

Package ‘memisc’

November 12, 2009

Type Package

Title Tools for Management of Survey Data, Graphics, Programming, Statistics, and Simulation

Version 0.95-23

Date 2009-11-11

Author Martin Elff

Maintainer Martin Elff <melff@essex.ac.uk>

Description One of the aims of this package is to make life easier for useRs who deal with survey data sets. It provides an infrastructure for the management of survey data including value labels, definable missing values, recoding of variables, production of code books, and import of (subsets of) SPSS and Stata files. Further, it provides functionality to produce tables and data frames of arbitrary descriptive statistics and (almost) publication-ready tables of regression model estimates. Also some convenience tools for graphics, programming, and simulation are provided.

License GPL-2

LazyLoad Yes

Depends lattice, grid, stats, methods, utils, MASS

URL <http://www.martin-elff.net/MartinsRPackages>

Repository CRAN

Date/Publication 2009-11-12 11:33:13

R topics documented:

aggregate.formula	3
annotations	5
applyTemplate	6
as.array	8
as.symbols	8
bucket	9

By	12
cases	12
codebook	14
collect	16
contr	18
data.set	20
Descriptives	22
dimrename	23
foreach	24
getSummary	25
importers	26
include	30
items	31
labels	34
measurement	35
Memisc	36
mkHelpDir	40
mtable	41
negative match	45
panel.errbars	46
percent	47
prediction.frame	48
query	50
recode	51
relabel	54
rename	55
reorder.array	56
retain	58
sample-methods	58
Sapply	59
Simulate	60
sort-methods	63
styles	64
Substitute	65
Table	66
Termplot	67
to.data.frame	69
toLatex	70
trimws	72
UnZip	73
Utility classes	73
value.filter	74
Index	77

Description

`aggregate.formula` constructs a data frame of summaries conditional on given values of independent variables given by a formula. It is a method of the generic function `aggregate` applied to formula objects.

`genTable` does the same, but produces a table.

`fapply` is a generic function that dispatches on its `data` argument. It is called internally by `aggregate.formula` and `genTable`. Methods for this function can be used to adapt `aggregate.formula` and `genTable` to data sources other than data frames.

Usage

```
## S3 method for class 'formula':
aggregate(x, data=parent.frame(), subset=NULL,
          sort = TRUE, names=NULL, addFreq=TRUE, as.vars=1,
          drop.constants=TRUE, ...)

genTable(formula, data=parent.frame(), subset=NULL,
          names=NULL, addFreq=TRUE, ...)

fapply(formula,data,...) # calls UseMethod("fapply",data)
## Default S3 method:
fapply(formula, data, subset=NULL,
        names=NULL, addFreq=TRUE, ...)
```

Arguments

<code>x, formula</code>	a formula. The right hand side includes one or more grouping variables separated by '+'. These may be factors, numeric, or character vectors. The left hand side may be empty, a numerical variable, a factor, or an expression. See details below.
<code>data</code>	an environment or data frame or an object coercable into a data frame.
<code>subset</code>	an optional vector specifying a subset of observations to be used.
<code>sort</code>	a logical value; determines the order in which the aggregated data appear in the data frame returned by <code>aggregate.formula</code> . If <code>sort</code> is <code>TRUE</code> , then the returned data frame is sorted by the values of the grouping variables, if <code>sort</code> is <code>FALSE</code> , the order of resulting data frame corresponds to the order in which the values of the grouping variables appear in the original data frame.
<code>names</code>	an optional character vector giving names to the result(s) yielded by the expression on the left hand side of <code>formula</code> . This argument may be redundant if the left hand side results in is a named vector. (See the example below.)

<code>addFreq</code>	a logical value. If TRUE and <code>data</code> is a table or a data frame with a variable named "Freq", a call to <code>table</code> , <code>Table</code> , <code>percent</code> , or <code>nvalid</code> is supplied by an additional argument <code>Freq</code> and a call to <code>table</code> is translated into a call to <code>Table</code> .
<code>as.vars</code>	an integer; relevant only if the left hand side of the formula returns an array or a matrix - which dimension (rows, columns, or layers etc.) will transformed to variables? Defaults to columns in case of matrices and to the highest dimensional extend in case of arrays.
<code>drop.constants</code>	logical; variables that are constant across levels dropped from the result?
<code>...</code>	further arguments, passed to methods or ignored.

Details

If an expression is given as left hand side of the formula, its value is computed for any combination of values of the values on the right hand side. If the right hand side is a dot, then all variables in `data` are added to the right hand side of the formula.

If no expression is given as left hand side, then the frequency counts for the respective value combinations of the right hand variables are computed.

If a single factor is on the left hand side, then the left hand side is translated into an appropriate call to `table()`. Note that also in this case `addFreq` takes effect.

If a single numeric variable is on the left hand side, frequency counts weighted by this variable are computed. In these cases, `genTable` is equivalent to `xtabs` and `aggregate.formula` is equivalent to `as.data.frame(xtabs(...))`.

Value

`aggregate.formula` results in a data frame with conditional summaries and unique value combinations of conditioning variables.

`genTable` returns a [table](#), that is, an array with class "table".

See Also

[aggregate.data.frame](#), [xtabs](#)

Examples

```
ex.data <- expand.grid(mu=c(0,100),sigma=c(1,10))[rep(1:4,rep(100,4)),]
ex.data <- within(ex.data,
  x<-rnorm(
    n=nrow(ex.data),
    mean=mu,
    sd=sigma
  )
)

aggregate(~mu+sigma,data=ex.data)
aggregate(mean(x)~mu+sigma,data=ex.data)
```

```

aggregate(mean(x)~mu+sigma,data=ex.data,name="Average")
aggregate(c(mean(x),sd(x))~mu+sigma,data=ex.data)
aggregate(c(Mean=mean(x),StDev=sd(x),N=length(x))~mu+sigma,data=ex.data)
genTable(c(Mean=mean(x),StDev=sd(x),N=length(x))~mu+sigma,data=ex.data)

aggregate(table(Admit)~.,data=UCBAdmissions)
aggregate(Table(Admit,Freq)~.,data=UCBAdmissions)
aggregate(Admit~.,data=UCBAdmissions)
aggregate(percent(Admit)~.,data=UCBAdmissions)
aggregate(percent(Admit)~Gender,data=UCBAdmissions)
aggregate(percent(Admit)~Dept,data=UCBAdmissions)
aggregate(percent(Gender)~Dept,data=UCBAdmissions)
aggregate(percent(Admit)~Dept,data=UCBAdmissions,Gender=="Female")
genTable(percent(Admit)~Dept,data=UCBAdmissions,Gender=="Female")

```

annotations

Adding Annotations to Objects

Description

Annotations, that is, objects of class "annotation", are character vectors with all their elements named. Only one method is defined for this subclass of character vectors, a method for `show`, that shows the annotation in a nicely formatted way. Annotations of an object can be obtained via the function `annotation(x)` and can be set via `annotation(x)<-value`.

Elements of an annotation with names "description" and "wording" have a special meaning. The first kind can be obtained and set via `description(x)` and `description(x)<-value`, the second kind can be obtained via `wording(x)` and `wording(x)<-value`. "description" elements are used in way the "variable labels" are used in SPSS and Stata. "wording" elements of annotation objects are meant to contain the question wording of a questionnaire item represented by an "item" objects. These elements of annotations are treated in a special way in the output of the `coodbook` function.

Usage

```

annotation(x)
## S4 method for signature 'ANY':
annotation(x)
## S4 method for signature 'item':
annotation(x)
## S4 method for signature 'data.set':
annotation(x)
annotation(x)<-value
## S4 method for signature 'ANY,character':
annotation(x)<-value
## S4 method for signature 'item,annotation':
annotation(x)<-value
## S4 method for signature 'vector,annotation':
annotation(x)<-value

```

```

description(x)
description(x) <-value

wording(x)
wording(x) <-value

## S4 method for signature 'data.set':
description(x)
## S4 method for signature 'importer':
description(x)

```

Arguments

x	an object
value	a character or annotation object

Value

`annotation(x)` returns an object of class "annotation", which is a named character. `description(x)` and `wording(x)` each usually return a character string. If `description(x)` is applied to a `data.set` or an `importer` object, however, a character vector is returned, which is named after the variables in the data set or the external file.

Examples

```

vote <- sample(c(1,2,3,8,9,97,99),size=30,replace=TRUE)
labels(vote) <- c(Conservatives      = 1,
                 Labour             = 2,
                 "Liberal Democrats" = 3,
                 "Don't know"       = 8,
                 "Answer refused"   = 9,
                 "Not applicable"    = 97,
                 "Not asked in survey" = 99
                 )
missing.values(vote) <- c(97,99)
description(vote) <- "Vote intention"
wording(vote) <- "If a general election would take place next tuesday,
                the candidate of which party would you vote for?"
annotation(vote)
annotation(vote)["Remark"] <- "This is not a real questionnaire item, of course ..."
codebook(vote)

```

applyTemplate

Apply a Formatting Template to a Numeric or Character Vector

Description

`applyTemplate` is called internally by `mtable` to format coefficients and summary statistics.

Usage

```
applyTemplate(x, template, float.style=getOption("float.style"),
             digits=min(3, getOption("digits")),
             signif.symbols=getOption("signif.symbols"))
```

Arguments

x a numeric or character vector to be formatted

template a character vector that defines the template, see details.

float.style A character string that is passed to `formatC` by `applyTemplate`; valid values are "e", "f", "g", "fg", "E", and "G". By default, the `float.style` setting of `options` is used. The 'factory fresh' setting is `options(float.style="f")`

digits number of significant digits to use if not specified in the template.

signif.symbols a named vector that specifies how significance levels are symbolically indicated, values of the vector specify significance levels and names specify the symbols. By default, the `signif.symbols` setting of `options` is used. The "factory-fresh" setting is `options(signif.symbols=c("***"=.001, "**"=.01, "*"=.05))`.

Details

Character vectors that are used as templates may be arbitrary. However, certain character sequences may form *template expressions*. A template expression is of the form (`$<POS>:<Format spec>`), where "`$`" indicates the start of a template expression, "`<POS>`" stands for either an index or name that selects an element from `x` and "`<Format spec>`" stands for a *format specifier*. It may contain an letter indicating the style in which the vector element selected by `<POS>` will be formatted by `formatC`, it may contain a number as the number of significant digits, a "`#`" indicating that the number of significant digits will be at most that given by `getOption("digits")`, or `*` that means that the value will be formatted as a significance symbol.

Value

`applyTemplate` returns a character vector in which template expressions in `template` are substituted by formatted values from `x`. If `template` is an array then the return value is also an array of the same shape.

Examples

```
applyTemplate(c(a=.0000000000000304, b=3), template=c("$1:g7#") ($a:*), " (($1:f2)) ")
applyTemplate(c(a=.0000000000000304, b=3), template=c("$a:g7#") ($a:*), " (($b:f2)) ")
```

as.array

Converting Data Frames into Arrays

Description

The `as.array` for data frames takes all factors in a data frame and uses them to define the dimensions of the resulting array, and fills the array with the values of the remaining numeric variables.

Currently, the data frame must contain all combinations of factor levels.

Usage

```
## S4 method for signature 'data.frame':
as.array(x, data.name=NULL, ...)
```

Arguments

<code>x</code>	a data frame
<code>data.name</code>	a character string, giving the name attached to the dimension that corresponds to the numerical variables in the data frame (that is, the name attached to the corresponding element of the <code>dimnames</code> list).
<code>...</code>	other arguments, ignored.

Value

An array

Examples

```
BerkeleyAdmissions <- to.data.frame(UCBAdmissions)
BerkeleyAdmissions
as.array(BerkeleyAdmissions, data.name="Admit")
try(as.array(BerkeleyAdmissions[-1, ], data.name="Admit"))
```

as.symbols

Construction of Lists of Symbols

Description

`as.symbols` and `syms` are functions potentially useful in connection with `foreach` and `xapply`. `as.symbols` produces a list of symbols from a character vector, while `syms` returns a list of symbols from symbols given as arguments, but it can be used to construct patterns of symbols.

Usage

```
as.symbols(x)
syms(..., paste=FALSE, sep="")
```

Arguments

x a character vector
... character strings or (unquoted) variable names
paste logical value; should the character strings **paste**d into one string?
sep a separator string, passed to **paste**.

Value

A list of language symbols (results of `as.symbol` - not graphical symbols!).

Examples

```

as.symbols(letters[1:8])
syms("a",1:3,paste=TRUE)

sapply(syms("a",1:3,paste=TRUE),typeof)

```

 bucket

Abstract Data Structures to Collect Simulation Results

Description

Buckets (objects that inherit from S3 class "bucket") are abstract data structures that are internally used by function `Simulate` to collect results from individual replications of a simulation study.

Usage

```

# Generic functions used by 'Simulate'
put_into(bucket,value)
pour_out(bucket,...)

# This generates a bucket that puts data
# into a data frame
default_bucket(size=1)

## S3 method for class 'default\_bucket':
put\_into(bucket,value)
## S3 method for class 'default\_bucket':
pour\_out(bucket,...)
## S3 method for class 'default\_bucket':
dim(x)
## S3 method for class 'default\_bucket':
as.matrix(x,...)
## S3 method for class 'default\_bucket':
as.data.frame(x,...)

```

```

# This generates a bucket that puts data
# into a text file.
textfile_bucket(size=1,name=date())
## S3 method for class 'textfile\_bucket':
put\_into(bucket,value)
## S3 method for class 'textfile\_bucket':
pour\_out(bucket,...)
## S3 method for class 'textfile\_bucket':
dim(x)
## S3 method for class 'textfile\_bucket':
as.data.frame(x,...)
## S3 method for class 'textfile\_bucket':
as.matrix(x,...)

## S3 method for class 'bucket':
print(x,...)
## S3 method for class 'bucket':
x[i,...]

```

Arguments

bucket, x, data	an object that inherits from class "bucket".
value	a named list of results of a replication to put into the bucket.
size	a numerical value, the number of rows by which to extend the bucket if it is full. This will be set by <code>Simulate</code> either according to its <code>nsim</code> argument or according to <code>getOption("Simulation.chunk.size")</code>
name	a name for the textfile into which results are written.
i	a numerical vector to select replication results from the bucket.
...	other arguments, ignored or passed to other methods.

Details

If a user wants to provide another class of buckets for collecting results for 'Simulate' (s)he needs to define a function that returns a bucket object, as `default_bucket` and `textfile_bucket` do; and to define methods of `put_into`, `pour_out`, `dim`, `as.data.frame`, `as.matrix` for the newly defined bucket class.

