Package 'gt.db'

October 30, 2009

Version 0.5-1

Date 2009-10-08

Title GT.DB: Genotype Data Management and Analysis

Author David Hinds

Maintainer David Hinds <dhinds@sonic.net>

Description Framework for storing and manipulating genotype data, phenotype data, and association study results

License GPL

Depends R (>= 2.7.0), lattice, grid, methods, DBI

Suggests nlme, cluster, mda, ROracle, RMySQL, RSQLite

R topics documented:

`adjust.gt.calls` *Manually Assign Genotype Calls*

Description

Interactively assign genotype calls by inspection of a plot of reference versus alternate signal intensities.

Usage

```
adjust.gt.calls(data, ..., radius=6)
```

Arguments

`data`	a data frame returned by `reshape.gt.data`.
`...`	additional arguments passed to `gt.cluster.plot`.
`radius`	the radius around the mouse pointer selected by clicking, in points.

Details

The intensity data is first plotted, then the user is iteratively prompted for a genotype value (in a/h/b/n notation) in the console window. Once a value has been entered, the plot window becomes active and the mouse pointer can be used to select points whose genotypes should be changed to that value. Each left click causes a zone around the mouse pointer position to be updated. The process is terminated by right clicking and selecting 'Stop' from the menu. The user is then prompted for another genotype value in the console window. To finish here, enter `q`.

Value

A data frame like `data` but with updated genotype calls. If a column called `orig.genotype` does not already exist, then it will be created and populated with the original unmodified genotypes.

See Also

`fetch.gt.data`, `reshape.gt.data`, `gt.cluster.plot`.

Examples

```
## Not run:
gt <- fetch.gt.data('Demo_2',raw.data=TRUE)
d <- reshape.gt.data(gt[32,], na.codes='n')
adjust.gt.calls(d, scales=list(log=TRUE))
## End(Not run)
```

`apply.gt.dataset` *Apply a Function to a Genotype Dataset*

Description

Apply a function to a genotype dataset, processed in manageable chunks, and aggregate the results.

Usage

```
apply.gt.dataset(gt.dataset, part.fn, aggr.fn, ..., GT.DATA='gt.data')
```

Arguments

`gt.dataset`	a dataset description from `gt.dataset`.
`part.fn`	a function to apply to each part of the dataset.
`aggr.fn`	an aggregator function to accumulate results.
`...`	additional arguments passed to `part.fn`.
`GT.DATA`	the name to use for passing the genotype argument to `part.fn`.

Details

Based on the specification in `gt.dataset`, the dataset is loaded in chunks defined by `parts` `binsz`, and `gt.filter`. `part.fn` is applied iteratively to each chunk. The genotype data is passed as the `GT.DATA` argument to `part.fn`. The aggregator function `aggr.fn` should accept two arguments: an accumulated result, initially set to NULL; and a result for a single chunk.

Value

Depends on form of `part.fn` and `aggr.fn`.

See Also

`fetch.gt.data`, `gt.dataset`.

Examples

```
gt.demo.check()
s <- apply.gt.dataset(gt.dataset('Demo_1'), summary,
                      rbind, GT.DATA='object')
str(s)
s <- apply.gt.dataset(gt.dataset('Demo_1'), summary,
                      aggr.fn=function(a,b) c(a,list(b)),
                      GT.DATA='object')
str(s)
```

---

apply.loadings *Apply PCA SNP Loadings to a New Genotype Set*

Description

Takes a table of SNP loadings from a principal components analysis, and applies them to another set of genotypes, yielding a new set of sample loadings.

Usage

```
apply.loadings(x, data)
```

Arguments

`x`	SNP loadings from `snp.loadings`.
`data`	either a data frame of genotype data from `fetch.gt.data`, or a dataset description from `gt.dataset`.

Details

If the SNP loadings were computed on the same platform as the new dataset, then corresponding assays are identified by name. Across platforms, assays are matched up by genomic position.

Value

A data frame of sample loadings, with one column per principal component, with samples ordered as they are in the supplied genotype strings.

See Also

`prcomp`, `snp.loadings`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_1')
# compute PCA for unrelated individuals
p1 <- prcomp(gt.dataset('Demo_1'), is.na(pt$father))
# now compute sample loadings for all individuals
s <- snp.loadings(p1)
p2 <- apply.loadings(s, gt.dataset('Demo_1'))
```

---

as.mask *Convert To/From Character Masks*

Description

Convert between logical vectors, and strings of T/F characters.

Usage

```
as.mask(x)
un.mask(s)
```

Arguments

x	a logical vector.
s	a string of T/F characters.

Value

For as.mask, a string of T/F characters with one character per element of x.

For un.mask, a logical vector with one element per character in s

See Also

mask.str.

Examples

```
as.mask(c(TRUE,FALSE,TRUE,FALSE,TRUE))
un.mask('TFFTTF')
```

`big.loadings` *Identify Large Sample or SNP Loadings from a PCA Analysis*

Description

Takes either a PCA result, or SNP loadings from PCA, and returns a data frame describing large loadings for each principal component.

Usage

```
big.loadings(x, sigma=6)
```

Arguments

`x`	either a PCA result structure, or SNP loadings.
`sigma`	number of standard deviations to qualify as "big".

Value

A data frame with sample or assay identifiers, and the following additional columns:

`name`	the principal component with high loading.
`value`	the loading for this sample and component.
`zscore`	the standardized loading for this sample.
`var`	the fraction of variance in this component accounted for by this sample.

See Also

`prcomp.gt.data`, `prcomp.gt.dataset`, `snp.loadings`.

Examples

```
gt.demo.check()
g <- fetch.gt.data('Demo_1')
pc <- prcomp(subset(g, (ploidy=='A')))
big.loadings(pc,5)
sl <- snp.loadings(pc, g)
big.loadings(sl,5)
```

ch.table *Character Based Contingency Table*

Description

Build contingency tables of character counts.

Usage

```
ch.table(s1, s2, chars)
```

Arguments

s1	a vector of character strings.
s2	an optional vector of character strings, with lengths matching strings in s1.
chars	a vector of individual characters to tabulate.

Value

If s2 is missing, a matrix of character counts with one row per element of s1 and one column per element of chars.

If s2 is present, a three dimensional array where the first dimension indicates a position within s1 and s2 (with recycling if necessary, and the second and third dimensions form contingency tables of corresponding characters in these strings.

Examples

```
s1 <- c('AACAGCTACAGT','TTGTCGATGTCA')
s2 <- 'AACAGCTACAGT'
ch.table(s1, chars=c('A','C','G','T'))
x <- ch.table(s1, s2, chars=c('A','C','G','T'))
x[1,,]
x[2,,]
```

demo_01 *Genotype Data for 5000 SNPs on 270 HapMap Samples*

Description

Genotype data for 5000 SNPs distributed uniformly across the genome, for 270 samples from the International HapMap Project. This is a subset of data generated by Perlegen for the GAIN collaboration, using a set of SNP arrays designed to tag LD bins across the genome.

Usage

```
data(demo_01)
```

Source

Perlegen Sciences, Inc., publicly released data.

References

The GAIN Collaborative Research Group (2007) New models of collaboration in genome-wide association studies: the Genetic Association Information Network. *Nat. Genet.* **39**: 1045-1051.

The Database of Genotypes and Phenotypes. http://www.ncbi.nlm.nih.gov/sites/entrez?db=gap.

The International HapMap Project. http://www.hapmap.org.

Examples

```
## Not run:
# load demo datasets into GT.DB
demo('setup.gt.demo')
## End(Not run)
```

---

demo_02 *Genotype Data for 371 Chr21 SNPs on 275 HapMap Samples*

Description

Genotype data for 371 SNPs distributed across a 300kb segment of human chromosome 21, for 275 samples from the International HapMap Project. The data was generated for the Phase II HapMap by Perlegen Sciences, using a custom array set developed for that project.

Usage

```
data(demo_02)
```

Source

Perlegen Sciences, Inc., publicly released data.

References

The International HapMap Consortium (2007). A second generation human haplotype map of over 3.1 million SNPs. *Nature* **449**: 851-861.

The International HapMap Project. http://www.hapmap.org.

Examples

```
## Not run:
# load demo datasets into GT.DB
demo('setup.gt.demo')
## End(Not run)
```

`fetch.gt.data` *Load Genotype Data for a Genotyping Dataset*

Description

Returns a data frame of assay information and genotypes for the specified dataset.

Usage

```
fetch.gt.data(dataset.name, mapping.name, assay.name, dbsnp.rsid,
              part, parts, by=c('assay','position'), binsz=1, where,
              show.ids=FALSE, genotype=TRUE, qscore=FALSE, raw.data=FALSE)
```

Arguments

`dataset.name`	the unique identifier for the dataset.
`mapping.name`	an identifier for the assay mapping to use.
`assay.name`	a vector of assay names.
`dbsnp.rsid`	a vector of integer dbSNP rsID values.
`part`	a vector of subsets 1..parts.
`parts`	the number of subsets to split the dataset into.
`by`	specifies how to construct subsets. See details.
`binsz`	specifies how to construct subsets. See details.
`where`	additional SQL `WHERE` clauses to limit the data returned.
`show.ids`	logical: indicates whether to include values of database keys.
`genotype`	logical: indicates if genotypes should be fetched.
`qscore`	logical: indicates if quality scores should be fetched.
`raw.data`	logical: indicates if underlying raw data should be fetched, if available.

Details

If an assay mapping is not explicitly specified, and there is just one visible mapping for this dataset's platform, that one will be used by default.

The `assay.name`, `dbsnp.rsid`, and `part` arguments are mutually exclusive ways of specifying a set of SNPs for which data should be retrieved.

If `part` and `parts` are specified, then the dataset is divided into `parts` roughly similarly sized subsets, and `part` specifies which subset to retrieve. The chunking can be done either by primary key (`by='assay.id'`) or by position (`by='position'`). To preserve some locality, assay subsets are interleaved such that `trunc(by/binsz)%%parts==(part-1)`.

Value

A data frame with one row per SNP. If any flags have been defined for this dataset, each flag is expanded to a logical column.

`assay.data.id`
: the unique integer identifier for this entry in the `ASSAY_DATA` table.

assay.id	if show.ids is set: the unique key for this assay.
assay.name	the name of this assay.
alleles	a slash-separated list of alleles for this assay.
scaffold	an identifier for the scaffold to which this assay has been mapped.
position	the 1-based genomic position in the specified scaffold.
strand	the strand (+/-) to which the assay was mapped.
ploidy	one of "A" (autosomal), "X" (X linked), "Y" (Y linked), or "M" (mitochondrial).
dbsnp.rsid	the dbSNP rsID to which this assay is mapped.
dbsnp.orient	the relative orientation of the dbSNP refsnp cluster.
genotype	a string of a/h/b/n/x genotypes.
qscore	a string of packed quality scores.
raw.data	a string containing packed raw data (such as signal intensities or read counts).

See Also

summary.gt.data, unpack.gt.matrix.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_1')
str(gt)
nrow(fetch.gt.data('Demo_1', part=1, parts=4))
nrow(fetch.gt.data('Demo_1', part=1, parts=100))
summary.gt.data(gt[1:10,])
```

---

fetch.prcomp *Load Principal Components Results for a Genotyping Dataset*

Description

Load principal components analysis results for a Genotyping Dataset.

Usage

```
fetch.prcomp(dataset.name, prcomp.name, nc)
```

Arguments

dataset.name	the unique identifier for the dataset.
prcomp.name	the unique identifier for the analysis.
nc	the number of components to retrieve.

Details

If prcomp.name is not present, then if there is just one visible analysis for this dataset, that one will be loaded.

Value

The same sort of structure as returned by prcomp.gt.data.

See Also

store.prcomp, prcomp.gt.data, prcomp.gt.dataset.

Examples

```
## Not run:
pc <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')))
store.prcomp(pc, 'demo_pc_1', 'Demo PCA results')
fetch.prcomp('Demo', 'demo_pc_1', nc=4)
## End(Not run)
```

---

fetch.pt.data *Load Phenotype Data for a Genotype Dataset*

Description

Returns a data frame of merged sample and subject phenotypes for the specified genotype dataset, mapped to the appropriate R datatypes.

Usage

```
fetch.pt.data(dataset.name, cols, pca=FALSE, show.all=FALSE)
```

Arguments

dataset.name	the short unique identifier for the dataset.
cols	a vector of either sample or subject attributes to be included.
pca	a logical indicating whether the current principal components analysis results should be included, or a list of arguments to be passed to fetch.prcomp to specify what to load.
show.all	logical: indicates if hidden attributes should be included in the output.

Value

A data frame with one row per sample, and one column per attribute defined for this dataset, combining sample data from fetch.sample.data and subject data from fetch.subject.data. The column datatypes are determined using the information in the SAMPLE_ATTR and SUBJECT_ATTR tables. The row order conforms to the order of samples in the genotype data, and row names are set to the sample names. The result also includes the predefined sample attributes: subject name, gender, and dataset position.

See Also

fetch.sample.data. fetch.subject.data. fetch.prcomp.

Examples

```
gt.demo.check()
head(fetch.pt.data('Demo_1'))
head(fetch.pt.data('Demo_1', cols='plate'))
```

---

fetch.sample.data *Load Sample Data for a Genotype Dataset*

Description

Returns a data frame of sample phenotypes for the specified dataset, mapped to the appropriate R datatypes.

Usage

```
fetch.sample.data(dataset.name, cols, show.all=FALSE)
```

Arguments

>

dataset.name	the short unique identifier for the dataset.
cols	a vector of column names to be included.
show.all	logical: indicates if hidden attributes should be included in the output.

Value

A data frame with one row per sample, and one column per attribute defined for this dataset. The column datatypes are determined using the information in the SAMPLE_ATTR table. The row order is arbitrary, but row names are set to the sample names. The result also includes the predefined sample attributes: subject name, gender, and dataset position.

See Also

mk.sample, ls.sample, fetch.pt.data, store.sample.data.

Examples

```
gt.demo.check()
head(fetch.sample.data('Demo_1'))
```

fetch.subject.data *Load Subject Data for a Genotyping Project*

Description

Returns a data frame of subject phenotypes for the specified project, mapped to the appropriate R datatypes.

Usage

```
fetch.subject.data(project.name, cols, show.all=FALSE)
```

Arguments

project.name	the short unique identifier for the project.
cols	a vector of column names to be included.
show.all	logical: indicates if hidden attributes should be included in the output.

Value

A data frame with one row per subject, and one column per attribute defined for this project. The column datatypes are determined using the information in the SUBJECT_ATTR table. The row order is arbitrary, but row names are set to the sample names.

See Also

mk.subject, ls.subject, store.subject.data.

Examples

```
gt.demo.check()
head(fetch.subject.data('Demo'))
```

fetch.test.scores *Load Association Test Results for a Genotyping Dataset*

Description

Load association test results for a Genotyping Dataset.

Usage

```
fetch.test.scores(dataset.name, test.name, term, max.pvalue)
```

Arguments

dataset.name	the unique identifier for a dataset.
test.name	the unique identifier for an analysis.
term	if present, for result sets with multiple results per assay, specifies which set of results to fetch.
max.pvalue	if present, specifies an upper bound for P values to be fetched.

Value

The same sort of structure as returned by score.gt.data.

See Also

store.test.scores, score.gt.data

Examples

```
## Not run:
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
x <- score.gt.data(status~genotype, pt, gt)
store.test.scores(x, 'status_1', 'Test analysis')
y <- fetch.test.scores('Demo_2', 'status_1')
str(y)
## End(Not run)
```

---

gplot *Genome-Wide Level Plot*

Description

Generates a level plot of genome-wide data organized by chromosome and position.

