcols(x): Individuals’ metadata

Generics
• summary(x): Compute the summary of a Ped object
• show(x): Convert the Ped object to a data.frame and print it with its summary.
• as.list(x): Convert a Ped object to a list with the metadata columns at the end.
• as.data.frame(x): Convert a Ped object to a data.frame with the metadata columns at the end.
• subset(x, i, del_parents = FALSE, keep = TRUE): Subset a Ped object based on the individuals identifiers given.
  – i: A vector of individuals identifiers to keep.
  – del_parents: A logical value indicating if the parents of the individuals should be deleted.
  – keep: A logical value indicating if the individuals should be kept or deleted.

See Also

Pedigree()
Pedigree-class

Examples

```r
data(sampleped)
Ped(sampleped)

Ped(
  obj = c("1", "2", "3", "4", "5", "6"),
  dadid = c("4", "4", "6", "0", "0", "0"),
  momid = c("5", "5", "0", "0", "0"),
  sex = c(1, 2, 3, 1, 2, 1),
  missid = "0"
)
```

Pedigree-class Pedigree object

Description

A pedigree is a ensemble of individuals linked to each other into a family tree. A Pedigree object store the informations of the individuals and the special relationships between them. It also permit to store the informations needed to plot the pedigree (i.e. scales and hints).

Constructor ::

Main constructor of the package. This constructor help to create a Pedigree object from different data.frame or a set of vectors.

If any errors are found in the data, the function will return the data.frame with the errors of the Ped object and the Rel object.

Usage

```r
Pedigree(obj, ...)
```

## S4 method for signature 'character OR integer'

Pedigree(
  obj,
  dadid,
  momid,
  sex,
  famid = NA,
  avail = NULL,
  affected = NULL,
  status = NULL,
  steril = NULL,
  rel_df = NULL,
  missid = NA_character_,
  col_aff = "affection",
  normalize = TRUE,
  ...
)
Pedigree-class

## S4 method for signature 'data.frame'

Pedigree(
  obj = data.frame(indId = character(), fatherId = character(), motherId = character(),
                  gender = numeric(), family = character(), available = numeric(), vitalStatus =
                  numeric(), affection = numeric(), sterilisation = numeric()),
  rel_df = data.frame(id1 = character(), id2 = character(), code = numeric(), famid =
                      character()),
  cols_ren_ped = list(indId = "id", fatherId = "dadid", motherId = "momid", family =
                     "famid", gender = "sex", sterilisation = "steril", affection = "affected", available
                     = "avail", vitalStatus = "status"),
  cols_ren_rel = list(id1 = "indId1", id2 = "indId2", famid = "family"),
  hints = list(horder = NULL, spouse = NULL),
  normalize = TRUE,
  missid = NA_character_,
  col_aff = "affection",
  ...
)

### Arguments

- **obj**: A vector of the individuals identifiers or a data.frame with the individuals informations. See **Ped()** for more informations.
- **...**: Arguments passed on to **generate_colors**
- **dadid**: A vector containing for each subject, the identifiers of the biologicals fathers.
- **momid**: A vector containing for each subject, the identifiers of the biologicals mothers.
- **sex**: A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: 
  - character() or factor(): "f", "m", "woman", "man", "male", "female", "unknown", "terminated"
  - numeric(): 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"
- **famid**: A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
- **avail**: A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
- **affected**: A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown). Can also be a data.frame with the same length as obj. If it is a matrix, it will be converted to a data.frame and the columns will be named after the col_aff argument.
- **status**: A logical vector with the affection status of the individuals (i.e. FALSE = alive, TRUE = dead, NA = unknown).
- **steril**: A logical vector with the sterilisation status of the individuals (i.e. FALSE = not sterilised, TRUE = sterilised, NA = unknown).
Pedigree-class

rel_df
A data.frame with the special relationships between individuals. See Rel() for more informations. The minimum columns required are id1, id2 and code. The famid column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as id1, id2, code and famid. The code values are:

- 1 = Monozygotic twin
- 2 = Dizygotic twin
- 3 = twin of unknown zygosity
- 4 = Spouse

The value relation code recognized by the function are the one defined by the rel_code_to_factor() function.

missid
A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

col_aff
A character vector with the name of the column to be used for the affection status.

normalize
A logical to know if the data should be normalised.

cols_ren_ped
A named list with the columns to rename for the pedigree dataframe. This is useful if you want to use a dataframe with different column names. The names of the list should be the new column names and the values should be the old column names. The default values are to be used with normalize = TRUE.

cols_ren_rel
A named list with the columns to rename for the relationship matrix. This is useful if you want to use a dataframe with different column names. The names of the list should be the new column names and the values should be the old column names.

hints
A Hints object or a named list containing horder and spouse.

Details

If the normalization is set to TRUE, then the data will be standardized using the function norm_ped() and norm_rel().

If a data.frame is given, the columns names needed will depend if the normalization is selected or not. If the normalization is selected, the columns names needed are as follow and if not the columns names needed are in parenthesis:

- indID: the individual identifier (id)
- fatherId: the identifier of the biological father (dadid)
- motherId: the identifier of the biological mother (momid)
- gender: the sex of the individual (sex)
- family: the family identifier of the individual (famid)
- sterilisation: the sterilisation status of the individual (steril)
- available: the availability status of the individual (avail)
- vitalStatus: the death status of the individual (status)
- affection: the affection status of the individual (affected)
The minimum columns required are:

- indID / id
- fatherId / dadid
- motherId / momid
- gender / sex

The family / famid column can also be used to specify the family of the individuals and will be merge to the indID / id field separated by an underscore. The columns sterilisation, available, vitalStatus, affection will be transformed with the `vect_to_binary()` function when the normalisation is selected. If you do not use the normalisation, the columns will be checked to be 0 or 1.

If affected is a data.frame, `col_aff` will be overwritten by the column names of the data.frame.

Value

A Pedigree object.

Slots

- ped: A Ped object for the identity informations. See `Ped()` for more informations.
- rel: A Rel object for the special relationships. See `Rel()` for more informations.
- scales: A Scales object for the filling and bordering colors used in the plot. See `Scales()` for more informations.
- hints: A Hints object for the ordering of the individuals in the plot. See `Hints()` for more informations.

Accessors

For all the following accessors, the x parameters is a Pedigree object. Each getters return a vector of the same length as x with the values of the corresponding slot.