The `put_into` method should add a list of values to the bucket, the `pour_out` method should close the bucket against further input and pour out in the data contained in the bucket into a fixed data structure. After `pour_out`, `put_into` should fail. The `default_bucket` method, e.g. puts the data into a data frame, the `textfile_bucket` method closes the textfile into which data were written.

Value

The function `default_bucket` returns an object of class "default_bucket", while function `textfile_bucket` returns an object of class "textfile_bucket".

The methods for `dim`, `as.data.frame`, and `as.matrix` give the usual return values, of the generic functions.

`put_into` returns nothing. `pour_out` returns the bucket.

Examples

```
Normal.example <- function(mean=0, sd=1, n=10) {
  x <- rnorm(n=n, mean=mean, sd=sd)
  c(
    Mean=mean(x),
    Median=median(x),
    Var=var(x)
  )
}
```

```
Sim_default_bucket <- Simulate(
  Normal.example(mean, sd, n),
  expand.grid(
    mean=0,
    sd=c(1, 10),
    n=c(10, 100)
  ),
  nsim=200)
```

```
tempfile_bucket <- function(n) textfile_bucket(n, name=tempfile())
```

```
Sim_textfile_bucket <- Simulate(
  Normal.example(mean, sd, n),
  expand.grid(
    mean=0,
    sd=c(1, 10),
    n=c(10, 100)
  ),
  nsim=200,
  bucket=tempfile_bucket
)
```

```
Sim_default_bucket
Sim_default_bucket[1:10]
```

```
Sim_textfile_bucket
Sim_textfile_bucket[1:10]
```

```
# Access of the textfile generated by 'textfile_bucket':
Sim_textfile_bucket$name
read.table(Sim_textfile_bucket$name, header=TRUE, nrow=10)
```

By *Conditional evaluation of an expression*

Description

The function `By` evaluates an expression within subsets of a data frame, where the subsets are defined by a formula.

Usage

```
By(formula, expr, data=parent.frame())
```

Arguments

<code>formula</code>	an expression or (preferably) a formula containing the names of conditioning variables or factors.
<code>expr</code>	an expression that is evaluated for any unique combination of values of the variables contained in <code>formula</code> .
<code>data</code>	a data frame, an object that can be coerced into a data frame (for example, a table), or an environment, from which values for the variables in <code>formula</code> or <code>expr</code> are taken.

Value

A list of class "by", giving the results for each combination of values of variables in `formula`.

Examples

```
berkeley <- aggregate(Table(Admit, Freq) ~ ., data=UCBAdmissions)
By(~Dept, glm(cbind(Admitted, Rejected) ~ Gender, family="binomial"), data=berkeley)

attach(berkeley)
By(~Dept, glm(cbind(Admitted, Rejected) ~ Gender, family="binomial"))
detach(berkeley)
```

cases *Distinguish between Cases Specified by Logical Conditions*

Description

`cases` allows to simultaneously several cases determined by logical conditions. It can be used to code these case into a factor or as a multi-condition generalization of [ifelse](#)

Usage

```
cases(..., check.xor=FALSE)
```

Arguments

`...` A sequence of logical expressions or assignment expressions containing logical expressions as "right hand side".

`check.xor` logical; if true, `cases` checks, whether the case conditions are mutually exclusive and exhaustive. If this is not satisfied and `check.xor` an error exception is raised.

Details

There are two distinct ways to use this function. Either the function can be used to construct a factor that represents several logical cases or it can be used to conditionally evaluate an expression in a manner similar to `ifelse`.

For the first use, the `...` arguments have to be a series of logical expressions. `cases` then returns a factor with as many levels as logical expressions given as `...` arguments. The resulting factor will attain its first level if the first condition is TRUE, otherwise it will attain its second level if the second condition is TRUE, etc. The levels will be named after the conditions or, if name tags are attached to the logical expressions, after the tags of the expressions. Note that the logical expressions all need to evaluate to logical vectors of the same length, otherwise an error condition is raised.

For the second use, the `...` arguments have to be a series of assignment expression of the type `<expression> <- <logical expression> or <logical expression> -> <expression>`. For cases in which the first logical expression is TRUE, the result of first expression that appears on the other side of the assignment operator become elements of the vector returned by `cases`, for cases in which the second logical expression is TRUE, the result of the second expression that appears on the other side of the assignment operator become elements of the vector returned by `cases`, etc. Note that the logical expressions also here all need to evaluate to logical vectors of the same length. The expressions on the other side of the assignment operator should also be either vectors of the same length and mode or should be scalars of the same mode, otherwise unpredictable results may occur.

Value

If it is called with logical expressions as `...` arguments, `cases` returns a factor, if it is called with assignment expressions the function returns a vector with the same mode as the results of the "assigned" expressions and with the same length as the logical conditions.

Examples

```
# Examples of the first kind of usage of the function
#
df <- data.frame(x = rnorm(n=20), y = rnorm(n=20))
df <- df[do.call(order,df),]
(df <- within(df, {
  x1=cases(x>0, x<=0)
  y1=cases(y>0, y<=0)
```

```

z1=cases(
  "Condition 1"=x<0,
  "Condition 2"=y<0,# only applies if x >= 0
  "Condition 3"=TRUE
)
z2=cases(x<0,(x>=0 & y <0), (x>=0 & y >=0))
}))
xtabs(~x1+y1,data=df)
dd <- with(df,
  try(cases(x<0,
            x>=0,
            x>1,
            check.xor=TRUE)# let's be fussy
      )
    )
dd <- with(df,
  try(cases(x<0,x>=0,x>1))
    )
genTable(range(x)~dd,data=df)

# An example of the second kind of usage of the function:
# A construction of a non-smooth function
#
fun <- function(x)
  cases(
    x==0      -> 1,
    abs(x)> 1 -> abs(x),
    abs(x)<=1 -> x^2
  )
x <- seq(from=-2,to=2,length=101)
plot(fun(x)~x)

```

codebook

Generate a Codebook of a Data Set

Description

Function `codebook` collects documentation about an item, or the items in a data set or external data file. It returns an object that, when shown, print this documentation in a nicely formatted way.

Usage

```

codebook(x)
## S4 method for signature 'data.set':
codebook(x)
## S4 method for signature 'importer':
codebook(x)

```

Arguments

`x` an `item`, `data.set` or `importer` object.

Value

An object of class "codebook", for which a `show` method exists that produces a nicely formatted output.

Examples

```
Data <- data.set(
  vote = sample(c(1,2,3,8,9,97,99),size=300,replace=TRUE),
  region = sample(c(rep(1,3),rep(2,2),3,99),size=300,replace=TRUE),
  income = exp(rnorm(300,sd=.7))*2000
)

Data <- within(Data,{
  description(vote) <- "Vote intention"
  description(region) <- "Region of residence"
  description(income) <- "Household income"
  wording(vote) <- "If a general election would take place next tuesday,
    the candidate of which party would you vote for?"
  wording(income) <- "All things taken into account, how much do all
    household members earn in sum?"
  foreach(x=c(vote,region),{
    measurement(x) <- "nominal"
  })
  measurement(income) <- "ratio"
  labels(vote) <- c(
    Conservatives = 1,
    Labour = 2,
    "Liberal Democrats" = 3,
    "Don't know" = 8,
    "Answer refused" = 9,
    "Not applicable" = 97,
    "Not asked in survey" = 99)
  labels(region) <- c(
    England = 1,
    Scotland = 2,
    Wales = 3,
    "Not applicable" = 97,
    "Not asked in survey" = 99)
  foreach(x=c(vote,region,income),{
    annotation(x)["Remark"] <- "This is not a real survey item, of course ..."
  })
  missing.values(vote) <- c(8,9,97,99)
  missing.values(region) <- c(97,99)
})
```

```
codebook(Data)
```

 collect

Collect Objects

Description

collect gathers several objects into one, matching the elements or subsets of the objects by [names](#) or [dimnames](#).

Usage

```
collect(..., names=NULL, inclusive=TRUE)
## Default S3 method:
collect(..., names=NULL, inclusive=TRUE)
## S3 method for class 'array':
collect(..., names=NULL, inclusive=TRUE)
## S3 method for class 'matrix':
collect(..., names=NULL, inclusive=TRUE)
## S3 method for class 'table':
collect(..., names=NULL, sourcename="arg", fill=0)
## S3 method for class 'data.frame':
collect(..., names=NULL, inclusive=TRUE,
        fussy=FALSE, warn=TRUE, sourcename="arg")
## S3 method for class 'data.set':
collect(..., names=NULL, inclusive=TRUE,
        fussy=FALSE, warn=TRUE, sourcename="arg")
```

Arguments

...	more atomic vectors, arrays, matrices, tables, data.frames or data.sets
names	optional character vector; in case of the default and array methods, giving dimnames for the new dimension that identifies the collected objects; in case of the data.frame and data.set methods, levels of a factor indentifying the collected objects.
inclusive	logical, defaults to TRUE; should unmatched elements included? See details below.
fussy	logical, defaults to FALSE; should it count as an error, if variables with same names of collected data.frames/data.sets have different attributes?
warn	logical, defaults to TRUE; should an warning be given, if variables with same names of collected data.frames/data.sets have different attributes?
sourcename	name of the factor that identifies the collected data.frames or data.sets
fill	numeric; with what to fill empty table cells, defaults to zero, assuming the table contains counts

Value

If x and all following \dots arguments are vectors of the same mode (numeric, character, or logical) the result is a matrix with as many columns as vectors. If argument `inclusive` is `TRUE`, then the number of rows equals the number of names that appear at least once in each of the vector names and the matrix is filled with `NA` where necessary, otherwise the number of rows equals the number of names that are present in *all* vector names.

If x and all \dots arguments are matrices or arrays of the same mode (numeric, character, or logical) and n dimension the result will be a $n+1$ dimensional array or table. The extend of the $n+1$ th dimension equals the number of matrix, array or table arguments, the extends of the lower dimension depends on the `inclusive` argument: either they equal to the number of dimnames that appear at least once for each given dimension and the array is filled with `NA` where necessary, or they equal to the number of dimnames that appear in all arguments for each given dimension.

If x and all \dots arguments are tables then the result will be a table. The result

If x and all \dots arguments are data frames or data sets, the result is a data frame or data set. The number of variables of the resulting data frame or data set depends on the `inclusive` argument. If it is true, the number of variables equals the number of variables that appear in each of the arguments at least once and variables are filled with `NA` where necessary, otherwise the number of variables equals the number of variables that are present in all arguments.

Examples

```
x <- c(a=1,b=2)
y <- c(a=10,c=30)

x
y

collect(x,y)
collect(x,y,inclusive=FALSE)

X <- matrix(1,nrow=2,ncol=2,dimnames=list(letters[1:2],LETTERS[1:2]))
Y <- matrix(2,nrow=3,ncol=2,dimnames=list(letters[1:3],LETTERS[1:2]))
Z <- matrix(3,nrow=2,ncol=3,dimnames=list(letters[1:2],LETTERS[1:3]))

X
Y
Z

collect(X,Y,Z)
collect(X,Y,Z,inclusive=FALSE)

X <- matrix(1,nrow=2,ncol=2,dimnames=list(a=letters[1:2],b=LETTERS[1:2]))
Y <- matrix(2,nrow=3,ncol=2,dimnames=list(a=letters[1:3],c=LETTERS[1:2]))
Z <- matrix(3,nrow=2,ncol=3,dimnames=list(a=letters[1:2],c=LETTERS[1:3]))

collect(X,Y,Z)
collect(X,Y,Z,inclusive=FALSE)

df1 <- data.frame(a=rep(1,5),b=rep(1,5))
df2 <- data.frame(a=rep(2,5),b=rep(2,5),c=rep(2,5))
```

```

collect(df1,df2)
collect(df1,df2,inclusive=FALSE)

data(UCBAdmissions)
Male <- as.table(UCBAdmissions[,1,])
Female <- as.table(UCBAdmissions[,2,])
collect(Male,Female,sourcename="Gender")
collect(unclass(Male),unclass(Female))

Male1 <- as.table(UCBAdmissions[,1,-1])
Female2 <- as.table(UCBAdmissions[,2,-2])
Female3 <- as.table(UCBAdmissions[,2,-3])
collect(Male=Male1,Female=Female2,sourcename="Gender")
collect(Male=Male1,Female=Female3,sourcename="Gender")
collect(Male=Male1,Female=Female3,sourcename="Gender",fill=NA)

f1 <- gl(3,5,labels=letters[1:3])
f2 <- gl(3,6,labels=letters[1:3])
collect(f1=table(f1),f2=table(f2))

```

 contr

Convenience Methods for Setting Contrasts

Description

This package provides modified versions of `contr.treatment` and `contr.sum`. `contr.sum` gains an optional base argument, analog to the one of `contr.treatment`, furthermore, the base argument may be the name of a factor level.

`contr` returns a function that calls either `contr.treatment`, `contr.sum`, etc., according to the value given to its first argument.

The `contrasts` method for "item" objects returns a contrast matrix or a function to produce a contrast matrix for the factor into which the item would be coerced via `as.factor` or `as.ordered`. This matrix or function can be specified by using `contrasts(x) <- value`

Usage

```

contr(type,...)
contr.treatment(n, base=1,contrasts=TRUE)
contr.sum(n,base=NULL,contrasts=TRUE)
## S4 method for signature 'item':
contrasts(x,contrasts=TRUE)
## S4 method for signature 'item':
contrasts(x,how.many) <- value
# These methods are defined implicitly by making 'contrasts' generic.
## S4 method for signature 'ANY':
contrasts(x,contrasts=TRUE)
## S4 method for signature 'ANY':
contrasts(x,how.many) <- value

```

Arguments

<code>type</code>	a character vector, specifying the type of the contrasts. This argument should have a value such that, if e.g. <code>type="something"</code> , then there is a function <code>contr.something</code> that produces a contrast matrix.
<code>...</code>	further arguments, passed to <code>contr.treatment</code> , etc.
<code>n</code>	a number of factor levels or a vector of factor levels names, see e.g. <code>contr.treatment</code> .
<code>base</code>	a number of a factor level or the names of a factor level, which specifies the baseline category, see e.g. <code>contr.treatment</code> or <code>NULL</code> .
<code>contrasts</code>	a logical value, see <code>contrasts</code>
<code>how.many</code>	the number of contrasts to generate, see <code>contrasts</code>
<code>x</code>	a factor or an object of class "item"
<code>value</code>	a matrix, a function or the name of a function

Value

`contr` returns a function that calls one of `contr.treatment`, `contr.sum`, ... `contr.treatment` and `contr.sum` return contrast matrices. `contrasts(x)` returns the "contrasts" attribute of an object, which may be a function name, a function, a contrast matrix or `NULL`.