Usage

```
gplot(formula, data, aggr.fn=max, rescale=FALSE, binsz=1e6,
      subset=TRUE, col.regions=rev(heat.colors(100)[10:100]),
      scales=list(x=list(at=seq(0,250,20), draw=TRUE)),
      shrink=list(x=1,y=0.75), colorkey=list(height=0.25),
      xlab='Position, Mb', ylab='Chromosome', ...)
```

Arguments

formula	a one-sided formula (i.e. ~ x) describing the values to be plotted.
data	a data frame containing values required to evaluate formula, as well as chromosome and position columns.
aggr.fn	an aggregating function to apply to data within each bin.
rescale	logical: indicates if results should be scaled based on the genome-wide average bin value.
binsz	bin size in base pairs
subset	a logical or integer vector identifying rows of data to be included in the plot.
col.regions,	scales, shrink, colorkey, xlab, ylab see levelplot.
...	additional arguments passed to levelplot.

Details

The supplied formula is first evaluated across the input data, and then the aggregating function `aggr.fn` is applied to results in bins of genomic coordinates. Appropriate aggregating functions include `max`, `min`, `sum`, etc.

Value

A plot object of class `"trellis"`.

See Also

`levelplot`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_1')
gt <- fetch.gt.data('Demo_1')
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
r <- score.gt.data(status~genotype, pt, gt, score.chisq)
gplot(~-log10(pvalue), merge(r,gt))
```

---

gplot.prcomp *Genome-Wide Level Plots of SNP Loadings from PCA*

Description

Generates level plots of genome-wide SNP loadings from a principal components analysis.

Usage

```
gplot.prcomp(x, col=1:5, aggr.fn=function(x) sum(x^2),
             rescale=TRUE, xlab=NULL, ylab=NULL, ...)
```

Arguments

`x`	a data structure returned by `snp.loadings`.
`col`	a vector of component numbers to plot.
`aggr.fn`	an aggregating function to apply to data within each bin.
`rescale`	logical: indicates if results should be scaled based on the genome-wide average bin value.
`xlab, ylab`	axis labels passed to `gplot`.
`...`	additional arguments passed to `gplot`.

Details

The output of `gplot.prcomp` is a vertically stacked series of genome-wide plots of SNP loadings for a specified set of principal components. The default aggregating function results in a plot of variance binned by genomic interval.

Value

A plot object of class "trellis".

See Also

prcomp.gt.data, prcomp.gt.dataset, snp.loadings, gplot.

Examples

```
gt.demo.check()
p <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')))
s <- snp.loadings(p, gt.dataset('Demo_1', gt.filter=TRUE))
par <- list(fontsize=list(text=8),
            background=list(alpha=0,col=gray(0.9)))
gplot.prcomp(s, col=1:3, par.settings=par)
```

---

gt.cluster.plot *Plot Genotype Cluster Data*

Description

Plot raw genotype cluster data (i.e. signal intensities), with predicted ellipsoid boundary regions.

Usage

```
gt.cluster.plot(data, rescale=TRUE, bounds=c(0.5,0.95), min.points=4,
                between=list(x=0.5,y=0.5), scales=list(alternating=0),
                xlab=NULL, ylab=NULL, par.settings=.gt.settings, ...)
## S3 method for class 'gt.data':
xyplot(x, ...)
```

Arguments

data	an unpacked data frame of genotype information from reshape.gt.data.
x	a data frame of genotypes from fetch.gt.data.
rescale	logical: indicates if each data in each panel should be scaled to fill the frame.
bounds	contours at which to draw ellipsoid boundaries.
min.points	the minimum number of points for which to compute ellipsoids.
between, scales, xlab, ylab, par.settings, ...	arguments passed to xyplot.

Details

gt.cluster.plot is a wrapper around xyplot to facilitate plotting raw data underlying genotype calls. A custom panel function draws minimal-area ellipsoid contours to capture proportions of genotype cluster density given by bounds.

xyplot.gt.data is a method for packed genotype data.

Value

An object of class "trellis".

See Also

panel.cluster, xyplot, ellipsoidPoints, fetch.gt.data, reshape.gt.data, adjust.gt.calls.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2',raw.data=TRUE)
head(reshape.gt.data(gt))
d <- reshape.gt.data(gt[seq(32,232,40),], na.codes='n')
gt.cluster.plot(d)
gt.cluster.plot(d, scales=list(log=TRUE))
xyplot(gt[seq(32,232,40),])
```

---

gt.dataset *Genotype Dataset Specification*

Description

Describes a genotype dataset in the database and how to process it.

Usage

```
gt.dataset(dataset.name, gt.filter=TRUE, parts=10,
           by='assay', binsz=1, progress=interactive())
```

Arguments

dataset.name	the unique identifier for the dataset.
gt.filter	an expression to use for subsetting the genotype data.
parts, by, binsz	specify how to construct chunks. See fetch.gt.data.
progress	logical: indicates whether to display a progress bar when processing the dataset.

Details

The specified dataset is loaded in chunks defined by parts and binsz, and part.fn is applied iteratively to each chunk. The genotype data is passed as the gt.data argument to part.fn. The aggregator function aggr.fn should accept two arguments: an accumulated result, initially set to NULL; and a result for a single chunk.

Value

A list of class gt.dataset, with elements representing all the arguments of gt.dataset.

See Also

fetch.gt.data, apply.gt.dataset.

Examples

```
gt.demo.check()
s <- apply.gt.dataset(gt.dataset('Demo_1'), summary,
                      rbind, GT.DATA='object')
str(s)
s <- apply.gt.dataset(gt.dataset('Demo_1', gt.filter=(ploidy=='A')),
                      summary, rbind, GT.DATA='object')
str(s)
```

---

gt.demo.check *Check for Presence of GT.DB Demo Datasets*

Description

This is used in many of the GT.DB examples to check for the presence of the demo datasets.

Usage

```
gt.demo.check()
```

Details

If the demo datasets are unavailable, then the example code is terminated with an error message. If invoked non-interactively (say, from `R CMD CHECK`), then a temporary SQLite database is created for the demo datasets.

See Also

`init.gt.db`, `demo(setup.gt.demo)`.

Examples

```
gt.demo.check()
```

---

gt.dist *Calculate Pairwise Genotype Distances*

Description

Calculate genotype-based distances (broadly defined) between pairs of samples, using a choice of distance operators and aggregator functions.

Usage

```
gt.dist(gt1, gt2=gt1, operation='==', aggregator='sum', na.value=NA)
```

Arguments

gt1, gt2	vectors of packed diploid genotypes as from fetch.gt.data, with 1:1 matching assays.
operation	one of 'n', '==', '!=', 'ibs==0', 'ibs==1', or 'ibs'.
aggregator	one of 'sum', 'min', or 'max'.
na.value	an integer value to use as the result of comparisons involving missing genotypes. The default excludes these comparisons.

Details

For each genotype assay, an operation is performed on each pairwise combination of samples:

'n' 1 if both genotypes are present, else 0.

'==' 1 if genotypes are identical, else 0.

'!=' 1 if genotypes are not identical, else 0.

'ibs==0' 1 if genotypes are identical by state (IBS) for 0 alleles, else 0.

'ibs==1' 1 if genotypes are IBS for 1 allele, else 0.

'ibs==2' 1 if genotypes are IBS for 2 alleles, else 0.

'ibs' the number of alleles IBS, 0..2.

The matrices of results are then aggregated across genotype assays, using one of the available aggregation functions.

Value

A matrix of integers with length(gt1) rows and length(gt2) columns, with elements giving the aggregated distance for the corresponding pair of samples.

See Also

match.gt.data, ibd.dataset.

Examples

```
gt <- fetch.gt.data('Demo_2')
gt <- substr(gt$genotype[1:20],1,10)
gt.dist(gt)
gt.dist(gt, operation='ibs', aggregator='min')
```

---

gt.split *Convert between Packed Genotype Strings and Genotype Vectors*

Description

gt.split takes a string of a/h/b/n genotypes and returns a vector with one genotype per element; gt.paste reverses this conversion.

Usage

```
gt.split(s, convert=c('score.b','score.a','char','none'),
         alleles, na.codes=c(), strand=c('+','-'), sep='/')
gt.paste(v, convert=c('score.b','score.a','char','none'),
         alleles, na.codes=c(), strand=c('+','-'), sep='/')
```

Arguments

s	a character string of packed a/h/b/n genotypes.
v	a vector of unpacked genotypes. See details.
convert	how genotypes should be represented.
alleles	a vector of length 2 specifying A and B alleles, for convert='char'.
na.codes	a vector of missing genotype codes: see details.
strand	for convert='char', strand to report.
sep	for convert='char', a string for separating alleles.

Details

If convert='score.a', then unpacked genotypes are coded as numeric scores, where "a"=2, "h"=1, and "b"=0. If convert='score.b', then unpacked genotypes are coded as numeric scores, where "a"=0, "h"=1, and "b"=2. if convert='char', then genotypes are coded as factors with levels formed from concatenated pairs of alleles. If convert='none', then genotypes are coded as factors with three levels, 'a', 'h', and 'b'.

Missing genotypes are by default coded as NA in the unpacked format. If na.codes is not empty, then the specified single character codes are instead reported as separate factor levels in the unpacked format.

Value

For gt.split, a vector of genotypes coded as specified by convert. For gt.paste, a packed genotype string.

See Also

fetch.gt.data, unpack.gt.matrix.

Examples

```
gt <- 'aahabbnnaxa'
gt.split(gt)
gt.split(gt, convert='char', alleles=c('A','C'))
gt.split(gt, convert='char', alleles=c('A','C'), strand='-')
gt.split(gt, convert='char', alleles=c('A','C'),
         na.codes='n', sep='')
gt.split(gt, convert='none', na.codes='n')
gt.split(gt, convert='none', na.codes=c('n','x'))

x <- gt.split(gt, convert='char', alleles=c('A','C'))
gt.paste(x, convert='char', alleles=c('A','C'))
gt.paste(x, convert='char', alleles=c('C','A'))
gt.paste(c(0,1,NA,2,1,2,NA), convert='score.b')
```

hapmap *Subject Data from the International HapMap Project*

Description

`hapmap.subjects` describes sample plate and panel membership, and parent/child relationships, for the 1301 individuals genotyped in the Phase II and Phase III HapMap Projects.

Usage

```
data(hapmap)
```

Source

The International HapMap Project

References

The International HapMap Project. http://www.hapmap.org.

See Also

`demo(setup.hapmap)`.

Examples

```
## Not run:
# create HapMap project in GT.DB
demo('setup.hapmap')
## End(Not run)
```

hwe.test *Tests for Hardy Weinberg Equilibrium*

Description

Asymtotic and exact tests for Hardy Weinberg equilibrium conditional on observed marginal allele frequencies, for biallelic genotype data.

Usage

```
hwe.test(aa, ab, bb, test=c('lratio','chisq','exact'),
         tail=c('both','lower','upper'))
```

Arguments

`aa, ab, bb`	biallelic genotype counts (may be vectors).
`test`	the type of test to perform. See details.
`tail`	See details.

Details

Three tests are implemented: the likelihood ratio test, the traditional chi-squared test without continuity correction, and the conditional exact test. All are defined as in Weir (1996). All the tests are vectorized for efficiency.

The default is to perform a two-sided test. If `tail='lower'`, then the result is the probability of seeing no more than the observed number of heterozygotes under the hypothesis of Hardy Weinberg equilibrium. If `tail='upper'`, then the result is the probability of seeing at least as many as the observed number of heterozygotes.

Value

A vector of P values.

References

Weir, B.S. (1996) *Genetic Data Analysis II*. Sinauer, Sunderland, MA.

Examples

```
# Table 3.1 in Weir (1996)
d <- data.frame(aa=0:9, ab=seq(19,1,-2), bb=21:30)
d$p.exact <- with(d, hwe.test(aa,ab,bb,'exact'))
d$p.chisq <- with(d, hwe.test(aa,ab,bb,'chisq'))
d$chisq <- qchisq(d$p.chisq,df=1,lower.tail=FALSE)
d[order(d$p.chisq),]
#
gt.demo.check()
gt <- fetch.gt.data('Demo_1')
pt <- fetch.pt.data('Demo_1')
s <- summary.gt.data(gt, pt$plate=='J+C')
d <- with(s,data.frame(
    lratio=hwe.test(AA,AB,BB,'lratio'),
    chisq=hwe.test(AA,AB,BB,'chisq'),
    exact=hwe.test(AA,AB,BB,'exact')
))
p <- list(limits=c(-10,0), at=c(-2,-4,-6,-8),
          labels=c('1E-2','1E-4','1E-6','1E-8'))
splom(~log(d,10), pscales=list(p,p,p))
```

---

ibd.dataset *Calculate Identity by Descent for a Genotype Dataset*

Description

Estimates identity by descent for all sample pairs in a genotype dataset, from observed identity by state information.

Usage

```
ibd.dataset(dataset.name, binsz=1e6, part=1, parts=10,
            maf.min=0.1, hw.p.min=0.01, gt.rate.min=0.98, ...)
```

Arguments

`dataset.name`	the identifier for the dataset to analyze.
`binsz, part, parts`	passed to `fetch.gt.data`, to identify the regions to be analyzed.
`maf.min`	minimum minor allele frequency to be included in the analysis.
`hw.p.min`	minimum Hardy-Weinberg equilibrium P value for SNPs to be included in the analysis.
`gt.rate.min`	minimum call rate for SNPs to be included.
`...`	additional arguments for `ibd.gt.data`.

Details

This is a wrapper around `ibd.gt.data` that helps to select a suitable genome-wide autosomal subset of a genotype dataset for IBD analysis. The `maf.min`, `hw.p.min`, and `gt.rate.min` filters improve the IBD estimates by excluding problematic data that may have elevated error rates.

Value

A list of two square matrices with row and column names set to sample names in the input data, containing the estimated proportions of the genome with IBD=1 and IBD=2, for all pairs of samples in the dataset.

See Also

`ibd.plot`, `ibd.summary`, `ibd.outliers`.

Examples

```
gt.demo.check()
ibd <- ibd.dataset('Demo_1', min.snps=10, binsz=10e6, parts=1)
ibd.plot(ibd, jitter=0.005)
```

`ibd.gt.data` *Estimate Pairwise Identity by Descent for Genotype Data*

Description

Estimate identity-by-descent for all sample pairs in a set of genotype data.

Usage

```
ibd.gt.data(gt.data, binsz=1e6, min.snps=25, max.snps=50,
            min.gt=0.8, ibs.limit=2)
```

Arguments

gt.data	a data frame of genotypes from fetch.gt.data.
binsz	Size (in base pairs) of bins to use for estimating IBD.
min.snps	The minimum number of SNPs to consider when estimating IBD. Bins with fewer SNPs will be excluded.
max.snps	The maximim number of SNPs to consider. Additional SNPs in a bin will be ignored.
min.gt	minimum sample call rate for inclusion in the analysis.
ibs.limit	bins with average IBS ibs.limit fold higher than the median IBS are excluded due to low information content.

Details

Pairwise identity-by-descent (IBD) is estimated by subdividing the input data into intervals of intervals of size binsz, equally spaced across the genome. The algorithm determines the minimum number of alleles identical by state (IBS) for SNPs within each bin, and averages these values across bins. The minimum IBS value gives an upper limit on IBD for that bin, and approaches IBD as the number of assayed SNPs increases.

