Package 'nprcgenekeepr'

June 2, 2020

Type Package

Title Genetic Tools for Colony Management

Version 1.0.3

Description Provides genetic tools for colony management and is a derivation of the work in Amanda Vinson and Michael J Raboin (2015) <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4671785/> ``A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques ('Macaca' 'mulatto')''.
It provides a 'Shiny' application with an exposed API.
The application supports five groups of functions:
(1) Quality control of studbooks contained in text files or 'Excel' workbooks and of pedigrees within 'LabKey' Electronic Health Records (EHR);
(2) Creation of pedigrees from a list of animals using the 'LabKey' EHR integration;
(3) Creation and display of an age by sex pyramid plot of the living animals within the designated pedigree;
(4) Generation of genetic value analysis reports; and
(5) Creation of potential breeding groups with and without proscribed sex ratios and defined maximum kinships.

URL https://rmsharp.github.io/nprcgenekeepr/, https://github.com/rmsharp/nprcgenekeepr

BugReports https://github.com/rmsharp/nprcgenekeepr/issues

Depends R (>= 3.6.0)

Imports anytime, futile.logger, htmlTable, lubridate, Matrix, plotrix, readxl, Rlabkey, shiny, shinyBS, stringi, utils, WriteXLS

Suggests covr, dplyr, ggplot2, grid, kableExtra, knitr, pkgdown, png, rmarkdown, roxygen2 (>= 7.0.0), testthat

Language en-US

Encoding UTF-8

License MIT + file LICENSE

RoxygenNote 7.1.0

LazyData TRUE

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Michael Raboin [aut],
Terry Therneau [aut],
Amanda Vinson [aut, dtc],
R. Mark Sharp [aut, cre, cph, dtc]
(<https://orcid.org/0000-0002-6170-6942>),
Southwest National Primate Research Center NIH grant P51 RR13986 [fnd],
Oregon National Primate Research Center grant P51 OD011092 [fnd]

Maintainer R. Mark Sharp <rmsharp@me.com>

Repository CRAN

Date/Publication 2020-06-02 12:40:03 UTC

R topics documented:

`addAnimalsWithNoRelative` *Adds an NA value for all animals without a relative*

Description

This allows `kin` to be used with `setdiff` when there are no relatives otherwise an error would occur because `kin[['animal_with_no_relative']]` would not be found. See the following: in **groupAddAssign**

Usage

```
addAnimalsWithNoRelative(kin, candidates)
```

Arguments

`kin`	dataframe with kinship values
`candidates`	character vector of IDs of the animals available for use in the group.

Details

`available[[i]] <-setdiff(available[[i]],kin[[id]])`

Value

A dataframe with kinships in long form after adding a row for each animal without a relative.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree, minParentAge = 2, reportChanges = FALSE,
                  reportErrors = FALSE)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1)
currentGroups[[1]] <- examplePedigree$id[1:3]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
```

```
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
                                 ignore = list(c("F", "F")), minAge = 1)
# Filtering out candidates related to current group members
conflicts <- unique(c(unlist(kin[unlist(currentGroups)]),
                                 unlist(currentGroups)))
candidates <- setdiff(candidates, conflicts)
kin <- addAnimalsWithNoRelative(kin, candidates)
length(kin) # should be 2416
kin[["1SPLS8"]] # should have 14 IDs
```

---

addBackSecondParents *Add back single parents trimmed pedigree*

Description

Uses the ped dataframe, which has full complement of parents and the uPed dataframe, which has all uninformative parents removed to add back single parents to the uPed dataframe where one parent is known. The parents are added back to the pedigree as an ID record with NA for both sire and dam of the added back ID.

Usage

```
addBackSecondParents(uPed, ped)
```

Arguments

uPed	a trimmed pedigree dataframe with uninformative founders removed.
ped	a trimmed pedigree

Value

A dataframe with pedigree with single parents added.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
probands <- breederPed$id[!(is.na(breederPed$sire) &
                              is.na(breederPed$dam)) &
                              is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
```

```
p <- addBackSecondParents(p, ped)
nrow(p)
```

---

addErrTxt *Concatenates any errors from nprcgenekeeprErr into narrative form*

Description

Concatenates any errors from nprcgenekeeprErr into narrative form

Usage

```
addErrTxt(txt, err, singularTxt, pluralTxt)
```

Arguments

txt	character string with initial error description value
err	ve from errorLst
singularTxt	character string with text used when the length of err is 1
pluralTxt	character string with text used when the length of err is greater than 1.

Value

Error from nprcgenekeepr

---

addGenotype *Add genotype data to pedigree file*

Description

Assumes genotype has been opened by checkGenotypeFile

Usage

```
addGenotype(ped, genotype)
```

Arguments

ped	pedigree dataframe. ped is to be provided by qcStudbook so it is not checked.
genotype	genotype dataframe. genotype is to be provided by checkGenotypeFile so it is not checked.

Value

A pedigree object with genotype data added.

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(ped = rhesusPedigree,
                                genotype = rhesusGenotypes)
```

---

addGroupOfUnusedAnimals

addGroupOfUnusedAnimals adds a group to the saved groups if needed

---

Description

addGroupOfUnusedAnimals adds a group to the saved groups if needed

Usage

```
addGroupOfUnusedAnimals(savedGroupMembers, candidates, ped, minAge, harem)
```

Arguments

savedGroupMembers	list of groups of animals in the form of a vector of animal Ids.
candidates	character vector of IDs of the animals available for use in the group.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
harem	logical variable when set to TRUE, the formed groups have a single male at least minAge old.

Value

A list of groups, which are each lists of animal Ids that are unused animals at the end of the iteration.

addIdRecords *addIdRecords Adds Ego records added having NAs for parent IDs*

Description

addIdRecords Adds Ego records added having NAs for parent IDs

Usage

```
addIdRecords(ids, fullPed, partialPed)
```

Arguments

ids	character vector of IDs to be added as Ego records having NAs for parent IDs
fullPed	a trimmed pedigree
partialPed	a trimmed pedigree dataframe with uninformative founders removed.

Value

Pedigree with Ego records added having NAs for parent IDs

Examples

```
uPedOne <- data.frame(id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                      sire = c("s0", "s4", NA, "s1", "s1", "s2", "s2"),
                      dam = c("d0", "d4", NA, "d1", "d2", "d2", "d2"),
                      sex = c("F", "M", "F", "F", "F", "F", "M"),
                      stringsAsFactors = FALSE)
pedOne <- data.frame(id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                     sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
                     dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
                     sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
                     stringsAsFactors = FALSE)
pedOne[!pedOne$id %in% uPedOne$id, ]
newPed <- addIdRecords(ids = "s1", pedOne, uPedOne)
pedOne[!pedOne$id %in% newPed$id, ]
newPed[newPed$id == "s1", ]
```

addParents *Add parents*

Description

Pedigree curation function Given a pedigree, find any IDs listed in the "sire" or "dam" columns that lack their own line entry and generate one.

Usage

```
addParents(ped)
```

Arguments

ped	datatable that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.

Details

This must be run after to addUIds since the IDs made there are used by addParents

Value

An updated pedigree with entries added as necessary. Entries have the id and sex specified; all remaining columns are filled with NA.

Examples

```
pedTwo <- data.frame(id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                     sire = c(NA, NA, NA, "s1", "s1", "s2", "s2"),
                     dam = c(NA, NA, NA, "d1", "d2", "d2", "d2"),
                     sex = c("F", "M", "F", "F", "F", "F", "M"),
                     stringsAsFactors = FALSE)
newPed <- addParents(pedTwo)
newPed
```

addSexAndAgeToGroup *Forms a dataframe with Id, Sex, and current Age given a list of Ids and a pedigree*

Description

Forms a dataframe with Id, Sex, and current Age given a list of Ids and a pedigree

Usage

```
addSexAndAgeToGroup(ids, ped)
```

Arguments

ids	character vector of animal Ids
ped	datatable that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.

Value

Dataframe with Id, Sex, and Current Age

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("qcPed")
df <- addSexAndAgeToGroup(ids = qcBreeders, ped = qcPed)
head(df)
```

addUIds *Eliminates partial parentage situations by adding unique placeholder IDs for the unknown parent.*

Description

This must be run prior to addParents since the IDs made herein are used by addParents

Usage

```
addUIds(ped)
```

Arguments

ped	datatable that is the 'Pedigree'. It contains pedigree information. The fields sire and dam are required.

Value

The updated pedigree with partial parentage removed.

Examples

```
pedTwo <- data.frame(id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                     sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
                     dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
                     sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
                     stringsAsFactors = FALSE)
newPed <- addUIds(pedTwo)
newPed[newPed$id == "s1", ]
pedThree <-
  data.frame(id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
             sire = c("s0", "s0", "s4", NA, "s1", "s1", "s2", "s2"),
             dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
             sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
             stringsAsFactors = FALSE)
newPed <- addUIds(pedThree)
newPed[newPed$id == "s1", ]
```

---

agePyramidPlot *Form age pyramid plot*

Description

Form age pyramid plot

Usage

```
agePyramidPlot(
  males,
  females,
  ageLabels,
  mcol,
  fcol,
  laxlab,
  raxlab,
  gap,
  currentDate
)
```

Arguments

males	integer vector with the number of males in age groups corresponding to the position in the vector
females	integer vector with the number of females in age groups corresponding to the position in the vector
ageLabels	character vector of labels for the categories represented by each pair of bars. There should be a label for each lx or rx value, even if empty. If labels is a matrix or data frame, the first two columns will be used for the left and right category labels respectively.
mcol	color for the male (left) side of the plot
fcol	color for the female (right) side of the plot
laxlab	label for the male (left) side of the plot
raxlab	label for the female (right) side of the plot
gap	numeric value for one half of the space between the two sets of bars for the ageLabels in user units
currentDate	POSIXct date object indicating the date corresponding to the date the pedigree census occurred.

Value

The return value of par("mar") when the function was called.

---

alleleFreq *Calculates the count of each allele in the provided vector.*

Description

Part of Genetic Value Analysis

Usage

```
alleleFreq(alleles, ids = NULL)
```

Arguments

alleles	an integer vector of alleles in the population
ids	character vector of IDs indicating to which animal each allele in alleles belongs.

Details

If ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A data.frame with columns `allele` and `freq`. This is a table of allele counts within the population.

Examples

```
library(nprcgenekeepr)
data("ped1Alleles")
ids <- ped1Alleles$id
alleles <- ped1Alleles[, !(names(ped1Alleles) %in% c("id", "parent"))]
aF <- alleleFreq(alleles[[1]], ids = NULL)
aF[aF$freq >= 10, ]
```

---

allTrueNoNA *Returns* TRUE *if every member of the vector is* TRUE.

Description

Part of Relations

Usage

```
allTrueNoNA(v)
```

Arguments

v	logical vector

Details

Considers NA values the same as false

---

assignAlleles *Assign parent alleles randomly*

Description

Assign parent alleles randomly

Usage

```
assignAlleles(alleles, parentType, parent, id, n)
```

Arguments

alleles	a list with a list alleles$alleles, which is a list of list containing the alleles for each individual's sire and dam that have been assigned thus far and alleles$counter that is the counter used to track the lists ofalleles$alleles.
parentType	character vector of length one with value of "sire" or "dam".
parent	either ped[id,"sire"] or ped[id,"dam"].
id	character vector of length one containing the animal ID
n	integer indicating the number of iterations to simulate. Default is 5000.

Value

The original list alleles passed into the function with newly randomly assigned alleles to each id based on dam and sire genotypes.

Examples

```
alleles <- list(alleles = list(), counter = 1)
alleles <- assignAlleles(alleles, parentType = "sire", parent = NA,
                         id = "o1", n = 4)
alleles
alleles <- assignAlleles(alleles, parentType = "dam", parent = NA,
                         id = "o1", n = 4)
alleles
```

---

calcA — *Calculates* a*, the number of an individual's alleles that are rare in each simulation.*

Description

Part of Genetic Value Analysis

Usage

```
calcA(alleles, threshold = 1, byID = FALSE)
```

Arguments

alleles	a matrix with id, parent, V1 ... Vn providing the alleles an animal received during each simulation. The first 2 columns provide the animal ID and the parent the allele came from. Remaining columns provide alleles.
threshold	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.

byID	logical variable of length 1 that is passed through to eventually be used by alleleFreq(), which calculates the count of each allele in the provided vector. If byID is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A matrix with named rows indicating the number of unique alleles an animal had during each round of simulation (indicated in columns).

Examples

```
library(nprcgenekeepr)
rare <- calcA(nprcgenekeepr::ped1Alleles, threshold = 3, byID = FALSE)
```

---

calcAge *Calculate animal ages.*

Description

Part of Pedigree Curation

Usage

```
calcAge(birth, exit)
```

Arguments

birth	Date vector of birth dates
exit	Date vector of exit dates.

Details

Given vectors of birth and exit dates, calculate an individuals age. If no exit date is provided, the calculation is based on the current date.

Value

A numeric vector (NA allowed) indicating age in decimal years from "birth" to "exit" or the current date if "exit" is NA.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
originalAge <- qcPed$age ## ages calculated at time of data collection
currentAge <- calcAge(qcPed$birth, qcPed$exit) ## assumes no changes in
                                               ## colony
```

`calcFE` *Calculates founder Equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFE(ped)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population).

Details

It is assumed that the pedigree has no partial parentage

Value

The founder equivalents `FE = 1 / sum(p ^ 2)`, where `p` is average number of descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
id = c("A", "B", "C", "D", "E", "F", "G"),
sire = c(NA, NA, "A", "A", NA, "D", "D"),
dam = c(NA, NA, "B", "B", NA, "E", "E"),
stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
```

```
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(pedFactors$id, pedFactors$sire,
                                    pedFactors$dam)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
fe <- calcFE(ped)
feFactors <- calcFE(pedFactors)
```

`calcFEFG` *Calculates Founder Equivalents and Founder Genome Equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFEFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). It is assumed that the pedigree has no partial parentage
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The list containing the founder equivalents, `FE = 1 / sum(p ^ 2)`, and the founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r` where `p` is average number of descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

Examples

```
data(lacy1989Ped)
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
```

```
ped <- nprcgenekeepr::lacy1989Ped
alleles <- lacy1989PedAlleles
pedFactors <- data.frame(
  id = as.factor(ped$id),
  sire = as.factor(ped$sire),
  dam = as.factor(ped$dam),
  gen = ped$gen,
  population = ped$population,
  stringsAsFactors = TRUE
)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
                           pedFactors$gen, genotype = NULL, n = 5000,
                           updateProgress = NULL)
feFg <- calcFEFG(ped, alleles)
feFgFactors <- calcFEFG(pedFactors, allelesFactors)
```

`calcFG` *Calculates Founder Genome Equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). It is assumed that the pedigree has no partial parentage
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r` where `p` is average number of descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
```

```
ped <- data.frame(
id = c("A", "B", "C", "D", "E", "F", "G"),
sire = c(NA, NA, "A", "A", NA, "D", "D"),
dam = c(NA, NA, "B", "B", NA, "E", "E"),
stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(pedFactors$id, pedFactors$sire,
                                    pedFactors$dam)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
alleles <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen, genotype = NULL,
                    n = 5000, updateProgress = NULL)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
                           pedFactors$gen, genotype = NULL, n = 5000,
                           updateProgress = NULL)
fg <- calcFG(ped, alleles)
fgFactors <- calcFG(pedFactors, allelesFactors)
```

`calcGU` *Calculates genome uniqueness for each ID that is part of the population.*

Description

Genome Uniqueness Functions

Usage

```
calcGU(alleles, threshold = 1, byID = FALSE, pop = NULL)
```

Arguments

`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`. An AlleleTable contains information about alleles an ego has inherited. It contains the following columns: • id — A character vector of IDs for a set of animals. • parent — A factor with levels of sire and dam. • V1 — Unnamed integer column representing allele 1. • V2 — Unnamed integer column representing allele 2. • ... — Unnamed integer columns representing alleles.

- Vn — Unnamed integer column representing the nth column.

`threshold` an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.

`byID` logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

`pop` character vector with animal IDs to consider as the population of interest, otherwise all animals will be considered. The default is NULL.

Details

Part of Genetic Value Analysis

The following functions calculate genome uniqueness according to the equation described in Ballou & Lacy.

It should be noted, however that this function differs slightly in that it does not distinguish between founders and non-founders in calculating the statistic.

Ballou & Lacy describe genome uniqueness as "the proportion of simulations in which an individual receives the only copy of a founder allele." We have interpreted this as meaning that genome uniqueness should only be calculated for living, non-founder animals. Alleles possessed by living founders are not considered when calculating genome uniqueness.

We have a differing view on this, since a living founder can still contribute to the population. The function below calculates genome uniqueness for all living animals and considers all alleles. It does not ignore living founders and their alleles.

Our results for genome uniqueness will, therefore differ slightly from those returned by Pedscope. Pedscope calculates genome uniqueness only for non-founders and ignores the contribution of any founders in the population. This will cause Pedscope's genome uniqueness estimates to possibly be slightly higher for non-founders than what this function will calculate.

The estimates of genome uniqueness for founders within the population calculated by this function should match the "founder genome uniqueness" measure calculated by Pedscope.

Value

Dataframe `rows: id,col: gu` A single-column table of genome uniqueness values as percentages. Rownames are set to 'id' values that are part of the population.

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

Examples

```
library(nprcgenekeepr)
ped1Alleles <- nprcgenekeepr::ped1Alleles
```

```
gu_1 <- calcGU(ped1Alleles, threshold = 1, byID = FALSE, pop = NULL)
gu_2 <- calcGU(ped1Alleles, threshold = 3, byID = FALSE, pop = NULL)
gu_3 <- calcGU(ped1Alleles, threshold = 3, byID = FALSE,
               pop = ped1Alleles$id[20:60])
```

`calcRetention` *Calculates Allelic Retention*

Description

Part of Genetic Value Analysis

Usage

```
calcRetention(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). It is assumed that the pedigree has no partial parentage
`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

A vector of the mean number of founder alleles retained in the gene dropping simulation.