Examples

```
ctr.t <- contr("treatment",base="c")
ctr.t
ctr.s <- contr("sum",base="c")
ctr.h <- contr("helmert")
ctr.t(letters[1:7])
ctr.s(letters[1:7])
ctr.h(letters[1:7])

x <- factor(rep(letters[1:5],3))
contrasts(x)
x <- as.item(x)
contrasts(x)
contrasts(x) <- contr.sum(letters[1:5],base="c")
contrasts(x)
missing.values(x) <- 5
contrasts(x)
contrasts(as.factor(x))

# Obviously setting missing values after specifying
# contrast matrix breaks the contrasts.
# Using the 'contr' function, however, prevents this:

missing.values(x) <- NULL
contrasts(x) <- contr("sum",base="c")
contrasts(x)
missing.values(x) <- 5
contrasts(x)
contrasts(as.factor(x))
```

Description

"data.set" objects are collections of "item" objects, with similar semantics as data frames. They are distinguished from data frames so that coercion by `as.data.frame` leads to a data frame that contains only vectors and factors. Nevertheless most methods for data frames are inherited by data sets, except for the method for the `within` generic function. For the `within` method for data sets, see the details section.

Thus data preparation using data sets retains all informations about item annotations, labels, missing values etc. While (mostly automatic) conversion of data sets into data frames makes the data amenable for the use of R's statistical functions.

Usage

```
data.set(..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
         stringsAsFactors = default.stringsAsFactors(),
         document = NULL)
as.data.set(x, row.names=NULL, optional=NULL, ...)
is.data.set(x)
## S3 method for class 'data.set':
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
## S4 method for signature 'data.set':
within(data, expr, ...)
```

Arguments

<code>...</code>	For the <code>data.set</code> function several vectors or items, for <code>within</code> further, ignored arguments.
<code>row.names</code> , <code>check.rows</code> , <code>check.names</code> , <code>stringsAsFactors</code> , <code>optional</code>	arguments as in data.frame or as.data.frame , respectively.
<code>document</code>	NULL or an optional character vector that contains documentation of the data.
<code>x</code>	for <code>is.data.set(x)</code> , any object; for <code>as.data.frame(x, ...)</code> a "data.set" object.
<code>data</code>	a data set, that is, an object of class "data.set".
<code>expr</code>	an expression, or several expressions enclosed in curly braces.

Details

The `as.data.frame` method for data sets is just a copy of the method for `list`. Consequently, all items in the data set are coerced in accordance to their `measurement` setting, see [item](#) and [measurement](#).

The `within` method for data sets has the same effect as the `within` method for data frames, apart from two differences: all results of the computations are coerced into items if they have the appropriate length, otherwise, they are automatically dropped.

Currently only one method for the generic function `as.data.set` is defined: a method for "importer" objects.

Value

`data.set` and the `within` method for data sets returns a "data.set" object, `is.data.set` returns a logical value, and `as.data.frame` returns a data frame.

Examples

```
Data <- data.set(
  vote = sample(c(1,2,3,8,9,97,99),size=300,replace=TRUE),
  region = sample(c(rep(1,3),rep(2,2),3,99),size=300,replace=TRUE),
  income = exp(rnorm(300,sd=.7))*2000
)

Data <- within(Data,{
  description(vote) <- "Vote intention"
  description(region) <- "Region of residence"
  description(income) <- "Household income"
  wording(vote) <- "If a general election would take place next tuesday,
    the candidate of which party would you vote for?"
  wording(income) <- "All things taken into account, how much do all
    household members earn in sum?"
  foreach(x=c(vote,region),{
    measurement(x) <- "nominal"
  })
  measurement(income) <- "ratio"
  labels(vote) <- c(
    Conservatives = 1,
    Labour = 2,
    "Liberal Democrats" = 3,
    "Don't know" = 8,
    "Answer refused" = 9,
    "Not applicable" = 97,
    "Not asked in survey" = 99)
  labels(region) <- c(
    England = 1,
    Scotland = 2,
    Wales = 3,
    "Not applicable" = 97,
    "Not asked in survey" = 99)
  foreach(x=c(vote,region,income),{
    annotation(x)["Remark"] <- "This is not a real survey item, of course ..."
  })
  missing.values(vote) <- c(8,9,97,99)
  missing.values(region) <- c(97,99)

  # These two variables do not appear in the
  # the resulting data set, since they have the wrong length.
  junk1 <- 1:5
  junk2 <- matrix(5,4,4)
```

```

})
# Since data sets may be huge, only a
# part of them are 'show'n
Data
# If we insist on seeing all, we can use 'print' instead
## Not run:
  print(Data)

## End(Not run)
str(Data)
summary(Data)

Data[[1]]
Data[1,]
head(as.data.frame(Data))

EnglandData <- subset(Data, region == "England")
EnglandData

xtabs(~vote+region, data=Data)
xtabs(~vote+region, data=within(Data, vote <- include.missings(vote)))

```

Descriptives

Vectors of Univariate Sample Statistics

Description

Descriptives(x) gives a vector of sample statistics for use in [codebook](#).

Usage

```

Descriptives(x, ...)
## S4 method for signature 'atomic':
Descriptives(x, ...)
## S4 method for signature 'item.vector':
Descriptives(x, ...)

```

Arguments

x an atomic vector or "item.vector" object.
... further arguments, to be passed to future methods.

Value

A numeric vector of sample statistics, containing the range, the mean, the standard deviation, the skewness and the (excess) kurtosis.

Examples

```
x <- rnorm(100)
Descriptives(x)
```

dimrename	<i>Change dimnames, rownames, or colnames</i>
-----------	---

Description

These functions provide an easy way to change the `dimnames`, `rownames` or `colnames` of an array.

Usage

```
dimrename(x, dim = 1, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
rowrename(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
colrename(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
```

Arguments

<code>x</code>	An array with <code>dimnames</code>
<code>dim</code>	A vector that indicates the dimensions
<code>...</code>	A sequence of named arguments
<code>gsub</code>	a logical value; if <code>TRUE</code> , <code>gsub</code> is used to change the <code>dimnames</code> of the object. That is, instead of substituting whole names, substrings of the <code>dimnames</code> of the object can be changed.
<code>fixed</code>	a logical value, passed to <code>gsub</code> . If <code>TRUE</code> , substitutions are by fixed strings and not by regular expressions.
<code>warn</code>	logical; should a warning be issued if the pattern is not found?

Details

`dimrename` changes the `dimnames` of `x` along dimension(s) `dim` according to the remaining arguments. The argument names are the *old* names, the values are the new names. `rowrename` is a shorthand for changing the `rownames`, `colrename` is a shorthand for changing the `colnames` of a matrix or matrix-like object.

If `gsub` is `FALSE`, argument tags are the *old* `dimnames`, the values are the new `dimnames`. If `gsub` is `TRUE`, arguments are substrings of the `dimnames` that are substituted by the argument values.

Value

Object `x` with changed `dimnames`.

Examples

```

m <- matrix(1,2,2)
rownames(m) <- letters[1:2]
colnames(m) <- LETTERS[1:2]
m
dimrename(m,1,a="first",b="second")
dimrename(m,1,A="first",B="second")
dimrename(m,2,"A"="first",B="second")

rowrename(m,a="first",b="second")
colrename(m,"A"="first",B="second")

```

 foreach

Loop over Variables in a Data Frame or Environment

Description

foreach evaluates an expression given as second argument by substituting in variables. The expression may also contain assignments, which take effect in the callers environment.

Usage

```
foreach(...)
```

Arguments

... tagged and untagged arguments. The tagged arguments define the 'variables' that are looped over, the first untagged argument defines the expression which is evaluated.

Examples

```

x <- 1:3
y <- -(1:3)
z <- c("Uri", "Schwyz", "Unterwalden")
print(x)
print(y)
print(z)
foreach(var=c(x,y,z),           # assigns names
        names(var) <- letters[1:3] # to the elements of x, y, and z
        )
print(x)
print(y)
print(z)

ds <- data.set(
  a = c(1,2,3,2,3,8,9),
  b = c(2,8,3,2,1,8,9),
  c = c(1,3,2,1,2,8,8)
)

```

```

    )
  print(ds)
  ds <- within(ds, {
    description(a) <- "First item in questionnaire"
    description(b) <- "Second item in questionnaire"
    description(c) <- "Third item in questionnaire"

    wording(a) <- "What number do you like first?"
    wording(b) <- "What number do you like second?"
    wording(c) <- "What number do you like third?"

    foreach(x=c(a,b,c), { # Lazy data documentation:
      labels(x) <- c( # a,b,c get value labels in one statement
                    one = 1,
                    two = 2,
                    three = 3,
                    "don't know" = 8,
                    "refused to answer" = 9)
      missing.values(x) <- c(8,9)
    })
  })

  as.data.frame(ds)

  ds <- within(ds, foreach(x=c(a,b,c), {
    measurement(x) <- "interval"
  })))

  as.data.frame(ds)

```

 getSummary

Get Model Summaries for Use with "mtable"

Description

A generic function and methods to collect coefficients and summary statistics from a model object. It is used in [mtable](#)

Usage

```

## S3 method for class 'lm':
getSummary(obj, alpha=.05, ...)
## S3 method for class 'glm':
getSummary(obj, alpha=.05, ...)

```

Arguments

obj	a model object, e.g. of class <code>lm</code> or <code>glm</code>
alpha	level of the confidence intervals; their coverage should be $1-\alpha/2$
...	further arguments; ignored.

Details

The generic function `getSummary` is called by `mtable` in order to obtain the coefficients and summaries of model objects. In order to adapt `mtable` to models of classes other than `lm` or `glm` one needs to define `getSummary` methods for these classes and to set a summary template via `setSummaryTemplate`

Value

Any method of `getSummary` must return a list with the following components:

`coef` an array with coefficient estimates; the lowest dimension *must* have the following names and meanings:

<code>est</code>	the coefficient estimates,
<code>se</code>	the estimated standard errors,
<code>stat</code>	t- or Wald-z statistics,
<code>p</code>	significance levels of the statistics,
<code>lwr</code>	lower confidence limits,
<code>upr</code>	upper confidence limits.

The higher dimensions of the array correspond to the individual coefficients and, in multi-equation models, to the model equations.

`sumstat` a vector containing the model summary statistics; the components may have arbitrary names.

importers

Object Oriented Intervace to Foreign Files

Description

Importer objects are objects that refer to an external data file. Currently only Stata files, SPSS system, portable, and fixed-column files are supported.

Data are actually imported by ‘translating’ an importer file into a `data.set` using `as.data.set` or `subset`.

The `importer` mechanism is more flexible and extensible than `read.spss` and `read.dta` of package "foreign", as most of the parsing of the file headers is done in R. They are also adapted to load efficiently large data sets. Most importantly, importer objects support the `labels`, `missing.values`, and `descriptions`, provided by this package.

Usage

```
spss.fixed.file(file,
  columns.file,
  varlab.file=NULL,
  codes.file=NULL,
```

```

    missval.file=NULL,
    count.cases=TRUE
  )

spss.portable.file(file,
  varlab.file=NULL,
  codes.file=NULL,
  missval.file=NULL,
  count.cases=TRUE)

spss.system.file(file,
  varlab.file=NULL,
  codes.file=NULL,
  missval.file=NULL,
  count.cases=TRUE)

Stata.file(file)

## The most important methods for "importer" objects are:
## S4 method for signature 'importer':
subset(x, subset, select, drop = FALSE, ...)

## S4 method for signature 'importer':
as.data.set(x, row.names=NULL, optional=NULL,
            compress.storage.modes=TRUE, ...)

```

Arguments

<code>x</code>	an object that inherits from class "importer".
<code>file</code>	character string; the path to the file containing the data
<code>columns.file</code>	character string; the path to an SPSS/PSPP syntax file with a DATA LIST FIXED statement
<code>varlab.file</code>	character string; the path to an SPSS/PSPP syntax file with a VARIABLE LABELS statement
<code>codes.file</code>	character string; the path to an SPSS/PSPP syntax file with a VALUE LABELS statement
<code>missval.file</code>	character string; the path to an SPSS/PSPP syntax file with a MISSING VALUES statement
<code>count.cases</code>	logical; should cases in file be counted? This takes effect only if the data file does not already contain information about the number of cases.
<code>subset</code>	a logical vector or an expression containing variables from the external data file that evaluates to logical.
<code>select</code>	a vector of variable names from the external data file. This may also be a named vector, where the names give the names into which the variables from the external data file are renamed.

<code>drop</code>	a logical value, that determines what happens if only one column is selected. If <code>TRUE</code> and only one column is selected, <code>subset</code> returns only a single <code>item</code> object and not a <code>data.set</code> .
<code>row.names</code>	ignored, present only for compatibility.
<code>optional</code>	ignored, present only for compatibility.
<code>compress.storage.modes</code>	logical value; if <code>TRUE</code> floating point values are converted to integers if possible without loss of information.
<code>...</code>	other arguments; ignored.

Details

A call to a ‘constructor’ for an importer object, that is, `spss.fixed.file`, `spss.portable.file`, `spss.syntax.file`, or `Stata.file`, causes R to read in the header of the data file and/or the syntax files that contain information about the variables, such as the columns that they occupy (in case of `spss.fixed.file`), variable labels, value labels and missing values.

The information in the file header and/or the accompanying files is then processed to prepare the file for importing. Thus the inner structure of an `importer` object may well vary according to what type of file is to be imported and what additional information is given.

The `as.data.set` and `subset` methods for "importer" objects internally use the generic functions `seekData`, `readData`, and `readSubset`, which have methods for the subclasses of "importer". These functions are not callable from outside the package, however.

Value

`spss.fixed.file`, `spss.portable.file`, `spss.system.file`, and `Stata.file` return, respectively, objects of class "spss.fixed.importer", "spss.portable.importer", "spss.system.importer", or "Stata.importer", which, by inheritance, are also objects of class "importer".

Objects of class "importer" have at least the following two slots:

<code>ptr</code>	an external pointer
<code>variables</code>	a list of objects of class "item.vector" which provides a ‘prototype’ for the "data.set" set objects returned by the <code>as.data.set</code> and <code>subset</code> methods for objects of class "importer"

The `as.data.frame` for `importer` objects does the actual data import and returns a data frame. Note that in contrast to `read.spss`, the variable names of the resulting data frame will be lower case. If long variable names are defined (in case of a PSPP/SPSS system file), they take precedence and are *not* coerced to lower case.