Determining IBD for close relatives requires only a fraction of the available data from a whole genome scan. Very accurate estimates of IBD are not particularly useful, because of intrinsic variability in the recombination process. The default values for binsz, part, and parts should not need to be changed.

Bins with few SNPs may not be sufficiently informative for IBD, if there is a substantial probability of unrelated samples sharing the same haplotype by chance. Rare genotyping errors can also impact apparent IBD, since a single error within a bin will change the minimum IBS. The min.snps setting ensures reasonable informativeness, and max.snps helps to limit the error rate.

The estimated genomic proportion with IBD=1 is biased upwards because a small proportion of bins are not sufficiently informative to distinguish IBD=0 from IBD=1. The estimate for IBD=2 seems to be less biased. The following table shows expected genomic IBD proportions for common familial relationships.

IBD=0	IBD=1	IBD=2	Relationship
1.00	0.00	0.00	unrelated
0.75	0.25	0.00	first cousins
0.50	0.50	0.00	half siblings
0.00	1.00	0.00	parent-child
0.25	0.50	0.25	full siblings
0.00	0.00	1.00	duplicate samples

It is important to note that the actual IBD proportions for most relative pairs (except for parent-child) show substantial natural variation, independent of the variation due to the estimation procedure.

Value

A list of two square matrices with row and column names set to sample names in the input data, containing the estimated proportions of the genome with IBD=1 and IBD=2, for all pairs of samples in the dataset.

See Also

ibd.plot, ibd.summary, ibd.outliers.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')
ibd <- ibd.gt.data(subset(gt, ploidy=='A'))
```

---

ibd.plot *Plot Identity by Descent Data*

Description

Plot identity by descent data for all sample pairs.

Usage

```
ibd.plot(ibd, jitter.amount, split=0.3,
         panel.width=list(0.8,'npc'))
```

Arguments

ibd	an IBD estimate from ibd.dataset.
jitter.amount	scale of random noise to be added to the IBD=2 values to resolve overlapping observations.
split	the proportion of plot height devoted to the IBD=1 histogram.
panel.width	the proportion of plot width to use for plotting data, as opposed to axes, tick labels, etc.

Details

Produces a two-panel plot, with the upper panel showing pairwise IBD=1 versus IBD=2 proportions, and the lower panel showing a histogram of the IBD=1 values. In the upper panel, outliers are colored by whether they are outliers on IBD=1, on IBD=2, or both. Outliers on IBD=1 are defined by a Bonferroni corrected P value of less than 0.05, assuming a normal distribution for unrelated individuals.

Value

A plot object of class "trellis".

See Also

ibd.dataset, ibd.summary, ibd.outliers.

Examples

```
gt.demo.check()
ibd <- ibd.dataset('Demo_1', min.snps=10, binsz=10e6, parts=1)
ibd.plot(ibd, jitter=0.005)
```

`ibd.summary` *Summarize Identity by Descent Analysis Results*

Description

Summarize results from an identity-by-descent analysis.

Usage

```
ibd.summary(ibd, alpha=0.05)
ibd.outliers(ibd, alpha=0.05)
```

Arguments

`ibd`	results from `ibd.dataset`.
`alpha`	a P value threshold for identifying IBD=1 outliers.

Details

`ibd.summary` and `ibd.outliers` both identify sample pairs with elevated genome-wide IBD=1 or IBD=2 proportions. For IBD=1, a threshold is selected assuming a normal distribution for unrelated individuals, using the specified `alpha` threshold with a Bonferroni correction. For IBD=2, an (arbitrary) threshold of 0.05 is used.

Unrelated pairs should have neither elevated; parent-sibling pairs, cousins and half sibs should have only IBD=1 elevated; duplicates should have only IBD=2 elevated; siblings should have both elevated.

Value

For `ibd.summary`, a table of counts of distinct sample pairs, indicating how many have neither IBD=1 or IBD=2 elevated; only elevated IBD=1; only elevated IBD=2; or both elevated.

For `ibd.outliers`, a data frame with 7 columns:

`rds.id.1, sample.name.1`
: identifiers for sample 1.

`rds.id.2, sample.name.2`
: identifiers for sample 2.

`grp`
: the type of outlier: one of `'IBD=1'`, `'IBD=2'`, `'Both'`.

`ibd.1`
: the inferred proportion of the genome with IBD=1.

`ibd.2`
: the inferred proportion of the genome with IBD=2.

See Also

`ibd.dataset`, `ibd.plot`.

Examples

```
gt.demo.check()
ibd <- ibd.dataset('Demo_1', min.snps=10, binsz=10e6, parts=1)
ibd.summary(ibd)
ibd.outliers(ibd)
```

`ibs.gt.data` *Calculate Pairwise Identity by State for Genotype Data*

Description

Calculates identity-by-state (IBS) for all sample pairs in a set of genotype data.

Usage

```
ibs(x, sample.mask=TRUE)
## S3 method for class 'gt.data':
ibs(x, sample.mask=TRUE)
## S3 method for class 'gt.dataset':
ibs(x, sample.mask=TRUE)
grr(x, sample.mask=TRUE)
```

Arguments

`x`	either a data frame of genotypes from `fetch.gt.data`, or a dataset description from `gt.dataset`.
`sample.mask`	an optional mask identifying a subset of samples to be included in the results.

Details

`ibs.gt.data` and `ibs.gt.dataset` assemble counts of markers versus the number of alleles IBS. `grr` computes the mean and standard deviation of IBS, as described in Abecasis *et al.* (2001).

Value

For `ibs.gt.data` and `ibs.gt.dataset`, a list of three square matrices of counts of markers with IBS=0, IBS=1, and IBS=2, for each sample pair.

For `grr`, a list of two square matrices giving the mean and standard deviation of the number of alleles IBS, for each sample pair.

References

Abecasis, G.R., Cherny, Stacey, S.S., Cookson, W.O.C., and Cardon, L.R. (2001) GRR: graphical representation of relationship errors. *Bioinformatics* **17**: 742-743.

See Also

`ibd.summary`, `gt.dataset`.

Examples

```
gt.demo.check()
ibs <- ibs(gt.dataset('Demo_1'))
str(ibs)
pt <- fetch.pt.data('Demo_1')
g <- grr(gt.dataset('Demo_1'), (pt$plate=='CEU'))
with(g, xyplot(sd[lower.tri(sd)]~mu[lower.tri(mu)]))
```

`if.na` *Conditional Element Selection for Missing Values*

Description

`if.na` returns a value with the same shape as `val`, populated from either `yes` or `no` depending on whether the element of `is.na(val)` is `TRUE` or `FALSE`.

Usage

```
if.na(val, yes, no=val)
```

Arguments

`val`	an object to be tested for `NA` values.
`yes`	return values for `NA` elements of `val`.
`no`	return values for non-`NA` elements of `val`.

Details

The result is equivalent to `ifelse(is.na(val), yes, no)`.

Value

A vector of the same length as `val`, where missing values in are taken from `yes`, and non-missing values are taken from `no` (or `val` itself, if `no` is missing).

References

Inspired by Oracle's `nvl()` and `nvl2()` functions.

See Also

`is.na`, `ifelse`.

Examples

```
x <- c(1:3,NA,NA)
if.na(x, 4)
if.na(x, 0, x+1)
```

init.gt.db *Initialize GT.DB Database*

Description

Create all the standard GT.DB database objects (tables, indexes) in an empty database.

Usage

```
init.gt.db()
```

Details

This should be called after connecting to a new database using dbConnect and use.gt.db. Scripts for creating GT.DB tables and indexes are installed under `library(help='gt.db')$path` in the 'schema' subdirectory.

See Also

dbConnect, use.gt.db, demo(setup.gt.demo).

Examples

```
## Not run:
# create new SQLite database in a temporary file
fn <- tempfile()
dbx <- dbConnect(dbDriver('SQLite'), fn)
# unlink it so it will go away at the end of the session
unlink(fn)
use.gt.db(dbx)
init.gt.db()
demo('setup.gt.demo')
## End(Not run)
```

jt.test *Jonckheere-Terpstra Nonparametric Test for Trend*

Description

Test for association between genotypes and a quantitative outcome, using the nonparametric Jonckheere-Terpstra test for ordered differences among genotype classes.

Usage

```
jt.test(x, y, alternative=c("two.sided", "decreasing", "increasing"),
        asymp=TRUE, correct=FALSE, perm=0, na.action=c("omit","fail"),
        permgraph=FALSE, permreps=FALSE)
```

Arguments

`x`	vector of quantitative response values.
`y`	group membership.
`alternative`	alternative hypothesis to be tested.
`asymp`	logical: use asymptotic formula for variance, or don't bother.
`correct`	logical: apply continuity correction?
`perm`	number of repetitions for a permutation test.
`na.action`	what to do with missing data.
`permgraph`	logical: draw a histogram of the permutations?
`permreps`	logical: return the permutations?

Details

The Jonckheere-Terpstra test computes Mann-Whitney rank sum statistics for ordered group labels, and does a two-sided test on the sum of those statistics. This tests for a monotonic trend in response as a function of group membership. The model formula should be of the form `outcome~genotype` without additional terms.

Value

A list of class htest, with the following components:

`statistic`	the observed J-T statistic.
`alternative`	same as input.
`method`	the string: `"Jonckheere-Terpstra test"`.
`data.name`	the names of the input data.
`EH`	the expected test statistic based on sample size.
`VH`	variance (adjusted for ties if necessary).
`p.value`	asymptotic p-value.

References

http://tolstoy.newcastle.edu.au/R/help/06/06/30112.html.

See Also

`kruskal.test`, `score.kruskal`.

Examples

```
x <- rnorm(30) + c(rep(1,10),rep(2,20))
y <- c(rep(1,10),rep(2,10),rep(3,10))
jt.test(x, y)
```

keep.attr *Keep User Attributes*

Description

This function creates objects with the property that they more systematically preserve user attributes when they are indexed.

Usage

```
keep.attr(.Data, ..., .Attr=NULL)
kept.attr(x)
## S3 method for class 'keep.attr':
x[...]
## S3 replacement method for class 'keep.attr':
x[...] <- value
```

Arguments

.Data	an R object for which attributes are to be kept.
...	attributes in key=value form to be attached to the object.
.Attr	additional attributes to attach, collected into a list.
x	an object of class 'keep.attr'.
value	a suitable replacement value.

Value

For keep.attr, the original object with additional class 'keep.attr', with additional attributes attached. Indexing on this object will propagate all user specified attributes, so long as the indexing operation returns an object of the same class. Thus, for a data frame, attributes will be preserved for operations that return a new data frame.

For kept.attr, a list of all the user defined attributes associated with the object.

See Also

attributes.

Examples

```
d <- data.frame(a=c(1,2,3), b=c(4,5,6), c=c(7,8,9))
d <- structure(d, xyz='something')
e <- keep.attr(d, abc='more')
d[3] <- 5
e[3] <- 5
str(d)
str(d[1])
str(d[,-1])
str(d[,1])
str(e)
str(e[1])
str(e[,-1])
str(e[,1])
```

`ld.gt.data` *Compute Pairwise Linkage Disequilibrium*

Description

Takes genotype data as returned by `fetch.gt.data` and computes a matrix of pairwise linkage disequilibrium values.

Usage

```
ld.gt.data(g1, g2=g1, outer=TRUE,
           measure=c('rsqr','dprime','delta','pvalue',
                     'lod','none','failed'),
           method=c('iterate','exact'), epsilon=1e-6, max.it=100)
```

Arguments

`g1, g2`	genotype data from `fetch.gt.data`.
`outer`	logical: specifies whether result should be an outer product of `g1` and `g2`.
`measure`	the measure of LD to report. See details.
`method`	the method to use for calculating LD.
`max.it`	the maximum number of EM iterations to perform.
`epsilon`	convergence criterion for the EM algorithm.

Details

LD is computed between unphased genotypes in `g1` and `g2` under an assumption of Hardy Weinberg equilibrium. Two algorithms are implemented: an iterative EM solution, and an exact solution.

All markers should have consistent ploidy status. The LD calculations account for haploid and diploid genotypes by using gender information at sex-linked loci.

With the iterative algorithm, a warning is printed if some pairwise LD calculations do not converge in the specified number of iterations. If `measure='failed'`, then the function returns an array of logical values indicating which cases failed to converge.

For the exact algorithm, the most-likely result is reported. With extreme deviations from HWE, it is possible for more than one solution to have the same likelihood, and in these cases, the one that is reported is essentially arbitrary.

Value

If `outer` is `TRUE`, then the result is a matrix of LD values with one row per element of `g1` and one column per element of `g2`. If `outer` is `FALSE`, then the result is a vector of values obtained by comparing corresponding elements of `g1` with `g2` with recycling of the shorter argument. This will usually only make sense when the lengths of `g1` and `g2` are either equal or equal to `1`.

The number of pairwise calculations that failed to converge is returned in an attribute with the name `'failed'`.

References

Weir, B.S. (1996) *Genetic Data Analysis II*. Sinauer, Sunderland, MA.

Gaunt, T. R., Rodriguez, S., & Day, I. N. M. (2007) Cubic exact solutions for the estimation of pairwise haplotype frequencies: implications for linkage disequilibrium analyses and a web tool 'CubeX'. *BMC Bioinformatics* **8**: 428.

CubeX. http://www.oege.org/software/cubex/.

See Also

fetch.gt.data, ld.plot.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')[1:6,]
ld.gt.data(gt, measure='rsqr')
ld.gt.data(gt, measure='dprime')
ld.gt.data(gt, measure='pvalue')
ld.gt.data(gt[1:3,], gt[3:5,], outer=TRUE)
ld.gt.data(gt[1:3,], gt[3:5,], outer=FALSE)
```

---

ld.plot *Pairwise Linkage Disequilibrium Plot*

Description

Generates a level plot of the strength of pairwise linkage disequilibrium among a set of SNPs.

Usage

```
ld.plot(gt.data, col=gray(seq(1,0,-0.01)), measure='rsqr',
        rotate=FALSE, equal=TRUE, colorkey=list(height=0.5),
        scales, ...)
```

Arguments

gt.data	genotype data from fetch.gt.data.
col	a color gradient.
measure	the measure(s) of LD to be plotted. See details.
rotate	logical: indicates if the plot should be rotated clockwise 45 degrees.
equal	logical: indicates if the LD matrix should be displayed with equal SNP spacing, or in genomic coordinates.
colorkey, scales	passed to levelplot.
...	additional arguments to pass to levelplot.

Details

SNPs are sorted by accession and contig position. Values of measure are as in ld.gt.data. If measure has two elements, then the first value is plotted in the upper left triangle and the second value is plotted in the lower right triangle.

Value

A plot object of class "trellis".

See Also

ld.gt.data levelplot.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')
pt <- fetch.pt.data('Demo_2')
gt <- mask.gt.data(gt, pt$plate=='CEU')
gs <- summary.gt.data(gt)
ld.plot(subset(gt, pass & gs$freq.a>0.05 & gs$freq.b>0.05),
        measure=c('dprime','rsqr'), rotate=TRUE, equal=FALSE)
```

---

ld.prune *Prune SNP List to Limit Linkage Disequilibrium*

Description

Takes genotype data as returned by fetch.gt.data and eliminates SNPs in high LD across a sliding window.

Usage

```
ld.prune(gt.data, min.maf=0.01, max.rsqr=0.2, span=20, subsets=TRUE)
```

Arguments

gt.data	genotype data for a single contiguous sequence, from fetch.gt.data.
min.maf	a minimum minor allele frequency for inclusion.
max.rsqr	the maximum allowed R-squared value.
span	the number of sequential SNPs to check for LD.
subsets	an optional list of masks identifying groups of subjects within which LD should be checked.

Details

LD is computed over a sliding window and SNPs are iteratively eliminated to guarantee that each remaining SNP has the specified maximum R-squared value for at least span SNPs on either side, ordered by sequence position.

If multiple subsets are specified, then the inclusion criteria are enforced for ALL subsets.

Value

A new data frame based on gt.data, containing a subset of SNPs satisfying the LD limitations.

See Also

fetch.gt.data, ld.gt.data.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')
pt <- fetch.pt.data('Demo_2')
by.plate <- list(pt$plate=='CEU', pt$plate=='YRI',
                 pt$plate=='J+C' & !pt$is_dup)
x <- ld.prune(gt, subsets=by.plate)
nrow(gt)
nrow(x)
```

---

ls.assay *List Assay Definitions*

Description

List definitions of assays associated with a genotyping platform.