- famid(x): Get the family identifiers of a Pedigree object. This function is a wrapper around famid(ped(x)).
- ped(x, slot): Get the value of a specific slot of the Ped object
- ped(x): Get the Ped object
- ped(x, slot) <- value: Set the value of a specific slot of the Ped object Wrapper of slot(ped(x)) <- value
- ped(x) <- value: Set the Ped object
- mcols(x): Get the metadata of a Pedigree object. This function is a wrapper around mcols(ped(x)).
- mcols(x) <- value: Set the metadata of a Pedigree object. This function is a wrapper around mcols(ped(x)) <- value.
• rel(x, slot) : Get the value of a specific slot of the Rel object
• rel(x) : Get the Rel object
• rel(x, slot) <- value : Set the value of a specific slot of the Rel object. Wrapper of slot(rel(x)) <- value
• rel(x) <- value : Set the Rel object
• scales(x) : Get the Scales object
• scales(x) <- value : Set the Scales object
• fill(x) : Get the fill.data.frame from the Scales object. Wrapper of fill(scales(x))
• fill(x) <- value : Set the fill.data.frame from the Scales object. Wrapper of fill(scales(x)) <- value
• border(x) : Get the border.data.frame from the Scales object. Wrapper of border(scales(x))
• border(x) <- value : Set the border.data.frame from the Scales object. Wrapper of border(scales(x)) <- value
• hints(x) : Get the Hints object
• hints(x) <- value : Set the Hints object
• horder(x) : Get the horder vector from the Hints object. Wrapper of horder(hints(x))
• horder(x) <- value : Set the horder vector from the Hints object. Wrapper of horder(hints(x)) <- value
• spouse(x) : Get the spouse.data.frame from the Hints object. Wrapper of spouse(hints(x)).
• spouse(x) <- value : Set the spouse.data.frame from the Hints object. Wrapper of spouse(hints(x)) <- value.

Generics

• length(x) : Get the length of a Pedigree object. Wrapper of length(ped(x)).
• show(x) : Print the information of the Ped and Rel object inside the Pedigree object.
• summary(x) : Compute the summary of the Ped and Rel object inside the Pedigree object.
• as.list(x) : Convert a Pedigree object to a list
• subset(x, i, keep = TRUE) : Subset a Pedigree object based on the individuals identifiers given.
  – i : A vector of individuals identifiers to keep.
  – del_parents : A logical value indicating if the parents of the individuals should be deleted.
  – keep : A logical value indicating if the individuals should be kept or deleted.
• x[i, del_parents, keep] : Subset a Pedigree object based on the individuals identifiers given.
ped_to_legdf

See Also

Pedigree(), Ped(), Rel(), Scales(), Hints()
Ped(), Rel(), Scales()

Examples

Pedigree(
  obj = c("1", "2", "3", "4", "5", "6"),
  dadid = c("4", "4", "6", "0", "0", "0"),
  momid = c("5", "5", "5", "0", "0", "0"),
  sex = c(1, 2, 3, 1, 2, 1),
  avail = c(0, 1, 0, 1, 0, 1),
  affected = matrix(c(
    0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1
  ), ncol = 2),
  col_aff = c("aff1", "aff2"),
  missid = "0",
  rel_df = matrix(c(
    "1", "2", 2
  ), ncol = 3, byrow = TRUE),
)

data(sampleped)
Pedigree(sampleped)

ped_to_legdf Create plotting legend data frame from a Pedigree

Description

Convert a Pedigree to a legend data frame for it to be plotted afterwards with plot_fromdf().

Usage

## S4 method for signature 'Pedigree'
ped_to_legdf(obj, boxh = 1, boxw = 1, cex = 1, adjx = 0, adjy = 0)

Arguments

obj A Pedigree object
boxh Height of the polygons elements
boxw Width of the polygons elements
cex Character expansion of the text
adjx default=0. Controls the horizontal text adjustment of the labels in the legend.
adxy default=0. Controls the vertical text adjustment of the labels in the legend.
Details

The data frame contains the following columns:

- \( x_0, y_0, x_1, y_1 \): coordinates of the elements
- \( \text{type} \): type of the elements
- \( \text{fill} \): fill color of the elements
- \( \text{border} \): border color of the elements
- \( \text{angle} \): angle of the shading of the elements
- \( \text{density} \): density of the shading of the elements
- \( \text{cex} \): size of the elements
- \( \text{label} \): label of the elements
- \( \text{tips} \): tips of the elements (used for the tooltips)
- \( \text{adjx} \): horizontal text adjustment of the labels
- \( \text{adjy} \): vertical text adjustment of the labels

All those columns are used by \( \text{plot_fromdf()} \) to plot the graph.

Value

A list containing the legend data frame and the user coordinates.

Examples

```r
data("sampleped")
ped <- Pedigree(sampleped)
leg_df <- ped_to_legdf(ped)
summary(leg_df$df)
plot_fromdf(leg_df$df, usr = c(-1,15,0,7))
```
Usage

```r
## S4 method for signature 'Pedigree'
ped_to_plotdf(
  obj,
  packed = TRUE,
  width = 6,
  align = c(1.5, 2),
  subreg = NULL,
  cex = 1,
  symbolsize = cex,
  pconnect = 0.5,
  branch = 0.6,
  aff_mark = TRUE,
  label = NULL,
  ...
)
```

Arguments

- `obj` A Pedigree object
- `...` Other arguments passed to `par()`
- `packed` Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
- `width` For a packed output, the minimum width of the plot, in inches.
- `align` For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine `alignped4()` will be called.
- `subreg` A 4-element vector for (min x, max x, min depth, max depth), used to edit away portions of the plot coordinates returned by `align()`. This is useful for zooming in on a particular region of the Pedigree.
- `cex` Character expansion of the text
- `symbolsize` Size of the symbols
- `pconnect` When connecting parent to children the program will try to make the connecting line as close to vertical as possible, subject to it lying inside the endpoints of the line that connects the children by at least pconnect people. Setting this option to a large number will force the line to connect at the midpoint of the children.
- `branch` defines how much angle is used to connect various levels of nuclear families.
- `aff_mark` If TRUE, add a aff_mark to each box corresponding to the value of the affection column for each filling scale.
- `label` If not NULL, add a label to each box corresponding to the value of the column given.
Details

The data frame contains the following columns:

- x0, y0, x1, y1: coordinates of the elements
- type: type of the elements
- fill: fill color of the elements
- border: border color of the elements
- angle: angle of the shading of the elements
- density: density of the shading of the elements
- cex: size of the elements
- label: label of the elements
- tips: tips of the elements (used for the tooltips)
- adjx: horizontal text adjustment of the labels
- adjy: vertical text adjustment of the labels

All those columns are used by `plot_fromdf()` to plot the graph.