See Also

[codebook](#), [description](#), [read.spss](#)

Examples

```
# Extract American National Election Study of 1948
nes1948.por <- UnZip("anes/NES1948.ZIP", "NES1948.POR",
                    package="memisc")

# Get information about the variables contained.
nes1948 <- spss.portable.file(nes1948.por)

# The data are not yet loaded:
show(nes1948)

# ... but one can see what variables are present:
description(nes1948)

# Now a subset of the data is loaded:
vote.socdem.48 <- subset(nes1948,
                        select=c(
                            v480018,
                            v480029,
                            v480030,
                            v480045,
                            v480046,
                            v480047,
                            v480048,
                            v480049,
                            v480050
                        ))

# Let's make the names more descriptive:
vote.socdem.48 <- rename(vote.socdem.48,
                        v480018 = "vote",
                        v480029 = "occupation.hh",
                        v480030 = "unionized.hh",
                        v480045 = "gender",
                        v480046 = "race",
                        v480047 = "age",
                        v480048 = "education",
                        v480049 = "total.income",
                        v480050 = "religious.pref"
                       )

# It is also possible to do both
# in one step:
# vote.socdem.48 <- subset(nes1948,
#                          select=c(
#                              vote           = v480018,
#                              occupation.hh = v480029,
#                              unionized.hh  = v480030,
#                              gender        = v480045,
#                              race          = v480046,
#                              age           = v480047,
#                              education     = v480048,
```

```

#           total.income  = v480049,
#           religious.pref = v480050
#           ))

# We examine the data more closely:
codebook(vote.socdem.48)

# ... and conduct some analyses.
#
t(genTable(percent(vote)~occupation.hh,data=vote.socdem.48))

# We consider only the two main candidates.
vote.socdem.48 <- within(vote.socdem.48,{
  truman.dewey <- vote
  valid.values(truman.dewey) <- 1:2
  truman.dewey <- relabel(truman.dewey,
    "VOTED - FOR TRUMAN" = "Truman",
    "VOTED - FOR DEWEY"  = "Dewey")
})

summary(truman.relig.glm <- glm((truman.dewey=="Truman")~religious.pref,
  data=vote.socdem.48,
  family="binomial",
  ))

```

include

Attach a Source File to the Search Path

Description

Functions `include` and `uninclude` provide a simple mechanism for modularisation without the need to code a full-blown package. Functions in an included sourcefile are attached to the search path, but are not visible in the global environment. If an experienced user has written quite a number of utility functions for her/his daily work, but does not want to have listed them in each `ls()` call, s/he can just `include` a source file containing these utility functions. `uninclude` just reverts the attachment of a source file into the search path. `detach.sources` detaches all source files from the search path.

Usage

```

include(filename, warn=FALSE)
uninclude(filename)
detach.sources()

```

Arguments

<code>filename</code>	character string, path to a file containing R code.
<code>warn</code>	logical value; should a warning issued if the file has already been included?

Examples

```
## Not run:
include("~/R/my-functions.R")
include("~/R/more-of-my-functions.R")

ls() # The functions defined in these files will not show up here

# ... but here
search()
ls("source:~/R/my-functions.R")
ls("source:~/R/more-of-my-functions.R")

z <- one.of.my.functions(100) # ... and are available.

uninclude("~/R/my-functions.R")

detach.sources()

## End(Not run)
```

 items

Survey Items

Description

Objects of class `item` are data vectors with additional information attached to them like “value labels” and “user-defined missing values” known from software packages like SPSS or Stata.

Usage

```
## The constructor for objects of class "item"
## more convenient than new("item",...)
## S4 method for signature 'numeric':
as.item(x,
  labels=NULL, missing.values=NULL,
  valid.values=NULL, valid.range=NULL,
  value.filter=NULL, measurement=NULL,
  annotation=attr(x,"annotation"), ...
)
## S4 method for signature 'character':
as.item(x,
  labels=NULL, missing.values=NULL,
  valid.values=NULL, valid.range=NULL,
  value.filter=NULL, measurement=NULL,
  annotation=attr(x,"annotation"), ...
)

## S4 method for signature 'logical':
```

```

as.item(x,...)
# x is first coerced to integer,
# arguments in ... are then passed to the "numeric"
# method.

## S4 method for signature 'factor':
as.item(x,...)
## S4 method for signature 'ordered':
as.item(x,...)

## S4 method for signature 'double.item':
as.item(x,
  labels=NULL, missing.values=NULL,
  valid.values=NULL, valid.range=NULL,
  value.filter=NULL, measurement=NULL,
  annotation=attr(x,"annotation"), ...
)

## S4 method for signature 'integer.item':
as.item(x,
  labels=NULL, missing.values=NULL,
  valid.values=NULL, valid.range=NULL,
  value.filter=NULL, measurement=NULL,
  annotation=attr(x,"annotation"), ...
)

## S4 method for signature 'character.item':
as.item(x,
  labels=NULL, missing.values=NULL,
  valid.values=NULL, valid.range=NULL,
  value.filter=NULL, measurement=NULL,
  annotation=attr(x,"annotation"), ...
)

```

Arguments

<code>x</code>	for <code>as.item</code> methods, any atomic vector; for the <code>as.character</code> , <code>as.factor</code> , <code>as.integer</code> , <code>as.double</code> , a vector with class "item"; for the <code>unique</code> , <code>summary</code> , <code>str</code> , <code>print</code> , <code>[</code> , and <code><-</code> methods, a vector with class labelled.
<code>labels</code>	a named vector of the same mode as <code>x</code> .
<code>missing.values</code>	either a vector of the same mode as <code>x</code> , or a list with components "values", vector of the same mode as <code>x</code> (which defines individual missing values) and "range" a matrix with two rows with the same mode as <code>x</code> (which defines a range of missing values), or an object of class "missing.values".
<code>valid.values</code>	either a vector of the same mode as <code>x</code> , defining those values of <code>x</code> that are to be considered as valid, or an object of class "valid.values".

`valid.range` either a vector of the same mode as `x` and length 2, defining a range of valid values of `x`, or an object of class `"valid.range"`.
`value.filter` an object of class `"value.filter"`, that is, of classes `"missing.values"`, `"valid.values"`, or `"valid.range"`.
`measurement` level of measurement; one of `"nominal"`, `"ordinal"`, `"interval"`, or `"ratio"`.
`annotation` a named character vector, or an object of class `"annotation"`
`...` further arguments, ignored.

See Also

[annotation labels value.filter](#)

Examples

```

x <- as.item(rep(1:5,4),
  labels=c(
    "First"      = 1,
    "Second"     = 2,
    "Third"      = 3,
    "Fourth"     = 4,
    "Don't know" = 5
  ),
  missing.values=5,
  annotation = c(
    description="test"
  )
)
str(x)
summary(x)
as.numeric(x)

test <- as.item(rep(1:6,2), labels=structure(1:6,
                                             names=letters[1:6]))

test
test == 1
test != 1
test == "a"
test != "a"
test == c("a", "z")
test != c("a", "z")
test
test

codebook(test)

Test <- as.item(rep(letters[1:6],2),
  labels=structure(letters[1:6],
                  names=LETTERS[1:6]))

Test
Test == "a"
Test != "a"
Test == "A"
  
```

```

Test != "A"
Test == c("a", "z")
Test != c("a", "z")
Test
Test

as.factor(test)
as.factor(Test)
as.numeric(test)
as.character(test)
as.character(Test)

as.data.frame(test)[[1]]

```

labels

Value Labels

Description

Value labels associate character labels to possible values of an encoded survey item. Value labels are represented as objects of class "value.labels".

Value labels of an item can be obtained using `labels(x)` and can be associated to items and to vectors using `labels(x) <- value`

Value labels also can be updated using the + and - operators.

Usage

```

labels(object, ...)
labels(x) <- value

```

Arguments

object	any object.
...	further arguments for other methods.
x	a vector or "item" object.
value	an object of class "value.labels" or a vector that can be coerced into an "value.labels" object or NULL

Examples

```

x <- as.item(rep(1:5, 4),
  labels=c(
    "First"      = 1,
    "Second"    = 2,
    "Third"     = 3,
    "Fourth"    = 4,
    "Don't know" = 5
  )
),

```

```

    missing.values=5,
    annotation = c(
      description="test "
    )
  labels(x)
  labels(x) <- labels(x) - c("Second"=2)
  labels(x)
  labels(x) <- labels(x) + c("Second"=2)
  labels(x)

  puv1 <- getOption("print.use.value.labels")
  options(print.use.value.labels=FALSE)
  x
  options(print.use.value.labels=TRUE)
  x
  options(print.use.value.labels=puv1)

```

 measurement

Levels of Measurement of Survey Items

Description

The measurement level of a "item" object, which is one of "nominal", "ordinal", "interval", "ratio", determines what happens to it, if it or the `data.set` containing it is coerced into a `data.frame`. If the level of measurement level is "nominal", the it will be converted into an (unordered) **factor**, if the level of measurement is "ordinal", the item will be converted into an **ordered** vector. If the measurement is "interval" or "ratio", the item will be converted into a numerical vector.

Usage

```

## S4 method for signature 'item':
measurement(x)
## S4 method for signature 'item':
measurement(x) <- value
is.nominal(x)
is.ordinal(x)
is.interval(x)
is.ratio(x)

```

Arguments

`x` an object, usually of class "item".
`value` a character string; either "nominal", "ordinal", "interval", or "ratio".

Value

`measurement(x)` returns a character string. `is.nominal`, `is.ordinal`, `is.interval`, `is.ratio` return a logical value.

References

Stevens, Stanley S. 1946. "On the theory of scales of measurement." *Science* 103: 677-680.

See Also

[data.set, item](#)

Examples

```
answer <- sample(c(1,2,3,8,9),size=30,replace=TRUE)
labels(answer) <- c(Conservatives      = 1,
                   Labour              = 2,
                   "Liberal Democrats" = 3,
                   "Don't know"        = 8,
                   "Answer refused"    = 9
                   )
missing.values(answer) <- c(8,9)
as.data.frame(answer)[[1]]
measurement(answer) <- "interval"
as.data.frame(answer)[[1]]
```

Memisc

Introduction to the 'memisc' Package

Description

This package collects an assortment of tools that are intended to make work with R easier for the author of this package and are submitted to the public in the hope that they will be also be useful to others.

The tools in this package can be grouped into five major categories:

- Data preparation and management
- Data analysis
- Presentation of analysis results
- Simulation
- Programming

Data preparation and management

Survey Items

memisc provides facilities to work with what users from other packages like SPSS, SAS, or Stata know as 'variable labels', 'value labels' and 'user-defined missing values'. In the context of this package these aspects of the data are represented by the "description", "labels", and "missing.values" attributes of a data vector. These facilities are useful, for example, if you work with survey data that contain coded items like vote intention that may have the following structure:

Question: "If there was a parliamentary election next tuesday, which party would you vote for?"

```

1 Conservative Party
2 Labour Party
3 Liberal Democrat Party
4 Scottish Nation Party
5 Plaid Cymru
6 Green Party
7 British National Party
8 Other party
96 Not allowed to vote
97 Would not vote
98 Would vote, do not know yet for which party
99 No answer

```

A statistical package like SPSS allows to attach labels like ‘Conservative Party’, ‘Labour Party’, etc. to the codes 1,2,3, etc. and to mark the codes 96, 97, 98, 99 as ‘missing’ and thus to exclude these variables from statistical analyses. `memisc` provides similar facilities. Labels can be attached to codes by calls like `labels(x) <- something` and expanded by calls like `labels(x) <- labels(x) + something`, codes can be marked as ‘missing’ by calls like `missing.values(x) <- something` and `missing.values(x) <- missing.values(x) + something`.

`memisc` defines a class called "data.set", which is similar to the class "data.frame". The main difference is that it is especially geared toward containing survey item data. Transformations of and within "data.set" objects retain the information about value labels, missing values etc. Using `as.data.frame` sets the data up for R’s statistical functions, but doing this explicitly is seldom necessary. See `data.set`.

More Convenient Import of External Data

Survey data sets are often relative large and contain up to a few thousand variables. For specific analyses one needs however only a relatively small subset of these variables. Although modern computers have enough RAM to load such data sets completely into an R session, this is not very efficient having to drop most of the variables after loading. Also, loading such a large data set completely can be time-consuming, because R has to allocate space for each of the many variables. Loading just the subset of variables really needed for an analysis is more efficient and convenient - it tends to be much quicker. Thus this package provides facilities to load such subsets of variables, without the need to load a complete data set. Further, the loading of data from SPSS files is organized in such a way that all informations about variable labels, value labels, and user-defined missing values are retained. This is made possible by the definition of `importer` objects, for which a `subset` method exists. `importer` objects contain only the information about the variables in the external data set but not the data. The data itself is loaded into memory when the functions `subset` or `as.data.set` are used.

Recoding

`memisc` also contains facilities for recoding survey items. Simple recodings, for example collapsing answer categories, can be done using the function `recode`. More complex recodings, for example the construction of indices from multiple items, and complex case distinctions, can be done using the function `cases`. This function may also be useful for programming, in so far as it is a generalization of `ifelse`.

Code Books

There is a function `codebook` which produces a code book of an external data set or an internal "data.set" object. A codebook contains in a conveniently formatted way concise information about every variable in a data set, such as which value labels and missing values are defined and some univariate statistics.

An extended example of all these facilities is contained in the vignette "anes48", and in `demo(anes48)`

Data Analysis

Tables and Data Frames of Descriptive Statistics

`genTable` is a generalization of `xtabs`: Instead of counts, also descriptive statistics like means or variances can be reported conditional on levels of factors. Also conditional percentages of a factor can be obtained using this function.

In addition a `formula` method for the `aggregate` generic function is provided, see. It has the same syntax as `genTable`, but gives a data frame of descriptive statistics instead of a `table` object.

Per-Subset Analysis

`By` is a variant of the standard function `by`: Conditioning factors are specified by a formula and are obtained from the data frame the subsets of which are to be analysed. Therefore there is no need to `attach` the data frame or to use the dollar operator.

Graphical Model Comparison

`Termplot` is a variant and an extension of `termplot`: The plots are similar to those of `termplot` but uses `lattice` graphics. Also `Termplot` can be used on more than one model and allows to compare the fit of linear or non-linear effect specifications of different models.

Use `example(Termplot)` or `demo(Termplot)` for an example.