Usage

```
ls.assay(platform.name, assay.group='%', show.ids=FALSE)
```

Arguments

platform.name	a genotyping platform name.
assay.group	an SQL LIKE expression for matching assay group names.
show.ids	logical: indicates whether to include values of database keys.

Value

A data frame with one row per assay, and 4 or 6 columns:

assay.group.id	if show.ids is set: a unique integer key for the assay group associated with this assay.
assay.group	the assay group associated with this assay.
assay.id	if show.ids is set: the unique integer ID for this assay.
assay.name	the assay name.
alleles	a slash separated list of allele sequences.
probe.seq	the genomic flanking sequence for the assay, with the variant position denoted by an underscore ('_').

See Also

ls.assay.group, ls.assay.position, mk.assay.

Examples

```
gt.demo.check()
head(ls.assay('Demo_Set_1'))
head(ls.mapping('Demo_Set_1', show.ids=TRUE))
```

---

ls.assay.group *List Assay Groups*

Description

This returns a list of assay groups for the specified genotyping platform.

Usage

```
ls.assay.group(platform.name, show.ids=FALSE)
```

Arguments

platform.name	a genotyping platform name.
show.ids	logical: indicates whether to include values of database keys.

Value

A data frame with one row per assay group, and 4 or 5 columns:

assay.group.id	if show.ids is set: a unique integer key for this assay group.
assay.group	a short, unique identifier for the assay group.
description	a free-text description of the assay group.
created.by	the user name that created the assay group.
created.dt	the creation date of the assay group.

See Also

mk.assay.group.

Examples

```
gt.demo.check()
ls.assay.group('Demo_Set_1')
ls.assay.group('Demo_Set_1', show.ids=TRUE)
```

`ls.assay.position` *List Assay Positions*

Description

List mapped genomic positions of assays associated with a genotyping platform.

Usage

```
ls.assay.position(platform.name, mapping.name, show.ids=FALSE)
```

Arguments

`platform.name`	a genotyping platform name.
`mapping.name`	an identifier for the assay mapping to use.
`show.ids`	logical: indicates whether to include values of database keys.

Details

If `mapping.name` is missing, it will default to the current (visible) mapping for the specified platform, if that is unique.

Value

A data frame with one row per assay, and 7 or 8 columns:

`assay.id`	if `show.ids` is set: the unique integer ID for this assay.
`assay.name`	the assay name.
`scaffold`	a string identifying the sequence to which the assay is mapped.
`position`	a one-based position within the specified scaffold.
`strand`	either `"+"`, `"-"`, or `NA`.
`ploidy`	describes the expected allele count: see `mk.assay.position`.
`dbsnp.rsid`	the dbSNP refSNP cluster ID for this assay.
`dbsnp.orient`	the orientation of the assay compared to the dbSNP cluster: either `"+"`, `"-"`, or `NA`.

See Also

`ls.assay`, `mk.assay.position`.

Examples

```
gt.demo.check()
head(ls.assay.position('Demo_Set_1'))
```

`ls.dataset` *List Genotype Datasets*

Description

This returns a list of genotype datasets defined in the current database.

Usage

```
ls.dataset(project.name='%', dataset.name='%',
           show.all=FALSE, show.ids=FALSE)
```

Arguments

`project.name`	an SQL `LIKE` expression for matching project names.
`dataset.name`	an SQL `LIKE` expression for matching dataset names.
`show.all`	logical: indicates if hidden datasets and members of hidden projects should be included in the output.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per dataset, and 8 or 11 columns:

`dataset.id`	if `show.ids` is set: the unique integer key for this dataset.
`dataset.name`	a short, unique identifier for the dataset.
`project.id`	if `show.ids` is set: the unique integer key for the project associated with this dataset.
`project.name`	the project associated with this dataset.
`platform.id`	if `show.ids` is set: the unique integer key for the platform associated with this dataset.
`platform.name`	the genotyping platform associated with this dataset.
`description`	a free-text description of the dataset.
`raw.layout`	a keyword indicating how raw data associated with individual assays is structured.
`is.hidden`	logical: indicates if the project is hidden.
`created.by`	the user name that created the dataset.
`created.dt`	the creation date of the dataset.

See Also

`ls.project`, `ls.platform`, `mk.dataset`.

Examples

```
gt.demo.check()
ls.dataset()
ls.dataset(show.ids=TRUE)
```

`ls.mapping` *List Assay Mapping Sets*

Description

This returns a list of mapping sets for the specified genotyping platform.

Usage

```
ls.mapping(platform.name, show.all=FALSE, show.ids=FALSE)
```

Arguments

`platform.name`	a genotyping platform name.
`show.all`	logical: indicates if hidden mapping sets should be included in the output.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per mapping, and 5 or 6 columns:

`mapping.id`	if `show.ids` is set: a unique integer key for this mapping.
`mapping`	a short, unique identifier for the mapping.
`description`	a free-text description of the mapping.
`assembly`	a short identifier for the target assembly.
`created.by`	the user name that created the mapping.
`created.dt`	the creation date of the mapping.

See Also

`mk.mapping`.

Examples

```
gt.demo.check()
ls.mapping('Demo_Set_1')
ls.mapping('Demo_Set_1', show.ids=TRUE)
```

`ls.platform` *List Genotyping Platforms*

Description

This returns a list of genotyping platforms defined in the current database.

Usage

```
ls.platform(platform.name='%', show.ids=FALSE)
```

Arguments

`platform.name`	an SQL `LIKE` expression for matching platform names.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per platform, and 6 or 7 columns:

`platform.id`	if `show.ids` is set: a unique integer key for this platform.
`platform.name`	a short, unique identifier for the platform.
`description`	a free-text description of the platform.
`assay.groups`	the number of assay groups for this platform.
`datasets`	a count of the number of datasets using this platform.
`created.by`	the user name that created the platform.
`created.dt`	the creation date of the platform.

See Also

`mk.platform`.

Examples

```
gt.demo.check()
ls.platform()
ls.platform(show.ids=TRUE)
```

`ls.prcomp` *List Principal Components Result Sets*

Description

This returns a list of principal components analysis result sets in the database for a specified genotype dataset.

Usage

```
ls.prcomp(dataset.name, prcomp.name='%', show.all=FALSE, show.ids=FALSE)
```

Arguments

`dataset.name`	a genotype dataset name.
`prcomp.name`	an SQL `LIKE` expression for matching principal components analysis set names.
`show.all`	logical: indicates if hidden result sets should be included in the output.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per result set, and 9 or 10 columns:

`prcomp.id`	if `show.ids` is set: the unique integer key for this test set.
`prcomp.name`	a short identifier for the result set, which is unique for the specified genotype dataset.
`description`	a free-text description of the result set.
`fn.call`	the text of the call used to perform the analysis.
`components`	the number of components retained in the database.
`samples`	the number of samples included in the analysis.
`assays`	the number of assays included in the analysis.
`is.hidden`	logical: indicates if the result set is hidden.
`created.by`	the user name that created the result set.
`created.dt`	the creation date of the result set.

See Also

`fetch.prcomp`, `store.prcomp`, `rm.prcomp`.

Examples

```
# FIXME
```

`ls.project` *List Genotyping Projects*

Description

This returns a list of genotyping projects defined in the current database.

Usage

```
ls.project(project.name='%', show.all=FALSE, show.ids=FALSE)
```

Arguments

`project.name`	an SQL `LIKE` expression for matching project names.
`show.all`	logical: indicates if hidden projects should be included in the output.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per project, and 6 or 7 columns:

`project.id`	if `show.ids` is set: a unique integer key for this project.
`project.name`	a short, unique identifier for the project.
`description`	a free-text description of the project.
`datasets`	a count of the number of datasets under this project.
`is.hidden`	logical: indicates if the project is hidden.
`created.by`	the user name that created the project.
`created.dt`	the creation date of the project.

See Also

`mk.project`.

Examples

```
gt.demo.check()
ls.project()
ls.project(show.ids=TRUE)
```

`ls.sample` *List Samples in a Genotype Dataset*

Description

This returns a description of all samples defined in the specified genotype dataset.

Usage

```
ls.sample(dataset.name, show.ids=FALSE)
```

Arguments

`dataset.name`	the unique identifier for the dataset.
`show.ids`	logical: indicates whether to include values of database keys.

Value

A data frame with one row per dataset, and 4 or 6 columns:

`sample.id`	if `show.ids` is set: the unique integer key for this sample.
`sample.name`	a short, unique identifier for the sample.
`subject.id`	if `show.ids` is set: the unique integer key for the subject associated with this sample.
`subject.name`	a short, unique identifier for the subject.
`gender`	a factor indicating the gender of this sample.
`position`	a 1-based integer indicating the position of this sample's genotyping data in the genotype arrays for this dataset.

See Also

`ls.dataset`, `mk.sample`.

Examples

```
gt.demo.check()
head(ls.sample('Demo_1'))
head(ls.sample('Demo_1', show.ids=TRUE))
```

ls.subject *List Subjects in a Genotyping Project*

Description

This returns a list of subjects defined in the specified genotyping project.

Usage

```
ls.subject(project.name, show.ids=FALSE)
```

Arguments

project.name	a unique project identifier.
show.ids	logical: indicates whether to include values of database keys.

Value

A data frame with one row per dataset, and 1 or 2 columns:

subject.id	if show.ids is set: the unique integer key for this subject.
subject.name	a short, unique identifier for the subject.

See Also

ls.project, mk.subject.

Examples

```
gt.demo.check()
head(ls.subject('Demo'))
head(ls.subject('Demo', show.ids=TRUE))
```

ls.test *List Association Test Result Sets*

Description

This returns a list of association test result sets in the database for a specified genotype dataset.

Usage

```
ls.test(dataset.name, test.name='%', show.all=FALSE, show.ids=FALSE)
```

Arguments

dataset.name	a genotype dataset name.
test.name	an SQL LIKE expression for matching test set names.
show.all	logical: indicates if hidden result sets should be included in the output.
show.ids	logical: indicates whether to include values of database keys.

Value

A data frame with one row per test set, and 8 or 9 columns:

test.id	if show.ids is set: the unique integer key for this test set.
test.name	a short identifier for the test set, unique when combined with term for the specified genotype dataset.
description	a free-text description of the test set.
fit	the type of model fit: 'lm', 'glm', etc.
model	the model formula used for scoring each genotype assay.
term	used for differentiating among multipe test result sets associated with a single analysis
is.hidden	logical: indicates if the test set is hidden.
created.by	the user name that created the test set.
created.dt	the creation date of the test set.

See Also

fetch.test.scores, store.test.scores, rm.test.

Examples

```
# FIXME
```

---

mask.gt.data *Mask Sample Genotypes*

Description

Given a data frame of genotype information, this masks out a subset of samples based on a logical vector.

Usage

```
mask.gt.data(gt.data, sample.mask, repack=FALSE)
```

Arguments

gt.data	a data frame of genotype information.
sample.mask	either a logical vector, or a string of T/F values, indicating which samples should be kept.
repack	logical: indicates whether masked-out samples should be entirely removed from the result, or left in place.

Details

If repack is FALSE, then genotypes of masked-out samples are coded as missing. If repack is TRUE, then masked-out samples are deleted from the result.

Value

An updated version of `gt.data` with genotypes for masked-out samples either marked as missing or deleted.

See Also

`fetch.gt.data`, `summary.gt.data`.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')
pt <- fetch.pt.data('Demo_2')
head(summary.gt.data(gt))
gm <- mask.gt.data(gt, pt$gender=='M')
gf <- mask.gt.data(gt, pt$gender=='F')
head(summary.gt.data(gm))
head(summary.gt.data(gf))
```

`mask.str` *Mask Character Strings*

Description

Mask selected positions from elements of a character vector.

Usage

```
mask.str(str, mask, ch='x')
```

Arguments

`str`	a character vector to be masked.
`mask`	either a logical vector or character mask, with a 1:1 correspondence to character positions in elements of `str`.
`ch`	a character that will replace masked positions in `str`.

Value

A character vector with the same form as `str`, with positions specified as `FALSE` in `mask` either replaced by the first character in `ch`, or squeezed out entirely, if `ch` is an empty string.

See Also

`as.mask`.

Examples

```
mask.str('12345678', 'TFTFTFTF', '_')
mask.str('12345678', 'TFTFTFTF', '')
```

`match.gt.data` *Identify Equivalent Genotyping Assays*

Description

This identifies assays from two genotype datasets that correspond to the same polymorphisms, based either on position or dbSNP rsIDs.

Usage

```
match.gt.data(gt.data.1, gt.data.2, by=c('position','dbsnp.rsid'))
```

Arguments

`gt.data.1, gt.data.2`	data frames with assay information to be compared.
`by`	the method to use to identify equivalent assays.

Details

This first identifies assays with corresponding positions (or dbSNP rsIDs) across the two datasets. For each matching pair, the reported genomic orientation and alleles are checked for compatibility. Putative matches are rejected if the alleles are inconsistent.

Value

A data frame with four columns. The first two columns contain corresponding assay names from the two datasets. The remaining two columns are:

`is.flipped`	logical: indicates that the assays are in opposing genomic orientations.
`is.swapped`	logical: indicates that the A/B alleles are interchanged between assays.

See Also

`fetch.gt.data`, `orient.gt.data`.

Examples

```
gt.demo.check()
g1 <- fetch.gt.data('Demo_2')
g2 <- orient.gt.data(g1, flip=(g1$dbsnp.orient == '-'))
head(match.gt.data(g1,g2))
```

`mk.assay` *Create Genotyping Assay Definitions*

Description

`mk.assay` defines genotyping assays associated with an already defined genotyping platform.

Usage

```
mk.assay(platform.name, data)
```

Arguments

`platform.name`	a short unique identifier for the platform.
`data`	a data frame with one row per assay. See details.

Details

A data frame of assay information should provide four columns:

assay.group an assay group defined with `mk.assay.group`.

assay.name a unique name (within this platform) for the assay.

alleles a slash separated list of valid alleles.

probe.seq the genomic flanking sequence for the assay, with the variant position denoted by an underscore (`'_'`).

Value

If successful, the number of rows inserted into the assay table.

See Also

`ls.assay`, `mk.assay.position`.

Examples

```
## Not run:
data(demo_01)
head(assay.def.01)
mk.assay('Demo_Set_1', assay.def.01)
## End(Not run)
```

`mk.assay.data` *Create Genotyping Assay Data*

Description

`mk.assay.data` defines genotype data associated with already defined genotype assays.

Usage