Examples

```
library(nprcgenekeepr)
data("lacy1989Ped")
data("lacy1989PedAlleles")
ped <- lacy1989Ped
alleles <- lacy1989PedAlleles
retention <- calcRetention(ped, alleles)
```

`calculateSexRatio` *Calculates the sex ratio (number of non-males / number of males) given animal Ids and their pedigree*

Description

The Males are counted when the `ped$sex` value is `"M"`. When females are counted when the `ped$sex` value is not `"M"`. This means animals with ambiguous sex are counted with the females.

Usage

```
calculateSexRatio(ids, ped, additionalMales = 0, additionalFemales = 0)
```

Arguments

`ids`	character vector of animal Ids
`ped`	datatable that is the 'Pedigree'. It contains pedigree information including the IDs listed in `candidates`.
`additionalMales`	Integer value of males to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.
`additionalFemales`	Integer value of females to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.

Value

Numeric value of sex ratio of the animals provided.

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("pedWithGenotype")
available <- c("JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "VFS0XB", "CQC133",
               "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
               "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL")
nonMales <- c("JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "CQC133",
              "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
              "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL")
male <- "VFS0XB"
calculateSexRatio(ids = male, ped = pedWithGenotype)
calculateSexRatio(ids = nonMales, ped = pedWithGenotype)
calculateSexRatio(ids = available, ped = pedWithGenotype)
calculateSexRatio(ids = available, ped = pedWithGenotype,
                  additionalMales = 1)
calculateSexRatio(ids = available, ped = pedWithGenotype,
```

```
                  additionalFemales = 1)
calculateSexRatio(ids = available, ped = pedWithGenotype,
                  additionalMales = 1, additionalFemales = 1)
calculateSexRatio(ids = nonMales, ped = pedWithGenotype,
                  additionalMales = 1, additionalFemales = 0)
calculateSexRatio(ids = character(0), ped = pedWithGenotype,
                 additionalMales = 1, additionalFemales = 0)
```

`checkChangedColAndErrorLst` *checkChangedColAndErrorLst examines errorLst for errors and errorLst$changeCols non-empty fields*

Description

checkChangedColAndErrorLst examines errorLst for errors and errorLst$changeCols non-empty fields

Usage

```
checkChangedColAndErrorLst(errorLst)
```

Arguments

`errorLst`	list with fields for each type of changed column and error detectable by `qcStudbook`.

Value

Returns `NULL` is all fields are empty else the entire list is returned.

`checkChangedColsLst` *checkChangedColsLst examines list for non-empty fields*

Description

checkChangedColsLst examines list for non-empty fields

Usage

```
checkChangedColsLst(changedCols)
```

Arguments

`changedCols`	list with fields for each type of column change `qcStudbook`.

Value

Returns NULL if all fields are empty else the entire list is returned.

Examples

```
library(nprcgenekeepr)
library(lubridate)
pedOne <- data.frame(ego_id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3",
                                "o4"),
                     `si re` = c(NA, NA, NA, NA, "s1", "s1", "s2", "s2"),
                     dam_id = c(NA, NA, NA, NA, "d1", "d2", "d2", "d2"),
                      sex = c("F", "M", "M", "F", "F", "F", "F", "M"),
                      birth_date = mdy(
                        paste0(sample(1:12, 8, replace = TRUE), "-",
                               sample(1:28, 8, replace = TRUE), "-",
                               sample(seq(0, 15, by = 3), 8, replace = TRUE) +
                                 2000)),
                      stringsAsFactors = FALSE, check.names = FALSE)

errorLst <- qcStudbook(pedOne, reportErrors = TRUE, reportChanges = TRUE)
checkChangedColsLst(errorLst$changedCols)
```

---

checkErrorLst *checkErrorLst examines list for non-empty fields*

Description

checkErrorLst examines list for non-empty fields

Usage

```
checkErrorLst(errorLst)
```

Arguments

errorLst	list with fields for each type of error detectable by qcStudbook.

Value

Returns NULL is all fields are empty else the entire list is returned.

Examples

```
errorLst <- qcStudbook(nprcgenekeepr::pedFemaleSireMaleDam,
                       reportErrors = TRUE)
checkErrorLst(errorLst)
```

---

checkGenotypeFile *Check genotype file*

Description

Checks to ensure the content and structure are appropriate for a genotype file. These checks are simply based on expected columns and legal domains.

Usage

```
checkGenotypeFile(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

A genotype file that has been checked to ensure the column types and number required are present. The returned genotype file has the first column name forced to "id".

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- ped[order(ped$id), ]
genotype <- data.frame(id = ped$id[50 + 1:20],
                       first_name = paste0("first_name", 1:20),
                       second_name = paste0("second_name", 1:20),
                       stringsAsFactors = FALSE)

## checkGenotypeFile disallows dataframe with < 3 columns
tryCatch({
  checkGenotypeFile(genotype[ , c("id", "first_name")])
  }, warning = function(w) {
    cat("Warning produced")
  }, error = function(e) {
    cat("Error produced")
})
```

checkParentAge *Check parent ages to be at least* minParentAge

Description

Ensure parents are sufficiently older than offspring

Usage

```
checkParentAge(sb, minParentAge = 2, reportErrors = FALSE)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information.
minParentAge	numeric values to set the minimum age in years for an animal to have an offspring. Defaults to 2 years. The check is not performed for animals with missing birth dates.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Value

A dataframe containing rows for each animal where one or more parent was less than minParentAge. It contains all of the columns in the original sb dataframe with the following added columns:

1. sireBirth sire's birth date
2. sireAge age of sire in years on the date indicated by birth.
3. damBirth dam's birth date damAge age of dam in years on the date indicated by birth.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
checkParentAge(qcPed, minParentAge = 2)
checkParentAge(qcPed, minParentAge = 3)
checkParentAge(qcPed, minParentAge = 5)
checkParentAge(qcPed, minParentAge = 6)
checkParentAge(qcPed, minParentAge = 10)
```

checkRequiredCols *Examines column names,* cols *for required column names*

Description

Examines column names, cols for required column names

Usage

```
checkRequiredCols(cols, reportErrors)
```

Arguments

cols	character vector of column names
reportErrors	logical value when TRUE and missing columns are found the errorLst object is updated with the names of the missing columns and returned and when FALSE and missing columns are found the program is stopped.

Value

NULL is returned if all required columns are present. See description of reportErrors for return values when required columns are missing.

Examples

```
library(nprcgenekeepr)
requiredCols <- getRequiredCols()
cols <-
  paste0("id,sire,siretype,dam,damtype,sex,numberofparentsknown,birth,",
         "arrivalatcenter,death,departure,status,ancestry,fromcenter?,",
         "origin")
all(requiredCols %in% checkRequiredCols(cols, reportErrors = TRUE))
```

chooseAlleles *Combines two vectors of alleles by randomly selecting one allele or the other at each position.*

Description

Combines two vectors of alleles by randomly selecting one allele or the other at each position.

Usage

```
chooseAlleles(a1, a2)
```

Arguments

a1	integer vector with first allele for each individual
a2	integer vector with second allele for each individual a1 and a2 are equal length vectors of alleles for one individual

Value

An integer vector with the result of sampling from a1 and a2 according to Mendelian inheritance.

Examples

```
chooseAlleles(0:4, 5:9)
```

---

chooseAllelesChar *Combines two vectors of alleles when alleles are character vectors.*

Description

Combines two vectors by randomly selecting one allele or the other at each position. Alleles may be of any class that does not require attributes as the vectors are combined with c().

Usage

```
chooseAllelesChar(a1, a2)
```

Arguments

a1	vector with first parent alleles for each individual
a2	vector with second parent alleles for each individual a1 and a2 are equal length vectors of alleles for one individual

Details

The current implementation is slower than the one using integer vectors (chooseAlleles).

Value

An integer vector with the result of sampling from a1 and a2 according to Mendelian inheritance.

chooseDate *Choose date based on* earlier *flag.*

Description

Part of Pedigree Curation

Usage

```
chooseDate(d1, d2, earlier = TRUE)
```

Arguments

d1	Date vector with the first of two dates to compare.
d2	Date vector with the second of two dates to compare.
earlier	logical variable with TRUE if the earlier of the two dates is to be returned, otherwise the later is returned. Default is TRUE.

Details

Given two dates, one is selected to be returned based on whether it occurred earlier or later than the other. NAs are ignored if possible.

Value

Date vector of chosen dates or NA where neither is provided

Examples

```
library(nprcgenekeepr)
someDates <- lubridate::mdy(paste0(sample(1:12, 2, replace = TRUE), "-",
                                   sample(1:28, 2, replace = TRUE), "-",
                                   sample(seq(0, 15, by = 3), 2,
                                          replace = TRUE) + 2000))
someDates
chooseDate(someDates[1], someDates[2], earlier = TRUE)
chooseDate(someDates[1], someDates[2], earlier = FALSE)
```

colChange *colChange internal function to describe column names transformation*

Description

colChange internal function to describe column names transformation

Usage

```
colChange(orgCols, cols)
```

Arguments

orgCols	character vector with column names to be transformed if needed.
cols	character vector with transformed column names

Value

Description of column name changes

convertAncestry *Converts the ancestry information to a standardized code*

Description

Part of Pedigree Curation

Usage

```
convertAncestry(ancestry)
```

Arguments

ancestry	character vector or NA with free-form text providing information about the geographic population of origin.

Value

A factor vector of standardized designators specifying if an animal is a Chinese rhesus, Indian rhesus, Chinese-Indian hybrid rhesus, or Japanese macaque. Levels: CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN.

Examples

```
original <- c("china", "india", "hybridized", NA, "human", "gorilla")
convertAncestry(original)
```

---

convertDate *Converts date columns formatted as characters to be of type datetime*

Description

Part of Pedigree Curation

Usage

```
convertDate(ped, time.origin = as.Date("1970-01-01"), reportErrors = FALSE)
```

Arguments

ped	a dataframe of pedigree information that may contain birth, death, departure, or exit dates. The fields are optional, but will be used if present.(optional fields: birth, death, departure, and exit).
time.origin	date object used by as.Date to set origin.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Value

A dataframe with an updated table with date columns converted from character data type to Date data type. Values that do not conform to the format

Examples

```
library(lubridate)
set_seed(10)
someBirthDates <- paste0(sample(seq(0, 15, by = 3), 10,
                                replace = TRUE) + 2000, "-",
                         sample(1:12, 10, replace = TRUE), "-",
                         sample(1:28, 10, replace = TRUE))
someBadBirthDates <- paste0(sample(1:12, 10, replace = TRUE), "-",
                            sample(1:28, 10, replace = TRUE), "-",
                            sample(seq(0, 15, by = 3), 10,
                                   replace = TRUE) + 2000)
someDeathDates <- sample(someBirthDates, length(someBirthDates),
                         replace = FALSE)
```

```
someDepartureDates <- sample(someBirthDates, length(someBirthDates),
                             replace = FALSE)
ped1 <- data.frame(birth = someBadBirthDates, death = someDeathDates,
                   departure = someDepartureDates)
someDates <- ymd(someBirthDates)
ped2 <- data.frame(birth = someDates, death = someDeathDates,
                   departure = someDepartureDates)
ped3 <- data.frame(birth = someBirthDates, death = someDeathDates,
                   departure = someDepartureDates)
someNADeathDates <- someDeathDates
someNADeathDates[c(1, 3, 5)] <- ""
someNABirthDates <- someDates
someNABirthDates[c(2, 4, 6)] <- NA
ped4 <- data.frame(birth = someNABirthDates, death = someNADeathDates,
                   departure = someDepartureDates)

## convertDate identifies bad dates
result = tryCatch({
  convertDate(ped1)
}, warning = function(w) {
  print("Warning in date")
}, error = function(e) {
  print("Error in date")
})

## convertDate with error flag returns error list and not an error
convertDate(ped1, reportErrors = TRUE)

## convertDate recognizes good dates
all(is.Date(convertDate(ped2)$birth))
all(is.Date(convertDate(ped3)$birth))

## convertDate handles NA and empty character string values correctly
convertDate(ped4)
```

`convertRelationships` *Converts pairwise kinship values to a relationship category descriptor.*

Description

Part of Relations

Usage

```
convertRelationships(kmat, ped, ids = NULL, updateProgress = NULL)
```

Arguments

kmat	a numeric matrix of pairwise kinship coefficients. Rows and columns should be named with IDs.
ped	the pedigree information in datatable format with required colnames id, sire, and dam.
ids	character vector of IDs or NULL to which the analysis should be restricted. If provided, only relationships between these IDs will be converted to relationships.
updateProgress	function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Value

A dataframe with columns id1, id2, kinship, relation. It is a long-form table of pairwise kinships, with relationship categories included for each pair.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
ids <- c("A", "B", "D", "E", "F", "G", "I", "J", "L", "M", "O", "P")
relIds <- convertRelationships(kmat, ped, ids)
rel <- convertRelationships(kmat, ped, updateProgress = function() {})
head(rel)
ped <- nprcgenekeepr::qcPped
bkmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
                 sparse = FALSE)
relBIds <- convertRelationships(bkmat, ped, c("4LFS70", "DD1U77"))
relBIds
```

---

convertSexCodes *Converts sex indicator for an individual to a standardized codes.*

Description

Part of Pedigree Curation

Usage

```
convertSexCodes(sex, ignoreHerm = TRUE)
```

Arguments

sex	factor with levels: "M", "F", "U". Sex specifier for an individual.
ignoreHerm	logical flag indicating if hermaphrodites should be treated as unknown sex ("U"), default is TRUE.

Details

Standard sex codes are

- F – replacing "FEMALE" or "2"
- M – replacing "MALE" or "1"
- H – replacing "HERMAPHRODITE" or "4", if ignore.herm == FALSE
- U – replacing "HERMAPHRODITE" or "4", if ignore.herm == TRUE
- U – replacing "UNKNOWN" or "3"

Value

A vector of factors representing standardized sex codes after transformation from non-standard codes.

Examples

```
library(nprcgenekeepr)
original <- c("m", "male", "1", "MALE", "M", "F", "f", "female",
              "FemAle", "U", "Unknown", "H", "hermaphrodite",
              "U", "Unknown", "3", "4")
sexCodes <- convertSexCodes(original)
sexCodes
```

---

convertStatusCodes *Converts status indicators to a Standardized code*

Description

Part of Pedigree Curation

Usage

```
convertStatusCodes(status)
```

Arguments

status	character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc.

Value

A factor vector of the standardized status codes with levels: ‘ALIVE‘, ‘DECEASED‘, ‘SHIPPED‘, and ‘UNKNOWN‘.