Value

A list containing the data frame and the user coordinates.

See Also

`plot_fromdf()` `ped_to_legdf()`

Examples

data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == 1,])
plot_df <- ped_to_plotdf(ped1)
summary(plot_df$df)
plot_fromdf(plot_df$df, usr = plot_df$par_usr$usr,
            boxh = plot_df$par_usr$boxh, boxw = plot_df$par_usr$boxw)

permute

Generate all possible permutation

Description

Given a vector of length n, generate all possible permutations of the numbers 1 to n. This is a recursive routine, and is not very efficient.
Usage
   permute(x)

Arguments
   x  A vector of length n

Value
   A matrix with n cols and n! rows

Description
This function is used to plot a Pedigree object.

   It is a wrapper for plot_fromdf() and ped_to_plotdf() as well as ped_to_legdf() if legend = TRUE.

Usage
   ## S4 method for signature 'Pedigree,missing'
   plot(
      x,
      aff_mark = TRUE,
      label = NULL,
      ggplot_gen = FALSE,
      cex = 1,
      symbolsize = 1,
      branch = 0.6,
      packed = TRUE,
      align = c(1.5, 2),
      width = 6,
      title = NULL,
      subreg = NULL,
      pconnect = 0.5,
      fam_to_plot = 1,
      legend = FALSE,
      leg_cex = 0.8,
      leg_symbolsize = 0.5,
      leg_loc = NULL,
      leg_adjx = 0,
      leg_adjy = 0,
      ...
   )
Arguments

x
A Pedigree object.

aff_mark
If TRUE, add a aff_mark to each box corresponding to the value of the affection column for each filling scale.

label
If not NULL, add a label to each box corresponding to the value of the column given.

ggplot_gen
If TRUE add the segments to the ggplot object

cex
Character expansion of the text

symbolsize
Size of the symbols

branch
defines how much angle is used to connect various levels of nuclear families.

packed
Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)

align
For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine alignped4() will be called.

width
For a packed output, the minimum width of the plot, in inches.

title
The title of the plot.

subreg
A 4-element vector for (min x, max x, min depth, max depth), used to edit away portions of the plot coordinates returned by align(). This is useful for zooming in on a particular region of the Pedigree.

pconnect
When connecting parent to children the program will try to make the connecting line as close to vertical as possible, subject to it lying inside the endpoints of the line that connects the children by at least pconnect people. Setting this option to a large number will force the line to connect at the midpoint of the children.

fam_to_plot
default=1. If the Pedigree contains multiple families, this parameter can be used to select which family to plot. It can be a numeric value or a character value. If numeric, it is the index of the family to plot returned by unique(x$ped$famid). If character, it is the family id to plot.

legend
default=FALSE. If TRUE, a legend will be added to the plot.

leg_cex
default=0.8. Controls the size of the legend text.

leg_symbolsize
default=0.5. Controls the size of the legend symbols.

leg_loc
default=NULL. If NULL, the legend will be placed in the upper right corner of the plot. Otherwise, a 4-element vector of the form (x0, x1, y0, y1) can be used to specify the location of the legend.

leg_adjx
default=0. Controls the horizontal labels adjustment of the legend.

leg_adjy
default=0. Controls the vertical labels adjustment of the legend.

...Extra options that feed into the plot function.
Details

Two important parameters control the looks of the result. One is the user specified maximum width. The smallest possible width is the maximum number of subjects on a line, if the user’s suggestion is too low it is increased to 1 + that amount (to give just a little wiggle room).

To make a Pedigree where all children are centered under parents simply make the width large enough, however, the symbols may get very small.

The second is align, a vector of 2 alignment parameters a and b. For each set of siblings at a set of locations x and with parents at p=c(p1, p2) the alignment penalty is

\[
\frac{1}{k^a} \sum_{i=1}^{k} \left[ \frac{x_i - (p1 + p2)/2}{k} \right]^2
\sum_{i=1}^{k} \frac{(x - \bar{p})^2}{(k^a)}
\]

Where k is the number of siblings in the set.

When a = 1 moving a sibship with k sibs one unit to the left or right of optimal will incur the same cost as moving one with only 1 or two sibs out of place.

If a = 0 then large sibships are harder to move than small ones, with the default value a = 1.5 they are slightly easier to move than small ones. The rationale for the default is as long as the parents are somewhere between the first and last siblings the result looks fairly good, so we are more flexible with the spacing of a large family. By tethering all the sibs to a single spot they are kept close to each other. The alignment penalty for spouses is \(b(x_1 - x_2)^2\), which tends to keep them together. The size of b controls the relative importance of sib-parent and spouse-spouse closeness.

Value

an invisible list containing

- df : the data.frame used to plot the Pedigree
- par_usr : the user coordinates used to plot the Pedigree
- ggplot : the ggplot object if ggplot_gen = TRUE

Side Effects

Creates plot on current plotting device.

See Also

Pedigree()

Examples

data(sampleped)
pedAll <- Pedigree(sampleped)
#plot(pedAll)
plot_fromdf

Create a plot from a data.frame

Description

This function is used to create a plot from a data.frame.

If ggplot_gen = TRUE, the plot will be generated with ggplot2 and will be returned invisibly.

Usage

plot_fromdf(
  df,
  usr = NULL,
  title = NULL,
  ggplot_gen = FALSE,
  boxw = 1,
  boxh = 1,
  add_to_existing = FALSE
)

Arguments

  df  A data.frame with the following columns:
      • type: The type of element to plot. Can be text, segments, arc or other polygons. For polygons, the name of the polygon must be in the form poly_*_* where poly is one of the type given by polygons(), the first * is the number of slice in the polygon and the second * is the position of the division of the polygon.
      • x0: The x coordinate of the center of the element.
      • y0: The y coordinate of the center of the element.
      • x1: The x coordinate of the end of the element. Only used for segments and arc.
      • y1: The y coordinate of the end of the element. Only used for segments and arc.
      • fill: The fill color of the element.
      • border: The border color of the element.
      • density: The density of the element.
      • angle: The angle of the element.
      • label: The label of the element. Only used for text.
      • cex: The size of the element.
      • adjx: The x adjustment of the element. Only used for text.
      • adjy: The y adjustment of the element. Only used for text.
  usr  The user coordinates of the plot.
  title  The title of the plot.
polyfun

### Description

Create a list of x and y coordinates for a polygon with a given number of slices and a list of coordinates for the polygon.