Presentation of Results of Statistical Analysis

Publication-Ready Tables of Coefficients

Journals of the Political and Social Sciences usually require that estimates of regression models are presented in the following form:

	Model 1	Model 2	Model 3
Coefficients			
(Intercept)	30.628*** (7.409)	6.360*** (1.252)	28.566*** (7.355)
pop15	-0.471** (0.147)		-0.461** (0.145)
pop75	-1.934 (1.041)		-1.691 (1.084)
dpi		0.001 (0.001)	-0.000 (0.001)
ddpi		0.529* (0.210)	0.410* (0.196)

```

Summaries
R-squared      0.262      0.162      0.338
adj. R-squared 0.230      0.126      0.280
N              50         50         50
=====

```

Such tables of coefficient estimates can be produced by `mtable`. To see some of the possibilities of this function, use `example(mtable)`.

LaTeX Representation of R Objects

Output produced by `mtable` can be transformed into LaTeX tables by an appropriate method of the generic function `toLatex` which is defined in the package `utils`. In addition, `memisc` defines `toLatex` methods for matrices and `fTable` objects. Note that results produced by `genTable` can be coerced into `fTable` objects. Also, a default method for the `toLatex` function is defined which coerces its argument to a matrix and applies the matrix method of `toLatex`.

Simulation

The `memisc` package defines a function `Simulate`, which can be used to conduct simulation experiments: For a given number of replications and given sets of parameters (which can be varied across experimental conditions) data are generated and can be summarized afterwards by other methods.

Use `example(Simulate)`, `demo(monte.carlo)`, `demo(lm.monte.carlo)`, `demo(random.walk)`, or `demo(schellings)` for examples.

Programming

Looping over Variables

Sometimes users want to construct loops that run over variables rather than values. For example, if one wants to set the missing values of a battery of items. For this purpose, the package contains the function `foreach`. To set 8 and 9 as missing values for the items `knowledge1`, `knowledge2`, `knowledge3`, one can use

```

foreach(x=c(knowledge1,knowledge2,knowledge3),
        missing.values(x) <- 8:9)

```

Changing Names of Objects and Labels of Factors

R already makes it possible to change the names of an object. Substituting the `names` or `dimnames` can be done with some programming tricks. This package defines the function `rename`, `dimrename`, `colrename`, and `rowrename` that implement these tricks in a convenient way, so that programmers (like the author of this package) need not reinvent the wheel in every instance of changing names of an object.

Dimension-Preserving Versions of `lapply` and `sapply`

If a function that is involved in a call to `sapply` returns a result an array or a matrix, the dimensional information gets lost. Also, if a list object to which `lapply` or `sapply` are applied have a dimension attribute, the result loses this information. The functions `Lapply` and `Sapply` defined in this package preserve such dimensional information.

Combining Vectors and Arrays by Names

The generic function `collect` collects several objects of the same mode into one object, using their names, `rownames`, `colnames` and/or `dimnames`. There are methods for atomic vectors, arrays (including matrices), and data frames. For example

```
a <- c(a=1,b=2)
b <- c(a=10,c=30)
collect(a,b)
```

leads to

```
      x  y
a    1 10
b    2 NA
c   NA 30
```

Reordering of Matrices and Arrays

The `memisc` package includes a `reorder` method for arrays and matrices. For example, the `matrix` method by default reorders the rows of a matrix according to the results of a function.

Other Facilities

`mkHelpDir` is a modification of `make.packages.html` that constructs HTML help pages permanently instead of temporarily.

mkHelpDir

Collect HTML Documentation

Description

`mkHelpDir` generates HTML documentation shown by `help.start` and puts it into a specified directory.

Usage

```
mkHelpDir(path="~/R")
```

Arguments

`path` a character string; specifies the directory where to put the documentation. By default the "R" subdirectory in the user's home directory.

mtable	<i>Comparative Table of Model Estimates</i>
--------	---

Description

mtable produces a table of estimates for several models.

Usage

```
mtable(..., coef.style=getOption("coef.style"),
        summary.stats=TRUE,
        factor.style=getOption("factor.style"),
        getSummary=function(obj, ...) UseMethod("getSummary"),
        float.style=getOption("float.style"),
        digits=min(3, getOption("digits")),
        drop=TRUE
)
## S3 method for class 'mtable':
relabel(x, ..., gsub = FALSE, fixed = !gsub, warn = FALSE)

## S3 method for class 'mtable':
format(x,
       coef.title="Coefficients",
       summary.title="Summaries",
       colsep="\t",
       rowsep="\n",
       trim=TRUE,
       trimleft=trim,
       trimright=trim,
       center.at=NULL,
       align.integers=c("dot", "right", "left"),
       topsep="",
       bottomsep="",
       sectionsep="",
       compact=TRUE,
       forLaTeX=FALSE,
       useDcolumn=TRUE,
       colspec=if (useDcolumn)
         paste("D{.}{", LaTeXdec, "}{", ddigits, "}", sep="")
         else "r",
       LaTeXdec=".",
       ddigits=getOption("digits"),
       useBooktabs=TRUE,
       toprule=if(useBooktabs) "\\toprule" else "\\hline\\hline",
       midrule=if(useBooktabs) "\\midrule" else "\\hline",
       cmidrule=if(useBooktabs) "\\cmidrule" else "\\cline",
       bottomrule=if(useBooktabs) "\\bottomrule" else "\\hline\\hline",
```

```

interaction.sep = if(forLaTeX) "  $\times$  " else " x ",
center.summaries=FALSE,
...
)

## S3 method for class 'mtable':
print(x, trim=FALSE, center.at=getOption("OutDec"),
      colsep=" ",
      topsep="=", bottomsep="=", sectionsep="-", ...)

## S3 method for class 'mtable':
toLatex(object, ...)

write.mtable(object, file="", ...)

```

Arguments

... as argument to `mtable`: several model objects, e.g. of class `lm`; as argument to `print.mtable`, `toLatex.mtable`, `write.mtable`: further arguments passed to `format.mtable`; as argument to `format.mtable`: further arguments passed to `format.default`; as argument to `relabel.mtable`: further arguments passed to `dimrename`.

`coef.style` a character string which specifies the style of coefficient values, whether standard errors, Wald/t-statistics, or significance levels are reported, etc. See [coef.style](#).

`summary.stats` if FALSE, no summary statistics are reported. If TRUE, all summary statistics produced by `getSummary` are reported. This argument may also contain a character vector with the names of the summary statistics to report

`factor.style` a character string that specifies the style in which factor contrasts are labeled. See [factor.style](#).

`getSummary` a function that computes model-related statistics that appear in the table. See [getSummary](#).

`float.style` default format for floating point numbers if no format is specified by `coef.style`; see `{float.style}`.

`digits` number of significant digits if not specified by the template returned from [getCoefTemplate](#) [getSummaryTemplate](#)

`drop` logical value; should redundant column headings be dropped if only one model is given as argument?

`x, object` an object of class `mtable`

`gsub, warn, fixed` logical values, see [relabel](#)

`coef.title` a character vector, the title for the reported coefficients.

`summary.title` a character vector, the title for the reported model summaries.

`colsep` a character string which separates the columns in the output.

<code>rowsep</code>	a character string which separates the rows in the output.
<code>trim</code>	should leading and trailing spaces be trimmed?
<code>trimleft</code>	should leading spaces be trimmed?
<code>trimright</code>	should trailing spaces be trimmed?
<code>center.at</code>	a character string on which resulting values are centered. Typically equal to ".". This is the default when <code>forLaTeX==TRUE</code> . If <code>NULL</code> , reported values are not centered.
<code>align.integers</code>	how to align integer values.
<code>topsep</code>	a character string that is recycled to a top rule.
<code>bottomsep</code>	a character string that is recycled to a bottom rule.
<code>sectionsep</code>	a character string that is recycled to separate coefficients from summary statistics.
<code>compact</code>	logical value; if <code>TRUE</code> , entries in the table are not aligned in any way, giving the table the most compact form.
<code>forLaTeX</code>	should LaTeX code be produced?
<code>useDcolumn</code>	should the <code>dcolumn</code> LaTeX package be used? If true, you will have to include <code>\usepackage{dcolumn}</code> into the preamble of your LaTeX document.
<code>colspec</code>	LaTeX table column format specifier(s).
<code>LaTeXdec</code>	the decimal point in the final LaTeX output.
<code>ddigits</code>	alignment specification or digits after the decimal point.
<code>useBooktabs</code>	should the <code>booktabs</code> LaTeX package be used? If true, you will have to include <code>\usepackage{booktabs}</code> into the preamble of your LaTeX document.
<code>toprule</code>	appearance of the top border of the LaTeX <code>tabular</code> environment.
<code>midrule</code>	how are coefficients and summary statistics separated in the LaTeX <code>tabular</code> environment.
<code>cmidrule</code>	appearance of rules under section headings.
<code>bottomrule</code>	appearance of the bottom border of the LaTeX <code>tabular</code> environment.
<code>interaction.sep</code>	a character string that separates factors that are involved in an interaction effect
<code>center.summaries</code>	logical value; if <code>TRUE</code> , summaries for each model are centered below the columns that correspond to the respective model coefficients.
<code>file</code>	a file where to write to; defaults to console output.

Details

`mtable` constructs a table of estimates for regression-type models. `format.mtable` formats suitable for use with output or conversion functions such as `print.mtable`, `toLatex.mtable`, or `write.mtable`.

Value

A call to `mtable` results in an object that inherits from `mtable` with the following components:

`coefficients` an array that contains the model coefficients.
`summaries` a matrix that contains the model summaries.

Examples

```
lm0 <- lm(sr ~ pop15 + pop75, data = LifeCycleSavings)
lm1 <- lm(sr ~ dpi + ddpi, data = LifeCycleSavings)
lm2 <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)

mtable123 <- mtable("Model 1"=lm0,"Model 2"=lm1,"Model 3"=lm2,
  summary.stats=c("sigma","R-squared","F","p","N"))

(mtable123 <- relabel(mtable123,
  "(Intercept)" = "Constant",
  pop15 = "Percentage of population under 15",
  pop75 = "Percentage of population over 75",
  dpi = "Real per-capita disposable income",
  ddpi = "Growth rate of real per-capita disp. income"
))

write.mtable(mtable123)
## Not run: file123 <- "mtable123.txt"
write.mtable(mtable123,file=file123)
file.show(file123)
## End(Not run)

## Not run: texfile123 <- "mtable123.tex"
cat(toLatex(mtable123),sep="\n",file=texfile123)
file.show(texfile123)
## End(Not run)

berkeley <- aggregate(Table(Admit,Freq)~.,data=UCBAdmissions)

berk0 <- glm(cbind(Admitted,Rejected)~1,data=berkeley,family="binomial")
berk1 <- glm(cbind(Admitted,Rejected)~Gender,data=berkeley,family="binomial")
berk2 <- glm(cbind(Admitted,Rejected)~Gender+Dept,data=berkeley,family="binomial")

mtable(berk0,summary.stats=c("Deviance","N"))
mtable(berk0,drop=FALSE,summary.stats=c("Deviance","N"))
mtable(berk1,summary.stats=c("Deviance","N"))
mtable(berk1,drop=FALSE,summary.stats=c("Deviance","N"))

mtable(berk0,berk1,berk2,summary.stats=c("Deviance","N"))

mtable(berk0,berk1,berk2,
  coef.style="stat",
  summary.stats=c("Deviance","AIC","N"))
mtable(berk0,berk1,berk2,
```

```

      coef.style="ci",
      summary.stats=c("Deviance", "AIC", "N"))
mtable(berk0,berk1,berk2,
      coef.style="ci.vertical",
      summary.stats=c("Deviance", "AIC", "N"))
mtable(berk0,berk1,berk2,
      coef.style="ci.horizontal",
      summary.stats=c("Deviance", "AIC", "N"))
mtable(berk0,berk1,berk2,
      coef.style="all",
      summary.stats=c("Deviance", "AIC", "N"))
mtable(berk0,berk1,berk2,
      coef.style="all.nostar",
      summary.stats=c("Deviance", "AIC", "N"))

mtable(by(berkeley,berkeley$Dept,function(x)glm(cbind(Admitted,Rejected)~Gender,
      data=x,family="binomial")),
      summary.stats=c("Likelihood-ratio", "N"))

mtable(By(~Gender,glm(cbind(Admitted,Rejected)~Dept,family="binomial"),
      data=berkeley),
      summary.stats=c("Likelihood-ratio", "N"))

berkfull <- glm(cbind(Admitted,Rejected)~Dept/Gender - 1,
      data=berkeley,family="binomial")
relabel(mtable(berkfull),Dept="Department",gsub=TRUE)

```

negative match *Negative Match*

Description

`%nin%` is a convenience operator: `x %nin% table` is equivalent to `!(x %in% table)`.

Usage

```
x %nin% table
```

Arguments

<code>x</code>	the values to be matched
<code>table</code>	a values to be match against

Value

A logical vector

Examples

```
x <- sample(1:6,12,replace=TRUE)
x %in% 1:3
x %nin% 1:3
```

panel.errbars

Panel Functions for Error Bars

Description

panel.errbars plots points and draws a line supplemented with error bars.

Usage

```
panel.errbars(x, y, ..., panel.xy=panel.xyplot,
              make.grid=c("horizontal", "vertical", "both", "none"), ewidth=0)
```

Arguments

x, y	numeric values, the points around which error bars are plotted. x is a numeric vector, y is a matrix with three columns, the values, the lower and the upper ends of the error bars.
...	graphical parameters passed to panel.xy
panel.xy	panel function to produce the plot to which error bars are added
make.grid	character string, determines the direction of grid lines added to the plot
ewidth	numerical value, width of the whiskers of the error bars

See Also

[panel.xyplot](#)

Examples

```
library(lattice)
library(grid)

applications <- aggregate(percent(Dept, weight=Freq, ci=TRUE)~Gender,
                          data=UCBAdmissions)
admissions <- aggregate(
  percent(Admit=="Admitted", weight=Freq, ci=TRUE)~Dept+Gender,
  data=UCBAdmissions)
xyplot(cbind(Percentage, lower, upper)~Gender|Dept, data=admissions,
       panel=panel.errbars,
       ewidth=.2, pch=19,
       ylab="Percentage applicants admitted by Department")
xyplot(cbind(Percentage, lower, upper)~Gender|Dept, data=applications,
```

```

panel=panel.errbars,
ewidth=.2,pch=19,
ylab="Percentage applications to the Department")

```

percent

Table of Percentages with Percentage Base

Description

`percent` returns a table of percentages along with the percentage base. It will be useful in conjunction with [aggregate.formula](#).