Examples

```
library(nprcgenekeepr)
original <- c("A", "alive", "Alive", "1", "S", "Sale", "sold", "shipped",
              "D", "d", "dead", "died", "deceased", "2",
              "shiped", "3", "U", "4", "unknown", NA,
              "Unknown", "H", "hermaphrodite", "U", "Unknown", "4")
convertStatusCodes(original)
```

---

correctParentSex *Sets sex for animals listed as either a sire or dam.*

Description

Part of Pedigree Curation

Usage

```
correctParentSex(id, sire, dam, sex, recordStatus, reportErrors = FALSE)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).
sex	factor with levels: "M", "F", "U". Sex specifier for an individual.
recordStatus	character vector with value of "added" or "original", which indicates whether an animal was added or an original animal.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Value

A factor with levels: "M", "F", "H", and "U" representing the sex codes for the ids provided

Examples

```
library(nprcgenekeepr)
pedOne <- data.frame(id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                     sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
                     dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
                     sex = c("F", "F", "M", "F", "F", "F", "F", "M"),
                     recordStatus = rep("original", 8),
                     stringsAsFactors = FALSE)
pedTwo <- data.frame(id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
                     sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
                     dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
                     sex = c("M", "M", "M", "F", "F", "F", "F", "M"),
                     recordStatus = rep("original", 8),
                     stringsAsFactors = FALSE)
pedOneCorrected <- pedOne
pedOneCorrected$sex <- correctParentSex(pedOne$id, pedOne$sire, pedOne$dam,
                                        pedOne$sex, pedOne$recordStatus)
pedOne[pedOne$sex != pedOneCorrected$sex, ]
pedOneCorrected[pedOne$sex != pedOneCorrected$sex, ]

pedTwoCorrected <- pedTwo
pedTwoCorrected$sex <- correctParentSex(pedTwo$id, pedTwo$sire, pedTwo$dam,
                                        pedTwo$sex, pedOne$recordStatus)
pedTwo[pedTwo$sex != pedTwoCorrected$sex, ]
pedTwoCorrected[pedTwo$sex != pedTwoCorrected$sex, ]
```

`countFirstOrder` *Count first-order relatives.*

Description

Part of Relations

Usage

```
countFirstOrder(ped, ids = NULL)
```

Arguments

`ped`	: 'Pedigree' Standardized pedigree information in a table.
`ids`	character vector of IDs or NULL These are the IDs to which the analysis should be restricted. First-order relationships will only be tallied for the listed IDs and will only consider relationships within the subset. If NULL, the analysis will include all IDs in the pedigree.

Details

Tallies the number of first-order relatives for each member of the provided pedigree. If 'ids' is provided, the analysis is restricted to only the specified subset.

Value

A dataframe with column `id`, `parents`, `offspring`, `siblings`, and `total`. A table of first-order relationship counts, broken down to indicate the number of parents, offspring, and siblings that are part of the subset under consideration.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ids <- c("B", "D", "E", "F", "G")
countIds <- countFirstOrder(ped, ids)
countIds
count <- countFirstOrder(ped, NULL)
count
```

`countLoops` *Count the number of loops in a pedigree tree.*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
countLoops(loops, ptree)
```

Arguments

`loops`	a named list of logical values where each named element is named with an `id` from `ptree`. The value of the list element is set to `TRUE` if the `id` has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.
`ptree`	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where `ped` is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Uses the `loops` data structure and the list of all ancestors for each individual to calculate the number of loops for each individual.

Value

A list indexed with each ID in the pedigree tree (`ptree`) containing the number of loops for each individual.

Examples

```
library(nprcgenekeepr)
exampleTree <- createPedTree(nprcgenekeepr::examplePedigree)
exampleLoops <- findLoops(exampleTree)
## You can count how many animals are in loops with the following code.
length(exampleLoops[exampleLoops == TRUE])
## You can count how many loops you have with the following code.
nLoops <- countLoops(exampleLoops, exampleTree)
sum(unlist(nLoops[nLoops > 0]))
## You can list the first 10 sets of ids, sires and dams in loops with
## the following line of code:
examplePedigree[exampleLoops == TRUE, c("id", "sire", "dam")][1:10, ]
```

---

createExampleFiles *Creates a folder with CSV files containing example pedigrees and ID lists used to demonstrate the package.*

Description

Creates a folder named `~/tmp/ExamplePedigrees` if it does not already exist. It then proceeds to write each example pedigree into a CSV file named based on the name of the example pedigree.

Usage

```
createExampleFiles()
```

Value

A vector of the names of the files written.

Examples

```
library(nprcgenekeepr)
files <- createExampleFiles()
```

---

createPedOne *createPedOne makes the pedOne data object*

Description

createPedOne makes the pedOne data object

Usage

```
createPedOne(savePed = TRUE)
```

Arguments

savePed	logical value if TRUE the pedigree is saved into the packages data directory

---

createPedSix *createPedSix makes the pedSix data object*

Description

createPedSix makes the pedSix data object

Usage

```
createPedSix(savePed = TRUE)
```

Arguments

savePed	logical value if TRUE the pedigree is saved into the packages data directory

`createPedTree` *Create a pedigree tree (PedTree).*

Description

The PedTree is a list containing sire and dam information for an individual.

Usage

```
createPedTree(ped)
```

Arguments

`ped`	dataframe of pedigree and demographic information potentially containing columns indicating the birth and death dates of an individual. The table may also contain dates of sale (departure). Optional columns are `birth`, `death`, `departure`.

Details

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

This function uses only `id`, `sire`, and `dam` columns.

Value

A list of named lists forming a pedigree tree (PedTree or ptree). Each sublist represents an ID in the pedigree and contains the sire ID and the dam ID as named elements.

Examples

```
library(nprcgenekeepr)
exampleTree <- createPedTree(nprcgenekeepr::examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

create_wkbk *Creates an Excel workbook with worksheets.*

Description

Creates an Excel workbook with worksheets.

Usage

```
create_wkbk(file, df_list, sheetnames, replace = FALSE)
```

Arguments

file	filename of workbook to be created
df_list	list of data frames to be added as worksheets to workbook
sheetnames	character vector of worksheet names
replace	Specifies if the file should be replaced if it already exist (default is FALSE).

Value

TRUE if the Excel file was successfully created. FALSE if any errors occurred.

Examples

```
library(nprcgenekeepr)

make_df_list <- function(size) {
  df_list <- list(size)
  if (size <= 0)
    return(df_list)
  for (i in seq_len(size)) {
    n <- sample(2:10, 2, replace = TRUE)
    df <- data.frame(matrix(data = rnorm(n[1] * n[2]), ncol = n[1]))
    df_list[[i]] <- df
  }
  names(df_list) <- paste0("A", seq_len(size))
  df_list
}
df_list <- make_df_list(3)
sheetnames <- names(df_list)
create_wkbk(file = file.path(tempdir(), "example_excel_wkbk.xlsx"),
            df_list = df_list,
            sheetnames = sheetnames, replace = FALSE)
```

dataframe2string *dataframe2string converts a data.frame object to a character vector*

Description

Adapted from print.data.frame

Usage

```
dataframe2string(object, ..., digits = NULL, row.names = TRUE)
```

Arguments

object	dataframe
...	optional arguments to print or plot methods.
digits	the minimum number of significant digits to be used: see print.default.
row.names	logical (or character vector), indicating whether (or what) row names should be printed.

Value

A character vector representation of the data.frame provided to the function.

Examples

```
library(nprcgenekeepr)
dataframe2string(nprcgenekeepr::pedOne)
```

exampleNprcgenekeeprConfig *exampleNprcgenekeeprConfig is a loadable version of the example configuration file* example_nprcgenekeepr_config

Description

It contains a working version of a **nprcgenekeepr** configuration file created the SNPRC. Users of LabKey's EHR can adapt it to their systems and put it in their home directory. Instructions are embedded as comments within the file.

Usage

```
exampleNprcgenekeeprConfig
```

Format

An object of class `character` of length 34.

Examples

```
library(nprcgenekeepr)
data("exampleNprcgenekeeprConfig")
head(exampleNprcgenekeeprConfig)
```

`examplePedigree` *examplePedigree is a pedigree object created by* `qcStudbook`

Description

Represents pedigree from *ExamplePedigree.csv*.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column.Unknown dams are indicated with `NA`

sex – factor with levels: "M", "F", "U". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – Date vector of birth dates

exit – Date vector of exit dates

age – numerical vector of age in years

ancestry – character vector or NA with free-form text providing information about the geographic population of origin.

origin – character vector or `NA` (optional) that indicates the name of the facility that the individual was imported from if other than local.

status – character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc. Transformed to factor levels: ALIVE, DECEASED, SHIPPED, UNKNOWN. Vector of standardized status codes with the possible values ALIVE, DECEASED, SHIPPED, or UNKNOWN

recordStats – character vector with value of `"added"` or `"original"`.

Usage

```
examplePedigree
```

Format

An object of class `data.frame` with 3694 rows and 12 columns.

Examples

```
library(nprcgenekeepr)
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`fillBins` *fillBins Fill bins represented by list of two lists* `males` *and* `females`.

Description

fillBins Fill bins represented by list of two lists `males` and `females`.

Usage

```
fillBins(ageDist, lowerAges, upperAges = NULL)
```

Arguments

`ageDist`	dataframe with `sex` and `age` columns
`lowerAges`	integer vector of lower age boundaries; must be the same length as `upperAges`
`upperAges`	integer vector of upper age boundaries; must be the same length as `lowerAges`

`fillGroupMembers` *Forms and fills list of animals groups based on provided constraints*

Description

Forms and fills list of animals groups based on provided constraints

Usage

```
fillGroupMembers(
  candidates,
  currentGroups,
  kin,
  ped,
  harem,
  minAge,
  numGp,
  sexRatio
)
```

Arguments

candidates	character vector of IDs of the animals available for use in the group.
currentGroups	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
kin	list of animals and those animals who are related above a threshold value.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
harem	logical variable when set to TRUE, the formed groups have a single male at least minAge old.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
sexRatio	numeric value indicating the ratio of females to males x (from 0.5 to 20 by increments of 0.5 within the accompanying Shiny application. A sex ratio of 0 ignores sex in making up groups.

Value

A list of animal groups and their member animals

---

fillGroupMembersWithSexRatio

Forms breeding group(s) with an effort to match a specified sex ratio

---

Description

The sex ratio is the ratio of females to males.

Usage

```
fillGroupMembersWithSexRatio(
  candidates,
  groupMembers,
  grpNum,
  kin,
  ped,
  minAge,
  numGp,
  sexRatio
)
```

Arguments

candidates	character vector of IDs of the animals available for use in the group.
groupMembers	list initialized and ready to receive groups with the desired sex ratios that are created within this function
grpNum	is a list numGp long with each member an integer vector of 1:numGp.
kin	list of animals and those animals who are related above a threshold value.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
sexRatio	numeric value indicating the ratio of females to males x from 0.5 to 20 by increments of 0.5.

---

filterAge *Removes kinship values where an animal is less than the minAge*

Description

Part of Group Formation

Usage

```
filterAge(kin, ped, minAge = 1)
```

Arguments

kin	a dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.
ped	dataframe of pedigree information including the IDs listed in "candidates".
minAge	numeric value representing minimum years of age of animals to retain.

`filterKinMatrix` *Filters a kinship matrix to include only the egos listed in 'ids'*

Description

Filters a kinship matrix to include only the egos listed in 'ids'

Usage

```
filterKinMatrix(ids, kmat)
```

Arguments

`ids`	character vector containing the IDs of interest. The kinship matrix should be reduced to only include these rows and columns.
`kmat`	a numeric matrix of pairwise kinship coefficients. Rows and columns should be named with IDs.

Value

A numeric matrix that is the reduced kinship matrix with named rows and columns (row and col names are 'ids').

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
                sparse = FALSE)
ids <- ped$id[c(189, 192, 194, 195)]
ncol(kmat)
nrow(kmat)
kmatFiltered <- filterKinMatrix(ids, kmat)
ncol(kmatFiltered)
nrow(kmatFiltered)
```

`filterPairs` *Filters kinship values from a long-format kinship table based on the sexes of the two animals involved.*

Description

Part of Group Formation

Usage

```
filterPairs(kin, ped, ignore = list(c("F", "F")))
```

Arguments

`kin`	a dataframe with columns `id1`, `id2`, and `kinship`. This is the kinship data reformatted from a matrix, to a long-format table.
`ped`	Dataframe of pedigree information including the IDs listed in `candidates`.
`ignore`	a list containing zero or more character vectors of length 2 indicating which sex pairs should be ignored with regard to kinship. Defaults to `list(c("F","F"))`.

Value

A dataframe representing a filtered long-format kinship table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, rm.dups = FALSE)
threshold <- 0.1
kin <- filterThreshold(kin, threshold = threshold)
ped$sex <- c("M", "F", "M", "M", "F", "F", "M")
kinNull <- filterPairs(kin, ped, ignore = NULL)
kinMM <- filterPairs(kin, ped, ignore = list(c("M", "M")))
ped
kin[kin$id1 == "C", ]
kinMM[kinMM$id1 == "C", ]
```

`filterReport` *Filters a genetic value report down to only the specified animals*

Description

Filters a genetic value report down to only the specified animals

Usage

```
filterReport(ids, rpt)
```

Arguments

`ids`	character vector of animal IDs
`rpt`	a dataframe with required colnames `id`, `gu`, `zScores`, `import`, `totalOffspring`, which is a data.frame of results from a genetic value analysis.

Value

A copy of report specific to the specified animals.

Examples

```
library(nprcgenekeepr)
rpt <- nprcgenekeepr::pedWithGenotypeReport$report
rpt1 <- filterReport(c("GHH9LB", "BD41WW"), rpt)
```

`filterThreshold` *Filters kinship to remove rows with kinship values less than the specified threshold*

Description

Part of Group Formation Filters kinship values less than the specified threshold from a long-format table of kinship values.

Usage

```
filterThreshold(kin, threshold = 0.015625)
```

Arguments

kin	a dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.
threshold	numeric value representing the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.

Value

The kinship matrix with all kinship relationships below the threshold value removed.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, rm.dups = FALSE)
kinFiltered_0.3 <- filterThreshold(kin, threshold = 0.3)
kinFiltered_0.1 <- filterThreshold(kin, threshold = 0.1)
```

---

finalRpt — *finalRpt is a list object created from the list object* rpt *prepared by* reportGV. *It is created inside* orderReport. *This version is at the state just prior to calling* rankSubjects *inside* orderReport.

Description

finalRpt is a list object created from the list object *rpt* prepared by reportGV. It is created inside orderReport. This version is at the state just prior to calling rankSubjects inside orderReport.

Usage

```
finalRpt
```

Format

An object of class list of length 3.

Examples

```
library(nprcgenekeepr)
data("finalRpt")
finalRpt <- rankSubjects(finalRpt)
```

`findGeneration` *Determines the generation number for each id.*

Description

This loops through the entire pedigree one generation at a time. It finds the zeroth generation during first loop. The first time through this loop no sire or dam is in parents. This means that the animals without a sire and without a dam are assigned to generation 0 and become the first parental generation. The second time through this loop finds all of the animals that do not have a sire or do not have a dam and at least one parent is in the vector of parents defined the first time through. The ids that were not assigned as parents in the previous loop are given the incremented generation number.

Subsequent trips in the loop repeat what was done the second time through until no further animals can be added to the `nextGen` vector.

This does not work if the pedigree does not have all parent IDs as ego IDs.

Usage

```
findGeneration(id, sire, dam)
```

Arguments

`id`	character vector with unique identifier for an individual
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).

Value

An integer vector indication the generation numbers for each id, starting at 0 for individuals lacking IDs for both parents.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped[ , c("id", "sire", "dam")]
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
ped
```

findLoops *Find loops in a pedigree tree*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
findLoops(ptree)
```

Arguments

ptree	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

A named list of logical values where each named element is named with an `id` from `ptree`. The value of the list element is set to `TRUE` if the `id` has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.

Examples

```
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

findOffspring *Finds the number of total offspring for each animal in the provided pedigree.*

Description

Part of Genetic Value Analysis

Usage

```
findOffspring(probands, ped)
```

Arguments

probands	character vector of egos for which offspring should be counted and returned.
ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). This requires complete pedigree information.

Value

A named vector containing the offspring counts for each animal in probands. Rownames are set to the IDs from probands.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
totalOffspring <- findOffspring(probands, ped)
```

---

findPedigreeNumber *Determines the generation number for each id.*

Description

One of Pedigree Curation functions

Usage

```
findPedigreeNumber(id, sire, dam)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).

Value

Integer vector indicating generation numbers for each id, starting at 0 for individuals lacking IDs for both parents.

Examples

```
library(nprcgenekeepr)
library(stringi)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- NULL
ped$population <- NULL
ped2 <- ped
ped2$id <- stri_c(ped$id, "2")
ped2$sire <- stri_c(ped$sire, "2")
ped2$dam <- stri_c(ped$dam, "2")
ped3 <- ped
ped3$id <- stri_c(ped$id, "3")
ped3$sire <- stri_c(ped$sire, "3")
ped3$dam <- stri_c(ped$dam, "3")
ped <- rbind(ped, ped2)
ped <- rbind(ped, ped3)
ped$pedigree <- findPedigreeNumber(ped$id, ped$sire, ped$dam)
ped
```

---

fixColumnNames *fixColumnNames changes original column names and into standardized names.*

Description

fixColumnNames changes original column names and into standardized names.

Usage

```
fixColumnNames(orgCols, errorLst)
```

Arguments

orgCols	character vector with ordered list of column names found in a pedigree file.
errorLst	list object with places to store the various column name changes.

Value

A list object with newColNames and errorLst with a record of all changes made.

Examples

```
library(nprcgenekeepr)
fixColumnNames(c("Sire_ID", "EGO", "DAM", "Id", "birth_date"),
               errorLst = getEmptyErrorLst())
```

`fixGenotypeCols` *Reformat names of observed genotype columns*

Description

This is not a good fix. A better solution is to avoid the problem. Currently qcStudbook() blindly changes all of the column names by removing the underscores.

Usage

```
fixGenotypeCols(ped)
```

Arguments

`ped`	the pedigree information in datatable format

`focalAnimals` *focalAnimals is a dataframe with one column (_id_) containing the of animal Ids from the __examplePedigree__ pedigree.*

Description

They can be used to illustrate the identification of a population of interest as is shown in the example below.

Usage

```
focalAnimals
```

Format

An object of class `data.frame` with 327 rows and 1 columns.

Examples