```
mk.assay.data(dataset.name, data)
```

Arguments

`dataset.name`	a short unique identifier for the genotype dataset.
`data`	a data frame with one row per assay. See details.

Details

A data frame of genotype data can provide up to five columns of information:

assay.id the unique integer ID for this assay.

flags an integer value composed of single-bit flags.

genotype a string of a/h/b/n/x genotypes.

qscore a string of packed quality scores.

raw.data a string containing packed raw data (such as signal intensities or read counts).

An `assay.name` column can be supplied in place of `assay.id`.

Two raw data layouts are currently supported. The 'signal' layout consists of a 16-bit unsigned little endian value for each allele. The 'seqread' layout consists of 8-bit unsigned counts for forward and reverse orientations for each allele, representing read counts from a sequencing experiment.

Value

If successful, the number of rows inserted into the assay data table.

See Also

`mk.assay.position`, `mk.assay`, `fetch.gt.data`.

Examples

```
## Not run:
data(demo_01)
head(assay.pos.01)
mk.mapping('Demo_Set_1', 'Demo_Set_1_b36',
           'Platform 1 Mapping to NCBI Build 36')
mk.assay.position('Demo_Set_1', data=assay.pos.01)
## End(Not run)
```

`mk.assay.group` *Create or Remove a Genotype Assay Group*

Description

These functions insert or remove entries from the genotyping assay group table.

Usage

```
mk.assay.group(platform.name, assay.group, description)
rm.assay.group(platform.name, assay.group)
```

Arguments

`platform.name`	the unique platform identifier.
`assay.group`	an identifier for the assay group.
`description`	a free-text description of the assay group.

Details

An assay group and its constituent assays can only be removed if it is not in use. Any mappings or datasets that refer to these assays must be explicitly deleted first.

Value

If successful, the number of rows inserted in or removed from the assay group table (i.e., typically, 1).

See Also

`ls.assay.group`, `ls.platform`, `mk.platform`, `mk.assay`.

Examples

```
## Not run:
mk.platform('My_600K', 'My 600K Array Set')
mk.assay.group('My_600K', 'Des01', '300K Array 1 of 2')
mk.assay.group('My_600K', 'Des02', '300K Array 2 of 2')
rm.assay.group('My_600K', c('Des01','Des02'))
rm.platform('My_600K')
## End(Not run)
```

mk.assay.position *Create Genotyping Assay Positions*

Description

mk.assay.position defines map positions associated with genotyping assays.

Usage

```
mk.assay.position(platform.name, mapping.name, data)
```

Arguments

platform.name	a short unique identifier for the platform.
mapping.name	the platform mapping to be populated.
data	a data frame with one row per assay. See details.

Details

If mapping.name is missing, it will default to the current (visible) mapping for the specified platform, if that is unique. A data frame of assay positions can provide up to seven columns of information:

assay.name the assay identifier.

scaffold a string identifying the sequence to which the assay is mapped.

position a one-based position within the specified scaffold.

strand either "+", "−", or NA.

ploidy describes the expected allele count: it can take values "A" (autosomal), "M" (mitochondrial, or haploid), "X" (diploid in females, haploid in males), or "Y" (haploid in males, absent in females). Pseudoautosomal positions are coded as "A".

dbsnp.rsid the dbSNP refSNP cluster ID for this assay.

dbsnp.orient the orientation of the assay compared to the dbSNP cluster: either "+", "−", or NA.

Value

If successful, the number of rows inserted into the assay map position table.

See Also

ls.assay.position, mk.mapping, mk.assay.

Examples

```
## Not run:
data(demo_01)
head(assay.pos.01)
mk.mapping('Demo_Set_1', 'Demo_Set_1_b36',
           'Platform 1 Mapping to NCBI Build 36')
mk.assay.position('Demo_Set_1', data=assay.pos.01)
## End(Not run)
```

mk.attr *Create, Remove, and List Attribute Definitions*

Description

`mk.attr` and `rm.attr` create or remove attribute definitions associated with various types of genotype data objects, such as samples and subjects, and `ls.attr` lists attribute definitions.

Usage

```
mk.attr(target, parent.name, attr.name, datatype,
        levels, description, is.hidden=FALSE)
rm.attr(target, parent.name, attr.name)
ls.attr(target, parent.name, show.all=FALSE, show.ids=FALSE)
```

Arguments

`target`	the type of attribute to be created: i.e., `'subject'` or `'sample'`.
`parent.name`	the name of the parent container for the new attribute, i.e. a project or dataset name.
`attr.name`	a short identifier for the new attribute.
`datatype`	the datatype for attribute values: `'number'`, `'string'`, `'boolean'`, or `'factor'`.
`levels`	for factors, a string describing the factor levels, formed by concatenating a comma-separated list of single quoted strings.
`description`	a free text description of the attribute.
`is.hidden`	logical: indicates if the attribute is hidden.
`show.all`	logical: indicates if hidden attributes should be included in the output.
`show.ids`	logical: indicates whether to include values of database keys.

Details

These functions are usually not called directly; instead, there are helper functions for each type of attribute (i.e. `mk.subject.attr`, `rm.subject.attr`, `ls.subject.attr` that are more convenient.

Multiple attributes can be created or removed with single calls to `mk.attr` and `rm.attr`. The `attr.name`, `datatype`, `levels`, and `description` arguments can all be vectors and the usual recycling rules apply.

Value

For `mk.attr` and `rm.attr`, if successful, the number of rows inserted or deleted from the target attribute table.

For `ls.attr`, a data frame with 7 columns:

`attr.name`	the attribute name.
`datatype`	one of `'number'`, `'string'`, `'boolean'`, or `'string'`.
`levels`	a single-quoted comma-separated list of factor levels.

description	a free text description.
is.hidden	logical: indicates if the attribute is hidden.
created.by	the user name that created the attribute.
created.dt	the creation date of the attribute.

See Also

mk.subject.attr, mk.sample.attr.

Examples

```
## Not run:
mk.attr('subject', 'Demo', 'stuff1', 'number',
        description='Something numeric')
mk.attr('subject', 'Demo', 'stuff2', 'factor', "'a','b','c'",
        description='A factor')
ls.attr('subject', 'Demo')
rm.attr('subject', 'Demo', c('stuff1','stuff2'))
## End(Not run)
```

---

mk.dataset *Create or Remove Genotype Datasets*

Description

These functions insert or delete entries from the genotype dataset table.

Usage

```
mk.dataset(dataset.name, project.name, platform.name,
           description, raw.layout=c(NA,'signal','seqread'),
           is.hidden=FALSE)
rm.dataset(dataset.name)
```

Arguments

dataset.name	a short unique identifier for the dataset.
project.name	the project associated with this dataset.
platform.name	the genotyping platform for this dataset.
description	a free-text description of the dataset.
raw.layout	a keyword describing how raw data associated with individual assays is structured.
is.hidden	logical: indicates if the dataset is hidden.

Details

While datasets are nested within projects, dataset names must be globally unique.

Value

If successful, the number of rows inserted in or deleted from the dataset table (i.e., 1).

See Also

ls.dataset, set.hidden.

Examples

```
## Not run:
mk.dataset('Demo_9', project.name='Demo', platform.name='Demo',
           description='Demo Dataset #9')
rm.dataset('Demo_9')
## End(Not run)
```

---

mk.mapping *Create or Remove a Mapping for a Genotyping Platform*

Description

These functions insert or remove entries from the platform mapping table.

Usage

```
mk.mapping(platform.name, mapping.name, description,
           assembly, is.hidden=FALSE)
rm.mapping(platform.name, mapping.name)
```

Arguments

platform.name	the unique platform identifier.
mapping.name	an identifier for the mapping.
description	a free-text description of the mapping.
assembly	the target assembly for the mapping. This is used to determine whether two mappings have compatible coordinates.
is.hidden	logical: indicates if the mapping is hidden.

Details

A platform may have multiple mapping sets, for different assemblies, or even for the same assembly with different alignment parameters. Removing a mapping results in deletion of all the associated assay map positions.

Value

If successful, the number of rows inserted in or removed from the mapping table (i.e., typically, 1).

See Also

ls.mapping, mk.platform, mk.assay.position.

Examples

```
## Not run:
mk.platform('My_600K', 'My 600K Array Set')
mk.mapping('My_600K', 'b36_1', 'Build 36 Vender Annotations', 'ncbi_b36')
rm.mapping('My_600K', 'b36_1')
rm.platform('My_600K')
## End(Not run)
```

---

mk.platform *Create or Remove a Genotyping Platform*

Description

These functions insert or remove entries from the genotyping platform table.

Usage

```
mk.platform(platform.name, description)
rm.platform(platform.name)
```

Arguments

platform.name	a short unique identifier for the platform.
description	a free-text description of the platform.

Value

If successful, the number of rows inserted in or removed from the platform table (i.e., 1).

See Also

ls.platform, mk.assay.group.

Examples

```
## Not run:
mk.platform('GT_800K', 'My 800K Genotyping Platform')
rm.platform('GT_800K')
## End(Not run)
```

mk.project — *Create or Remove a Genotyping Project*

Description

These functions insert or remove entries from the project table.

Usage

```
mk.project(project.name, description, is.hidden=FALSE)
rm.project(project.name)
```

Arguments

project.name	a short unique identifier for the project.
description	a free-text description of the project.
is.hidden	logical: indicates if the project is hidden.

Value

If successful, the number of rows inserted or removed from the project table (i.e., 1).

See Also

ls.project, set.hidden.

Examples

```
## Not run:
mk.project('Demo_9', 'Demo Genome-Wide Association Study')
rm.project('Demo_9')
## End(Not run)
```

mk.sample — *Create or Remove Dataset Samples*

Description

mk.sample and rm.sample insert or remove samples from a genotype dataset.

Usage

```
mk.sample(dataset.name, data)
rm.sample(dataset.name, sample.name)
```

Arguments

dataset.name	a short unique identifier for the dataset.
data	a data frame with one row per sample. See details.
sample.name	a vector of sample identifiers.

Details

A data frame of sample information should have (at least) four columns: `sample.name`, `subject.name`, `gender`, and `position`. This information is important because it affects how various functions interpret genotyping data in this dataset. Sample gender is used to determine ploidy for sex linked assays, and the position is used to index arrays of packed genotype data. Positions are 1-based integers.

Value

If successful, the number of rows inserted or deleted from the sample table.

See Also

`ls.sample`, `mk.subject`.

Examples

```
## Not run:
s <- data.frame(
    sample.name='NA_12345_1'
    subject.name='NA_12345',
    gender='M',
    position=1
)
mk.sample('Demo_1', s)
rm.sample('Demo_1, s$sample.name)
## End(Not run)
```

`mk.sample.attr` *Create, Remove, or List Sample Attribute Definitions*

Description

`mk.sample.attr` and `rm.sample.attr` create or remove sample attributes associated with a particular genotype dataset, and `ls.sample.attr` lists sample attributes for this dataset.

Usage

```
mk.sample.attr(dataset.name, data, description=names(data))
rm.sample.attr(dataset.name, attr.name)
ls.sample.attr(dataset.name)
```

Arguments

`dataset.name`	the unique identifier for a dataset.
`data`	a data frame with one or more columns to be used to model the new attributes. See details.
`description`	a vector of free-text descriptions with one element per column in `data`.
`attr.name`	a vector of attribute names to be removed.

Details

These functions are wrappers around `mk.attr`, `rm.attr`, and `ls.attr`. A call to `mk.sample.attr` results in creation of a new attribute for each column in the `data` argument, based on the data type, factor levels, etc of that column.

Value

For `mk.sample.attr` and `rm.sample.attr`, the number of attribute definitions inserted or removed from the sample attribute table.

For `ls.sample.attr`, a data frame with 7 columns describing subject attributes defined for the specified project, as defined in `ls.attr`.

See Also

`mk.attr`, `rm.attr`, `ls.attr`.

Examples

```
## Not run:
sd <- fetch.sample.data('Demo_1')
sd$stuff.1 <- rep(1:3, length.out=nrow(sd))
sd$stuff.2 <- factor(sd$stuff.1, levels=1:3, labels=c('a','b','c'))
sd$stuff.3 <- as.character(sd$stuff.2)
str(sd)
mk.sample.attr('Demo_1', sd[c('stuff.1','stuff.2','stuff.3')])
ls.sample.attr('Demo_1')
rm.sample.attr('Demo_1', c('stuff.1','stuff.2','stuff.3'))
## End(Not run)
```

`mk.subject` *Create or Remove Project Subjects*

Description

`mk.subject` and `rm.subject` insert or remove subjects from a genotyping project.

Usage

```
mk.subject(project.name, data)
rm.subject(project.name, subject.name)
```

Arguments

`project.name`	a short unique identifier for the project.
`data`	a data frame with a `subject.name` column.
`subject.name`	a vector of subject identifiers.

Value

If successful, the number of rows inserted or deleted from the subject table.

See Also

ls.subject, mk.sample.

Examples

```
## Not run:
mk.subject('Demo_1', data.frame(subject.name='NA_12345'))
rm.subject('Demo_1', 'NA_12345')
## End(Not run)
```

---

mk.subject.attr *Create, Remove, or List Subject Attribute Definitions*

Description

mk.subject.attr and rm.subject.attr create or remove subject attributes associated with a particular genotyping project, and ls.subject.attr lists subject attributes for this project.

Usage

```
mk.subject.attr(project.name, data, description=names(data))
rm.subject.attr(project.name, attr.name)
ls.subject.attr(project.name)
```

Arguments

project.name	the unique identifier for a project.
data	a data frame with one or more columns to be used to model the new attributes. See details.
description	a vector of free-text descriptions with one element per column in data.
attr.name	a vector of attribute names to be removed.

Details

These functions are wrappers around mk.attr and rm.attr, and ls.attr. A call to mk.subject.attr results in creation of a new attribute for each column in the data argument, based on the data type, factor levels, etc of that column.

Value

For mk.sample.attr and rm.sample.attr, the number of attribute definitions inserted or removed from the sample attribute table.

For ls.sample.attr, a data frame with 7 columns describing subject attributes defined for the specified project, as defined in ls.attr.

See Also

mk.attr, rm.attr, ls.attr.

Examples

```
## Not run:
sd <- fetch.subject.data('Demo')
sd$stuff.1 <- rep(1:3, length.out=nrow(sd))
sd$stuff.2 <- factor(sd$stuff.1, levels=1:3, labels=c('a','b','c'))
sd$stuff.3 <- as.character(sd$stuff.2)
str(sd)
mk.subject.attr('Demo', sd[c('stuff.1','stuff.2','stuff.3')])
ls.subject.attr('Demo')
rm.subject.attr('Demo', c('stuff.1','stuff.2','stuff.3'))
## End(Not run)
```

---

na.if *Conditional Conversion to Missing Values*

Description

`na.if` returns a value with the same shape as `v1`, with values equal to `v2` set to `NA`.

Usage

```
na.if(v1, v2)
```

Arguments

`v1`	an object to be tested.
`v2`	a value to be compared to elements of `v1`.

Details

The result is equivalent to `ifelse(v1==v2, NA, v1)`.

Value

An object of the same shape as `v1`, where values equal to `v2` are set to `NA`.

References

Inspired by Oracle's `nullif()` function.

See Also

`if.na`, `ifelse`.

Examples

```
x <- c(1:3,1:3)
na.if(x, 3)
```

nsubstr *Count Substring Instances*

Description

Counts exact instances of a target string in each element of a character vector.

Usage

```
nsubstr(a,b)
```

Arguments

a	a character vector.
b	a character string with positive length.

Value

An integer vector consisting of the number of instances of b in each element of a.

Examples

```
nsubstr(c('aabbcc',NA,'aabbbbccbb'), 'bb')
```

nw.align *Needleman and Wunsch Sequence Alignment*

Description

Returns a global or partial-local optimal alignment of two nucleotide sequences, with linear gap penalty. nw.score returns just the alignment score.

Usage