Usage

```

percent(x,...)
## Default S3 method:
percent(x,weights=NULL,total=!(se || ci),
        se=FALSE,ci=FALSE,ci.level=.95,
        total.name="N",perc.label="Percentage",...)
## S3 method for class 'logical':
percent(x,weights=NULL,total=!(se || ci),
        se=FALSE,ci=FALSE,ci.level=.95,
        total.name="N",perc.label="Percentage",...)

```

Arguments

<code>x</code>	a numeric vector or factor.
<code>weights</code>	a optional numeric vector of weights of the same length as <code>x</code> .
<code>total</code>	logical; should the total sum of counts from which the percentages are computed be included into the output?
<code>se</code>	logical; should standard errors of the percentages be included?
<code>ci</code>	logical; should confidence intervals of the percentages be included?
<code>ci.level</code>	numeric; nominal coverage of confidence intervals
<code>total.name</code>	character; name given for the total sum of counts
<code>perc.label</code>	character; label given for the percentages if the table has more than one dimensions, e.g. if <code>se</code> or <code>ci</code> is TRUE.
<code>...</code>	for <code>percent.mresp</code> : one or several 1-0 vectors or matrices otherwise, further arguments, currently ignored.

Value

A table of percentages.

Examples

```

x <- rnorm(100)
y <- rnorm(100)
z <- rnorm(100)
f <- sample(1:3,100,replace=TRUE)
f <- factor(f,labels=c("a","b","c"))

percent(x>0)
percent(f)

genTable(
  cbind(percent(x>0),
        percent(y>0),
        percent(z>0)) ~ f
)

gt <- genTable(
  cbind(percent(x>0,ci=TRUE),
        percent(y>0,ci=TRUE),
        percent(z>0,ci=TRUE)) ~ f
)

print(gt)
ftable(gt,row.vars=2,col.vars=c(3,1))

ex.data <- expand.grid(mean=c(0,25,50),sd=c(1,10,100))[rep(1:9,rep(250,9)),]
ex.data <- within(ex.data,x <- rnorm(n=nrow(ex.data),mean=ex.data$mean,sd=ex.data$sd))
ex.data <- within(ex.data,x.grp <- cases( x < 0,
                                       x >= 0 & x < 50,
                                       x >= 50 & x < 100,
                                       x >= 100
                                       ))
genTable(percent(x.grp)~mean+sd,data=ex.data)

aggregate(percent(Admit,weight=Freq)~Gender+Dept,data=UCBAdmissions)

```

prediction.frame *Produce a Data Frame of Predictions and Independent Variables*

Description

prediction.frame produces a data frame that contains the independent variables of a model together with model generated predictions.

Usage

```
## Default S3 method:
```

```
prediction.frame(object, newdata=NULL, ...,
                residuals=c("none", "deviance", "pearson", "working",
                           "standardized", "studentized"))
```

Arguments

object	a model object from which predictions are generated.
newdata	an optional data frame for out-of-sample predictions.
...	further arguments passed to <code>predict</code> .
residuals	a character vector that specifies residuals that are added to the resulting data frame.

Value

A data frame.

Examples

```
lm1 <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
lm2 <- lm(sr ~ pop15 + pop75 + dpi + ddpi + pop15*dpi, data = LifeCycleSavings)

str(prediction.frame(lm1))
str(prediction.frame(lm1, se.fit=TRUE))
str(prediction.frame(lm1, interval="prediction"))
str(prediction.frame(lm1, type="terms"))
str(prediction.frame(lm1,
                    se.fit=TRUE,
                    type="terms"))

str(prediction.frame(lm1,
                    se.fit=TRUE,
                    type="terms",
                    residuals="working"))

str(prediction.frame(lm1,
                    se.fit=TRUE,
                    type="terms",
                    residuals="deviance"))

str(prediction.frame(lm2,
                    se.fit=TRUE,
                    type="terms",
                    residuals="standardized"))

berkeley <- aggregate(Table(Admit, Freq) ~ ., data=UCBAdmissions)
berk2 <- glm(cbind(Admitted, Rejected) ~ Gender+Dept, data=berkeley, family="binomial")

str(prediction.frame(berk2,
                    se.fit=TRUE,
                    type="terms",
                    residuals="studentized"))
```

 query

Query an Object for Information

Description

The function `query` can be used to search an object for a keyword.

The `data.set` and `importer` methods perform such a search through the annotations and value labels of the items in the data set.

Usage

```
query(x,pattern,...)
## S4 method for signature 'data.set':
query(x,pattern,...)
## S4 method for signature 'importer':
query(x,pattern,...)

## S4 method for signature 'item':
query(x,pattern,...)
# (Called by the methods above.)
```

Arguments

<code>x</code>	an object
<code>pattern</code>	a character string that gives the pattern to be searched for
<code>...</code>	optional arguments such as
	<code>fuzzy</code> logical, TRUE by default; use fuzzy search via <code>agrep</code> or regexp search via <code>grep</code>
	<code>extended</code> logical, defaults to FALSE; passed to <code>grep</code>
	<code>perl</code> logical, defaults to TRUE; passed to <code>grep</code>
	<code>fixed</code> logical, defaults to TRUE; passed to <code>grep</code>
	<code>ignore.case</code> logical, defaults to TRUE; passed to <code>grep</code> or <code>agrep</code>
	<code>insertions</code> numerical value, defaults to 0.999999999; passed to <code>agrep</code>
	<code>deletions</code> numerical value, defaults to 0; passed to <code>agrep</code>
	<code>substitutions</code> numerical value, defaults to 0; passed to <code>agrep</code>

Value

If both the annotation and the value labels of an item match the pattern the `query` method for 'item' objects returns a list containing the annotation and the value labels, otherwise if only the annotation or the value labels match the pattern, either the annotation or the value labels are returned, otherwise if neither matches the pattern, `query` returns NULL.

The methods of `query` for 'data.set' and 'importer' objects return a list of all non-NULL query results of all items contained by these objects, or NULL.

Examples

```
nes1948.por <- UnZip("anes/NES1948.ZIP", "NES1948.POR",
                   package="memisc")
nes1948 <- spss.portable.file(nes1948.por)
query(nes1948, "TRUMAN")
```

recode

*Recode Items, Factors and Numeric Vectors***Description**

recode substitutes old values of a factor or a numeric vector by new ones, just like the recoding facilities in some commercial statistical packages.

Usage

```
## S4 method for signature 'vector':
recode(x, ..., otherwise="NA")
## S4 method for signature 'factor':
recode(x, ..., otherwise="NA")
## S4 method for signature 'item':
recode(x, ..., otherwise="NA")
```

Arguments

x	An object
...	<p>One or more assignment expressions, each of the form <code>new.value <- old.values</code>. <code>new.value</code> should be a scalar numeric value or character string. If one of the <code>new.values</code> is a character string, the return value of <code>recode</code> will be a factor and each <code>new.value</code> will be coerced to a character string that labels a level of the factor.</p> <p>Each <code>old.value</code> in an assignment expression may be a (numeric or character) vector. If <code>x</code> is numeric such an assignment expression may have the form <code>new.value <- range(lower, upper)</code>. In that case, values between <code>lower</code> and <code>upper</code> are exchanged by <code>new.value</code>. If one of the arguments to <code>range</code> is <code>min</code>, it is substituted by the minimum of <code>x</code>. If one of the arguments to <code>range</code> is <code>max</code>, it is substituted by the maximum of <code>x</code>.</p> <p>In case of the method for labelled vectors, the <i>tags</i> of arguments of the form <code>tag = new.value <- old.values</code> will define the labels of the new codes.</p> <p>If the <code>old.values</code> of different assignment expressions overlap, an error will be raised because the recoding is ambiguous.</p>
otherwise	a character string or some other value that the result may obtain. If equal to <code>NA</code> or <code>"NA"</code> , original codes not given an explicit new code are recoded into <code>NA</code> . If equal to <code>"copy"</code> , original codes not given an explicit new code are copied.

Details

`recode` relies on the lazy evaluation mechanism of *R*: Arguments are not evaluated until required by the function they are given to. `recode` does not cause arguments that appear in `...` to be evaluated. Instead, `recode` parses the `...` arguments. Therefore, although expressions like `1 <- 1:4` would cause an error action, if evaluated at any place elsewhere in *R*, they will not cause an error action, if given to `recode` as an argument. However, a call of the form `recode(x, 1=1:4)`, would be a syntax error.

If John Fox' package "car" is installed, `recode` will also be callable with the syntax of the `recode` function of that package.

Value

A numerical vector, factor or an `item` object.

See Also

[recode](#) of package `car`.

Examples

```
x <- as.item(sample(1:6,20,replace=TRUE),
             labels=c( a=1,
                       b=2,
                       c=3,
                       d=4,
                       e=5,
                       f=6))

print(x)

# A recoded version of x is returned
# containing the values 1, 2, 3, which are
# labelled as "A", "B", "C".
recode(x,
       A = 1 <- range(min,2),
       B = 2 <- 3:4,
       C = 3 <- range(5,max), # this last comma is ignored
       )

# This causes an error action: the sets
# of original values overlap.
try(recode(x,
          A = 1 <- range(min,2),
          B = 2 <- 2:4,
          C = 3 <- range(5,max)
          ))

recode(x,
       A = 1 <- range(min,2),
       B = 2 <- 3:4,
       C = 3 <- range(5,6),
       D = 4 <- 7
```

```
)

# This results in an all-missing vector:
recode(x,
  D = 4 <- 7,
  E = 5 <- 8
)

f <- as.factor(x)
x <- as.integer(x)

recode(x,
  1 <- range(min,2),
  2 <- 3:4,
  3 <- range(5,max)
)

# This causes another error action:
# the third argument is an invalid
# expression for a recoding.
try(recode(x,
  1 <- range(min,2),
  3:4,
  3 <- range(5,max)
))

# The new values are character strings,
# therefore a factor is returned.
recode(x,
  "a" <- range(min,2),
  "b" <- 3:4,
  "c" <- range(5,6)
)

recode(x,
  1 <- 1:3,
  2 <- 4:6
)

recode(x,
  4 <- 7,
  5 <- 8,
  otherwise = "copy"
)

recode(f,
  "A" <- c("a","b"),
  "B" <- c("c","d"),
  otherwise="copy"
)

recode(f,
  "A" <- c("a","b"),
```

```

    "B" <- c("c","d"),
    otherwise="C"
  )

  recode(f,
    "A" <- c("a","b"),
    "B" <- c("c","d")
  )

```

relabel

Change labels of factors or labelled objects

Description

Function `relabel` changes the labels of a factor or any object that has a `names`, `labels`, `value.labels`, or `variable.labels` attribute. Function `relabel4` is an (internal) generic which is called by `relabel` to handle S4 objects.

Usage

```

## Default S3 method:
relabel(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
## S3 method for class 'factor':
relabel(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)

## S4 method for signature 'item':
relabel4(x, ...)
# This is an internal method, see details.
# Use relabel(x, ...) for 'item' objects

```

Arguments

<code>x</code>	An object with a <code>names</code> , <code>labels</code> , <code>value.labels</code> , or <code>variable.labels</code> attribute
<code>...</code>	A sequence of named arguments, all of type character
<code>gsub</code>	a logical value; if <code>TRUE</code> , <code>gsub</code> is used to change the labels of the object. That is, instead of substituting whole labels, substrings of the labels of the object can be changed.
<code>fixed</code>	a logical value, passed to <code>gsub</code> . If <code>TRUE</code> , substitutions are by fixed strings and not by regular expressions.
<code>warn</code>	a logical value; if <code>TRUE</code> , a warning is issued if a change of labels was unsuccessful.

Details

This function changes the names or labels of `x` according to the remaining arguments. If `gsub` is `FALSE`, argument tags are the *old* labels, the values are the new labels. If `gsub` is `TRUE`, arguments are substrings of the labels that are substituted by the argument values.

Function `relabel` is S3 generic. If its first argument is an S4 object, it calls the (internal) `relabel4` generic function.

Value

The object `x` with new labels defined by the `...` arguments.

Examples

```
f <- as.factor(rep(letters[1:4],5))
levels(f)
F <- relabel(f,
  a="A",
  b="B",
  c="C",
  d="D"
)
levels(F)

f <- as.item(f)
labels(f)
F <- relabel(f,
  a="A",
  b="B",
  c="C",
  d="D"
)
labels(F)
```

rename

Change Names of a Named Object

Description

`rename` changes the names of a named object.

Usage

```
rename(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
```

Arguments

<code>x</code>	Any named object
<code>...</code>	A sequence of named arguments, all of type character
<code>gsub</code>	a logical value; if TRUE, <code>gsub</code> is used to change the row and column labels of the resulting table. That is, instead of substituting whole names, substrings of the names of the object can be changed.
<code>fixed</code>	a logical value, passed to <code>gsub</code> . If TRUE, substitutions are by fixed strings and not by regular expressions.
<code>warn</code>	a logical value; should a warning be issued if those names to be changed are not found?

Details

This function changes the names of `x` according to the remaining arguments. If `gsub` is FALSE, argument tags are the *old* names, the values are the new names. If `gsub` is TRUE, arguments are substrings of the names that are substituted by the argument values.

Value

The object `x` with new names defined by the `...` arguments.

Examples

```
x <- c(a=1, b=2)
rename(x, a="A", b="B")

str(rename(iris,
           Sepal.Length="Sepal_Length",
           Sepal.Width="Sepal_Width",
           Petal.Length="Petal_Length",
           Petal.Width="Petal_Width"
         ))
str(rename(iris,
           .="_"
           , gsub=TRUE))
```

reorder.array

Reorder an Array or Matrix

Description

`reorder.array` reorders an array along a specified dimension according to given names, indices or results of a function applied.

Usage

```
## S3 method for class 'array':
reorder(x, dim=1, names=NULL, indices=NULL, FUN=mean, ...)
## S3 method for class 'matrix':
reorder(x, dim=1, names=NULL, indices=NULL, FUN=mean, ...)
```

Arguments

<code>x</code>	An array
<code>dim</code>	An integer specifying the dimension along which <code>x</code> should be ordered.
<code>names</code>	A character vector
<code>indices</code>	A numeric vector
<code>FUN</code>	A function that can be used in <code>apply(x, dim, FUN)</code>
<code>...</code>	further arguments, ignored.

Details

Typical usages are

```
reorder(x, dim, names)
reorder(x, dim, indices)
reorder(x, dim, FUN)
```

The result of `reorder(x, dim, names)` is `x` reordered such that `dimnames(x)[[dim]]` is equal to the concatenation of those elements of `names` that are in `dimnames(x)[[dim]]` and the remaining elements of `dimnames(x)[[dim]]`.