```
library(nprcgenekeepr)
data("focalAnimals")
data("examplePedigree")
any(names(examplePedigree) == "population")
nrow(examplePedigree)
examplePedigree <- setPopulation(ped = examplePedigree,
                                 ids = focalAnimals$id)
any(names(examplePedigree) == "population")
nrow(examplePedigree)
nrow(examplePedigree[examplePedigree$population, ])
```

---

geneDrop *Gene drop simulation based on the provided pedigree information*

Description

Part of Genetic Value Analysis

Usage

```
geneDrop(
  ids,
  sires,
  dams,
  gen,
  genotype = NULL,
  n = 5000,
  updateProgress = NULL
)
```

Arguments

ids	A character vector of IDs for a set of animals.
sires	A character vector with IDS of the sires for the set of animals. NA is used for missing sires.
dams	A character vector with IDS of the dams for the set of animals. NA is used for missing dams.
gen	An integer vector indicating the generation number for each animal.
genotype	A dataframe containing known genotypes. It has three columns: id, first, and second. The second and third columns contain the integers indicating the observed genotypes. The gene dropping method from *Pedigree analysis by computer simulation* by Jean W MacCluer, John L Vandeberg, and Oliver A Ryder (1986) <doi:10.1002/zoo.1430050209> is used in the genetic value calculations.

Currently there is no means of handling knowing only one haplotype. It will be easy to add another column to handle situations where only one allele is observed and it is not known to be homozygous or heterozygous. The new fourth column could have a frequency for homozygosity that could be used in the gene dropping algorithm.

The genotypes are using indirection (integer instead of character) to indicate the genes because the manipulation of character strings was found to take 20-35 times longer to perform.

Adding additional columns to genotype does not significantly affect the time require. Thus, it is convenient to add the corresponding haplotype names to the dataframe using `first_name` and `second_name`.

`n` integer indicating the number of iterations to simulate. Default is 5000.

`updateProgress` function or NULL. If this function is defined, it will be called during each iteration to update a `shiny::Progress` object.

Value

A data.frame `id,parent,V1 ... Vn` A data.frame providing the maternal and paternal alleles for an animal for each iteration. The first two columns provide the animal's ID and whether the allele came from the sire or dam. These are followed by `n` columns indicating the allele for that iteration.

Examples

```
## We usually defined `n` to be >= 5000
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
                       genotype = NULL, n = 50, updateProgress = NULL)
genotype <- data.frame(id = ped$id,
                       first_allele = c(NA, NA, "A001_B001", "A001_B002",
                                        NA, "A001_B002", "A001_B001"),
                       second_allele = c(NA, NA, "A010_B001", "A001_B001",
                                         NA, NA, NA),
                       stringsAsFactors = FALSE)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
                          genotype = pedGenotype,
                          n = 5, updateProgress = NULL)
```

`getAncestors` *Recursively create a character vector of ancestors for an individual ID.*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
getAncestors(id, ptree)
```

Arguments

id	character vector of length 1 having the ID of interest
ptree	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

A character vector of ancestors for an individual ID.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- qcStudbook(ped, minParentAge = 0)
pedTree <- createPedTree(ped)
pedLoops <- findLoops(pedTree)
ids <- names(pedTree)
allAncestors <- list()

for (i in seq_along(ids)) {
  id <- ids[[i]]
  anc <- getAncestors(id, pedTree)
  allAncestors[[id]] <- anc
}
head(allAncestors)
countOfAncestors <- unlist(lapply(allAncestors, length))
idsWithMostAncestors <-
  names(allAncestors)[countOfAncestors == max(countOfAncestors)]
allAncestors[idsWithMostAncestors]
```

getAnimalsWithHighKinship *Forms a list of animal Ids and animals related to them*

Description

Forms a list of animal Ids and animals related to them

Usage

```
getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups, ignore, minAge)
```

Arguments

kmat	numeric matrix of pairwise kinship values. Rows and columns are named with animal IDs.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
threshold	numeric value indicating the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.
currentGroups	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
ignore	list of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Default is to ignore all pairwise kinship between females.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.

Value

A list of named character vectors where each name is an animal Id and the character vectors are made up of animals sharing a kinship value greater than our equal to the threshold value.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree, minParentAge = 2, reportChanges = FALSE,
                  reportErrors = FALSE)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1)
currentGroups[[1]] <- examplePedigree$id[1:3]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
                                 ignore = list(c("F", "F")), minAge = 1)
```

```
length(kin) # should be 2412
kin[["1SPLS8"]] # should have 14 IDs
```

`getChangedColsTab` *getChangedColsTab skeleton of list of errors*

Description

getChangedColsTab skeleton of list of errors

Usage

```
getChangedColsTab(errorLst, pedigreeFileName)
```

Arguments

`errorLst`	list of errors and changes made by `qcStudbook`
`pedigreeFileName`	name of file provided by user on Input tab

Value

HTML formatted error list

`getConfigFileName` *getConfigFileName returns the configuration file name appropriate for the system.*

Description

getConfigFileName returns the configuration file name appropriate for the system.

Usage

```
getConfigFileName(sysInfo)
```

Arguments

`sysInfo`	object returned by Sys.info()

Value

Character vector with expected configuration file

Examples

```
library(nprcgenekeepr)
sysInfo <- Sys.info()
config <- getConfigFileName(sysInfo)
```

`getCurrentAge` *Age in years using the provided birthdate.*

Description

Assumes current date for calculating age.

Usage

```
getCurrentAge(birth)
```

Arguments

`birth`	birth date(s)

Value

Age in years using the provided birthdate.

Examples

```
library(nprcgenekeepr)
age <- getCurrentAge(birth = as.Date("06/02/2000", format = "%m/%d/%Y"))
```

`getDateColNames` *Vector of date column names*

Description

Vector of date column names

Usage

```
getDateColNames()
```

Value

Vector of column names in a standardized pedigree object that are dates.

`getDatedFilename` *Returns a character vector with an file name having the date prepended.*

Description

Returns a character vector with an file name having the date prepended.

Usage

```
getDatedFilename(filename)
```

Arguments

`filename`	character vector with name to use in file name

Value

A character string with a file name prepended with the date and time in YYYY-MM-DD_hh_mm_ss_basename format.

Examples

```
library(nprcgenekeepr)
getDatedFilename("testName")
```

`getDateErrorsAndConvertDatesInPed` *Converts columns of dates in text form to* `Date` *object columns*

Description

Finds date errors in columns defined in `convertDate` as dates and converts date strings to `Date` objects.

Usage

```
getDateErrorsAndConvertDatesInPed(sb, errorLst)
```

Arguments

`sb`	A dataframe containing a table of pedigree and demographic information.
`errorLst`	object with placeholders for error types found in a pedigree file by `qcStudbook` through the functions it calls.

Details

If there are no errors that prevent the calculation of exit dates, they are calculated and added to the pedigree otherwise the pedigree is not updated.

Value

A list with the pedigree, sb, and the `errorLst` with invalid date rows (`errorLst$invalidDateRows`)

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedInvalidDates
ped
errorLst <- getEmptyErrorLst()
colNamesAndErrors <- fixColumnNames(names(ped), errorLst)
names(ped) <- colNamesAndErrors$newColNames
pedAndErrors <- getDateErrorsAndConvertDatesInPed(ped, errorLst)
pedAndErrors$sb
pedAndErrors$errorLst
```

`getDemographics` *Get demographic data*

Description

This is a thin wrapper around `labkey.selectRows()`.

Usage

```
getDemographics(colSelect = NULL)
```

Arguments

`colSelect`	(optional) a vector of comma separated strings specifying which columns of a dataset or view to import

Value

A data.frame containing LabKey demographic data with the columns specified in the single parameter provided.

Examples

```
library(nprcgenekeepr)
siteInfo <- getSiteInfo()
colSet <- siteInfo$lkPedColumns
source <- " generated by getDemographics: "
pedSourceDf <- tryCatch(getDemographics(colSelect = colSet),
                        warning = function(wCond) {
                          cat(paste0("Warning", source, wCond),
                                     name = "nprcgenekeepr")
                          return(NULL)},
                        error = function(eCond) {
                          cat(paste0("Error", source, eCond),
                                     name = "nprcgenekeepr")
                          return(NULL)}
)
```

---

getEmptyErrorLst *Creates a empty errorLst object*

Description

Creates a empty errorLst object

Usage

```
getEmptyErrorLst()
```

Value

An errorLst object with placeholders for error types found in a pedigree file by qcStudbook.

Examples

```
library(nprcgenekeepr)
getEmptyErrorLst()
```

getErrorTab *getErrorTab skeleton of list of errors*

Description

getErrorTab skeleton of list of errors

Usage

```
getErrorTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

HTML formatted error list

getFocalAnimalPed *Get pedigree based on list of focal animals*

Description

Get pedigree based on list of focal animals

Usage

```
getFocalAnimalPed(fileName, sep = ",")
```

Arguments

fileName	character vector of temporary file path.
sep	column separator in CSV file

Value

A pedigree file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
siteInfo <- getSiteInfo()
source <- " generated by getFocalAnimalPed: "
tryCatch(getFocalAnimalPed(fileName = "breeding file.csv"),
                           warning = function(wCond) {
                             cat(paste0("Warning", source, wCond),
                                        name = "nprcgenekeepr")
                             return(NULL)},
                           error = function(eCond) {
                             cat(paste0("Error", source, eCond),
                                        name = "nprcgenekeepr")
                             return(NULL)}
)
```

`getGenoDefinedParentGenotypes`

Assigns parental genotype contributions to an IDs genotype by attributing alleles to sire or dam

---

Description

Assigns parental genotype contributions to an IDs genotype by attributing alleles to sire or dam

Usage

```
getGenoDefinedParentGenotypes(alleles, genotype, id, sire, dam, n)
```

Arguments

`alleles`	data.frame `id,parent,V1 ... Vn` A data.frame providing the maternal and paternal alleles for an animal for each iteration. The first two columns provide the animal's ID and whether the allele came from the sire or dam. These are followed by `n` columns indicating the allele for that iteration.
`genotype`	A dataframe containing known genotypes. It has three columns: `id`, `first`, and `second`. The second and third columns contain the integers indicating the observed genotypes.
`id`	A character vector of length one having the ID of interest
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).
`n`	integer indicating the number of iterations to simulate.

Value

data.frame `id,parent,V1 ... Vn` A data.frame providing the maternal and paternal alleles for an animal for each iteration. The first two columns provide the animal's ID and whether the allele came from the sire or dam. These are followed by `n` columns indicating the allele for that iteration.

This is not correct for situations where one haplotype is not known.

`getGenotypes` *Get genotypes from file*

Description

Get genotypes from file

Usage

```
getGenotypes(fileName, sep = ",")
```

Arguments

`fileName`	character vector of temporary file path.
`sep`	column separator in CSV file

Value

A genotype file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
pedCsv <- getGenotypes(fileName = system.file("testdata", "qcPed.csv",
                       package="nprcgenekeepr"))
```

getGVGenotype *Get Genetic Value Genotype data structure for reportGV function.*

Description

Extracts genotype data if available otherwise NULL is returned.

Usage

```
getGVGenotype(ped)
```

Arguments

ped	the pedigree information in datatable format

Value

A data.frame with the columns `id`, `first`, and `second` extracted from a pedigree object (a data.frame) containing genotypic data. If the pedigree object does not contain genotypic data the `NULL` is returned.

Examples

```
## We usually defined `n` to be >= 5000
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
                       genotype = NULL, n = 50, updateProgress = NULL)
genotype <- data.frame(id = ped$id,
                       first_allele = c(NA, NA, "A001_B001", "A001_B002",
                                        NA, "A001_B002", "A001_B001"),
                       second_allele = c(NA, NA, "A010_B001", "A001_B001",
                                         NA, NA, NA),
                       stringsAsFactors = FALSE)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
                          genotype = pedGenotype,
                          n = 5, updateProgress = NULL)
```

getGVPopulation *Get the population of interest for the Genetic Value analysis.*

Description

If user has limited the population of interest by defining pop, that information is incorporated via the ped$population column.

Usage

```
getGVPopulation(ped, pop)
```

Arguments

ped	the pedigree information in datatable format
pop	character vector with animal IDs to consider as the population of interest. The default is NULL.

Value

A logical vector corresponding to the IDs in the vector of animal IDs provided to the function in pop.

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
id = c("A", "B", "C", "D", "E", "F", "G"),
sire = c(NA, NA, "A", "A", NA, "D", "D"),
dam = c(NA, NA, "B", "B", NA, "E", "E"),
stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
```

getIdsWithOneParent *getIdsWithOneParent extracts IDs of animals pedigree without either a sire or a dam*

Description

getIdsWithOneParent extracts IDs of animals pedigree without either a sire or a dam

Usage

```
getIdsWithOneParent(uPed)
```

Arguments

uPed	a trimmed pedigree dataframe with uninformative founders removed.

Value

Character vector of all single parents

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
probands <- breederPed$id[!(is.na(breederPed$sire) &
                              is.na(breederPed$dam)) &
                              is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
```

getIncludeColumns *Get the superset of columns that can be in a pedigree file.*

Description

Part of Genetic Value Functions

Usage

```
getIncludeColumns()
```

Details

Replaces INCLUDE.COLUMNS data statement.

Value

Superset of columns that can be in a pedigree file.

Examples

```
getIncludeColumns()
```

`getIndianOriginStatus` *Get Indian-origin status of group*

Description

Get Indian-origin status of group

Usage

```
getIndianOriginStatus(origin)
```

Arguments

`origin`	character vector of the animal origins. This vector is to have already been filtered to remove animals that should not be included in the calculation.

Value

`ancestry` list of number of Chinese animals (`chinese`), number of hybrid (`hybrid`), number of borderline hybrid animals (`borderline`), number of Indian ancestry animals (`indian`), and the dashboard color (`color`) to be assigned based on the number of animals of each type counted.

getLkDirectAncestors *Get the direct ancestors of selected animals*

Description

Gets direct ancestors from labkey study schema and demographics table.

Usage

```
getLkDirectAncestors(ids)
```

Arguments

ids	character vector with Ids.

Value

data.frame with pedigree structure having all of the direct ancestors for the Ids provided.

Examples

```
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectAncestors(ids = focalAnimals))
```

getLkDirectRelatives *Get the direct ancestors of selected animals*

Description

Gets direct ancestors from labkey study schema and demographics table.

Usage

```
getLkDirectRelatives(ids, unrelatedParents = FALSE)
```

Arguments

ids	character vector with Ids.
unrelatedParents	logical vector when FALSE the unrelated parents of offspring do not get a record as an ego; when TRUE a place holder record where parent (sire, dam) IDs are set to NA.

Value

A data.frame with pedigree structure having all of the direct ancestors for the Ids provided.

Examples

```
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectRelatives(ids = focalAnimals))
```

`getLogo` *Get Logo file name*

Description

Get Logo file name

Usage

```
getLogo()
```

Value

A character vector of length one having the name of the logo file used in the `Input` tab. A warning is returned if the configuration file is not found.

Examples

```
result = tryCatch({
  getLogo()
}, warning = function(w) {
print(paste0("Warning in getLogo: ", w, ". File is to be ",
             suppressWarnings(getLogo())$file))
}, error = function(e) {
print(paste0("Error in in getLogo: ", e))
})
```

`getMaxAx` *Get the maximum of the absolute values of the negative (males) and positive (female) animal counts.*

Description

This is used to scale the pyramid plot symmetrically.

Usage

```
getMaxAx(bins, axModulus)
```

Arguments

`bins`	integer vector with numbers of individuals in each bin
`axModulus`	integer value used in the modulus function to determine the interval between possible maxAx values.

`getMinParentAge` *Get minimum parent age.*

Description

This can be set to anything greater than or equal to 0.

Usage

```
getMinParentAge(input)
```

Arguments

`input`	shiny's input

Details

Set to 0 if you do not want to enforce parents being sexually mature by age. Animals that do not have an age are ignored.

getOffspring *Get offspring to corresponding animal IDs provided*

Description

Get offspring to corresponding animal IDs provided

Usage

```
getOffspring(pedSourceDf, ids)
```

Arguments

pedSourceDf	dataframe with pedigree structure having at least the columns id, sire, and dam.
ids	character vector of animal IDs

Value

A character vector containing all of the ancestor IDs for all of the IDs provided in the second argument `ids`. All ancestors are combined and duplicates are removed.

Examples

```
library(nprcgenekeepr)

pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getOffspring(pedOne, c("s1", "d2"))
```

getParamDef *Get parameter definitions from tokens found in configuration file.*

Description

Get parameter definitions from tokens found in configuration file.

Usage

```
getParamDef(tokenList, param)
```

Arguments

tokenList	list of parameters and their definitions, which are character vectors
param	character vector representing the parameter being defined.

getParents *Get parents to corresponding animal IDs provided*

Description

Get parents to corresponding animal IDs provided

Usage

```
getParents(pedSourceDf, ids)
```

Arguments

pedSourceDf	dataframe with pedigree structure having at least the columns id, sire, and dam.
ids	character vector of animal IDs

Value

A character vector with the IDs of the parents of the provided ID list.