```
nw.align(gt1, gt2, gap=-1, pm=1, mm=0, ends=FALSE)
nw.score(gt1, gt2, gap=-1, pm=1, mm=0, ends=FALSE)
```

Arguments

gt1, gt2	nucleotide sequences to be aligned.
gap	gap penalty.
pm	score for a perfect match.
mm	score for a mismatch.
ends	logical: specifies whether overhanging ends are not penalized. See details.

Details

The default is to perform a global alignment across the full lengths of both sequences. The `ends` argument can be used to specify that overhangs at one or both ends of one or both sequences should not be penalized. If `ends` is a scalar, then it controls all four possible overhangs. It may also be set to a vector of four logical values, to control scoring of each possible overhang: the left and right sides of `gt1`, and left and right sides of `gt2`.

The dynamic programming algorithm in `nw.align` is O(m*n) in time and space, and hence is not appropriate for aligning very long sequences. The method in `nw.score` is O(n) in space.

To get the longest common subsequence, use

Value

A list:

`score`	the optimal alignment score.
`pct.matched`	over the aligned intervals of the two sequences, the percentage identity.
`ends`	the starting and ending base positions of the aligned intervals in `gt1` and `gt2`.
`alignment`	a vector of three strings: the aligned `gt1`, a representation of the matched positions, and the aligned `gt2`.

References

Needleman, S. B., & Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48**: 443-453.

See Also

`nw.orient.assay`.

Examples

```
s1 <-       'GAGTTC'
s2 <- 'AACATCGAGGTCTACGGATT'
nw.align(s1, s2, ends=FALSE)
nw.align(s1, s2, ends=TRUE)
```

`nw.orient.assay` *Orient Assay Sequences*

Description

Determines the relative orientation of pairs of genotyping assays by sequence alignment.

Usage

```
nw.orient.assay(gt1, gt2, delta=1)
```

Arguments

`gt1, gt2`	assay sequence(s) with a variant position denoted by an underscore (`'_'`).
`delta`	the minimum difference in scores to establish orientation.

Details

Corresponding elements of `gt1` and `gt2` are split into left and right flanks around the variant position, and then aligned in forward and reverse-complement orientations. The best-scoring orientation is reported if its score exceeds the score of the opposing orientation by at least `delta`.

Value

A vector of `'+'`, `'-'`, or `NA` values.

See Also

`nw.align`.

Examples

```
a1 <- 'ACAAC_ATGCT'
a2 <-  'GCAT_GTTG'
nw.orient.assay(a1,a2)
```

`orient.gt.data` *Flip Assay Strands and/or Swap Alleles*

Description

Given a data frame of genotype information, this updates genomic orientations and/or swaps allele coding for individual assays.

Usage

```
orient.gt.data(gt.data, flip=FALSE, swap=FALSE)
```

Arguments

`gt.data`	a data frame of genotype information.
`flip`	logical: indicates if strands should be flipped.
`swap`	logical: indicates if alleles should be swapped.

Details

Both `flip` and `swap` are evaluated for each assay in `gt.data`, so these can either be single logical values or vectors with one value per row in `gt.data`.

Value

An updated version of `gt.data` with the specified adjustments applied.

See Also

`fetch.gt.data`, `match.gt.data`.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2')

# orient to dbSNP strand
gt <- orient.gt.data(gt, flip=(gt$dbsnp.orient == '-'))
with(gt, table(strand,dbsnp.orient))

# orient to '+' strand
gt <- orient.gt.data(gt, flip=(gt$strand=='-'))
with(gt, table(strand,dbsnp.orient))

# swap to major/minor allele order
swap <- (summary.gt.data(gt)$freq.a < 0.5)
table(swap)
gt <- orient.gt.data(gt, swap=swap)
table(summary.gt.data(gt)$freq.a < 0.5)

# sort alleles
head(gt$alleles)
swap <- (sapply(strsplit(gt$alleles,'/'), order)[1,] == 2)
gt <- orient.gt.data(gt, swap=swap)
head(gt$alleles)
```

---

panel.cluster *Panel Function for Drawing Elliptical Cluster Boundaries*

Description

Panel function for plotting two-dimensional clusters of points with elliptical boundary regions.

Usage

```
panel.cluster(x, y, group.number=1, bounds=c(), min.points=4, ...)
```

Arguments

x, y	as in panel.xyplot.
group.number	as in panel.superpose.
bounds	contours at which to draw ellipsoid boundaries.
min.points	the minimum number of points for which to compute ellipsoids.
...	additional arguments passed to panel.xyplot.

Details

Points are first plotted as in panel.xyplot. Then elliptical boundaries are drawn using settings from superpose.line, using trellis.par.get.

See Also

gt.cluster.plot, panel.xyplot, panel.superpose, ellipsoidPoints.

Examples

```
g <- rep(0:1,100)
x <- rnorm(200)
y <- g*6 + 2*g*x + rnorm(100)
xyplot(y~x, groups=g, panel=panel.superpose,
       panel.groups=panel.cluster, bounds=c(0.5,0.8,0.95))
```

---

panel.qqpval *Quantile-Quantile Plots for P Values: Panel Functions*

Description

Panel functions used by qqpval for generating Quantile-Quantile plots of log transformed P values.

Usage

```
prepanel.qqpval(x, n, groups=NULL, subscripts, ...)
panel.qqpval(x, n, max.pts, groups=NULL, ...)
```

Arguments

x	a vector of P values.
n	the total number of tests drawn from.
max.pts	the maximum number of discrete quantiles to plot.
groups, subscripts	see xyplot.
...	additional arguments passed to prepanel.default.xyplot or panel.qq.

See Also

qqpval, prepanel.default.xyplot, panel.qq.

---

panel.qqthin *Sparse Normal Quantile-Quantile Plots: Panel Function*

Description

Panel function used by qqthin for generating quantile-quantile plots of very large numbers of observations.

Usage

```
panel.qqthin(x, max.pts, groups=NULL, ...)
```

Arguments

x	a numeric vector.
max.pts	the maximum number of discrete quantiles to plot.
groups	see xyplot.
...	additional arguments passed to panel.qqmath.

See Also

qqthin, panel.qqmath.

---

prcomp.gt.data *Principal Components Analysis of Genotype Data*

Description

Performs a principal components analysis of a matrix of genotypes and scores each sample against the most significant components.

Usage

```
## S3 method for class 'gt.data':
prcomp(x, sample.mask=TRUE, nc=20, ...)
## S3 method for class 'gt.dataset':
prcomp(x, sample.mask=TRUE, nc=20, ...)
```

Arguments

x	a data frame of genotypes from fetch.gt.data, or a dataset description from gt.dataset.
sample.mask	a logical vector for subsetting samples.
nc	the number of components to return.
...	not used.

Details

This function is useful for computing principal components in cases where an entire genotype matrix can be accommodated in memory. Genotype vectors are centered and scaled, and missing values are imputed, as in Price *et al.* (2006). The principal components are computed using prcomp.

Value

A list with the following components:

loadings	a data frame with one row per row in the data table, and nc columns, containing the loadings of each sample onto the top principal components.
sdev	the standard deviations of the top nc principal components.
assays	the number of assays included in the analysis.
dataset.name	the name of the genotype dataset.
call	the call used to generate the analysis.

References

Price, A. L., *et al.* (2006) Principal components analysis corrects for stratification in genome-wide association studies. *Nat. Genet.* **38**: 904-909.

See Also

prcomp, fetch.gt.data, gt.dataset, snp.loadings, apply.loadings.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_1')
pt <- fetch.pt.data('Demo_1')
pc <- prcomp(subset(gt, (ploidy=='A')))
screeplot(pc)
xyplot(PC1~PC2, pc$loadings, groups=pt$plate, auto.key=TRUE)
```

---

progress.bar *Console Text-Based Progress Bar*

Description

Displays a text-based progress bar on the R console, to indicate how much of a long-running calculation has been completed. The elapsed time and estimated time remaining are also reported.

Usage

```
progress.bar(done, total, len = 40)
```

Arguments

done	the amount of work already done.
total	the total amount of work to do.
len	the length of the bar, in characters.

Details

After writing the progress bar, the cursor is repositioned at the start of the line so that successive calls will update the bar in place. The console buffer is flushed after each call. The first call should have done=0 to ensure proper initialization of internal data structures.

Value

Invisible NULL.

Examples

```
fn <- function(x) { Sys.sleep(1); progress.bar(x, 20) }
x <- sapply(0:20, fn)
```

qqprcomp *Quantile-Quantile Plots of PCA Loadings*

Description

Generates a series of quantile-quantile plots of either sample or SNP loadings from a principal components analysis.

Usage

```
qqprcomp(x, col=1:6, layout, ...)
```

Arguments

x	either a PCA result structure returned by prcomp.gt.data or prcomp.gt.dataset, or a table of SNP loadings from snp.loadings.
col	a vector of component numbers to plot.
layout	a vector with two elements giving the numbers of columns and rows to use for arranging the plot panels. If not specified, the plots are arranged to minimize wasted space.
...	additional arguments passed to qqthin.

Details

This function generates a series of normal Q-Q plots of loadings from a principal components analysis. It accepts either a PCA result structure (for sample loadings), or a table of SNP loadings returned by snp.loadings. Sparse Q-Q plots are generated automatically when the number of loadings is very large.

Q-Q plots are useful for distinguishing components that reveal either sample or genomic structure in the data, from components that do not. If a component has flat Q-Q plots for both sample and SNP loadings, it is unlikely to be informative.

Value

A plot object of class "trellis".

See Also

prcomp.gt.data, prcomp.gt.dataset, snp.loadings, qqthin.

Examples

```
gt.demo.check()
p1 <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')))
qqprcomp(p1)
pt <- fetch.pt.data('Demo_1')
p2 <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')),
             sample.mask=is.na(pt$father))
qqprcomp(p2)
```

qqpval *Quantile-Quantile Plots for P Values*

Description

Quantile-Quantile plots of log transformed P values versus their expected uniform distribution, with support for very large numbers of tests.

Usage

```
qqpval(x, ..., n=NA, max.pts=1000, prepanel=prepanel.qqpval,
       panel=panel.qqpval, xlab='theoretical quantiles')
```

Arguments

`x`	either a vector of P values, or a formula to be evaluated in the context of a `data` argument.
`...`	additional arguments passed to `qqmath`.
`n`	the total number of tests drawn from.
`max.pts`	the maximum number of discrete quantiles to plot.
`prepanel, panel`	as in `qqmath`.
`xlab`	X axis label.

Details

Conventional Q-Q plots for very large numbers of tests are slow and most of the "ink" is spent plotting uninteresting large P values. This function avoids this problem by limiting the number of quantiles plotted. If the number of supplied P values is larger than `max.pts`, then a threshold is chosen to switch between plotting each observed value, and plotting evenly spaced quantiles. The threshold is optimized to minimize the quantile spacing in the equal-spaced region.

A valid (truncated) Q-Q plot can also be generated from the smallest P values selected from a larger set of tests, by specifying `n`, the total number of tests the supplied values were selected from.

A reference line is drawn using settings specified by `trellis.par.get('reference.line')`.

Value

A plot object of class `"trellis"`.

See Also

`qqthin`, `qqmath`.

Examples

```
pval <- runif(250000)
qqpval(pval)
qqpval(pval, max.pts=100)
qqpval(pval[pval<0.001], n=250000, max.pts=100)

p <- runif(3000)
s <- replicate(400, sort(runif(3000)))
q <- apply(s, 1, quantile, c(0.025,0.975))
rc <- '#b0b0b0'
qqpval(~q[1,]+q[2,]+p, type=c('l','p'), col=c(rc,rc,2),
       lty=c(2,2,0), pch=c(32,32,1), max.pts=100,
       par.settings=list(reference.line=list(col=rc)))
```

---

qqthin *Sparse Normal Quantile-Quantile Plots*

Description

Quantile-Quantile plots of a sample versus the normal distribution, with support for very large numbers of observations.

Usage

```
qqthin(x, ..., max.pts=1000, panel=panel.qqthin)
```

Arguments

`x`	either a numeric vector, or a formula to be evaluated in the context of a `data` argument.
`...`	additional arguments passed to `qqmath`.
`max.pts`	the maximum number of discrete quantiles to plot.
`panel`	as in `qqmath`.

Details

Conventional Q-Q plots for very large numbers of observations are slow and most of the "ink" is spent plotting uninteresting values. This function avoids this problem by limiting the number of distinct quantiles plotted. If the number of values is larger than `max.pts`, then a threshold is chosen to switch between plotting each extreme value, and plotting evenly spaced quantiles. The threshold is optimized to minimize the quantile spacing in the equal-spaced region.

A reference line is drawn using settings specified by `trellis.par.get('reference.line')`.

Value

A plot object of class `"trellis"`.

See Also

`qqpval`, `qqmath`.

Examples

```
v <- rnorm(100000)
qqthin(v)
qqthin(v, max.pts=100)

v <- rnorm(3000)
s <- replicate(400, sort(rnorm(3000)))
q <- apply(s, 1, quantile, c(0.025,0.975))
rc <- '#b0b0b0'
qqthin(~q[1,]+q[2,]+v, type=c('l','p'), col=c(rc,rc,2),
       lty=c(2,2,0), pch=c(32,32,1), max.pts=100,
       par.settings=list(reference.line=list(col=rc)))
```

---

hexToRaw *Convert between Raw Vectors and Hex Strings*

Description

Convert between raw vectors and hex strings.

Usage

```
hexToRaw(hex)
rawToHex(raw)
```

Arguments

hex	a string of hex digits.
raw	a raw vector.

Value

For hexToRaw, a raw vector constructed by converting each consecutive pair of hex digits to one byte. For rawToHex, a character string formed by converting each byte of the input vector to its two-digit hex representation.

See Also

charToRaw, rawToChar.

Examples

```
hexToRaw('1a3b1a3b')
rawToHex(as.raw(seq(0,100,10)))
```

`reshape.gt.data` *Reshape Genotype Data*

Description

This takes a data frame of genotypes with one row per assay, and returns a "long" data frame with one row per genotype per sample.

Usage

```
reshape.gt.data(gt.data, ...)
```

Arguments

`gt.data`	a data frame of genotypes from `fetch.gt.data`.
`...`	additional arguments passed to `gt.split`.

Value

A data frame with one row per assay per sample, and columns corresponding to available data extracted from `gt.data`. This normally includes genotypes, and may include quality scores and/or one or more columns of additional underlying raw data.

`assay.name`	an assay identifier.
`sample.name`	a sample identifier.
`genotype`	the genotype for the specified sample and assay.
`qscore`	the corresponding quality score.
`...`	additional columns of raw data, depending on the dataset. For datasets with signal intensities, there will be two columns: `signal.a` and `signal.b`. For data with sequencing read counts, there will be four columns: `fwd.a`, `rev.a`, `fwd.b`, and `rev.b`.

See Also

`fetch.gt.data`, `unpack.gt.matrix`, `gt.split`.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_2',raw.data=TRUE)
head(reshape.gt.data(gt))
d <- reshape.gt.data(gt[seq(32,232,40),], na.codes='n')
gt.cluster.plot(d)
```

revcomp *Reverse Complement DNA Sequences*

Description

Reverse complement DNA sequences, including IUPAC ambiguity codes.

Usage

```
revcomp(x)
```

Arguments

x	a character vector of DNA sequences

Value

A vector with the same shape as x, where each sequence has been replaced by its reverse complement, preserving case.