### Usage

```r
polyfun(nslicer, coor)
```

### Arguments

- `nslicer`: Number of slices in the polygon
- `coor`: Element form which to generate the polygon containing x and y coordinates and theta

### Value

a list of x and y coordinates

### Examples

```r
polyfun(2, list(
  x = c(-0.5, -0.5, 0.5, 0.5),
  y = c(-0.5, 0.5, 0.5, -0.5),
  theta = -c(3, 5, 7, 9) * pi / 4
))
```
**polygons**  
*List of polygonal elements*

**Description**
Create a list of polygonal elements with x, y coordinates and theta for the square, circle, diamond and triangle. The number of slices in each element can be specified.

**Usage**

```r
polygons(nslice = 1)
```

**Arguments**

- `nslice` Number of slices in each element If nslice > 1, the elements are created with `polyfun()`.

**Value**
a list of polygonal elements with x, y coordinates and theta by slice.

**Examples**

```r
polygons()
polygons()
polygons(4)
```

---

**Rel-class**  
*Rel object*

**Description**
S4 class to represent the special relationships in a Pedigree.

**Constructor ::**
You either need to provide a vector of the same size for each slot or a `data.frame` with the corresponding columns.

**Usage**

```r
## S4 method for signature 'data.frame'
Rel(obj)
```

```r
## S4 method for signature 'character.OR.integer'
Rel(obj, id2, code, famid = NA_character_)
```
**Arguments**

- `obj` A character vector with the id of the first individuals of each pairs or a data.frame with all the informations in corresponding columns.
- `id2` A character vector with the id of the second individuals of each pairs.
- `code` A character, factor or numeric vector corresponding to the relation code of the individuals:
  - MZ twin = Monozygotic twin
  - DZ twin = Dizygotic twin
  - UZ twin = twin of unknown zygosity
  - Spouse = Spouse The following values are recognized:
    - character() or factor(): "MZ twin", "DZ twin", "UZ twin", "Spouse" with of without space between the words. The case is not important.
    - numeric(): 1 = "MZ twin", 2 = "DZ twin", 3 = "UZ twin", 4 = "Spouse"
- `famid` A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.

**Details**

A Rel object is a list of special relationships between individuals in the pedigree. It is used to create a Pedigree object. The minimal needed informations are `id1`, `id2` and `code`.

If a `famid` is provided, the individuals `id` will be aggregated to the `famid` character to ensure the uniqueness of the `id`.

**Value**

A Rel object.

**Slots**

- `id1` A character vector with the id of the first individual.
- `id2` A character vector with the id of the second individual.
- `code` An ordered factor vector with the code of the special relationship. (i.e. MZ twin < DZ twin < UZ twin < Spouse).
- `famid` A character vector with the famid of the individuals.

**Accessors**

For all the following accessors, the `x` parameters is a Rel object. Each getters return a vector of the same length as `x` with the values of the corresponding slot.

- `code(x)` : Relationships' code
- `id1(x)` : Relationships' first individuals' identifier
- `id2(x)` : Relationships' second individuals' identifier
• `famid(x)` : Relationships’ individuals’ family identifier

• `famid(x) <- value` : Set the relationships’ individuals’ family identifier
  – `value` : A character or integer vector of the same length as `x` with the family identifiers

Generics

• `summary(x)` : Compute the summary of a Rel object

• `show(x)` : Convert the Rel object to a data.frame and print it with its summary.

• `as.list(x)` : Convert a Rel object to a list

• `as.data.frame(x)` : Convert a Rel object to a data.frame

• `subset(x, i, keep = TRUE)` : Subset a Rel object based on the individuals identifiers given.
  – `i` : A vector of individuals identifiers to keep.
  – `keep` : A logical value indicating if the individuals should be kept or deleted.

See Also

`Pedigree()`

Examples

```r
rel_df <- data.frame(
  id1 = c("1", "2", "3"),
  id2 = c("2", "3", "4"),
  code = c(1, 2, 3)
)
Rel(rel_df)

Rel(
  obj = c("1", "2", "3"),
  id2 = c("2", "3", "4"),
  code = c(1, 2, 3)
)
```

rel_code_to_factor

`Relationship code variable to ordered factor`

Description

Relationship code variable to ordered factor

Usage

```r
rel_code_to_factor(code)
```
Arguments
code A character, factor or numeric vector corresponding to the relation code of the individuals:
- MZ twin = Monozygotic twin
- DZ twin = Dizygotic twin
- UZ twin = twin of unknown zygosity
- Spouse = Spouse The following values are recognized:
  - character() or factor(): "MZ twin", "DZ twin", "UZ twin", "Spouse" with of without space between the words. The case is not important.
  - numeric(): 1 = "MZ twin", 2 = "DZ twin", 3 = "UZ twin", 4 = "Spouse"

Value
an ordered factor vector containing the transformed variable "MZ twin" < "DZ twin" < "UZ twin" < "Spouse"

Examples
rel_code_to_factor(c(1, 2, 3, 4, "MZ twin", "DZ twin", "UZ twin", "Spouse"))

sampleped Sampleped data

Description
Small sample pedigree data set for testing purposes.

Usage
data("sampleped")

Format
A data frame with 55 observations, one line per subject, on the following 7 variables.
- famid: Family identifier
- id: Subject identifier
- dadid: Identifier of the father, if the father is part of the data set; zero otherwise
- momid: Identifier of the mother, if the mother is part of the data set; zero otherwise
- sex: 1 for male or 2 for female
- affected: 1 or 0
- avail: 1 or 0
Details

This is a small fictive pedigree data set, with 55 individuals in 2 families. The aim was to create a data set with a variety of pedigree structures.

Examples

```r
data("sampleped")
ped <- Pedigree(sampleped)
summary(ped)
#plot(ped)
```

### Scales-class

**Scales object**

**Description**

A Scales object is a list of two data.frame. The first one is used to represent the affection status of the individuals and therefore the filling of the individuals in the pedigree plot. The second one is used to represent the availability status of the individuals and therefore the border color of the individuals in the pedigree plot.