Examples

```
library(nprcgenekeepr)

pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getParents(pedOne, c("o1", "d4"))
```

getPedigree *Get pedigree from file*

Description

Get pedigree from file

Usage

```
getPedigree(fileName, sep = ",")
```

Arguments

fileName	character vector of temporary file path.
sep	column separator in CSV file

Value

A pedigree file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
ped <- getPedigree(fileName = system.file("testdata", "qcPed.csv",
                   package="nprcgenekeepr"))
```

`getPedMaxAge` *Get the maximum age of live animals in the pedigree.*

Description

Get the maximum age of live animals in the pedigree.

Usage

```
getPedMaxAge(ped)
```

Arguments

ped	dataframe with pedigree

Value

Numeric value representing the maximum age of animals in the pedigree.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree, minParentAge = 2,
                        reportChanges = FALSE,
                        reportErrors = FALSE)
getPedMaxAge(ped)
```

`getPossibleCols` *Get possible column names for a studbook.*

Description

Pedigree curation function

Usage

```
getPossibleCols()
```

Details

@return A character vector of the possible columns that can be in a studbook. The possible columns are as follows:

- id – character vector with unique identifier for an individual
- sire – character vector with unique identifier for an individual's father (NA if unknown).
- dam – character vector with unique identifier for an individual's mother (NA if unknown).
- sex – factor levels: "M", "F", "U" Sex specifier for an individual
- gen – integer vector with the generation number of the individual
- birth – Date or NA (optional) with the individual's birth date
- exit – Date or NA (optional) with the individual's exit date (death, or departure if applicable)
- ancestry – character vector or NA (optional) that indicates the geographic population to which the individual belongs.
- age – numeric or NA (optional) indicating the individual's current age or age at exit.
- population – an optional logical argument indicating whether or not the `id` is part of the extant population.
- origin – character vector or NA (optional) that indicates the name of the facility that the individual was imported from. NA indicates the individual was not imported.
- status – an optional factor indicating the status of an individual with levels ALIVE, DEAD, and SHIPPED.
- condition – character vector or NA (optional) that indicates the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive.
- spf – character vector or NA (optional) indicating the specific pathogen-free status of an individual.
- vasxOvx – character vector indicating the vasectomy/ovariectomy status of an animal where NA indicates an intact animal and all other values indicate surgical alteration.
- pedNum – integer vector indicating generation numbers for each id, starting at 0 for individuals lacking IDs for both parents.

Examples

```
library(nprcgenekeepr)
getPossibleCols()
```

`getPotentialSires` *Provides list of potential sires*

Description

Provides list of potential sires

Usage

```
getPotentialSires(ids, minAge = 1, ped)
```

Arguments

`ids`	character vector of IDs of the animals
`minAge`	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
`ped`	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in `candidates`.

Value

A character vector of potential sire Ids

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
getPotentialSires(ids, minAge = 1, ped)
```

getProbandPedigree *Gets pedigree to ancestors of provided group leaving uninformative ancestors.*

Description

Filters a pedigree down to only the ancestors of the provided group, removing unnecessary individuals from the studbook. This version builds the pedigree back in time starting from a group of probands. This will include all ancestors of the probands, even ones that might be uninformative.

Usage

```
getProbandPedigree(probands, ped)
```

Arguments

probands	a character vector with the list of animals whose ancestors should be included in the final pedigree.
ped	datatable that is the 'Pedigree'. It contains pedigree information. The fields `sire` and `dam` are required.

Value

A reduced pedigree.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
sires <- getPotentialSires(ids, minAge = 1, ped)
head(getProbandPedigree(probands = sires, ped = ped))
```

getProductionStatus *Get production status of group*

Description

Get production status of group

Usage

```
getProductionStatus(
  ped,
  minParentAge = 3,
  maxOffspringAge = NULL,
  housing = "shelter_pens",
  currentDate = Sys.Date()
)
```

Arguments

ped	Dataframe that is the 'Pedigree'. It contains pedigree information. The id, dam, sex and age (in years) columns are required.
minParentAge	Numeric values to set the minimum age in years for an animal to have an offspring. Defaults to 2 years. The check is not performed for animals with missing birth dates.
maxOffspringAge	Numeric values to set the maximum age in years for an animal to be counted as birth in calculation of production status ratio.
housing	character vector of length 1 having the housing type, which is either *"shelter_pens"* or *"corral"*.
currentDate	Date to be used for calculating age. Defaults to Sys.Date().

Details

Description of how Production and Production Status (color) is calculated.

1. The Production Status is calculated on September 09, 2019, Births = count of all animals in group born since January 1, 2017 through December 31, 2018, that lived at least 30 days.
2. Dams = count of all females in group that have a birth date on or prior to September 09, 2016.
3. Production = Births / Dams
4. Production Status (color)
   (a) Shelter and pens
      i. Production < 0.6; Red
      ii. Production >= 0.6 and Production <= 0.63; Yellow
      iii. Production > 0.63; Green
   (b) Corrals
      i. Production < 0.5; Red
      ii. Production >= 0.5 and Production <= 0.53; Yellow
      iii. Production > 0.53; Green

This code may need to be modified to allow the user to supply a list of IDs to include as group members. Currently each animal in the provided pedigree (ped) is considered to be a member of the group.

Value

`production` – Ratio of the number of births that live >30 days to the number of females >= 3 years of age.

---

getProportionLow *Get proportion of Low genetic value animals*

Description

Get proportion of Low genetic value animals

Usage

```
getProportionLow(geneticValues)
```

Arguments

`geneticValues`	character vector of the genetic values. This vector is to have already been filtered to remove animals that should not be included in the calculation.

Value

List of the proportion of Low genetic value animals and the dashboard color to be assigned base on that proportion.

---

getPyramidAgeDist *Get the age distribution for the pedigree*

Description

Forms a dataframe with columns `id`, `birth`, `sex`, and age for those animals with a status of `Alive` in the pedigree.

Usage

```
getPyramidAgeDist(ped = NULL)
```

Arguments

`ped`	dataframe with pedigree

Details

The lubridate package is used here because of the way the modern Gregorian calendar is constructed, there is no straightforward arithmetic method that produces a person's age, stated according to common usage — common usage meaning that a person's age should always be an integer that increases exactly on a birthday.

Value

A pedigree with `status` column added, which describes the animal as `ALIVE` or `DECEASED` and a `age` column added, which has the animal's age in years or `NA` if it cannot be calculated. The `exit` column values have been remapped to valid dates or `NA`.

Examples

```
library(nprcgenekeepr)
ped <- getPyramidAgeDist()
```

`getPyramidPlot` *Creates a pyramid plot of the pedigree provided.*

Description

The pedigree provided must have the following columns: `sex` and `age`. This needs to be augmented to allow pedigrees structures that are provided by the nprcgenekeepr package.

Usage

```
getPyramidPlot(ped = NULL)
```

Arguments

`ped`	dataframe with pedigree data.

Value

The return value of par("mar") when the function was called.

Examples

```
library(nprcgenekeepr)
data(qcPed)
getPyramidPlot(qcPed)
```

getRecordStatusIndex *Returns record numbers with selected* recordStatus.

Description

Returns record numbers with selected recordStatus.

Usage

```
getRecordStatusIndex(ped, status = "added")
```

Arguments

ped	pedigree dataframe
status	character vector with value of "added" or "original".

Value

An integer vector of records with recordStatus == status.

getRequiredCols *Get required column names for a studbook.*

Description

Pedigree curation function

Usage

```
getRequiredCols()
```

Value

A character vector of the required columns that can be in a studbook. The required columns are as follows:

- id – character vector with unique identifier for an individual
- sire – character vector with unique identifier for an individual's father (NA if unknown).
- dam – character vector with unique identifier for an individual's mother (NA if unknown).
- sex – factor levels: "M", "F", "U" Sex specifier for an individual
- birth – Date or NA (optional) with the individual's birth date

Examples

```
library(nprcgenekeepr)
getRequiredCols()
```

`getSexRatioWithAdditions` *getSexRatioWithAdditions returns the sex ratio of a group.*

Description

Adding males and females to the ratio calculation is possible, but the default behavior is to simply return the sex ratio of the group. This is a helper routine for the main one `calculateSexRatio`.

Usage

```
getSexRatioWithAdditions(ids, ped, additionalMales, additionalFemales)
```

Arguments

`ids`	character vector of animal Ids
`ped`	datatable that is the 'Pedigree'. It contains pedigree information including the IDs listed in `candidates`.
`additionalMales`	Integer value of males to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.
`additionalFemales`	Integer value of females to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.

`getSiteInfo` *Get site information*

Description

Get site information

Usage

```
getSiteInfo(expectConfigFile = TRUE)
```

Arguments

`expectConfigFile`	logical parameter when set to `FALSE`, no configuration is looked for. Default value is `TRUE`.

Value

A list of site specific information used by the application.

Currently this returns the following character strings in a named list.

1. centerOne of "SNPRC" or "ONPRC"
2. baseUrlIf `center` is "SNPRC", baseUrl is one of "https://boomer.txbiomed.local:8080/labkey" or "https://vger.txbiomed.local:8080/labkey". To allow testing, if `center` is "ONPRC" baseUrl is "https://boomer.txbiomed.local:8080/labkey".
3. schemaNameIf `center` is "SNPRC", schemaName is "study". If `center` is "ONPRC", schemaName is "study"
4. folderPath If `center` is "SNPRC", folderPath is "/SNPRC". If `center` is "ONPRC", folderPath is "/ONPRC"
5. queryNameis "demographics"

Examples

```
library(nprcgenekeepr)
getSiteInfo()
```

---

getTokenList *Gets tokens from character vector of lines*

Description

Copyright(c) 2017-2020 R. Mark Sharp

Usage

```
getTokenList(lines)
```

Arguments

`lines`	character vector with text from configuration file

Value

First right and left space trimmed token from first character vector element.

Examples

```
lines <- c("center = \"SNPRC\"",
           " baseUrl = \"https://boomer.txbiomed.local:8080/labkey\"",
           " schemaName = \"study\"", " folderPath = \"/SNPRC\"",
           " queryName = \"demographics\"",
           "lkPedColumns = (\"Id\", \"gender\", \"birth\", \"death\",",
           "              \"lastDayAtCenter\", \"dam\", \"sire\")",
           "mapPedColumns = (\"id\", \"sex\", \"birth\", \"death\", ",
           "  \"exit\", \"dam\", \"sire\")")
lkVec <- c("Id", "gender", "birth", "death",
           "lastDayAtCenter", "dam", "sire")
mapVec <- c("id", "sex", "birth", "death", "exit", "dam", "sire")
tokenList <- getTokenList(lines)
params <- tokenList$param
tokenVectors <- tokenList$tokenVec
```

`getVersion` *getVersion Get the version number of nprcgenekeepr*

Description

getVersion Get the version number of nprcgenekeepr

Usage

```
getVersion(date = TRUE)
```

Arguments

`date`	A logical value when TRUE (default) a date in YYYYMMDD format within parentheses is appended.

Value

Current Version

Examples

```
library(nprcgenekeepr)
getVersion()
```

`get_and_or_list` *Returns a one element character string with correct punctuation for a list made up of the elements of the character vector argument.*

Description

Returns a one element character string with correct punctuation for a list made up of the elements of the character vector argument.

Usage

```
get_and_or_list(c_vector, conjunction = "and")
```

Arguments

`c_vector`	Character vector containing the list of words to be put in a list.
`conjunction`	The conjunction to be used as the connector. This is usually 'and' or 'or' with 'and' being the default.

Value

A character vector of length one containing the a single correctly punctuated character string that list each element in the first arguments vector with commas between if there are more than two elements with the last two elements joined by the selected conjunction.

Examples

```
get_and_or_list(c("Bob", "John")) # "Bob and John"
get_and_or_list(c("Bob", "John"), "or") # "Bob or John"
get_and_or_list(c("Bob", "John", "Sam", "Bill"), "or")
# "Bob, John, Sam, or Bill"
```

`get_elapsed_time_str` *Returns the elapsed time since start_time.*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
get_elapsed_time_str(start_time)
```

Arguments

start_time	a POSIXct time object

Value

A character vector describing the passage of time in hours, minutes, and seconds.

Examples

```
start_time <- proc.time()
## do something
elapsed_time <- get_elapsed_time_str(start_time)
```

---

groupAddAssign *Add animals to an existing breeding group or forms groups:*

Description

groupAddAssign finds the largest group that can be formed by adding unrelated animals from a set of candidate IDs to an existing group, to a new group it has formed from a set of candidate IDs or if more than 1 group is desired, it finds the set of groups with the largest average size.

The function implements a maximal independent set (MIS) algorithm to find groups of unrelated animals. A set of animals may have many different MISs of varying sizes, and finding the largest would require traversing all possible combinations of animals. Since this could be very time consuming, this algorithm produces a random sample of the possible MISs, and selects from these. The size of the random sample is determined by the specified number of iterations.

Usage

```
groupAddAssign(
  candidates,
  currentGroups = list(character(0)),
  kmat,
  ped,
  threshold = 0.015625,
  ignore = list(c("F", "F")),
  minAge = 1,
  iter = 1000,
  numGp = 1,
  harem = FALSE,
  sexRatio = 0,
  withKin = FALSE,
  updateProgress = NULL
)
```

Arguments

candidates	Character vector of IDs of the animals available for use in forming the groups. The animals that may be present in currentGroups are not included within candidates.
currentGroups	List of character vectors of IDs of animals currently assigned to groups. Defaults to a list with character(0) in each sublist element (one for each group being formed) assuming no groups are prepopulated.
kmat	Numeric matrix of pairwise kinship values. Rows and columns are named with animal IDs.
ped	Dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
threshold	Numeric value indicating the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored. The default value is 0.015625.
ignore	List of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Default is to ignore all pairwise kinship between females.
minAge	Integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
iter	Integer indicating the number of times to perform the random group formation process. Default value is 1000 iterations.
numGp	Integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
harem	Logical variable when set to TRUE, the formed groups have a single male at least minAge old.
sexRatio	Numeric value indicating the ratio of females to males x from 0.5 to 20 by increments of 0.5.
withKin	Logical variable when set to TRUE, the kinship matrix for the group is returned along with the group and score. Defaults to not return the kinship matrix. This maintains compatibility with earlier versions.
updateProgress	Function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Details

Part of Group Formation

Value

A list with list items group, score and optionally groupKin. The list item group contains a list of the best group(s) produced during the simulation. The list item score provides the score associated with the group(s). The list item groupKin contains the subset of the kinship matrix that is specific for each group formed.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped, removeUninformative = FALSE,
                    addBackParents = FALSE)
geneticValue <- reportGV(ped, guIter = 50, # should be >= 1000
                         guThresh = 3,
                         byID = TRUE,
                         updateProgress = NULL)
trimmedGeneticValue <- reportGV(trimmedPed, guIter = 50, # should be >= 1000
                                guThresh = 3,
                                byID = TRUE,
                                updateProgress = NULL)
candidates <- trimmedPed$id[trimmedPed$birth < as.Date("2013-01-01") &
                              !is.na(trimmedPed$birth) &
                              is.na(trimmedPed$exit)]
haremGrp <- groupAddAssign(candidates = candidates,
                           kmat = trimmedGeneticValue[["kinship"]],
                           ped = trimmedPed,
                           iter = 10, # should be >= 1000
                           numGp = 6,
                           harem = TRUE)
haremGrp$group
sexRatioGrp <- groupAddAssign(candidates = candidates,
                              kmat = trimmedGeneticValue[["kinship"]],
                              ped = trimmedPed,
                              iter = 10, # should be >= 1000
                              numGp = 6,
                              sexRatio = 9)
sexRatioGrp$group
```

`groupMembersReturn` *Forms return list of groupAddAssign function*

Description

@return A list with members `savedGroupMembers`, `savedScore`, and if `withKin == TRUE` `groupKin` as well.

Usage

```
groupMembersReturn(savedGroupMembers, savedScore, withKin, kmat)
```

Arguments

savedGroupMembers	selected animal group
savedScore	score of selected group, which is the group having the largest minimum group size
withKin	logical variable indicating to return kinship coefficients when TRUE.
kmat	numeric matrix of pairwise kinship values. Rows and columns are named with animal IDs.

---

hasBothParents *hasBothParents checks to see if both parents are identified.*

Description

hasBothParents checks to see if both parents are identified.

Usage

```
hasBothParents(id, ped)
```

Arguments

id	character vector of IDs to examine for parents
ped	a pedigree

Value

TRUE if ID has both sire and dam identified in ped.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedOne
names(ped) <- c("id", "sire", "dam", "sex", "birth")
hasBothParents("o2", ped)
ped$sire[ped$id == "o2"] <- NA
hasBothParents("o2", ped)
```

hasGenotype *Check for genotype data in dataframe*

Description

Checks to ensure the content and structure are appropriate for genotype data are in the dataframe and ready for the geneDrop function by already being mapped to integers and placed in columns named first and second. These checks are simply based on expected columns and legal domains.

Usage

```
hasGenotype(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

A logical value representing whether or not the data.frame passed in contains genotypic data that can be used. Non-standard column names are accepted for this assessment.