Examples

```
revcomp(c('AACAGTAGA','AACCNNRacgt'))
```

score.and.store *Test SNPs for Association and Store Results*

Description

Perform a series of single-point SNP association tests on a genotype dataset, using an arbitrary scoring function, and store the results in the TEST and TEST_RESULT tables.

Usage

```
score.and.store(dataset.name, test.name, description, formula,
                score.fn=NULL, pt.filter=TRUE, gt.filter=TRUE,
                pca=FALSE, part=1:parts, parts=200,
                dryrun=FALSE, ...)
```

Arguments

dataset.name	the name of the dataset to be analyzed.
test.name	a unique identifier to associate with these test results.
description	a description of this test result set.
formula	a symbolic description of the model to be fit.
score.fn	the scoring function to be applied to each SNP.
pt.filter	an expression to use for subsetting on the phenotype table.
gt.filter	an expression to use for subsetting on the genotype table.

pca	a logical indicating whether the current principal components analysis results for this dataset should be loaded, or a list of arguments to be passed to fetch.prcomp to specify an analysis to be loaded.
part	a vector of slices 1..parts.
parts	the number of slices to split the dataset into.
dryrun	logical: if TRUE, then do not write results to the database.
...	additional arguments to pass to score.gt.data.

Details

This is essentially a wrapper around score.gt.data, that handles fetching phenotype and genotype data, and storing results back into the database. The formula, score.fn, pt.filter, and gt.filter arguments are passed to score.gt.data. The pca argument is passed to fetch.pt.data. And the part and parts arguments are used with fetch.gt.data. The description argument is used for constructing new entries in the TEST table in GTPUB.

If executed as part of an LSF "job array", this function will automatically initialize parts on each node to divide the computation evenly across the job array.

See Also

fetch.pt.data, fetch.gt.data, score.gt.data.

Examples

```
gt.demo.check()
## score 5% of the SNPs in Demo_1 dataset
score.and.store('Demo_1', 'Test_1', 'Example Analysis',
                plate ~ genotype, score.chisq,
                gt.filter=(pass==1),
                part=1, parts=20, dryrun=TRUE,
                progress=TRUE)
```

---

score.chisq.2x2 *Chi-Squared Test for Allelic Association*

Description

Test for allelic association with a binary outcome, using a chi-squared test on a 2x2 table of allele counts.

Usage

```
score.chisq.2x2(formula, data, ploidy)
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data and details.

Details

The 2x2 chi-squared test is the simplest test for association between genotypes and a binary outcome. The model formula should be of the form `outcome~genotype` without additional terms. We do not apply a continuity correction because that makes the overall distribution of P values slightly conservative.

Sex linked data is handled by counting males as having a single allele, which while technically correct may not be reasonable.

Use of this test is not recommended as it is sensitive to deviations from Hardy Weinberg equilibrium. Usually, `score.trend` or `score.glm` are better choices.

Value

A data frame with one row and three columns:

`pvalue`	P value for the test.
`effect`	the log odds ratio for the 2x2 table.
`stderr`	the standard error of the log odds ratio.

See Also

`chisq.test`, `score.chisq`, `score.trend`, `score.glm`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt, score.chisq.2x2)
score.gt.data(status~genotype, pt, gt, score.glm)
```

`score.chisq` *Chi-Squared Test for Genotypic Association*

Description

Test for genotype association with a categorical outcome, using a chi-squared test on an NxM table of genotype counts.

Usage

```
score.chisq(formula, data, ploidy,
            mode=c('general','recessive','dominant'))
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`ploidy`	see `fetch.gt.data`. Ignored.
`mode`	genetic mode of action to test.

Details

This performs a simple chi squared test of association using the contingency table for a categorical outcome (with any number of levels) versus diploid genotypes. The model formula should be of the form `outcome~genotype` without additional terms. Empty levels for the outcome or the genotype will be dropped. We do not apply a continuity correction because that makes the overall distribution of P values slightly conservative.

Sex linked data is handled by treating males as diploid homozygotes for the corresponding haploid alleles, which may or may not be a reasonable thing to do.

Value

A data frame with one row and three columns:

`pvalue`	P value for the test.
`effect`	for 2x2 tables, the log odds ratio.
`stderr`	for 2x2 tables, the standard error of the log odds ratio.

See Also

`chisq.test`, `score.fisher`, `score.trend`, `score.chisq.2x2`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt, score.chisq.2x2)
score.gt.data(status~genotype, pt, gt, score.chisq)
```

`score.fisher` *Fisher's Exact Test for Genotypic Association*

Description

Performs Fisher's exact test for genotype association with a categorical outcome, on an NxM table of genotype counts.

Usage

```
score.fisher(formula, data, ploidy,
             mode=c('general','recessive','dominant'), ...)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`ploidy`	see `fetch.gt.data`. Ignored.
`mode`	genetic mode of action to test.
`...`	additional arguments passed to `fisher.test`.

Details

This evaluates Fisher's exact test for association on a contingency table for a categorical outcome (with any number of levels) versus diploid genotypes. The model formula should be of the form `outcome~genotype` without additional terms. Empty levels for the outcome or the genotype will be dropped.

Sex linked data is handled by treating males as diploid homozygotes for the corresponding haploid alleles, which may or may not be a reasonable thing to do.

Value

A data frame with one row and two columns:

`pvalue`	P value for the test.
`effect`	for 2x2 tables, the log odds ratio.

See Also

`fisher.test`, `score.chisq`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt, score.chisq)
score.gt.data(status~genotype, pt, gt, score.fisher)
```

`score.glm` *Test for Association using Logistic Regression*

Description

Test for association with a binary outcome using logistic regression.

Usage

```
score.glm(formula, data, ploidy, drop='genotype',
          mode=c('additive','recessive','dominant','general'))
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`ploidy`	see `fetch.gt.data`. Ignored.
`drop`	the model term to test for association.
`mode`	genetic mode of action to be tested.

Details

The model formula may include covariates and should contain a single genotype term without interactions. Significance is assessed by ANOVA comparing the full model with a null model constructed by removing this term.

At sex linked loci, haploid males are treated the same as the corresponding diploid female homozygotes.

Value

A data frame with one row and three columns. An effect size is not reported if the mode of action is `'general'`.

`pvalue`	P value for an F test comparing the full and null models.
`effect`	the regression coefficient (log odds) for `term`.
`stderr`	the standard error of the effect size estimate.

See Also

`lm`, `score.trend`, `score.chisq`, `score.glm.general`, `score.glm.groups`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~plate+genotype, pt, gt, score.glm)
score.gt.data(status~plate+genotype, pt, gt, score.glm.general)
```

`score.glm.general` *Association Test with a Logistic Model and General Mode of Action*

Description

Test for association using logistic regression, with a general mode of action.

Usage

```
score.glm.general(formula, data, ploidy)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`ploidy`	see `fetch.gt.data`. Ignored.

Details

This is similar to `score.glm`, except that it considers a general mode of action with (log) additive and dominance effects. Overall significance is still assessed by ANOVA comparing the full model with a null model constructed by removing this term.

Value

A data frame with four columns. If there are only two distinct genotypes, then results are equivalent to `score.glm` with additive mode of action. If there are three distinct genotypes, then the first row of results describes the overall ANOVA test; the second describes the fitted additive effect; and the third describes the fitted dominance effect.

`term`	`NA`, `'additive'`, or `'dominance'`.
`pvalue`	For the overall test, the ANOVA F test result. For additive and dominance effects, Wald test results.
`effect`	the regression coefficient, for the additive or dominant effect.
`stderr`	the standard error of the effect size estimate.

See Also

`glm`, `score.glm`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt, score.glm)
score.gt.data(status~genotype, pt, gt, score.glm.general)
```

`score.glm.groups` *Test for Association using a Logistic Model with Subgroup Effects*

Description

Test for association with a binary outcome, using logistic regression assuming a log-additive allelic effect on risk for the genotype term, where effects are allowed to vary by subgroup.

Usage

```
score.glm.groups(formula, data, ploidy, quick=FALSE,
                 mode=c('additive','recessive','dominant'))
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`ploidy`	see `fetch.gt.data`. Ignored.
`quick`	logical: if set, just return the overall test result.
`mode`	genetic mode of action to test.

Details

The model formula can contain multiple genotype terms with interactions, where the interacting variables are factors. Overall significance is assessed by a likelihood ratio test comparing the full model with a null model constructed by removing all genotype terms. Significance of the interaction term(s) is assessed by a likelihood ratio test comparing the full model with a model with no interactions. Finally, genotype effects are computed for each subgroup. At sex linked loci, haploid males are treated the same as the corresponding diploid female homozygotes.

Value

A data frame with four columns and at least two rows. The first two rows describe likelihood ratio tests for the full model versus a null model, and the full model versus a model with no subgroup interaction. Additional rows describe tests for effects within each subgroup.

`term`	for subgroup effects, the corresponding genotype:subgroup interaction term.
`pvalue`	for tests on nested models, the P value for a likelihood ratio test comparing the two models. For subgroup effects, the P value of a Wald test on the effects size.
`effect`	for subgroups, the estimated allelic effect (log odds) from the regression.
`stderr`	for subgroups, the estimated standard error of the effect size.

See Also

`glm`, `score.glm`, `score.gt.data`.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:5,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~plate+genotype, pt, gt, score.glm)
score.gt.data(status~plate*genotype, pt, gt, score.glm.groups)
```

`score.gt.data` *Test SNPs for Association*

Description

Perform a series of single-point SNP association tests, using an arbitrary scoring function.

Usage

```
score.gt.data(formula, pt.data, gt.data, score.fn=NULL,
              pt.filter=TRUE, gt.filter=TRUE,
              progress=FALSE, ...)
```

Arguments

formula	a symbolic description of the model to be fit.
pt.data	a data frame of phenotypes from fetch.pt.data.
gt.data	a data frame of genotypes from fetch.gt.data.
score.fn	the scoring function to be applied to each SNP.
pt.filter	an expression to use for subsetting on the phenotype table.
gt.filter	an expression to use for subsetting on the genotype table.
progress	logical: specifies if a progress bar should be displayed on the R console.
...	additional arguments to pass to score.fn.

Details

For each row of genotypes in gt.data, a data frame will be constructed by merging columns of pt.data referenced in the model formula with those genotypes. The specified score function will be invoked for each of these data frames in turn.

If no scoring function is specified, then one will be deduced from the form of the model formula. A trend test will be used for simple binary-outcome models of the form "status genotype". Logistic regression will be used for binary outcomes with more complicated model functions. And linear regression will be used for quantitative outcomes.

The filters pt.filter and gt.filter are evaluated in the context of the pt.data and gt.data tables, respectively, similar to how subset works.

It is expected that some tests will fail for some SNPs (say, due to degeneracies where a SNP or trait is invariant across the samples for which data is available). Errors from the scoring function are converted to warnings.

Value

A data frame with up to five columns, and one or more rows per SNP tested, depending on the form of the test. Some columns may not be populated for certain test types.

assay.name	an identifier for the assay tested.
term	for tests that return more than one result per assay, something specifying the individual results.
pvalue	an uncorrected P value for this test.
effect	an estimated effect size, where that makes sense.
stderr	the estimated standard error of the effect size.

See Also

score.chisq.2x2, score.chisq, score.fisher, score.trend, score.kruskal, score.jt, score.lm, score.lm.general, score.lm.groups, score.glm, score.glm.general, score.glm.groups, score.and.store.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt)
score.gt.data(status~plate+genotype, pt, gt, score.glm,
              pt.filter=(is.na(father)))
```

---

score.jt *Jonckheere-Terpstra Nonparametric Test for Association*

Description

Test for association between genotypes and a quantitative outcome, using the nonparametric Jonckheere-Terpstra test for ordered differences among genotype classes.

Usage

```
score.jt(formula, data, ploidy, ...)
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.
...	additional arguments passed to jt.test.

Details

The Jonckheere-Terpstra test computes Mann-Whitney rank sum statistics for ordered diploid genotypes, and does a two-sided test on the sum of those statistics. This tests for a monotonic trend in outcomes as a function of genotype. The model formula should be of the form outcome~genotype without additional terms.

The test does not directly provide an estimate of an effect size. Instead, we center and rescale the test statistic to fall in the range −1 to +1. The expected value of this score is independent of sample size and indicates the direction of effect.

Value

A data frame with one row and three columns:

pvalue	P value for the test assuming an asymptotic normal distribution for the test statistic.
effect	the test statistic centered and scaled to be in the range −1 to +1.
stderr	the estimated standard error of effect.

References

http://tolstoy.newcastle.edu.au/R/help/06/06/30112.html.

See Also

jt.test, score.kruskal, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$score <- rnorm(nrow(pt))
score.gt.data(score~genotype, pt, gt, score.kruskal)
score.gt.data(score~genotype, pt, gt, score.jt)
```

---

score.kruskal *Kruskal-Wallis Nonparametric Test for Association*

Description

Test for association between genotypes and a quantitative outcome, using the nonparametric Kruskal-Wallis test for differences among genotype classes.

Usage

```
score.kruskal(formula, data, ploidy,
              mode=c('general','recessive','dominant'))
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.
mode	genetic mode of action to test.

Details

This tests whether the summed ranks of the outcome are independent of genotype. The model formula should be of the form outcome~genotype without additional terms. The default is to make no assumptions about mode of action.

Value

A data frame with one row and one column:

pvalue	asymptotic chi-squared P value for the test.

See Also

kruskal.test, score.jt, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$score <- rnorm(nrow(pt))
score.gt.data(score~genotype, pt, gt, score.kruskal)
score.gt.data(score~genotype, pt, gt, score.lm)
```

---

score.lm *Test for Association using a Simple Linear Model*

Description

Test for association using linear regression.

Usage

```
score.lm(formula, data, ploidy, drop='genotype',
         mode=c('additive','recessive','dominant','general'))
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.
drop	the model term to test for association.
mode	genetic mode of action to test.

Details

The model formula may include covariates and should contain a single genotype term without interactions. Significance is assessed by ANOVA comparing the full model with a null model constructed by removing this term.

At sex linked loci, haploid males are treated the same as the corresponding diploid female homozygotes.

Value

A data frame with one row and three columns. An effect size is not reported if the mode of action is 'general'.

pvalue	P value for an F test comparing the full and null models.
effect	the regression coefficient for term.
stderr	the standard error of the effect size estimate.

See Also

lm, score.lm.general, score.lm.groups, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$score <- rnorm(nrow(pt))
score.gt.data(score~plate+genotype, pt, gt, score.lm)
score.gt.data(score~genotype, pt, gt, score.lm, mode='recessive')
score.gt.data(score~genotype, pt, gt, score.lm, mode='general')
score.gt.data(score~genotype, pt, gt, score.lm.general)
```

---

score.lm.general *Test for Association using a Linear Model and General*

Description

Test for association using linear regression, for a general mode of action.

Usage

```
score.lm.general(formula, data, ploidy)
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.

Details

This is similar to score.lm, except that it considers a general mode of action with additive and dominance effects. Overall significance is still assessed by ANOVA comparing the full model with a null model constructed by removing this term.

Value

A data frame with four columns. If there are only two distinct genotypes, then results are equivalent to score.lm with additive mode of action. If there are three distinct genotypes, then the first row of results describes the overall ANOVA test; the second describes the fitted additive effect; and the third describes the fitted dominance effect.

term	NA, 'additive', or 'dominance'.
pvalue	For the overall test, the ANOVA F test result. For additive and dominance effects, T test results.
effect	the regression coefficient, for the additive or dominant effect.
stderr	the standard error of the effect size estimate.