**Constructor ::**

You need to provide both `fill` and `border` in the dedicated parameters. However this is usually done using the `generate_colors()` function with a Pedigree object.

**Usage**

```r
Scales(fill, border)
```

```r
## S4 method for signature 'data.frame,data.frame'
Scales(fill, border)
```

**Arguments**

`fill` A data.frame with the informations for the affection status. The columns needed are:

- 'order': the order of the affection to be used
- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'affected': a logical value indicating if the mod correspond to an affected individuals
- 'fill': the color to use for this mods
A data.frame with the informations for the availability status. The columns needed are:

- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'border': the color to use for this mods

Value

A Scales object.

Slots

fill A data.frame with the informations for the affection status. The columns needed are:

- 'order': the order of the affection to be used
- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'affected': a logical value indicating if the mod correspond to an affected individuals
- 'fill': the color to use for this mods
- 'density': the density of the shading
- 'angle': the angle of the shading

border A data.frame with the informations for the availability status. The columns needed are:

- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'border': the color to use for this mods

Accessors

- fill(x) : Get the fill data.frame
- fill(x) <- value : Set the fill data.frame
- border(x) : Get the border data.frame
- border(x) <- value : Set the border data.frame
Generics

• as.list(x): Convert a Scales object to a list

See Also

Pedigree()
generate_colors()

Examples

Scales(
  fill = data.frame(
    order = 1,
    column_values = "affected",
    column.mods = "affected.mods",
    mods = c(0, 1),
    labels = c("unaffected", "affected"),
    affected = c(FALSE, TRUE),
    fill = c("white", "red"),
    density = c(NA, 20),
    angle = c(NA, 45)
  ),
  border = data.frame(
    column_values = "avail",
    column.mods = "avail.mods",
    mods = c(0, 1),
    labels = c("not available", "available"),
    border = c("black", "blue")
  )
)

Description

Set plotting area

Usage

set_plot_area(cex, id, maxlev, xrange, symbolsize, ...)

Arguments

cex Character expansion of the text
id A character vector with the identifiers of each individuals
maxlev Maximum level
sex_to_factor

\texttt{xrange} 	 Range of x values
\texttt{symbolsize} 	 Size of the symbols
... 	 Other arguments passed to \texttt{par()}

\textbf{Value}

List of user coordinates, old par, box width, box height, label height and leg height

\begin{center}
\textbf{sex_to_factor} \hspace{1cm} \textit{Gender variable to ordered factor}
\end{center}

\textbf{Description}

Gender variable to ordered factor

\textbf{Usage}

\texttt{sex_to_factor(sex)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{sex} 	 A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: male < female < unknown < \texttt{terminated} The following values are recognized:
  \begin{itemize}
    \item character() or factor() : "f", "m", "woman", "man", "male", "female", "unknown", "terminated"
    \item numeric() : 1 = "male", 2 = "female", 3 = "unknown", 4 = "terminated"
  \end{itemize}
\end{itemize}

\textbf{Value}

an ordered factor vector containing the transformed variable "male" < "female" < "unknown" < "terminated"

\textbf{Examples}

\texttt{sex_to_factor(c(1, 2, 3, 4, "f", "m", "man", "female"))}
shift

*Shift set of siblings to the left or right*

Description

Shift set of siblings to the left or right

Usage

```
shift(id, sibs, goleft, hint, twinrel, twinset)
```

Arguments

- **id**: The id of the subject to be shifted
- **sibs**: The ids of the siblings
- **goleft**: If TRUE, shift to the left, otherwise to the right
- **hint**: The current hint vector
- **twinrel**: The twin relationship matrix
- **twinset**: The twinset vector

Details

This routine is used by `auto_hint()`. It shifts a set of siblings to the left or right, so that the marriage is on the edge of the set of siblings, closest to the spouse. It also shifts the subject himself, so that he is on the edge of the set of siblings, closest to the spouse. It also shifts the monozygotic twins, if any, so that they are together within the set of twins.

Value

The updated hint vector

See Also

`auto_hint()`
Shrink Pedigree object

Description

Shrink Pedigree object to specified bit size with priority placed on trimming uninformative subjects. The algorithm is useful for getting a Pedigree condensed to a minimally informative size for algorithms or testing that are limited by size of the Pedigree.

If avail or affected are NULL, they are extracted with their corresponding accessors from the Ped object.

Usage

```r
## S4 method for signature 'Pedigree'
shrink(obj, avail = NULL, affected = NULL, max_bits = 16)

## S4 method for signature 'Ped'
shrink(obj, avail = NULL, affected = NULL, max_bits = 16)
```

Arguments

- `obj` A Pedigree or Ped object.
- `avail` A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
- `affected` A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
- `max_bits` Optional, the bit size for which to shrink the Pedigree

Details

Iteratively remove subjects from the Pedigree. The random removal of members was previously controlled by a seed argument, but we remove this, forcing users to control randomness outside the function. First remove uninformative subjects, i.e., unavailable (not genotyped) with no available descendants. Next, available terminal subjects with unknown phenotype if both parents available. Last, iteratively shrinks Pedigrees by preferentially removing individuals (chosen at random if there are multiple of the same status):

1. Subjects with unknown affected status
2. Subjects with unaffected affected status
3. Affected subjects.
Value

A list containing the following elements:

- pedObj: Pedigree object after trimming
- id_trim: Vector of ids trimmed from Pedigree
- id_lst: List of ids trimmed by category
- bit_size: Vector of bit sizes after each trimming step
- avail: Vector of availability status after trimming
- pedSizeOriginal: Number of subjects in original Pedigree
- pedSizeIntermed: Number of subjects after initial trimming
- pedSizeFinal: Number of subjects after final trimming

Author(s)

Original by Dan Schaid, updated by Jason Sinnwell and Louis Le Nézet

See Also

Pedigree(), bit_size()

Examples

data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == '1',])
shrink(ped1, max_bits = 12)

subregion

Subset a region of a Pedigree

Description

Subset a region of a Pedigree

Usage

subregion(plist, subreg)

Arguments

plist  The alignment structure representing the Pedigree layout. See align() for details.

subreg  A 4-element vector for (min x, max x, min depth, max depth), used to edit away portions of the plot coordinates returned by align(). This is useful for zooming in on a particular region of the Pedigree.
unrelated

Value

A Pedigree structure with the specified region

---

**unrelated** *Find Unrelated subjects*

---

Description

Determine set of maximum number of unrelated available subjects from a Pedigree.