The result of `reorder(x, dim, indices)` is `x` reordered along `dim` according to `indices`.

The result of `reorder(x, dim, FUN)` is `x` reordered along `dim` according to `order(apply(x, dim, FUN))`.

Value

The reordered object `x`.

See Also

The default method of `reorder` in package `stats`.

Examples

```
(M <- matrix(rnorm(n=25), 5, 5, dimnames=list(LETTERS[1:5], letters[1:5])))
reorder(M, dim=1, names=c("E", "A"))
reorder(M, dim=2, indices=3:1)
reorder(M, dim=1)
reorder(M, dim=2)
```

retain	<i>Retain Objects in an Environment</i>
--------	---

Description

`retain` removes all objects from the environment except those mentioned as argument.

Usage

```
retain(..., list = character(0), envir = parent.frame(), force=FALSE)
```

Arguments

<code>...</code>	names of objects to be retained, as names (unquoted) or character strings(quoted).
<code>list</code>	a character vector naming the objects to be retained.
<code>envir</code>	the environment from which the objects are removed that are not to be retained.
<code>force</code>	logical value. As a measure of caution, this function removes objects only from local environments, unless <code>force</code> equals TRUE. In that case, <code>retain</code> can also be used to clear the global environment, the user's workspace.

Examples

```
local({
  foreach(x=c(a,b,c,d,e,f,g,h), x<-1)
  cat("Objects before call to 'retain':\n")
  print(ls())
  retain(a)
  cat("Objects after call to 'retain':\n")
  print(ls())
})
x <- 1
y <- 2
retain(x)
```

sample-methods	<i>Take a Sample from a Data Frame-like Object</i>
----------------	--

Description

The methods below are convenience short-cuts to take samples from data frames and data sets. They result in a data frame or data set, respectively, the rows of which are a sample of the complete data frame/data set.

Usage

```
## S3 method for class 'data.frame':
sample(x, size, replace = FALSE, prob = NULL, ...)
## S3 method for class 'data.set':
sample(x, size, replace = FALSE, prob = NULL, ...)
```

Arguments

<code>x</code>	a data frame or data set.
<code>size</code>	an (optional) numerical value, the sample size, defaults to the total number of rows of <code>x</code> .
<code>replace</code>	a logical value, determines whether sampling takes place with or without replacement.
<code>prob</code>	a vector of sampling probabilities or <code>NULL</code> .
<code>...</code>	further arguments, ignored.

Value

A data frame or data set.

Sapply

A Dimension Preserving Variant of "sapply" and "lapply"

Description

`Sapply` is equivalent to `sapply`, except that it preserves the dimension and dimension names of the argument `X`. It also preserves the dimension of results of the function `FUN`. It is intended for application to results e.g. of a call to `by`. `Lapply` is an analog to `lapply` insofar as it does not try to simplify the resulting `list` of results of `FUN`.

Usage

```
Sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
Lapply(X, FUN, ...)
```

Arguments

<code>X</code>	a vector or list appropriate to a call to <code>sapply</code> .
<code>FUN</code>	a function.
<code>...</code>	optional arguments to <code>FUN</code> .
<code>simplify</code>	a logical value; should the result be simplified to a vector or matrix if possible?
<code>USE.NAMES</code>	logical; if <code>TRUE</code> and if <code>X</code> is character, use <code>X</code> as names for the result unless it had names already.

Value

If FUN returns a scalar, then the result has the same dimension as X, otherwise the dimension of the result is enhanced relative to X.

Examples

```
berkeley <- aggregate(Table(Admit, Freq) ~ ., data=UCBAdmissions)
berktest1 <- By(~Dept+Gender,
               glm(cbind(Admitted, Rejected) ~ 1, family="binomial"),
               data=berkeley)
berktest2 <- By(~Dept,
               glm(cbind(Admitted, Rejected) ~ Gender, family="binomial"),
               data=berkeley)

sapply(berktest1, coef)
Sapply(berktest1, coef)

sapply(berktest1, function(x) drop(coef(summary(x))))
Sapply(berktest1, function(x) drop(coef(summary(x))))

sapply(berktest2, coef)
Sapply(berktest2, coef)
sapply(berktest2, function(x) coef(summary(x)))
Sapply(berktest2, function(x) coef(summary(x)))
```

 Simulate

Simulation Experiments

Description

`Simulate` is a function to simplify simulation studies. It can be used to conduct Monte Carlo studies of statistical estimators, discrete event, and agent based simulations.

Usage

```
Simulate(step,
         conditions = NULL,
         start = NULL,
         cleanup = NULL,
         ...,
         nsim = 1,
         seed = NULL,
         trace=0,
         keep.data=TRUE,
         keep.states=FALSE,
         keep.seed = !is.null(seed),
         restore.seed = !is.null(seed),
         bucket = default_bucket)
```

```
)

# signal an interrupt condition
interrupt(msg=NULL)
```

Arguments

<code>step</code>	an expression that produces simulation results for each replication; can be a function call, or a braced expression that “returns” a value like a function body.
<code>conditions</code>	an optional data frame or object coerceable into a data frame. Each row of this data frame defines an experimental condition.
<code>start</code>	either <code>NULL</code> or an expression that computes starting values for <code>step</code> .
<code>cleanup</code>	either <code>NULL</code> or an expression does some cleaning up after the execution of all steps.
<code>...</code>	other substitutions for <code>step</code> , held fixed in the simulation experiment
<code>nsim</code>	an integer value; the number of replication in each experimental setting. If <code>nsim</code> is infinite or <code>NA</code> , <code>step</code> is replicated (in each setting) until either a user interrupt is signalled (<code>CTRL-C</code> is pressed) or <code>interrupt</code> is called.
<code>seed</code>	either <code>NULL</code> or an integer value suitable for <code>set.seed</code> . Note that the random state before the call to <code>Simulate</code> is restored.
<code>trace</code>	an integer value determining the amount of information output during the simulation process. If <code>trace</code> equals zero nothing is reported during the simulation run. Otherwise, the replication number is output for each multiple of <code>trace</code> .
<code>keep.data</code>	logical value; if <code>TRUE</code> , return values of the expression in <code>step</code> are collected into a data frame.
<code>keep.states</code>	logical value; if <code>TRUE</code> , a list of all variables defined in <code>step</code> (after execution of <code>cleanup</code> if present) is returned.
<code>keep.seed</code>	logical value; if <code>TRUE</code> , the state of the random number generator is saved in an attribute "seed" of the return value of <code>Simulate</code> .
<code>restore.seed</code>	logical value; if <code>TRUE</code> , the state of the random number generator is restored after conducting the simulations.
<code>bucket</code>	a function that returns a <code>bucket</code> object, in which simulation results are collected.
<code>msg</code>	a character string, the message shown if an interrupt condition is signalled.

Details

`Simulate` calls or evaluates its first argument, `step`, or, if a `conditions` argument is given, `nsim` times for each row of the `conditions` data frame.

Before repeatedly evaluating `step`, the expression `start`, if present, is evaluated, which may be used to create starting values for a simulation or to setup up the scenery for an agent-based simulation. *After* repeatedly evaluating `step`, the expression `cleanup`, if present, is evaluated.

If `restore.seed` is given, the state of the random generator is saved before conducting the simulation and restored afterwards. Therefore `step`, `start`, or `cleanup` may call `set.seed` without affecting the generation of random numbers after a call to `Simulate`.

`interrupt` raises an interrupt condition, which acts like a user interrupt.

Note that if an interrupt condition is signalled during a (replicated) evaluation of `step` the results of previous replications are still saved and `Simulate` jumps to the next condition of the simulation experiment (if there is any). That is, if a simulation is interrupted by the user because it takes too long, the results so far produced by the simulation are not lost.

On the other hand, `interrupt` can be used to determine at run-time how often `step` is evaluated.

Value

A data frame that contains experimental conditions and simulation results.

Examples

```
Normal.example <- function(mean=0, sd=1, n=10) {
  x <- rnorm(n=n, mean=mean, sd=sd)
  c(
    Mean=mean(x),
    Median=median(x),
    Var=var(x)
  )
}

Normal.simres <- Simulate(
  Normal.example(mean, sd, n),
  expand.grid(
    mean=0,
    sd=c(1, 10),
    n=c(10, 100)
  ),
  nsim=200,
  trace=50)

genTable(sd(Median)~sd+n, data=Normal.simres)

expr.simres <- Simulate(
  median(rnorm(n, mean, sd)),
  expand.grid(
    n=c(10, 100),
    mean=c(0, 1),
    sd=c(1, 10)
  ),
  nsim=200,
  trace=50)

genTable(c(mean(result), sd(result))~sd+n+mean, data=expr.simres)

## Not run:
## This takes a little bit longer
lm.example <- function(a=0, b=1, n=101, xrange=c(-1, 1), serr=1) {
  x <- seq(from=xrange[1], to=xrange[2], length=n)
```

```

y <- a + b*x + rnorm(n,sd=serr)
lm.res <- lm(y~x)
coef <- lm.res$coef
names(coef) <- c("a","b")
coef
}

lm.simres <- Simulate(
  lm.example(n=n,serr=serr),
  expand.grid(
    serr=c(0.1,1,10),
    n=c(11,101,501)
  ),
  nsim=200,
  trace=50
)
genTable(c(sd(a),sd(b))~serr+n,data=lm.simres)

## End(Not run)

```

Description

The methods below return a sorted version of the data frame or data set, given as first argument.

Usage

```

## S3 method for class 'data.frame':
sort(x,decreasing=FALSE,by=NULL,na.last=NA,...)
## S3 method for class 'data.set':
sort(x,decreasing=FALSE,by=NULL,na.last=NA,...)

```

Arguments

x	a data frame or data set.
decreasing	a logical value, should sorting be in increasing or decreasing order?
by	a character name of variable names, by which to sort; a formula giving the variables, by which to sort; NULL, in which case, the data frame / data set is sorted by all of its variables.
na.last	for controlling the treatment of 'NA's. If 'TRUE', missing values in the data are put last; if 'FALSE', they are put first; if 'NA', they are removed
...	other arguments, currently ignored.

Value

A sorted copy of `x`.

Examples

```
DF <- data.frame(
  a = sample(1:2, size=20, replace=TRUE),
  b = sample(1:4, size=20, replace=TRUE))
sort(DF)
sort(DF, by=~a+b)
sort(DF, by=~b+a)
sort(DF, by=c("b", "a"))
sort(DF, by=c("a", "b"))
```

styles

Formatting Styles for Coefficients and Factor Contrasts

Description

Methods for setting and getting templates for formatting model coefficients and summaries for use in `mtable`.

Usage

```
setCoefTemplate(...)
getCoefTemplate(style)
getSummaryTemplate(x)
setSummaryTemplate(...)
```

Arguments

<code>...</code>	several tagged arguments; in case of <code>setCoefTemplate</code> the tags specify the <code>coef.styles</code> , in case of <code>setSummaryTemplate</code> they specify model classes. The associated values are templates .
<code>style</code>	a character string with the name of a coefficient style, if left empty, all coefficient templates are returned.
<code>x</code>	a model or a name of a model class, for example <code>"lm"</code> or <code>"glm"</code> ; if left empty, all summary templates are returned.

Details

The style in which model coefficients are formatted by `mtable` is by default selected from the `coef.style` setting of `options`, "factory-fresh" setting being `options(coef.style="default")`.

The appearance of factor levels in an `mtable` can be influenced by the `factor.style` setting of `options`. The "factory-fresh" setting is `options(factor.style="($f): ($l)")`, where `($f)` stands for the factor name and `($l)` stands for the factor level. In case of treatment

contrasts, the baseline level will also appear in an `mtable` separated from the current factor level by the `baselevel.sep` setting of `options`. The "factory-fresh" setting is `options(baselevel.sep="-")`,

Users may specify additional coefficient styles by a call to `setCoefTemplate`. In order to adapt `mtable` to other model classes, users need to set a template for model summaries via a call to `setSummaryTemplate`.

 Substitute

Substitutions in Language Objects

Description

`Substitute` differs from `substitute` in so far as its first argument can be a variable that contains an object of mode "language". In that case, substitutions take place inside this object.

Usage

```
Substitute(lang, with)
```

Arguments

<code>lang</code>	any object, unevaluated expression, or unevaluated language construct, such as a sequence of calls inside braces
<code>with</code>	a named list, environment, data frame or data set.

Details

The function body is just `do.call("substitute", list(lang, with))`.

Value

An object of storage mode "language" or "symbol".

Examples

```
lang <- quote(sin(x)+z)
substitute(lang, list(x=1, z=2))
Substitute(lang, list(x=1, z=2))
```

Table

*One-Dimensional Table of Frequencies and/or Percentages***Description**

Table is a generic function that produces a table of counts or weighted counts and/or the corresponding percentages of an atomic vector, factor or "item.vector" object. This function is intended for use with `aggregate.formula` or `genTable`. The "item.vector" method is the workhorse of `codebook`.

Usage

```
## S4 method for signature 'atomic':
Table(x, weights=NULL, counts=TRUE, percentage=FALSE, ...)
## S4 method for signature 'factor':
Table(x, weights=NULL, counts=TRUE, percentage=FALSE, ...)
## S4 method for signature 'item.vector':
Table(x, weights=NULL, counts=TRUE, percentage=(style=="codebook"),
      style=c("table", "codebook", "nolabels"),
      include.missings=(style=="codebook"),
      missing.marker=if(style=="codebook") "M" else "*", ...)
```

Arguments

<code>x</code>	an atomic vector, factor or "item.vector" object
<code>counts</code>	logical value, should the table contain counts?
<code>percentage</code>	logical value, should the table contain percentages? Either the <code>counts</code> or the <code>percentage</code> arguments or both should be TRUE.
<code>style</code>	character string, the style of the names or rownames of the table.
<code>weights</code>	a numeric vector of weights of the same length as <code>x</code> .
<code>include.missings</code>	a logical value; should missing values included into the table?
<code>missing.marker</code>	a character string, used to mark missing values in the table (row)names.
<code>...</code>	other, currently ignored arguments.

Value

The atomic vector and factor methods return either a vector of counts or vector of percentages or a matrix of counts and percentages. The same applies to the "item.vector" vector method unless `include.missing=TRUE` and `percentage=TRUE`, in which case total percentages and percentages of valid values are given.

Examples

```
with(as.data.frame(UCBAdmissions), Table(Admit, Freq))
aggregate(Table(Admit, Freq) ~ ., data=UCBAdmissions)

A <- sample(c(1:5, 9), size=100, replace=TRUE)
labels(A) <- c(a=1, b=2, c=3, d=4, e=5, dk=9)
missing.values(A) <- 9
Table(A, percentage=TRUE)
```

Termplot

Produce a Term Plot Lattice

Description

Termplot produces a lattice plot of termplots. Terms are not plotted individually, rather the terms in which a variable appears are summed and plotted.