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(ped = rhesusPedigree,
                                genotype = rhesusGenotypes)
hasGenotype(pedWithGenotypes)
```

headerDisplayNames *Convert internal column names to display or header names.*

Description

Converts the column names of a Pedigree or Genetic value Report to something more descriptive.

Usage

```
headerDisplayNames(headers)
```

Arguments

headers	a character vector of column (header) names

Value

Updated list of column names

Examples

```
library(nprcgenekeepr)
headerDisplayNames(headers = c("id", "sire", "dam", "sex", "birth", "age"))
```

`initializeHaremGroups` *Make the initial groupMembers animal list*

Description

Make the initial groupMembers animal list

Usage

```
initializeHaremGroups(numGp, currentGroups, candidates, ped, minAge)
```

Arguments

`numGp`	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
`currentGroups`	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
`candidates`	character vector of IDs of the animals available for use in the group.
`ped`	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in `candidates`.
`minAge`	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.

Value

Initial groupMembers list

insertChangedColsTab *insertChangedColsTab insert a list of changed columns found by* qcStudbook *in the pedigree file*

Description

insertChangedColsTab insert a list of changed columns found by qcStudbook in the pedigree file

Usage

```
insertChangedColsTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

Text of the error list formatted as an HTML page

insertErrorTab *insertErrorTab insert a list of errors found by* qcStudbook *in the pedigree file*

Description

insertErrorTab insert a list of errors found by qcStudbook in the pedigree file

Usage

```
insertErrorTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

Text of the error list formatted as an HTML page

insertSeparators *insertSeparators inserts the character "-" between year and month and between month and day portions of a date string in %Y%m%d format.*

Description

This function is not exported because it is not general purpose and is missing several defensive programming measures.

Usage

```
insertSeparators(dates)
```

Arguments

dates	character vector of potential dates

Value

A character vector of potential dates in %Y-%m-%d format.

isEmpty *Is vector empty or all NA values.*

Description

Is vector empty or all NA values.

Usage

```
isEmpty(x)
```

Arguments

x	vector of any type.

Value

TRUE if x is a zero-length vector else FALSE.

is_valid_date_str *Returns TRUE if the string is a valid date.*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
is_valid_date_str(
  date_str,
  format = "%d-%m-%Y %H:%M:%S",
  optional = FALSE
)
```

Arguments

date_str	character vector with 0 or more dates
format	character vector of length one having the date format
optional	parameter to as.Date. Logical value indicating to return NA (instead of signaling an error) if the format guessing does not succeed. Defaults to FALSE.

Value

A logical value or NA indicating whether or not the provided character vector represented a valid date string.

Examples

```
is_valid_date_str(c("13-21-1995", "20-13-98", "5-28-1014",
  "1-21-15", "2-13-2098", "25-28-2014"), format = "%m-%d-%y")
```

kinMatrix2LongForm *Reformats a kinship matrix into a long-format table.*

Description

Part of Group Formation

Usage

```
kinMatrix2LongForm(kinMatrix, rm.dups = FALSE)
```

Arguments

kinMatrix	numerical matrix of pairwise kinship values. The row and column names correspond to animal IDs.
rm.dups	logical value indication whether or not reverse-order ID pairs be filtered out? (i.e., "ID1 ID2 kin_val" and "ID2 ID1 kin_val" will be collapsed into a single entry if rm.dups = TRUE)

Value

A dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
reformattedKmat <- kinMatrix2LongForm(kmat, rm.dups = FALSE)
nrow(reformattedKmat)
reformattedNoDupsKmat <- kinMatrix2LongForm(kmat, rm.dups = TRUE)
nrow(reformattedNoDupsKmat)
```

---

kinship *Generates a kinship matrix.*

Description

Kinship Matrix Functions The code for the kinship function was written by Terry Therneau at the Mayo clinic and taken from his website. This function is part of a package written in S (and later ported to R) for calculating kinship and other statistics.

Usage

```
kinship(id, father.id, mother.id, pdepth, sparse = FALSE)
```

Arguments

id	character vector of IDs for a set of animals.
father.id	character vector or NA for the IDs of the sires for the set of animals.
mother.id	character vector or NA for the IDs of the dams for the set of animals.
pdepth	integer vector indicating the generation number for each animal.
sparse	logical flag. If TRUE, Matrix::Diagnol() is used to make a unit diagonal matrix. If FALSE, base::diag() is used to make a unit square matrix.

Details

The function previously had an internal call to the kindepth function in order to provide the parameter pdepth (the generation number). This version requires the generation number to be calculated elsewhere and passed into the function.

The rows (cols) of founders are just .5 * identity matrix, no further processing is needed for them. Parents must be processed before their children, and then a child's kinship is just a sum of the kinship's for his/her parents.

Value

A kinship square matrix

Author(s)

Terry Therneau, original version

as modified by, M Raboin, 2014-09-08 14:44:26

References

Main website http://www.mayo.edu/research/faculty/therneau-terry-m-ph-d/bio-00025991

S-Plus/R Function Page *www.mayo.edu/research/departments-divisions/department-health-sciences-research/division-biomedical-statistics-informatics/software/* @description s-plus-r-functions Downloaded 2014-08-26 This page address is now (2019-10-03) stale.

All of the code on the S-Plus page was stated to be released under the GNU General Public License (version 2 or later).

The R version became the kinship2 package available on CRAN:

https://cran.r-project.org/package=kinship2

$Id: kinship.s,v 1.5 2003/01/04 19:07:53 therneau Exp $

Create the kinship matrix, using the algorithm of K Lange, Mathematical and Statistical Methods for Genetic Analysis, Springer, 1997, p 71-72.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
ped
kmat
```

lacy1989Ped *lacy1989Ped small hypothetical pedigree*

Description

lacy1989Ped small hypothetical pedigree

Usage

```
lacy1989Ped
```

Format

An object of class `data.frame` with 7 rows and 5 columns.

Source

lacy1989Ped is a dataframe containing the small hypothetical pedigree of three founders and four descendants used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

The founders (`A`, `B`, `E`) have unknown parentages and are assumed to have independent ancestries.

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam the female parent of the animal indicated by the `id` column.Unknown dams are indicated with `NA`

gen generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

population logical vector with all values set TRUE

lacy1989PedAlleles *lacy1989PedAlleles is a dataframe produced by* `geneDrop` *on* `lacy1989Ped` *with 5000 iterations.*

Description

lacy1989PedAlleles is a dataframe produced by `geneDrop` on `lacy1989Ped` with 5000 iterations.

Usage

```
lacy1989PedAlleles
```

Format

An object of class `data.frame` with 14 rows and 5002 columns.

Source

lacy1989Ped is a dataframe containing the small example pedigree used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

There are 5000 columns, one for each iteration in `geneDrop` containing alleles randomly selected at each generation of the pedigree using Mendelian rules.

Column 5001 is the `id` column with two rows for each member of the pedigree (2 * 7).

Column 5002 is the `parent` column with values of `sire` and `dam` alternating.

---

makeAvailable *Convenience function to make the initial available animal list*

Description

Convenience function to make the initial available animal list

Usage

```
makeAvailable(candidates, numGp)
```

Arguments

`candidates`	character vector of IDs of the animals available for use in the group.
`numGp`	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial available animals list

makeCEPH *Make a CEPH-style pedigree for each id*

Description

Part of Relations

Usage

```
makeCEPH(id, sire, dam)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).

Details

Creates a CEPH-style pedigree for each id, consisting of three generations: the id, the parents, and the grandparents. Inserts NA for unknown pedigree members.

Calculates the first-order relationships in a pedigree, and to convert pairwise kinships to the appropriate relationship category. Relationships categories: For each ID in the pair, find a CEPH-style pedigree and compare them

- If one is the parent of the other — Designate the relationship as parent-offspring
- Else if both parents are shared — Designate the relationship as full-siblings
- Else if one parent is shared — Designate the relationship as half-siblings
- Else if one is the grandparent of the other — Designate the relationship as grandparent-grandchild
- Else if both grand parents are shared — Designate the relationship as cousin
- Else if at least one grand parent is shared — Designate the relationship as cousin -other
- Else if the parents of one are the grandparents of the other — Designate the relationship as full-avuncular
- Else if a single parent of one is the grandparent of the other — Designate the relationship as avuncular -other
- Else if the kinship is greater than 0, but the pair don't fall into the above categories — Designate the relationship as other
- Else — Designate the relationships as no relation.

Value

List of lists: {fields: id, {subfields: parents, pgp, mgp}}. Pedigree information converted into a CEPH-style list. The top level list elements are the IDs from id. Below each ID is a list of three elements: parents (sire, dam), paternal grandparents (pgp: sire, dam), and maternal grandparents (mgp: sire, dam).

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
pedCEPH <- makeCEPH(ped$id, ped$sire, ped$dam)
head(ped)
head(pedCEPH$F)
```

`makeExamplePedigreeFile` *Write copy of nprcgenekeepr::examplePedigree into a file*

Description

Uses `examplePedigree` data structure to create an example data file

Usage

```
makeExamplePedigreeFile(
  file = file.path(tempdir(), "examplePedigree.csv"),
  fileType = "csv"
)
```

Arguments

`file`	character vector of length one providing the file name
`fileType`	character vector of length one with possible values of `"txt"`, `"csv"`, or `"xlsx"`. Default value is `"csv"`.

Value

Full path name of file saved.

Examples

```
library(nprcgenekeepr)
pedigreeFile <- makeExamplePedigreeFile()
```

makeGroupMembers *Convenience function to make the initial groupMembers animal list*

Description

Convenience function to make the initial groupMembers animal list

Usage

```
makeGroupMembers(numGp, currentGroups, candidates, ped, harem, minAge)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
currentGroups	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
candidates	character vector of IDs of the animals available for use in the group.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.
harem	logical variable when set to TRUE, the formed groups have a single male at least minAge old.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.

Value

Initial groupMembers list

makeGrpNum *Convenience function to make the initial grpNum list*

Description

Convenience function to make the initial grpNum list

Usage

```
makeGrpNum(numGp)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial grpNum list

`makeRelationClassesTable` *Make relation classes table from* `kin` *dataframe.*

Description

From Relations

Usage

```
makeRelationClassesTable(kin)
```

Arguments

`kin`	a dataframe with columns `id1`, `id2`, `kinship`, and `relation`. It is a long-form table of pairwise kinships, with relationship categories included for each pair.

Value

A data.frame with the number of instances of following relationship classes: Parent-Offspring, Full-Siblings, Half-Siblings, Grandparent-Grandchild, Full-Cousins, Cousin - Other, Full-Avuncular, Avuncular - Other, Other, and No Relation.

Examples

```
library(nprcgenekeepr)
suppressMessages(library(dplyr))

qcPed <- nprcgenekeepr::qcPed
bkmat <- kinship(qcPed$id, qcPed$sire, qcPed$dam, qcPed$gen,
                 sparse = FALSE)
kin <- convertRelationships(bkmat, qcPed)
relClasses <- makeRelationClassesTable(kin)
relClasses$`Relationship Class` <-
   as.character(relClasses$`Relationship Class`)
relClassTbl <- kin[!kin$relation == "Self", ] %>%
  group_by(relation) %>%
  summarise(count = n())
relClassTbl
```

makeRoundUp *Round up the provided integer vector* int *according to the* modulus.

Description

Round up the provided integer vector int according to the modulus.

Usage

```
makeRoundUp(int, modulus)
```

Arguments

int	integer vector
modulus	integer value to use as the divisor.

makesLoop makesLoop *tests for a common ancestor.*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
makesLoop(id, ptree)
```

Arguments

id	character vector of length 1 having the ID of interest
ptree	a list of lists forming a pedigree tree as constructed by createPedTree(ped) where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

TRUE if there is one or more common ancestors for the sire and dam.

Tests to see if sires and dams for an individual in a ptree have a common ancestor.

mapIdsToObfuscated *Map IDs to Obfuscated IDs*

Description

This is not robust as it fails if all IDs are found not within map.

Usage

```
mapIdsToObfuscated(ids, map)
```

Arguments

ids	character vector with original IDs
map	named character vector where the values are the obfuscated IDs and the vector of names (names(map)) is the vector of original names.

Value

A dataframe or vector with original IDs replaced by their obfuscated counterparts.

Examples

```
set_seed(1)
ped <- qcStudbook(nprcgenekeepr::pedSix)
obfuscated <- obfuscatePed(ped, map = TRUE)
someIds <- c("s1", "s2", "d1", "d1")
mapIdsToObfuscated(someIds, obfuscated$map)
```

meanKinship *Calculates the mean kinship for each animal in a kinship matrix*

Description

Part of Genetic Value Analysis

Usage

```
meanKinship(kmat)
```

Arguments

kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

Details

The mean kinship of animal *i* is

$$MK_i = \Sigma f_i j / N$$

, in which the summation is over all animals, *j*, including the kinship of animal *i* to itself.

Value

A named numeric vector of average kinship coefficients for each animal ID. Elements are named with the IDs from the columns of kmat.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
head(meanKinship(kmat))
```

`nprcgenekeepr` *Genetic Management Functions*

Description

Primary Data Structure — Pedigree

Contains studbook information for a number of individuals. ASSUME: All IDs listed in the sire or dam columns must have a row entry in the id column

See Also

`getIncludeColumns` to get set of columns that can be used in a pedigree file

A Pedigree is a data frame within the `R` environment with the following possible columns:

- id – character vector with unique identifier for an individual
- sire – character vector with unique identifier for an individual's father (`NA` if unknown).
- dam – character vector with unique identifier for an individual's mother (`NA` if unknown).
- sex – factor levels: "M", "F", "U" Sex specifier for an individual
- gen – integer vector with the generation number of the individual
- birth – Date or `NA` (optional) with the individual's birth date
- exit – Date or `NA` (optional) with the individual's exit date (death, or departure if applicable)
- ancestry – character vector or `NA` (optional) that indicates the geographic population to which the individual belongs.
- age – numeric or `NA` (optional) indicating the individual's current age or age at exit.

- population – logical (optional) Is the id part of the extant population?
- origin – character vector or NA (optional) that indicates the name of the facility that the individual was imported from if other than local.NA indicates the individual was not imported.

Pedigree File Testing Functions

- qcStudbook — Main pedigree curation function that performs basic quality control on pedigree information
- fixColumnNames — Changes original column names and into standardized names.
- checkRequiredCols — Examines column names, cols, to see if all required column names are present.
- correctParentSex — Sets sex for animals listed as either a sire or dam.
- getDateErrorsAndConvertDatesInPed — Converts columns of dates in text form to Date object columns
- checkParentAge — Check parent ages to be at least minParentAge
- removeDuplicates — Remove duplicate records from pedigree

Gene Dropping Function

- geneDrop — Performs a gene drop simulation based on the provided pedigree information

Genetic Value Analysis Functions

Contains functions to calculate the kinship coefficient and genome uniqueness for animals listed in a Pedigree table.

- meanKinship — Calculates the mean kinship for each animal in a kinship matrix
- calcA — Calculates a, the number of an individual's alleles that are rare in each simulation.
- alleleFreq — Calculates the count of each allele in the provided vector.
- calcFE — Calculates founder equivalents.
- calcFG — Calculates founder genome equivalents.
- calcFEFG— Returns founder equivalents FE and FG as elements in a list.
- calcGU — Calculates genome uniqueness for each ID that is part of the population.
- geneDrop — Performs a gene drop simulation based on the pedigree information.
- chooseAlleles — Combines two vectors of alleles by randomly selecting one allele or the other at each position.
- calcRetention — Calculates allelic retention.
- filterKinMatrix — Filters a kinship matrix to include only the egos listed in 'ids'
- kinship — Generates a kinship matrix
- reportGV — Generates a genetic value report for a provided pedigree.

Plotting Functions

- meanKinship — Calculates the mean kinship for each animal in a kinship matrix

Breeding Group Formation Functions

- meanKinship — Calculates the mean kinship for each animal in a kinship matrix

obfuscateDate *obfucateDate adds a random number of days bounded by plus and minus max delta*

Description

Get the base_date add a random number of days taken from a uniform distribution bounded by -max_delta and max_delta. Insure the resulting date is as least as large as the min_date.

Usage

```
obfuscateDate(baseDate, maxDelta = 30, minDate)
```

Arguments

baseDate	list of Date objects with dates to be obfuscated
maxDelta	integer vector that is used to create min and max arguments to runif (runif(n,min = 0,max = 1))
minDate	list object of Date objects that has the lower bound of resulting obfuscated dates

Value

A vector of dates that have be obfuscated.

Examples

```
library(nprcgenekeepr)
someDates <- rep(as.Date(c("2009-2-16", "2016-2-16"), format = "%Y-%m-%d"),
                 10)
minBirthDate <- rep(as.Date("2009-2-16", format = "%Y-%m-%d"), 20)
obfuscateDate(someDates, 30, minBirthDate)
```

obfuscateId *obfucateId creates a vector of ID aliases of specified length*

Description

ID aliases are pseudorandom sequences of alphanumeric upper case characters where the letter "O" is not included for readability.. User has the option of providing a character vector of aliases to avoid using.

Usage

```
obfuscateId(id, size = 10, existingIds = character(0))
```

Arguments

id	character vector of IDs to be obfuscated (alias creation).
size	character length of each alias
existingIds	character vector of existing aliases to avoid duplication.

Value

A named character vector of aliases where the name is the original ID value.