Usage

```r
## S4 method for signature 'Ped'
unrelated(obj, avail = NULL)

## S4 method for signature 'Pedigree'
unrelated(obj, avail = NULL)
```

Arguments

- **obj**: A Pedigree or Ped object.
- **avail**: A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).

Details

Determine set of maximum number of unrelated available subjects from a Pedigree, given vectors id, father, and mother for a Pedigree structure, and status vector of TRUE / FALSE for whether each subject is available (e.g. has DNA).

This is a greedy algorithm that uses the kinship matrix, sequentially removing rows/cols that are non-zero for subjects that have the most number of zero kinship coefficients (greedy by choosing a row of kinship matrix that has the most number of zeros, and then remove any cols and their corresponding rows that are non-zero. To account for ties of the count of zeros for rows, a random choice is made. Hence, running this function multiple times can return different sets of unrelated subjects.

If **avail** is NULL, it is extracted with its corresponding accessor from the Ped object.

Value

A vector of the ids of subjects that are unrelated.

Author(s)

Dan Schaid and Shannon McDonnell updated by Jason Sinnwell
Examples

data(sampleped)
fam1 <- sampleped[sampleped$famid == 1,]
ped1 <- Pedigree(fam1)
unrelated(ped1)
## some possible vectors
## [1] '110' '113' '133' '109'
## [1] '113' '118' '141' '109'
## [1] '113' '118' '140' '109'
## [1] '110' '113' '116' '109'
## [1] '113' '133' '141' '109'

upd_famid_id  Update family prefix in individuals id

Description

Update the family prefix in the individuals identifiers. Individuals identifiers are constructed as follow famid_id. Therefore to update their family prefix the ids are split by the first underscore and the first part is overwritten by famid.

Usage

## S4 method for signature 'character,ANY'
upd_famid_id(obj, famid, missid = NA_character_)

## S4 method for signature 'Ped,character.OR.integer'
upd_famid_id(obj, famid)

## S4 method for signature 'Ped,missing'
upd_famid_id(obj)

## S4 method for signature 'Rel,character.OR.integer'
upd_famid_id(obj, famid)

## S4 method for signature 'Rel,missing'
upd_famid_id(obj)

## S4 method for signature 'Pedigree,character.OR.integer'
upd_famid_id(obj, famid)

## S4 method for signature 'Pedigree,missing'
upd_famid_id(obj)
useful_inds  

Arguments

obj  Ped or Pedigree object or a character vector of individual ids
famid  A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
missid  A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Details

If famid is missing, then the famid() function will be called on the object.

Value

A character vector of individual ids with family prefix updated

Examples

upd_famid_id(c("1", "2", "B_3"), c("A", "B", "A"))
upd_famid_id(c("1", "B_2", "C_3", "4"), c("A", NA, "A", NA))

data(sampleped)
ped1 <- Pedigree(sampleped[, -1])
id(ped1)
new_fam <- make_famid(id(ped(ped1)), dadid(ped(ped1)), momid(ped(ped1)))
id(ped(upd_famid_id(ped1, new_fam)))

data(sampleped)
ped1 <- Pedigree(sampleped[, -1])
make_famid(ped1)

useful_inds  Usefulness of individuals

Description

Compute the usefulness of individuals

Usage

## S4 method for signature 'character'
useful_inds(  
  obj,  
  dadid,  
  momid,  
  avail,  
  affected,  
  num_child_tot,
useful_inds

```r
useful_inds = function(obj, informative = "AvAf", keep_infos = FALSE, reset = FALSE) {
  ## S4 method for signature 'Pedigree'
  useful_inds(obj, informative = "AvAf", keep_infos = FALSE, reset = FALSE)
  ## S4 method for signature 'Ped'
  useful_inds(obj, informative = "AvAf", keep_infos = FALSE, reset = FALSE)
}
```

**Arguments**

- `obj` A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
- `dadid` A vector containing for each subject, the identifiers of the biologicals fathers.
- `momid` A vector containing for each subject, the identifiers of the biologicals mothers.
- `avail` A logical vector with the availability status of the individuals (i.e. `FALSE` = not available, `TRUE` = available, `NA` = unknown).
- `affected` A logical vector with the affection status of the individuals (i.e. `FALSE` = unaffected, `TRUE` = affected, `NA` = unknown).
- `num_child_tot` A numeric vector of the number of children of each individuals
- `informative` Informative individuals selection can take 5 values:
  - 'AvAf' (available and affected),
  - 'AvOrAf' (available or affected),
  - 'Av' (available only),
  - 'Af' (affected only),
  - 'All' (all individuals)
  - A numeric/character vector of individuals id
  - A boolean
- `keep_infos` Boolean to indicate if individuals with unknown status but available or reverse should be kept
- `reset` Boolean to indicate if the `useful` column should be reset

**Details**

Check for the informativeness of the individuals based on the informative parameter given, the number of children and the usefulness of their parents. A `useful` slot is added to the Ped object with the usefulness of the individual. This boolean is hereditary.

**Value**

- **When obj is a vector:**
  A vector of useful individuals identifiers

- **When obj is a Pedigree or Ped object:**
  The Pedigree or Ped object with the slot 'useful' containing `TRUE` for useful individuals and `FALSE` otherwise.
Examples

data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
ped(useful inds(ped1, informative = "AvAf"))

Description

Transform a vector to a binary vector. All values that are not 0, 1, TRUE, FALSE, or NA are transformed to NA.

Usage

vect_to_binary(vect, logical = FALSE)

Arguments

vect A character, factor, logical or numeric vector corresponding to a binary variable (i.e. 0 or 1). The following values are recognized:

- character() or factor(): "TRUE", "FALSE", "0", "1", "NA" will be respectively transformed to 1, 0, 0, 1, NA. Spaces and case are ignored. All other values will be transformed to NA.
- numeric(): 0 and 1 are kept, all other values are transformed to NA.
- logical(): TRUE and FALSE are transformed to 1 and 0.

logical Boolean defining if the output should be a logical vector instead of a numeric vector (i.e. 0 and 1 becomes FALSE and 'TRUE').