See Also

lm, score.lm, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$score <- rnorm(nrow(pt))
score.gt.data(score~genotype, pt, gt, score.lm, mode='general')
score.gt.data(score~genotype, pt, gt, score.lm.general)
```

---

score.lm.groups *Test for Association using a Linear Model with Subgroup Effects*

Description

Test for association using linear regression assuming an additive allelic effect for the genotype term, where effects are allowed to vary by subgroup.

Usage

```
score.lm.groups(formula, data, ploidy, quick=FALSE,
                mode=c('additive','recessive','dominant'))
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.
quick	logical: if set, just return the overall test result.
mode	genetic mode of action to test.

Details

The model formula can contain multiple genotype terms with interactions, where the interacting variables are factors. Overall significance is assessed by ANOVA comparing the full model with a null model constructed by removing all genotype terms. Significance of the interaction term(s) is assessed by ANOVA comparing the full model with a model with no interactions. Finally, genotype effects are computed for each subgroup. At sex linked loci, haploid males are treated the same as the corresponding diploid female homozygotes.

Value

A data frame with four columns and at least two rows. The first two rows describe ANOVA tests for the full model versus a null model, and the full model versus a model with no subgroup interaction. Additional rows describe tests for effects within each subgroup.

term	for subgroup effects, the corresponding genotype:subgroup interaction term.
pvalue	for ANOVA tests, the P value for an F test comparing the full and null models. For subgroup effects, the P value for a T test on the effect size.
effect	for subgroups, the estimated allelic effect from the regression.
stderr	for subgroups, the estimated standard error of the effect size.

See Also

lm, score.lm, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:5,]
pt$score <- rnorm(nrow(pt))
score.gt.data(score~plate+genotype, pt, gt, score.lm)
score.gt.data(score~plate*genotype, pt, gt, score.lm.groups)
```

---

score.prcomp *Test Phenotypic Association with Principal Components*

Description

Evaluates evidence for association between sample loadings from a principal components analysis, and one or more phenotypes.

Usage

```
score.prcomp(formula, pc.data, pt.data, ...)
```

Arguments

formula	a symbolic description of the model to be fitted. The left hand side should be given as PC.
pc.data	a PCA result structure returned by prcomp.gt.data or prcomp.gt.dataset.
pt.data	phenotype data from fetch.pt.data.
...	additional arguments passed to lm.

Details

The specified model formula is evaluated by linear regression for each principal component, and results are summarized as in an ANOVA based on the proportion of variance of the principal component explained.

Value

A data frame with one row per principal component and three columns:

R^2	the adjusted multivariate $R\hat{2}$ statistic for the model.
F value	an F statistic for the model.
Pr(>F)	the tail probability associated with this F statistic.

See Also

prcomp.gt.data, prcomp.gt.dataset, lm.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_1')
pt <- fetch.pt.data('Demo_1')
pc <- prcomp(subset(gt, (ploidy=='A')),
             sample.mask=(is.na(pt$father)))
score.prcomp(PC~panel, pc, pt)
```

---

score.trend *Cochran-Armitage Trend Test for Association*

Description

Test for genotype assocation with a binary outcome, using the Cochran-Armitage test for trend in proportions.

Usage

```
score.trend(formula, data, ploidy)
```

Arguments

formula	a symbolic description of the model to be fit.
data	a data frame containing the variables in the model.
ploidy	see fetch.gt.data. Ignored.

Details

The Cochran-Armitage test for trend is a test for a linear dependence of outcome proportions on genotype expressed as an allele count. The model formula should be of the form `outcome~genotype` without additional terms.

In place of a conventional effect size, we return the slope and standard error for the trend in proportions. This slope is thus the fitted change in outcome proportion per allele.

Value

A data frame with one row and three columns:

pvalue	P value for test.
effect	the slope of the trend in proportions.
stderr	the standard error of the slope.

See Also

prop.trend.test, score.glm, score.gt.data.

Examples

```
gt.demo.check()
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')[1:10,]
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
score.gt.data(status~genotype, pt, gt, score.trend)
score.gt.data(status~genotype, pt, gt, score.glm)
```

---

set.hidden *Update Hidden Status*

Description

This updates the "hidden" status of a database object (project, dataset, platform mapping, sample attribute, etc).

Usage

```
set.hidden(table, name, is.hidden=TRUE, ...)
```

Arguments

table	the type of object to be hidden (or revealed).
name	the name of the object to be hidden (or revealed).
is.hidden	logical: the new status for the object.
...	additional qualifiers to identify the target object.

Value

If successful, the number of rows updated in the corresponding table (i.e., 1).

See Also

mk.project, mk.dataset.

Examples

```
## Not run:
set.hidden('dataset','Demo_1',is.hidden=TRUE)
ls.dataset('Demo')
set.hidden('dataset','Demo_1',is.hidden=FALSE)
ls.dataset('Demo')
set.hidden('subject_attr','plate',is.hidden=TRUE,project.name='Demo')
## End(Not run)
```

snp.loadings *SNP Loadings from Principal Components Analyses*

Description

Computes loadings of assays onto previously computed principal components. Results can be computed for either a specific set of SNPs or the entire original genotype dataset.

Usage

```
snp.loadings(x, data)
```

Arguments

x	a structure returned by prcomp.
data	either a data frame of genotypes from fetch.gt.data, or a dataset description from gt.dataset.

Details

snp.loadings computes SNP loadings across either an entire dataset or a specific set of genotype assays. The genotype data needs to cover the same individuals included in the original principal components analysis.

If the analysis was done with prcomp.gt.dataset and data is missing, then loadings are computed across the same dataset specification used in the original analysis.

Value

A list with the following components:

loadings	a data frame with one row per assay, and nc columns, containing the loadings of each assay onto the top principal components.
sdev	the standard deviations of the top nc principal components.
assays	annotations for the assays in loadings.
dataset.name	the name of the source genotype dataset.
call	the call used to generate the analysis.

See Also

prcomp.gt.data, fetch.gt.data, gt.dataset, apply.loadings.

Examples

```
gt.demo.check()
p <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')))
s <- snp.loadings(p, gt.dataset('Demo_1'))
p$loadings[1:5,1:5]
s$loadings[1:5,1:5]
```

`sql.query` *Simplified SQL Statement Execution*

Description

These functions parse and execute SQL statements, and either fetch query results or return the number of affected rows. All result sets are also cleaned up. Some database differences and differences in DBI implementations are also concealed.

Usage

```
sql.query(db, sql, ...)
sql.exec(db, sql, ..., chunk.kb=256)
```

Arguments

`db`	a DBI connection object returned by `dbConnect`.
`sql`	a valid SQL statement, optionally with embedded bind variables.
`...`	optionally, a data frame or arguments to be used to construct a data frame of bind variables.
`chunk.kb`	when processing multiple rows of bind variables, a rough limit on the amount of data to send per query.

Details

To facilitate database agnostic code, several special elements can be used in SQL statements, which are internally replaced with database specific forms:

:user: the database user name.

:sysdate: the current date.

:unhex:(...) a function to convert from a hex string to binary.

:clob:(...) a wrapper for passing long character data back to R.

:blob:(...) a wrapper for passing long binary data back to R, rendered as a hexadecimal string.

:fromdual: a placeholder for the Oracle `FROM DUAL` idiom for `SELECT` statements that do not need table data.

Both `sql.query` and `sql.exec` emulate prepared statements and bind variables in a consistent way across databases. The emulation uses the Oracle syntax for bind variables (i.e., ':1', ':2', etc in SQL are substituted with values of the bind variables).

A single call to these functions may result in multiple SQL statements. These will be grouped into a single transaction for databases that support that, so that a call to `sql.exec` should be "all or none". RMySQL currently does not support transactions, so a call to `sql.exec` may partially fail.

Value

For `sql.query`, a data frame constructed from the results returned by the query. Column names are transformed by converting '_' to '.' and changing to lower case. For `sql.exec`, the return value is the number of affected rows.

See Also

dbConnect, fetch.

Examples

```
## Not run:
sql.exec(db, 'create table xyzzy (a number, b number)')
sql.exec(db, 'insert into xyzzy values (:1,:2)', a=6, b=1:4)
sql.exec(db, 'insert into xyzzy values (:1,:2)', data.frame(1,2))
sql.query(db, 'select * from xyzzy')
sql.exec(db, 'drop table xyzzy')
sql.exec(db, 'select :user:, :sysdate: :fromdual:')
## End(Not run)
```

---

store.prcomp *Store or Remove Principal Components Results*

Description

Store, or remove, principal components analysis results in the database.

Usage

```
store.prcomp(x, prcomp.name, description,
             nc=ncol(x$loadings), is.hidden=FALSE)
rm.prcomp(dataset.name, prcomp.name)
```

Arguments

x	a principal components result returned by prcomp.gt.data or prcomp.gt.dataset.
prcomp.name	a short unique identifier for the analysis.
description	a free-text description of this analysis.
nc	the number of components to store.
is.hidden	logical: indicates if the dataset is hidden.
dataset.name	the name of a parent dataset for which an analysis is to be removed.

Details

Analysis names only need to be unique within the scope of a particular dataset.

Value

The number of principal components stored into the database.

See Also

fetch.prcomp, prcomp.gt.data, prcomp.gt.dataset.

Examples

```
## Not run:
pc <- prcomp(gt.dataset('Demo_1', gt.filter=(ploidy=='A')))
store.prcomp(pc, 'demo_pc_1', 'Demo PCA results')
fetch.prcomp('Demo', 'demo_pc_1', nc=4)
## End(Not run)
```

---

store.sample.data *Store Sample Data*

Description

Stores sample phenotype data into the database from a data frame.

Usage

```
store.sample.data(dataset.name, data)
```

Arguments

dataset.name	the short unique identifier for the dataset.
data	a data frame of sample phenotypes. See details.

Details

The data argument should have a sample.name column, with samples previously defined by mk.sample. Additional columns should supply values of sample attributes previously defined by mk.sample.attr. Missing values are not explicitly recorded in the database.

Columns with reserved names "subject_name", "gender", and "position" will be ignored.

Value

A vector of counts of rows inserted in the database for each column of phenotype information in the input data.

See Also

mk.sample, mk.sample.attr, fetch.sample.data, fetch.pt.data.

Examples

```
## Not run:
p <- ls.sample('Demo_1')['sample.name']
p$stuff <- rnorm(nrow(p))
mk.sample.attr('Demo_1', p['stuff'], 'Some random stuff')
store.sample.data('Demo_1', p)
head(fetch.sample.data('Demo_1')
rm.sample.attr('Demo_1', 'stuff')
## End(Not run)
```

`store.subject.data` *Store Subject Data*

Description

Stores subject phenotype data into the database from a data frame.

Usage

```
store.subject.data(project.name, data)
```

Arguments

`project.name`	the short unique identifier for the project.
`data`	a data frame of subject phenotypes. See details.

Details

The `data` argument should have a `subject.name` column, with subjects previously defined by `mk.subject`. Additional columns should supply values of subject attributes previously defined by `mk.subject.attr`. Missing values are not explicitly recorded in the database.

Value

A vector of counts of rows inserted in the database for each column of phenotype information in the input data.

See Also

`mk.subject`, `mk.subject.attr`, `fetch.subject.data`, `fetch.pt.data`.

Examples

```
## Not run:
p <- ls.subject('Demo')['subject.name']
p$stuff <- rnorm(nrow(p))
mk.subject.attr('Demo', p['stuff'], 'Some random stuff')
store.subject.data('Demo', p)
head(fetch.subject.data('Demo')
rm.subject.attr('Demo', 'stuff')
## End(Not run)
```

`store.test.scores` *Store or Remove Association Test Results*

Description

Store, or remove, association test results in the database.

Usage

```
store.test.scores(x, test.name, description, is.hidden=FALSE)
rm.test(dataset.name, test.name)
```

Arguments

`x`	a principal components result returned by `prcomp.gt.data` or `prcomp.gt.dataset`.
`test.name`	a short unique identifier for the analysis.
`description`	a free-text description of this analysis.
`is.hidden`	logical: indicates if the result set is hidden.
`dataset.name`	the name of a parent dataset for which an analysis is to be removed.

Details

Analysis names only need to be unique within the scope of a particular dataset.

Value

The number of results stored into the database.

See Also

`fetch.test.scores`, `score.gt.data`.

Examples

```
## Not run:
pt <- fetch.pt.data('Demo_2')
gt <- fetch.gt.data('Demo_2')
pt$status <- as.logical(rbinom(nrow(pt), 1, 0.5))
x <- score.gt.data(status~genotype, pt, gt)
store.test.scores(x, 'status_1', 'Test analysis')
y <- fetch.test.scores('Demo_2', 'status_1')
str(y)
## End(Not run)
```

summary.gt.data *Genotype Data Summary*

Description

Computes genotype counts, allele frequencies, call rates, and tests for Hardy Weinberg equilibrium for a packed genotype data structure.

Usage

```
## S3 method for class 'gt.data':
summary(object, sample.mask, by.sample=FALSE, ...)
## S3 method for class 'gt.dataset':
summary(object, sample.mask, by.sample=FALSE, ...)
```

Arguments

object	either a data frame of genotypes from fetch.gt.data, or a dataset description from gt.dataset.
sample.mask	an optional mask identifying a subset of samples to be included in the summaries.
by.sample	logical: indicates if statistics should be computed by sample (as opposed to by SNP).
...	not used.

Details

The sample.mask argument may either be a character string or a logical vector with one element per genotype.

Value

If by.sample is FALSE: a data frame with one row per genotype assay, and 9 columns:

NN	count of missing ('n') genotypes.
AA, AB, BB	counts of diploid r/h/a genotypes.
A_, B_	counts of haploid r/a genotypes.
gt.rate	genotype call rate for this SNP.
freq.a, freq.b	allele frequencies.
hw.p.value	Hardy-Weinberg equilibrium P value, calculated from a likelihood ratio test.

If by.sample is TRUE: a data frame with one row per sample and 9 columns, with samples ordered by position in the genotype strings, and row names set to RNA_DNA_SOURCE_ID.

See Also

fetch.gt.data, gt.dataset.

Examples

```
gt.demo.check()
g <- fetch.gt.data('Demo_1')[1:10,]
summary(g)
summary(g, attr(g,'gender')[['F']])
summary(g, by.sample=TRUE)[1:10,]
```

---

unpack.gt.matrix *Convert Packed Genotype Strings to a Genotype Matrix*

Description

This takes a data frame of genotypes with one row per assay, and returns a new data frame with one column per assay and one row per sample.

Usage

```
unpack.gt.matrix(gt.data, names=gt.data$assay.name, ...)
```

Arguments

gt.data	a data frame of genotypes from fetch.gt.data.
names	column names for the resulting data frame.
...	additional arguments passed to gt.split.

Value

A data frame with one column per row in gt.data, and one row per sample.

See Also

fetch.gt.data, gt.split.

Examples

```
gt.demo.check()
gt <- fetch.gt.data('Demo_1')
head(unpack.gt.matrix(gt[1:5,]))
head(unpack.gt.matrix(gt[1:5,], convert='char'))
```

use.gt.db *Define GT.DB Database Connection*

Description

Define a DBI connection to be used to access a GT.DB database. This connection is used implicitly by other GT.DB functions that interact with the database.

Usage

```
use.gt.db(dbConnection)
```

Arguments

`dbConnection` a connection object from `dbConnect`.

Value

None.

See Also

`dbDriver`, `dbConnect`, `init.gt.db`.

Examples

```
## Not run:
library(RSQLite)
tmpname <- tempfile('db')
gt.db <- dbConnect(dbDriver('SQLite'), tmpname)
use.gt.db(gt.db)
## End(Not run)
```

Index