Examples

```
library(nprcgenekeepr)
integerIds <- 1:10
obfuscateId(integerIds, size = 4)
characterIds <- paste0(paste0(sample(LETTERS, 1, replace = FALSE)), 1:10)
obfuscateId(characterIds, size = 4)
```

---

obfuscatePed *obfuscatePed takes a pedigree object and creates aliases for all IDs and adjusts all date within a specified amount.*

Description

User provides a pedigree object (ped), the number of characters to be used for alias IDs (size), and the maximum number of days that the birthdate can be shifted (maxDelta).

Usage

```
obfuscatePed(
  ped,
  size = 6,
  maxDelta = 30,
  existingIds = character(0),
  map = FALSE
)
```

Arguments

ped	pedigree object
size	integer value indicating number of characters in alias IDs
maxDelta	integer value indicating maximum number of days that the birthdate can be shifted
existingIds	character vector of existing aliases to avoid duplication.
map	logical if TRUE a list object is returned with the new pedigree and a named character vector with the names being the original IDs and the values being the new alias values. Defaults to FALSE.

Value

An obfuscated pedigree

Examples

```
library(nprcgenekeepr)
ped <- qcStudbook(nprcgenekeepr::pedGood)
obfuscatedPed <- obfuscatePed(ped)
ped
obfuscatedPed
```

---

offspringCounts *Finds the total number of offspring for each animal in the pedigree*

Description

Optionally find the number that are part of the population of interest.

Usage

```
offspringCounts(probands, ped, considerPop = FALSE)
```

Arguments

probands	character vector of egos for which offspring should be counted.
ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). This is the complete pedigree.
considerPop	logical value indication whether or not the number of offspring that are part of the focal population are to be counted? Default is FALSE.

Value

A dataframe with at least id and totalOffspring required and livingOffspring optional.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
```

```
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
counts <- offspringCounts(probands, ped)
```

`orderReport` *Order the results of the genetic value analysis for use in a report.*

Description

Part of Genetic Value Analysis

Usage

```
orderReport(rpt, ped)
```

Arguments

`rpt`	a dataframe with required colnames `id`, `gu`, `zScores`, `import`, `totalOffspring`, which is a data.frame of results from a genetic value analysis.
`ped`	the pedigree information in datatable format with required colnames `id`, `sire`, `dam`, `gen`, `population`). This requires complete pedigree information..

Details

Takes in the results from a genetic value analysis and orders the report according to the ranking scheme we have developed.

Value

A dataframe, which is `rpt` sorted according to the ranking scheme:

- imported animals with no offspring
- animals with genome uniqueness above 10
- animals with mean kinship less than 0.25, ranked by ascending mk
- all remaining animals, ranked by ascending mk

ped1Alleles *ped1Alleles is a dataframe created by the geneDrop function*

Description

ped1Alleles is a dataframe created by the geneDrop function

Usage

```
ped1Alleles
```

Format

A dataframe with 554 rows and 6 variables

V1 alleles assigned to the parents of the animals identified in the `id` column during iteration 1 of gene dropping performed by `geneDrop`.

V2 alleles assigned to the parents of the animals identified in the `id` column during iteration 1 of gene dropping performed by `geneDrop`.

V3 alleles assigned to the parents of the animals identified in the `id` column during iteration 1 of gene dropping performed by `geneDrop`.

V4 alleles assigned to the parents of the animals identified in the `id` column during iteration 1 of gene dropping performed by `geneDrop`.

id character vector of animal IDs provided to the gene dropping function `geneDrop`.

parent the parent type ("sire" or "dam") of the parent who supplied the alleles as assigned during each of the 4 gene dropping iterations performed by `geneDrop`.

Source

example baboon pedigree file provided by Deborah Newman, Southwest National Primate Center.

pedDuplicateIds *pedDuplicateIds is a dataframe with 9 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with a duplicated record.*

Description

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedgood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedDuplicateIds
```

Format

An object of class `data.frame` with 9 rows and 5 columns.

`pedFemaleSireMaleDam` *pedFemaleSireMaleDam is a dataframe with 8 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with the errors of having a sire labeled as female and a dam labeled as male.*

Description

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedgood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedFemaleSireMaleDam
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

`pedGood` *pedGood is a dataframe with 8 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with no errors.*

Description

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedgood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedGood
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

pedInvalidDates	*pedInvalidDates is a dataframe with 8 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with values in the* birth_date *column that are not valid dates.*

Description

It is one of six pedigrees (pedDuplicateIds, pedFemaleSireMaleDam, pedgood, pedInvalidDates, pedMissingBirth, pedSameMaleIsSireAndDam) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedInvalidDates
```

Format

An object of class data.frame with 8 rows and 5 columns.

pedMissingBirth	*pedMissingBirth is a dataframe with 8 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with no errors.*

Description

It is one of six pedigrees (pedDuplicateIds, pedFemaleSireMaleDam, pedgood, pedInvalidDates, pedMissingBirth, pedSameMaleIsSireAndDam) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedMissingBirth
```

Format

An object of class data.frame with 8 rows and 4 columns.

pedOne *pedOne is a loadable version of a pedigree file fragment used for testing and demonstration*

Description

This is used for testing and demonstration.

Usage

```
pedOne
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Examples

```
library(nprcgenekeepr)
data("pedOne")
head(pedOne)
```

pedSameMaleIsSireAndDam *pedSameMaleIsSireAndDam is a dataframe with 8 rows and 5 columns (ego_id, sire, dam_id, sex, birth_date) representing a full pedigree with no errors.*

Description

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedgood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

Usage

```
pedSameMaleIsSireAndDam
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

pedSix — *pedSix is a loadable version of a pedigree file fragment used for testing and demonstration*

Description

This is used for testing and demonstration.

Usage

```
pedSix
```

Format

An object of class `data.frame` with 8 rows and 7 columns.

Examples

```
library(nprcgenekeepr)
data("pedSix")
head(pedSix)
```

pedWithGenotype — *pedWithGenotype is a dataframe produced from qcPed by adding made up genotypes.*

Description

A dataframe containing 280 records with 12 columns: `id`, `sire`, `dam`, `sex`, `gen`, `birth`, `exit`, `age`, `first`, `second`, `first_name`, and `second_name`.

Usage

```
pedWithGenotype
```

Format

An object of class `data.frame` with 280 rows and 12 columns.

pedWithGenotypeReport *pedWithGenotypeReport is a list containing the output of* reportGV.

Description

pedWithGenotypeReport is a list containing the output of reportGV.

Usage

```
pedWithGenotypeReport
```

Format

An object of class list (inherits from GVnprcmanag) of length 8.

Source

pedWithGenotypeReport was made with pedWithGenotype as input into reportGV with 10,000 iterations.

pedWithGenotypeReport is a simple example report for use in examples and unit tests. It was created using the following commands.

- set_seed(10)
- pedWithGenotypeReport <- reportGV(nprcgenekeepr::pedWithGenotype, guIter = 10000)
- save(pedWithGenotypeReport, file = "data/pedWithGenotypeReport.RData")

Examples

```
pedWithGenotypeReport <- nprcgenekeepr::pedWithGenotypeReport
```

print.summary.nprcgenekeeprErr *print.summary.nprcgenekeepr print.summary.nprcgenekeeprGV*

Description

print.summary.nprcgenekeepr print.summary.nprcgenekeeprGV

Usage

```
## S3 method for class 'summary.nprcgenekeeprErr'
print(x, ...)

## S3 method for class 'summary.nprcgenekeeprGV'
print(x, ...)
```

Arguments

x	object of class summary.nprcgenekeeprErr and class list
...	additional arguments for the summary.default statement

Value

An object to send to the generic print function

object to send to generic print function

Examples

```
library(nprcgenekeepr)
errorLst <- qcStudbook(nprcgenekeepr::pedInvalidDates,
                       reportChanges = TRUE, reportErrors = TRUE)
summary(errorLst)



library(nprcgenekeepr)
ped <- nprcgenekeepr::pedGood
ped <- suppressWarnings(qcStudbook(ped, reportErrors = FALSE))
summary(reportGV(ped, guIter = 10))
```

---

qcBreeders *qcBreeders is a list of 29 baboon IDs that are potential breeders*

Description

qcBreeders is a list of 29 baboon IDs that are potential breeders

Usage

```
qcBreeders
```

Format

An object of class character of length 29.

Source

qcBreeders is a list of 3 males and 26 females from the qcPed data set.

These 29 animal IDs are used for examples and unit tests. They were initially selected for having low kinship coefficients.

qcPed *qcPed is a dataframe with 277 rows and 6 columns*

Description

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column.

dam the female parent of the animal indicated by the `id` column.

sex sex of the animal indicated by the `id` column.

gen generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth birth date in `Date` format of the animal indicated by the `id` column.

exit exit date in `Date` format of the animal indicated by the `id` column.

age age in year (numeric) of the animal indicated by the `id` column.

Usage

```
qcPed
```

Format

An object of class `data.frame` with 280 rows and 8 columns.

qcPedGvReport *qcPedGvReport is a genetic value report*

Description

qcPedGvReport is a genetic value report for illustrative purposes only. It is used in examples and unit tests with the nprcgenekeepr package. It was created using the following commands.

- set_seed(10)
- qcPedGvReport <- reportGV(nprcgenekeepr::qcPed, guIter = 10000)
- save(qcPedGvReport, file = "data/qcPedGvReport.RData")

Usage

```
qcPedGvReport
```

Format

An object of class `list` (inherits from `GVnprcmanag`) of length 8.

Examples

```
qcPedGvReport <- nprcgenekeepr::qcPedGvReport
```

---

qcStudbook *Quality Control for the Studbook or pedigree*

Description

Main pedigree curation function that performs basic quality control on pedigree information

Usage

```
qcStudbook(sb, minParentAge = 2, reportChanges = FALSE, reportErrors = FALSE)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information. The function recognizes the following columns (optional columns will be used if present, but are not required): • id — Character vector with Unique identifier for all individuals • sire — Character vector with unique identifier for the father of the current id • dam — Character vector with unique identifier for the mother of the current id • sex — Factor levels: "M", "F", "U" Sex specifier for an individual • birth — Date or NA (optional) with the individual's birth date • departure — Date or NA (optional) an individual was sold or shipped from the colony • death — date or NA (optional) Date of death, if applicable • status — Factor levels: ALIVE, DEAD, SHIPPED (optional) Status of an individual • origin — Character or NA (optional) Facility an individual originated from, if other than ONPRC • ancestry — Character or NA (optional) Geographic population to which the individual belongs • spf — Character or NA (optional) Specific pathogen-free status of an individual • vasxOvx — Character or NA (optional) Indicator of the vasectomy/ovariectomy status of an animal; NA if animal is intact, assume all other values indicate surgical alteration • condition — Character or NA (optional) Indicator of the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive.

`minParentAge` numeric values to set the minimum age in years for an animal to have an offspring. Defaults to 2 years. The check is not performed for animals with missing birth dates.

`reportChanges` logical value that if `TRUE`, the `errorLst` contains the list of changes made to the column names. Default is `FALSE`.

`reportErrors` logical value if `TRUE` will scan the entire file and report back changes made to input and errors in a list of list where each sublist is a type of change or error found. Changes will include column names, case of categorical values (male, female, unknown), etc. Errors will include missing columns, invalid date rows, male dams, female sires, and records with one or more parents below minimum age of parents.

The following changes are made to the cols.

- Column cols are converted to all lower case
- Periods (".") within column cols are collapsed to no space ""
- `egoid` is converted to `id`
- `sireid` is convert to `sire`
- `damid` is converted to `dam`

If the dataframe (`sb` does not contain the five required columns (`id`, `sire`, `dam`, `sex`), and `birth` the function throws an error by calling `stop()`.

If the `id` field has the string *UNKNOWN* (any case) or both the fields `sire` or `dam` have `NA` or *UNKNOWN* (any case), the record is removed. If either of the fields `sire` or `dam` have the string *UNKNOWN* (any case), they are replaced with a unique identifier with the form `Unnnn`, where `nnnn` represents one of a series of sequential integers representing the number of missing sires and dams right justified in a pattern of `0000`. See `addUIds` function.

The function `addParents` is used to add records for parents missing their own record in the pedigree.

The function `convertSexCodes` is used with `ignoreHerm == TRUE` to convert sex codes according to the following factors of standardized codes:

- F – replacing "FEMALE" or "2"
- M – replacing "MALE" or "1"
- H – replacing "HERMAPHRODITE" or "4", if ignore.herm == FALSE
- U – replacing "HERMAPHRODITE" or "4", if ignore.herm == TRUE
- U – replacing "UNKNOWN" or "3"

The function `correctParentSex` is used to ensure no parent is both a sire and a dam. If this error is detected, the function throws an error and halts the program.

The function `convertStatusCodes` converts status indicators to the following factors of standardized codes. Case of the original status value is ignored.

- "ALIVE" — replacing "alive", "A" and "1"
- "DECEASED" — replacing "deceased", "DEAD", "D", "2"
- "SHIPPED" — replacing "shipped", "sold", "sale", "s", "3"
- "UNKNOWN" — replacing is.na(status)
- "UNKNOWN" — replacing "unknown", "U", "4"

The function `convertAncestry` coverts ancestry indicators using regular expressions such that the following conversions are made from character strings that match selected substrings to the following factors.

- "INDIAN" — replacing "ind" and not "chin"
- "CHINESE" — replacing "chin" and not "ind"
- "HYBRID" — replacing "hyb" or "chin" and "ind"
- "JAPANESE" — replacing "jap"
- "UNKNOWN" — replacing `NA`
- "OTHER" — replacing not matching any of the above

The function `convertDate` converts character representations of dates in the columns `birth`, `death`, `departure`, and `exit` to dates using the `as.Date` function.

The function `setExit` uses heuristics and the columns `death` and `departure` to set `exit` if it is not already defined.

The function `calcAge` uses the `birth` and the `exit` columns to define the `age` column. The numerical values is rounded to the nearest 0.1 of a year. If `exit` is not defined, the current system date (`Sys.Date()`) is used.

The function `findGeneration` is used to define the generation number for each animal in the pedigree.

The function `removeDuplicates` checks for any duplicated records and removes the duplicates. I also throws an error and stops the program if an ID appears in more than one record where one or more of the other columns have a difference.

Columns that cannot be used subsequently are removed and the rows are ordered by generation number and then ID.

Finally the columns `id` `sire`, and `dam` are coerce to character.

Value

A data.frame with standardized and quality controlled pedigree information.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree, minParentAge = 2, reportChanges = FALSE,
                  reportErrors = FALSE)
names(ped)
```

rankSubjects *Ranks animals based on genetic value.*

Description

Part of Genetic Value Analysis Adds a column to rpt containing integers from 1 to nrow, and provides a value designation for each animal of "high value" or "low value"

Usage

```
rankSubjects(rpt)
```

Arguments

rpt	a list of data.frame req. colnames: value containing genetic value data for the population. Dataframes separate out those animals that are imports, those that have high genome uniqueness (gu > 10 have low mean kinship (mk < 0.25), and the remainder.

Value

A list of dataframes with value and ranking information added.

Examples

```
library(nprcgenekeepr)
finalRpt <- nprcgenekeepr::finalRpt
rpt <- rankSubjects(nprcgenekeepr::finalRpt)
rpt[["highGu"]][1, "value"]
rpt[["highGu"]][1, "rank"]
rpt[["lowMk"]][1, "value"]
rpt[["lowMk"]][1, "rank"]
rpt[["lowVal"]][1, "value"]
rpt[["lowVal"]][1, "rank"]
```

rbindFill *Append the rows of one dataframe to another.*

Description

Part of Pedigree Curation

Usage

```
rbindFill(df1, df2)
```

Arguments

df1	the target dataframe to append to.
df2	the the donor dataframe information should be appended from

Details

Appends the rows of df2 to df1, can handle cases where df2 has a subset of the columns of df1

Value

The appended dataframe with NA inserted into columns as needed.

---

readExcelPOSIXToCharacter

Read in Excel file and convert POSIX dates to character

Description

Read in Excel file and convert POSIX dates to character

Usage

```
readExcelPOSIXToCharacter(fileName)
```

Arguments

fileName	character vector of temporary file path.

Value

A pedigree file compatible with others in this package.

`removeDuplicates` *Remove duplicate records from pedigree*

Description

Part of Pedigree Curation

Usage

```
removeDuplicates(ped, reportErrors = FALSE)
```

Arguments

`ped`	dataframe that is the 'Pedigree'. It contains pedigree information. The `id` column is required.
`reportErrors`	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Details

Returns an updated dataframe with duplicate rows removed.

Returns an error if the table has duplicate IDs with differing data.

Value

Pedigree object with all duplicates removed.

Examples

```
ped <- nprcgenekeepr::smallPed
newPed <- cbind(ped, recordStatus = rep("original", nrow(ped)))
ped1 <- removeDuplicates(newPed)
nrow(newPed)
nrow(ped1)
pedWithDups <- rbind(newPed, newPed[1:3, ])
ped2 <- removeDuplicates(pedWithDups)
nrow(pedWithDups)
nrow(ped2)
```

removeEarlyDates *removeEarlyDates removes dates before a specified year*

Description

Dates before a specified year are set to NA. This is often used for dates formed from malformed character representations such as a date in

Usage

```
removeEarlyDates(dates, firstYear)
```

Arguments

dates	vector of dates
firstYear	integer value of first (earliest) year in the allowed date range.