Value

numeric binary vector of the same size as vect with 0 and 1

Examples

vect_to_binary(
  c(0, 1, 2, 3.6, "TRUE", "FALSE", "0", "1", "NA", "B", TRUE, FALSE, NA)
)


Index

* Pedigree-plot
  circfun, 21
  draw_arc, 22
  draw_polygon, 23
  draw_segment, 24
  draw_text, 25
  ped_to_legdf, 75
  ped_to_plotdf, 76
  plot, Pedigree, missing-method, 79
  plot_fromdf, 82
  polyfun, 83
  polygons, 84
  set_plot_area, 90
  subregion, 94
* alignment,
  auto_hint, 15
  best_hint, 16
* alignment
  alignped1, 7
  alignped2, 9
  alignped3, 10
  alignped4, 12
* auto_hint
  auto_hint, 15
  best_hint, 16
  duporder, 26
  findsibs, 29
  findspouse, 30
  get_twin_rel, 41
  permute, 78
  shift, 92
* datasets
  minnbreast, 57
  sampleped, 87
* generate_scales
  generate_aff inds, 35
  generate_border, 37
  generate_colors, 37
  generate_fill, 40
* internal,
  alignped1, 7
  alignped2, 9
  alignped3, 10
  alignped4, 12
  auto_hint, 15
  bit_size, 17
  circfun, 21
  draw_arc, 22
  draw_polygon, 23
  draw_segment, 24
  draw_text, 25
  duporder, 26
  exclude_stray_marinyin, 26
  exclude_unavail_founders, 27
  find_avail_affected, 31
  find_avail_noninform, 32
  find_unavailable, 33
  findsibs, 29
  findspouse, 30
  get_twin_rel, 41
  ped_to_legdf, 75
  ped_to_plotdf, 76
  permute, 78
  plot_fromdf, 82
  polyfun, 83
  polygons, 84
  set_plot_area, 90
  shift, 92
  subregion, 94
* internal
  ancestors, 13
  check_columns, 18
  check_num_na, 20
  check_slot_fd, 20
  check_values, 21
  descendants, 22
  family_check, 28
  is_disconnected, 45
is_founder, 46
is_parent, 48
is_valid_hints, 49
is_valid_ped, 50
is_valid_pedigree, 50
is_valid_ped, 51
is_valid_scales, 52
na_to_length, 60
paste0max, 66

∗ shrink
bit_size, 17
exclude_stray_marryin, 26
exclude_unavail_founders, 27
find_avail_affected, 31
find_avail_noninform, 32
find_unavailable, 33
shrink, 93
useful_inds, 97

[, Pedigree, ANY, missing, ANY-method
(Pedigree-class), 70

affected (Ped-class), 66
affected, Ped-method (Ped-class), 66
affected<-, (Ped-class), 66
affected<-, Ped, numeric_OR_logical-method
(Ped-class), 66

align, 5
align(), 9–11, 13, 14, 16, 17, 26, 30, 53, 77, 80, 94

align, Pedigree-method (align), 5
alignped1, 7
alignped1(), 6, 7
alignped2, 9
alignped2(), 6–8
alignped3, 10
alignped3(), 6, 7
alignped4, 12
alignped4(), 6, 7
ancestors, 13
anchor_to_factor, 14
as.data.frame, Ped-method (Ped-class), 66
as.data.frame, Rel-method (Rel-class), 84
as.list, Hints-method (Hints-class), 42
as.list, Ped-method (Ped-class), 66

as.list, Pedigree-method
(Pedigree-class), 70
as.list, Rel-method (Rel-class), 84
as.list, Scales-method (Scales-class), 88

auto_hint(), 6, 7, 16, 17, 26, 30, 42, 92

auto_hint, Pedigree-method (auto_hint),
15

avail (Ped-class), 66
avail, Ped-method (Ped-class), 66
avail<-, (Ped-class), 66
avail<-, Ped, numeric_OR_logical-method
(Ped-class), 66

best_hint, 16
best_hint(), 16
best_hint, Pedigree-method (best_hint),
16

bit_size, 17
bit_size(), 4, 94

bit_size, character/OR_integer-method
(bit_size), 17

bit_size, Ped-method (bit_size), 17
bit_size, Pedigree-method (bit_size), 17

border (Scales-class), 88

border, Pedigree-method
(Pedigree-class), 70

border, Scales-method (Scales-class), 88
border<-, (Scales-class), 88
border<-, Pedigree, data.frame-method
(Pedigree-class), 70

border<-, Scales, data.frame-method
(Scales-class), 88

code (Rel-class), 84
code, Rel-method (Rel-class), 84

dadid (Ped-class), 66
dadid, Ped-method (Ped-class), 66
dadid<-, (Ped-class), 66
dadid<-, Ped, character/OR_integer-method
(Ped-class), 66

descendants, 22
descendants, character/OR_integer, character/OR_integer-method
(descendants), 22

descendants, character/OR_integer, Ped-method
(descendants), 22
descendants, character/OR_integer, Pedigree-method
(descendants), 22
draw_arc, 22
draw_polygon, 23
draw_segment, 24
draw_text, 25
duporder, 26

exclude_stray_marryin, 26
exclude_stray_marryin(), 33
exclude_unavail_founders, 27
exclude_unavail_founders(), 33

famid (Ped-class), 66
famid, Ped-method (Ped-class), 66
famid, Pedigree-method (Pedigree-class), 70
famid, Rel-method (Rel-class), 84
famid<-, (Ped-class), 66
famid<-, Ped, character OR integer-method (Ped-class), 66
famid<-, Rel, character OR integer-method (Rel-class), 84

family_check, 28
family_check, character OR integer-method (family_check), 28
family_check, Ped-method (family_check), 28
family_check, Pedigree-method (family_check), 28

fill (Scales-class), 88
fill, Pedigree-method (Pedigree-class), 70
fill, Scales-method (Scales-class), 88
fill<-, (Scales-class), 88
fill<-, Pedigree, data.frame-method (Pedigree-class), 70
fill<-, Scales, data.frame-method (Scales-class), 88

find_avail_affected, 31
find_avail_affected, Ped-method (find_avail_affected), 31
find_avail_affected, Pedigree-method (find avail_affected), 31
find_avail_noninform, 32
find_avail_noninform, Ped-method (find_avail_noninform), 32
find_avail_noninform, Pedigree-method (find_avail_noninform), 32
find_unavailable, 33

find_unavailable, Ped-method (find_unavailable), 33
find_unavailable, Pedigree-method (find_unavailable), 33
findsibs, 29
findspouse, 30
fix_parents, 34
fix_parents, character-method (fix_parents), 34
findsibs, 29
findspouse, 30
fix_parents, 34
fix_parents, character-method (fix_parents), 34

Hints, 15, 16
Hints (Hints-class), 42
find_unavailable, Ped-method (find_unavailable), 33
find_unavailable, Pedigree-method (find_unavailable), 33
findsibs, 29
findspouse, 30
fix_parents, 34
fix_parents, character-method (fix_parents), 34
findsibs, 29
findspouse, 30
fix_parents, 34
fix_parents, character-method (fix_parents), 34

Hints-class, 42
hints<-, (Pedigree-class), 70
Hints(), 73, 75
Hints, Hints, missing OR NULL-method (Hints-class), 42
Hints, list, missing OR NULL-method (Hints-class), 42
Hints, missing OR NULL, missing OR NULL-method (Hints-class), 42
Hints, numeric, data.frame-method (Hints-class), 42
Hints, numeric, missing OR NULL-method (Hints-class), 42
hints, Pedigree-method (Pedigree-class), 70
Hints-class, 42
horder (Hints-class), 42
INDEX

Ped, 62
Ped (Ped-class), 66
ped (Pedigree-class), 70
Ped(), 62, 71, 73, 75
Ped, character_OR_integer-method (Ped-class), 66
Ped, data.frame-method (Ped-class), 66
Ped, missing-method (Ped-class), 66
ped, Pedigree, ANY-method (Pedigree-class), 70
ped, Pedigree, missing-method (Pedigree-class), 70
Ped-class, 66
ped<-(Pedigree-class), 70
ped<-, Pedigree, ANY, ANY-method (Pedigree-class), 70
ped<-, Pedigree, missing, Ped-method (Pedigree-class), 70
ped_to_legdf, 75
ped_to_legdf(), 4, 78, 79
ped_to_legdf, Pedigree-method (ped_to_legdf), 75
ped_to_plotdf, 76
ped_to_plotdf(), 6, 79
ped_to_plotdf, Pedigree-method (ped_to_plotdf), 76
Pedigree (Pedigree-class), 70
Pedigree(), 4, 43, 62, 69, 75, 81, 86, 90, 94
Pedigree, character_OR_integer-method (Pedigree-class), 70
Pedigree, data.frame-method (Pedigree-class), 70
Pedigree, missing-method (Pedigree-class), 70
Pedigree-class, 70
Pedixplorer (Pedixplorer-package), 4
Pedixplorer-package, 4
permute, 78
plot(), 4
plot, Pedigree (plot, Pedigree, missing-method), 79
plot, Pedigree, missing-method, 79
plot, Pedigree (plot, Pedigree, missing-method), 79
plot_fromdf, 82
plot_fromdf(), 4, 75, 76, 78, 79
polyfun, 83
polyfun(), 84
polygons, 84
polygons(), 82
Rel (Rel-class), 84
rel (Pedigree-class), 70
Rel(), 63, 65, 72, 73, 75
Rel, character_OR_integer-method (Rel-class), 84
Rel, data.frame-method (Rel-class), 84
rel, Pedigree, ANY-method (Pedigree-class), 70
rel, Pedigree, missing-method (Pedigree-class), 70
Rel-class, 84
rel<-(Pedigree-class), 70
rel<-, Pedigree, ANY, ANY-method (Pedigree-class), 70
rel<-, Pedigree, missing, Rel-method (Pedigree-class), 70
rel_code_to_factor, 86
rel_code_to_factor(), 63–65, 72
sampleped, 87
sampleped(), 4
Scales (Scales-class), 88
scales (Pedigree-class), 70
Scales(), 73, 75
Scales, data.frame, data.frame-method (Scales-class), 88
Scales, missing, missing-method (Scales-class), 88
scales, Pedigree-method (Pedigree-class), 70
Scales-class, 88
scales<-(Pedigree-class), 70
scales<-, Pedigree, Scales-method (Pedigree-class), 70
set_plot_area, 90
sex (Ped-class), 66
sex, Ped-method (Ped-class), 66
sex<-(Ped-class), 66
sex<-, Ped, character_OR_integer-method (Ped-class), 66
sex_to_factor, 91
shift, 92
show, Ped-method (Ped-class), 66
show, Pedigree-method (Pedigree-class), 70
show, Rel-method (Rel-class), 84
shrink, 93
shrink(), 4, 18, 27, 28, 31–33
shrink, Ped-method (shrink), 93
shrink, Pedigree-method (shrink), 93
spouse (Hints-class), 42
spouse, Hints-method (Hints-class), 42
spouse, Pedigree-method (Pedigree-class), 70
spouse<-, Pedigree-method (Pedigree-class), 70
spouse<-, Hints, data.frame-method (Hints-class), 42
spouse<-, Pedigree, data.frame-method (Pedigree-class), 70
status (Ped-class), 66
status, Ped-method (Ped-class), 66
status<-, Ped-method (Ped-class), 66
status<-, Ped, numeric OR logical-method (Ped-class), 66
subregion, 94
subset, Hints-method (Hints-class), 42
subset, Ped-method (Ped-class), 66
subset, Pedigree-method (Pedigree-class), 70
subset, Rel-method (Rel-class), 84
summary, Ped-method (Ped-class), 66
summary, Pedigree-method (Pedigree-class), 70
summary, Rel-method (Rel-class), 84
unrelated, 95
unrelated, Ped-method (unrelated), 95
unrelated, Pedigree-method (unrelated), 95
upd_famid_id, 96
upd_famid_id(), 61, 64
upd_famid_id, character, ANY-method (upd_famid_id), 96
upd_famid_id, Ped, character OR integer-method (upd_famid_id), 96
upd_famid_id, Ped, missing-method (upd_famid_id), 96
upd_famid_id, Pedigree, character OR integer-method (upd_famid_id), 96
upd_famid_id, Pedigree, missing-method (upd_famid_id), 96
useful (Ped-class), 66
useful, Ped-method (Ped-class), 66
useful<-, Ped, numeric OR logical-method (Ped-class), 66
useful_inds, 97
useful_inds, character-method (useful_inds), 97
useful_inds, Ped-method (useful_inds), 97
useful_inds, Pedigree-method (useful_inds), 97
vect_to_binary, 99
vect_to_binary(), 37, 61, 73