Details

NA values are ignored and not changed.

Value

A vector of dates after the year indicated by the numeric value of `firstYear`.

Examples

```
dates <- structure(c(12361, 14400, 15413, NA, 11189, NA, 13224, 10971,
                     -432000, 13262), class = "Date")
cleanedDates <- removeEarlyDates(dates, firstYear = 1000)
dates
cleanedDates
```

removeGroupIfNoAvailableAnimals *Remove group numbers when all available animals have been used*

Description

@return The grpNum list after removing any list element corresponding to a group with no available animals left using in filling a group.

Usage

```
removeGroupIfNoAvailableAnimals(grpNum, available)
```

Arguments

grpNum	as list of integer vectors initially populated with one list named by the integers 1:numGrp, where numGrp is the number of groups to be formed. Each list member is initially populated with a integer vector seq_len(numGrp).
available	is a list of numGrp named members and each member is initially defined as the character vector made up of candidate animal Ids.

---

removePotentialSires *Removes potential sires from list of Ids*

Description

@return character vector of Ids with any potential sire Ids removed.

Usage

```
removePotentialSires(ids, minAge, ped)
```

Arguments

ids	character vector of IDs of the animals
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
ped	dataframe that is the 'Pedigree'. It contains pedigree information including the IDs listed in candidates.

Examples

```
library(nprcgenekeepr)
qcBreeders <- nprcgenekeepr::qcBreeders
pedWithGenotype <- nprcgenekeepr::pedWithGenotype
noSires <- removePotentialSires(ids = qcBreeders, minAge = 2,
                                ped = pedWithGenotype)
sires <- getPotentialSires(qcBreeders, minAge = 2, ped = pedWithGenotype)
pedWithGenotype[pedWithGenotype$id %in% noSires, c("sex", "age")]
pedWithGenotype[pedWithGenotype$id %in% sires, c("sex", "age")]
```

`removeSelectedAnimalFromAvailableAnimals`
Updates list of available animals by removing the selected animal

Description

Updates list of available animals by removing the selected animal

Usage

```
removeSelectedAnimalFromAvailableAnimals(available, ids, numGp)
```

Arguments

`available`	list of available animals for each group
`ids`	character vector having the selected animal Ids
`numGp`	integer indicating the number of groups being formed.

Value

list of available animals

`removeUninformativeFounders`
Remove uninformative founders.

Description

Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information.

Usage

```
removeUninformativeFounders(ped)
```

Arguments

`ped`	datatable that is the 'Pedigree'. It contains pedigree information. The fields `sire` and `dam` are required.

Value

A reduced pedigree.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
probands <- breederPed$id[!(is.na(breederPed$sire) &
                              is.na(breederPed$dam)) &
                              is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
```

---

removeUnknownAnimals *removeUnknownAnimals Removes unknown animals added to pedigree that serve as placeholders for unknown parents.*

Description

removeUnknownAnimals Removes unknown animals added to pedigree that serve as placeholders for unknown parents.

Usage

```
removeUnknownAnimals(ped)
```

Arguments

ped	pedigree dataframe

Value

Pedigree with unknown animals removed

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
addedPed <- cbind(ped, recordStatus = rep("original", nrow(ped)),
                  stringsAsFactors = FALSE)
addedPed[1:3, "recordStatus"] <- "added"
ped2 <- removeUnknownAnimals(addedPed)
nrow(ped)
```

```
nrow(ped2)
```

`reportGV` *Generates a genetic value report for a provided pedigree.*

Description

This is the main function for the Genetic Value Analysis.

Usage

```
reportGV(
  ped,
  guIter = 5000,
  guThresh = 1,
  pop = NULL,
  byID = TRUE,
  updateProgress = NULL
)
```

Arguments

`ped`	The pedigree information in data.frame format
`guIter`	Integer indicating the number of iterations for the gene-drop analysis. Default is 5000 iterations
`guThresh`	Integer indicating the threshold number of animals for defining a unique allele. Default considers an allele "unique" if it is found in only 1 animal.
`pop`	Character vector with animal IDs to consider as the population of interest. The default is NULL.
`byID`	Logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
`updateProgress`	Function or NULL. If this function is defined, it will be called during each iteration to update a `shiny::Progress` object.

Value

A dataframe with the genetic value report. Animals are ranked in order of descending value.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped, removeUninformative = FALSE,
                    addBackParents = FALSE)
geneticValue <- reportGV(ped, guIter = 50, # should be >= 1000
                         guThresh = 3,
                         byID = TRUE,
                         updateProgress = NULL)
trimmedGeneticValue <- reportGV(trimmedPed, guIter = 50, # should be >= 1000
                                guThresh = 3,
                                byID = TRUE,
                                updateProgress = NULL)
rpt <- trimmedGeneticValue[["report"]]
kmat <- trimmedGeneticValue[["kinship"]]
f <- trimmedGeneticValue[["total"]]
mf <- trimmedGeneticValue[["maleFounders"]]
ff <- trimmedGeneticValue[["femaleFounders"]]
nmf <- trimmedGeneticValue[["nMaleFounders"]]
nff <- trimmedGeneticValue[["nFemaleFounders"]]
fe <- trimmedGeneticValue[["fe"]]
fg <- trimmedGeneticValue[["fg"]]
```

`resetGroup` *Update or add the "group" field of a Pedigree.*

Description

Part of the pedigree filtering toolset

Usage

```
resetGroup(ped, ids)
```

Arguments

`ped`	datatable that is the 'Pedigree'. It contains pedigree information. The `id` column is required.

ids character vector of IDs to be flagged as part of the group under consideration.

Value

An updated pedigree with the group column added or updated by being set to TRUE for the animal IDs in ped$id and FALSE otherwise.

---

rhesusGenotypes *rhesusGenotypes is a dataframe with two haplotypes per animal*

Description

There are object.

Usage

```
rhesusGenotypes
```

Format

An object of class data.frame with 31 rows and 3 columns.

Details

Represents 31 animals that are also in the obfuscated rhesusPedigree pedigree from *rhesusGenotypes.csv*.

id – character column of animal IDs

first_name – a generic name for the first haplotype

second_name – a generic name for the second haplotype

Examples

```
library(nprcgenekeepr)
data("rhesusGenotypes")
```

rhesusPedigree *rhesusPedigree is a pedigree object*

Description

Represents an obfuscated pedigree from *rhesusPedigree.csv* where the IDs and dates have been modified to de-identify the data.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column.Unknown dams are indicated with `NA`

sex – factor with levels: "M", "F", "U". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – Date vector of birth dates

exit – Date vector of exit dates

age – numerical vector of age in years

Usage

```
rhesusPedigree
```

Format

An object of class `data.frame` with 375 rows and 8 columns.

Examples

```
library(nprcgenekeepr)
data("rhesusPedigree")
```

`runGeneKeepR` *Allows running* `shiny` *application with* `nprcgenekeepr::runGeneKeepR()`

Description

Allows running `shiny` application with `nprcgenekeepr::runGeneKeepR()`

Usage

```
runGeneKeepR()
```

Value

Returns the error condition of the Shiny application when it terminates.

Examples

```
## Not run:
library(nprcgenekeepr)
runGeneKeepR()

## End(Not run)
```

`saveDataframesAsFiles` *Write copy of dataframes to either CSV or Excel file.*

Description

Uses `examplePedigree` data structure to create an example data file

Usage

```
saveDataframesAsFiles(dfList, baseDir, fileType = "csv")
```

Arguments

`dfList`	list of dataframes to be stored as files. `"txt"`, `"csv"`, or `"xlsx"`. Default value is `"csv"`.
`baseDir`	character vector of length on with the directory path.
`fileType`	character vector of length one with possible values of `"txt"`, `"csv"`, or `"xlsx"`. Default value is `"csv"`.

Value

Full path name of file saved.

setExit *Sets the exit date, if there is no exit column in the table*

Description

Part of Pedigree Curation

Usage

```
setExit(ped, time.origin = as.Date("1970-01-01"))
```

Arguments

ped	dataframe of pedigree and demographic information potentially containing columns indicating the birth and death dates of an individual. The table may also contain dates of sale (departure). Optional columns are birth, death, and departure.
time.origin	date object used by as.Date to set origin.

Value

A dataframe with an updated pedigree with exit dates specified based on date information that was available.

Examples

```
library(lubridate)
library(nprcgenekeepr)
death <- mdy(paste0(sample(1:12, 10, replace = TRUE), "-",
                    sample(1:28, 10, replace = TRUE), "-",
                    sample(seq(0, 15, by = 3), 10, replace = TRUE) + 2000))
departure <- as.Date(rep(NA, 10), origin = as.Date("1970-01-01"))
departure[c(1, 3, 6)] <- as.Date(death[c(1, 3, 6)],
                                 origin = as.Date("1970-01-01"))
death[c(1, 3, 5)] <- NA
death[6] <- death[6] + days(1)
ped <- data.frame(
  id = paste0(100 + 1:10),
  birth = mdy(paste0(sample(1:12, 10, replace = TRUE), "-",
                     sample(1:28, 10, replace = TRUE), "-",
                     sample(seq(0, 20, by = 3), 10, replace = TRUE) + 1980)),
  death = death,
  departure = departure,
  stringsAsFactors = FALSE)
pedWithExit <- setExit(ped)
```

setPopulation *Population designation function*

Description

Part of the pedigree filtering toolset.

Usage

```
setPopulation(ped, ids)
```

Arguments

ped	datatable that is the 'Pedigree'. It contains pedigree information. The id column is required.
ids	character vector of IDs to be flagged as part of the population under consideration.

Value

An updated pedigree with the population column added or updated by being set to TRUE for the animal IDs in ped$id and FALSE otherwise.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
nrow(breederPed[breederPed$population, ])
```

set_seed *Work around for unit tests using sample() among various versions of R*

Description

The change in how 'set.seed' works in R 3.6 prompted the creation of this R version agnostic replacement to get unit test code to work on multiple versions of R in a Travis-CI build.

Usage

```
set_seed(seed = 1)
```

Arguments

seed	argument to `set.seed`

Details

It seems RNGkind(sample.kind="Rounding") does not work prior to version 3.6 so I resorted to using version dependent construction of the argument list to set.seed() in do.call().

Value

NULL, invisibly.

Examples

```
set_seed(1)
rnorm(5)
```

---

smallPed *smallPed is a hypothetical pedigree*

Description

It has the following structure: structure(list(id = c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q"), sire = c("Q", NA, "A", "A", NA, "D", "D", "A", "A", NA, NA, "C", "A", NA, NA, "M", NA), dam = c(NA, NA, "B", "B", NA, "E", "E", "B", "J", NA, NA, "K", "N", NA, NA, "O", NA), sex = c("M", "F", "M", "M", "F", "F", "F", "M", "F", "F", "F", "F", "M", "F", "F", "F", "M"), gen = c(1, 1, 2, 2, 1, 3, 3, 2, 2, 1, 1, 2, 1, 1, 2, 3, 0), population = c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE)), .Names = c("id", "sire", "dam", "sex", "gen", "population"), row.names = c(NA, -17L), class = "data.frame")

Usage

```
smallPed
```

Format

An object of class `data.frame` with 17 rows and 6 columns.

smallPedTree *smallPedTree is a pedigree tree made from* smallPed

Description

Access it using the following commands.

Usage

```
smallPedTree
```

Format

An object of class list of length 17.

Examples

```
library(nprcgenekeepr)
data("smallPedTree")
```

str_detect_fixed_all *Returns a logical vector with results of stri_detect() for each pattern in second parameters character vector.*

Description

Returns a logical vector with results of stri_detect() for each pattern in second parameters character vector.

Usage

```
str_detect_fixed_all(strings, patterns, ignore_na, ...)
```

Arguments

strings	input vector. This must be an atomic vector and will be coerced to a character vector.
patterns	patterns to look for, as defined by a POSIX regular expression. See fixed, ignore.case and perl sections for details. See *Extended Regular Expressions* for how to use regular expressions for matching.
ignore_na	if TRUE NA values are trimmed out of strings and patterns before comparison
...	further arguments for stri_detect_fixed

summary.nprcgenekeeprErr *summary.nprcgenekeeprErr Summary function for class nprcgenekeeprErr*

Description

summary.nprcgenekeeprErr Summary function for class nprcgenekeeprErr

Usage

```
## S3 method for class 'nprcgenekeeprErr'
summary(object, ...)

## S3 method for class 'nprcgenekeeprGV'
summary(object, ...)
```

Arguments

object	object of class nprcgenekeeprErr and class list
...	additional arguments for the summary.default statement

Value

Object of class summary.nprcgenekeeprErr

object of class summary.nprcgenekeeprGV

Examples

```
errorList <- qcStudbook(nprcgenekeepr::pedOne, minParentAge = 0,
reportChanges = TRUE,
reportErrors = TRUE)
summary(errorList)



examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
                         reportChanges = FALSE,
                         reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped, removeUninformative = FALSE,
```

```
                    addBackParents = FALSE)
geneticValue <- reportGV(ped, guIter = 50, # should be >= 1000
                         guThresh = 3,
                         byID = TRUE,
                         updateProgress = NULL)
trimmedGeneticValue <- reportGV(trimmedPed, guIter = 50, # should be >= 1000
                                guThresh = 3,
                                byID = TRUE,
                                updateProgress = NULL)
summary(geneticValue)
summary(trimmedGeneticValue)
```

`toCharacter` *Force dataframe columns to character*

Description

Converts designated columns of a dataframe to character. Defaults to converting columns `id`, `sire`, and `dam`.

Usage

```
toCharacter(df, headers = c("id", "sire", "dam"))
```

Arguments

`df`	a dataframe where the first three columns can be coerced to character.
`headers`	character vector with the columns to be converted to character class. Defaults to `c("id","sire","dam")/`

Value

A dataframe with the specified columns converted to class "character" for display with xtables (in shiny)

Examples

```
library(nprcgenekeepr)
pedGood <- nprcgenekeepr::pedGood
names(pedGood) <- c("id", "sire", "dam", "sex", "birth")
class(pedGood[["id"]])
pedGood <- toCharacter(pedGood)
class(pedGood[["id"]])
```

`trimPedigree` *Trim pedigree to ancestors of provided group by removing uninformative individuals*

Description

Filters a pedigree down to only the ancestors of the provided group, removing unnecessary individuals from the studbook. This version builds the pedigree back in time starting from a group of probands, then moves back down the tree trimming off uninformative ancestors.

Usage

```
trimPedigree(
  probands,
  ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
```

Arguments

`probands`	a character vector with the list of animals whose ancestors should be included in the final pedigree.
`ped`	datatable that is the 'Pedigree'. It contains pedigree information. The fields `sire` and `dam` are required.
`removeUninformative`	logical defaults to `FALSE`. If set to `TRUE`, uninformative founders are removed. Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information.
`addBackParents`	logical defaults to `FALSE`. If set to `TRUE`, the function adds back single parents to the `p` dataframe when one parent is known. The function `addBackSecondParents` uses the `ped` dataframe, which has full complement of parents and the `p` dataframe, which has all uninformative parents removed to add back single parents to the `p` dataframe.

Value

A pedigree that has been trimmed, had uninformative founders removed and single parents added back.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree, minParentAge = 2,
```

```
                          reportChanges = FALSE,
                          reportErrors = FALSE)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
                                  is.na(breederPed$dam)) &
                                 is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
trimmedPedInformative <- trimPedigree(focalAnimals, breederPed,
                                      removeUninformative = TRUE)
nrow(breederPed)
nrow(trimmedPed)
nrow(trimmedPedInformative)
```

---

unknown2NA *Removing IDs having "UNKNOWN" regardless of case*

Description

Someone started entering "unknown" for unknown parents instead of leaving the field blank in PRIMe.

Usage

```
unknown2NA(ped)
```

Arguments

ped	A dataframe containing at least and "id" field

---

withinIntegerRange *Get integer within a range*

Description

Assures that what is returned is an integer within the specified range. Real values are truncated. Non-numerics are forced to minimum without warning.

Usage

```
withinIntegerRange(int = 0, minimum = 0, maximum = 0, na = "min")
```

Arguments

int	value to be forced within a range
minimum	minimum integer value.
maximum	maximum integer value
na	if "min" then non-numerics are forced to the minimum in the range If "max" then non-numerics are forced to the maximum in the range. If not either "min" or "max" it is forced to "min".

Value

A vector of integers forced to be within the specified range.

Examples

```
library(nprcgenekeepr)
withinIntegerRange()
withinIntegerRange( , 0, 10)
withinIntegerRange(NA, 0, 10, na = "max")
withinIntegerRange( , 0, 10, na = "max") # no argument is not NA
withinIntegerRange(LETTERS, 0, 10)
withinIntegerRange(2.6, 1, 5)
withinIntegerRange(2.6, 0, 2)
withinIntegerRange(c(0, 2.6, -1), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "max")
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "min")
```

Index