Usage

```
## Default S3 method:
Termplot(object,
  ...,
  variables=NULL,
  col.term = 2,
  lty.term = 1,
  lwd.term = 1.5,
  se = TRUE,
  col.se = "orange",
  lty.se = 2,
  lwd.se = 1,
  col.res = "gray",
  residuals = c("deviance", "none", "pearson", "working"),
  cex = 1,
  pch = 1,
  jitter.resid=FALSE,
  smooth = TRUE,
  col.smth = "darkred",
  lty.smth = 2,
  lwd.smth = 1,
  span.smth = 2/3,
  aspect="fill",
  xlab=NULL,
  ylab=NULL,
  main=paste(deparse(object$call), collapse="\n"),
  models=c("rows", "columns"),
  xrot = 0,
  layout=NULL)
```

Arguments

<code>object</code>	an model fit object, or a list of model objects
<code>...</code>	further model objects.
<code>variables</code>	a character vector giving the names of independent variables; note that the combined effect of all terms containing the respective values will be plotted; if empty, the effect of each independent variable will be plotted. Currently, higher-order terms will be ignored.
<code>col.term, lty.term, lwd.term</code>	parameters for the graphical representation of the terms with the same meaning as in termplot .
<code>se</code>	a logical value; should standard error curves added to the plot?
<code>col.se, lty.se, lwd.se</code>	graphical parameters for the depiction of the standard error curves, see termplot .
<code>residuals</code>	a character string, to select the type of residuals added to the plot.
<code>col.res, cex, pch</code>	graphical parameters for the depiction the residuals as in termplot .
<code>jitter.resid</code>	a logical vector of at most length 2. Determines whether residuals should be jittered along the x-axis (first element) and along the y-axis. If this argument has length 1, its setting applies to both axes.
<code>smooth</code>	a logical value; should a LOWESS smooth added to the plot?
<code>span.smth</code>	a numerical value, the span of the smoother.
<code>col.smth, lty.smth, lwd.smth</code>	graphical parameters for the smoother curve.
<code>aspect</code>	aspect ratio of the plot(s), see xyplot .
<code>xlab</code>	label of the x axis, see xyplot .
<code>ylab</code>	label of the y axis, see xyplot .
<code>main</code>	main heading, see xyplot .
<code>models</code>	character; should models arranged in rows or columns?
<code>xrot</code>	angle by which labels along the x-axis are rotated.
<code>layout</code>	layout specification, see xyplot .

Value

A trellis object.

Examples

```
library(lattice)
library(grid)

lm0 <- lm(sr ~ pop15 + pop75, data = LifeCycleSavings)
lm1 <- lm(sr ~ dpi + ddpi, data = LifeCycleSavings)
```

```

lm2 <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)

berkeley <- aggregate(Table(Admit,Freq)~.,data=UCBAdmissions)
berk0 <- glm(cbind(Admitted,Rejected)~1,data=berkeley,family="binomial")
berk1 <- glm(cbind(Admitted,Rejected)~Gender,data=berkeley,family="binomial")
berk2 <- glm(cbind(Admitted,Rejected)~Gender+Dept,data=berkeley,family="binomial")

Termplot(lm2)
Termplot(berk2)
Termplot(lm0,lm1,lm2)
Termplot(berk0,berk1,berk2)

Termplot(By(~Gender,glm(cbind(Admitted,Rejected)~Dept,family="binomial"),
            data=berkeley))
Termplot(By(~Dept,glm(cbind(Admitted,Rejected)~Gender,family="binomial"),
            data=berkeley))

require(splines)
xyz <- data.frame(
  x = 1:100,
  z = factor(rep(LETTERS[1:4],25))
)
xyz <- within(xyz,
  y <- rnorm(100,sin(x/10)+x/50+as.numeric(z))
)
yxz.lin <- glm(y ~ x + z, data=xyz)
yxz.bs <- glm(y ~ bs(x,6) + z, data=xyz)
yxz.ns <- glm(y ~ ns(x,6) + z, data=xyz)
yxz.poly <- glm(y ~ poly(x,6) + z, data=xyz)
yxz.sincos <- glm(y ~ sin(x/10) + cos(x/10) + x + z, data=xyz)

# Terms containing
# the same variable are not plotted
# individually but their combined effect is plotted
#
Termplot(yxz.lin,yxz.bs,yxz.ns,yxz.poly,yxz.sincos,models="columns",
  span.smth=1/3)

Termplot(yxz.lin,yxz.bs,yxz.ns,yxz.poly,yxz.sincos,variables="x",
  span.smth=1/3)

```

to.data.frame

Convert an Array into a Data Frame

Description

to.data.frame converts an array into a data frame, in such a way that a chosen dimensional

extent forms variables in the data frame. The elements of the array must be either atomic, data frames with matching variables, or coercable into such data frames.

Usage

```
to.data.frame(X, as.vars=1, name="Freq")
```

Arguments

X	an array.
as.vars	a numeric value; indicates the dimensional extend which defines the variables. Takes effect only if X is an atomic array. If as.vars equals zero, a new variable is created that contains the values of the array, that is, to.data.frame acts on the array X like as.data.frame(as.table(X))
name	a character string; the name of the variable created if X is an atomic array and as.vars equals zero.

Value

A data frame.

Examples

```
berkeley <- aggregate(Table(Admit, Freq) ~ ., data=UCBAdmissions)
berktest1 <- By(~Dept+Gender,
               glm(cbind(Admitted, Rejected) ~ 1, family="binomial"),
               data=berkeley)
berktest2 <- By(~Dept,
               glm(cbind(Admitted, Rejected) ~ Gender, family="binomial"),
               data=berkeley)
Stest1 <- Lapply(berktest2, function(x) predict(x, , se.fit=TRUE) [c("fit", "se.fit")])
Stest2 <- Sapply(berktest2, function(x) coef(summary(x)))
Stest2.1 <- Lapply(berktest1, function(x) predict(x, , se.fit=TRUE) [c("fit", "se.fit")])
to.data.frame(Stest1)
to.data.frame(Stest2, as.vars=2)
to.data.frame(Stest2.1)
```

Description

Methods for the generic function `toLatex` of package “`utils`” are provided for generating LaTeX representations of matrices and flat contingency tables (see `ftable`). Also a default method is defined that coerces its first argument into a matrix and applies the matrix method.

Usage

```

## Default S3 method:
toLatex(object,...)
## S3 method for class 'ftable':
toLatex(object,
  show.titles=TRUE,
  digits=0,
  format="f",
  useDcolumn=TRUE,
  colspec= if (useDcolumn)
    paste("D{.}{", LaTeXDec, "}{", ddigits, "}", sep="")
    else "r",
  LaTeXDec=".",
  ddigits=digits,
  useBooktabs=TRUE,
  toprule=if(useBooktabs) "\\toprule" else "\\hline\\hline",
  midrule=if(useBooktabs) "\\midrule" else "\\hline\\n",
  cmidrule=if(useBooktabs) "\\cmidrule" else "\\cline",
  bottomrule=if(useBooktabs) "\\bottomrule" else "\\hline\\hline",
  extrarowsep = NULL,
  ...)
## S3 method for class 'matrix':
toLatex(object,
  show.titles=TRUE,
  digits=0,
  format="f",
  useDcolumn=TRUE,
  colspec= if (useDcolumn)
    paste("D{.}{", LaTeXDec, "}{", ddigits, "}", sep="")
    else "r",
  LaTeXDec=".",
  ddigits=digits,
  useBooktabs=TRUE,
  toprule=if(useBooktabs) "\\toprule" else "\\hline\\hline",
  midrule=if(useBooktabs) "\\midrule" else "\\hline",
  cmidrule=if(useBooktabs) "\\cmidrule" else "\\cline",
  bottomrule=if(useBooktabs) "\\bottomrule" else "\\hline\\hline",
  ...)

```

Arguments

object	an ftable , a matrix or an object coercable into a matrix.
show.titles	logical; should variable names (in case of the <code>ftable</code> method) or row and column names (in case of the <code>matrix</code> method) be appear in the LaTeX code?
digits	number of significant digits.
format	format specifier, see format .
useDcolumn	should the <code>dcolumn</code> LaTeX package be used?

<code>colspec</code>	LaTeX table column format specifier(s).
<code>LaTeXdec</code>	the decimal point in the final LaTeX output.
<code>ddigits</code>	alignment specification or digits after the decimal point.
<code>useBooktabs</code>	should the <code>booktabs</code> LaTeX package be used?
<code>toprule</code>	appearance of the top border of the LaTeX <code>tabular</code> environment.
<code>midrule</code>	how are coefficients and summary statistics separated in the LaTeX <code>tabular</code> environment.
<code>cmidrule</code>	appearance of rules under section headings.
<code>bottomrule</code>	appearance of the bottom border of the LaTeX <code>tabular</code> environment.
<code>extrarowsep</code>	extra code to be inserted between the column titles and the table body produced by <code>toLatex</code> .
<code>...</code>	further argument, currently ignored.

Examples

```
toLatex(diag(5))
toLatex(ftable(UCBAdmissions))
```

`trimws` *Trim Leading and Trailing Whitespaces*

Description

`trimws` returns a character string with leading and/or trailing whitespaces removed. This is just a shorthand for an appropriate call to [sub](#).

Usage

```
trimws(x, left=TRUE, right=TRUE)
```

Arguments

<code>x</code>	a character vector
<code>left</code>	logical, if TRUE leading whitespace is removed
<code>right</code>	logical, if TRUE trailing whitespace is removed

Value

The character vector `x` with whitespace removed.

Examples

```
teststr <- c(" Hello World! ", "I am here. ")
trimws(teststr)
trimws(teststr, left=FALSE)
trimws(teststr, right=FALSE)
```

 UnZip

Extract Files from Zip Files

Description

Unzip extracts a file from a zip archive and puts them into a directory specified by the user or into the temporary directory of the current session.

Usage

```
UnZip(zipfile, item,
      dir=tempdir(), package=NULL)
```

Arguments

zipfile	a character string, the path to the zip file.
item	a character string, full path (from the root of the zip file) to the file to extract.
dir	path to the directory were to place the extracted file.
package	optional package name, if given, the path to the zipfile starts in the packages' root directory.

Examples

```
# Extract American National Election Study of 1948
# It is item "NES1948.POR" in zip file "anes/NES1948.ZIP"
# where this path is relative to the packages'
# root directory.
nes1948.por <- UnZip("anes/NES1948.ZIP", "NES1948.POR",
                   package="memisc")
nes1948.por
```

 Utility classes

Named Lists, Lists of Items, and Atomic Vectors

Description

The classes "named.list" and "item.list" are merely some 'helper classes' for the construction of the classes "data.set" and "importer".

Class "named.list" extends the basic class "list" by an additional slot "names". Its `initialize` method assures that the names of the list are unique.

Class "item.list" extends the class "named.list", but does not add any slots. From "named.list" it differs only by the `initialize` method, which calls that for "named.list" and makes sure that all elements of the list belong to class "item".

Classes "atomic" and "double" are merely used for method selection.

Examples

```

new("named.list", a=1, b=2)

# This should generate an error, since the names
# are not unique.
try(new("named.list", a=1, a=2))

# Another error, one name is missing.
try(new("named.list", a=1, 2))

# Also an error, the resulting list would be unnamed.
try(new("named.list", 1, 2))

new("item.list", a=1, b=2)

# Also an error: "item.list"s are "named.lists",
# and here the names would be non-unique.
try(new("item.list", a=1, a=2))

```

value.filter

Value Filters

Description

Value filters, that is objects that inherit from class "value.filter", are a mechanism to distinguish between valid codes of a survey item and codes that are considered to be missing, such as the codes for answers like "don't know" or "answer refused".

Value filters are optional slot values of "item" objects. They determine which codes of "item" objects are replaced by NA when they are coerced into a vector or a factor.

There are three (sub)classes of value filters: "missing.values", which specify individual missing values and/or a range of missing values; "valid.values", which specify individual valid values (that is, all other values of the item are considered as missing); "valid.range", which specify a range of valid values (that is, all values outside the range are considered as missing). Value filters of class "missing.values" correspond to missing-values declarations in SPSS files, imported by [spss.fixed.file](#), [spss.portable.file](#), or [spss.system.file](#).

Value filters also can be updated using the + and - operators.

Usage

```

value.filter(x)

missing.values(x)
missing.values(x) <- value

valid.values(x)
valid.values(x) <- value

```

```

valid.range(x)
valid.range(x) <-value

is.valid(x)
nvalid(x)
is.missing(x)
include.missings(x,mark="*")

```

Arguments

`x`, `value` objects of the appropriate class.
`mark` a character string, used to pasted to value labels of `x` (if present).

Value

`value.filter(x)`, `missing.values(x)`, `valid.values(x)`, and `valid.range(x)`, return the value filter associated with `x`, an object of class "value.filter", that is, of class "missing.values", "valid.values", or "valid.range", respectively.

`is.missing(x)` returns a logical vector indicating for each element of `x` whether it is a missing value or not. `is.valid(x)` returns a logical vector indicating for each element of `x` whether it is a valid value or not. `nvalid(x)` returns the number of elements of `x` that are valid.

For convenience, `is.missing(x)` and `is.valid(x)` also work for atomic vectors and factors, where they are equivalent to `is.na(x)` and `!is.na(x)`. For atomic vectors and factors, `nvalid(x)` returns the number of elements of `x` for which `!is.na(x)` is TRUE.

`include.missings(x, ...)` returns a copy of `x` that has all values declared as valid.

Examples

```

x <- rep(c(1:4,8,9),2,length=60)
labels(x) <- c(
  a=1,
  b=2,
  c=3,
  d=4,
  dk=8,
  refused=9
)
missing.values(x) <- 9
missing.values(x)
missing.values(x) <- missing.values(x) + 8
missing.values(x)
missing.values(x) <- NULL
missing.values(x)
missing.values(x) <- list(range=c(8,Inf))
missing.values(x)
valid.values(x)
print(x)
is.missing(x)
is.valid(x)

```

```
as.factor(x)
as.factor(include.missings(x))
as.integer(x)
as.integer(include.missings(x))
```

Index

- *Topic **datagen**
 - prediction.frame, 47
- *Topic **file**
 - importers, 25
- *Topic **hplot**
 - panel.errbars, 45
 - Termplot, 66
- *Topic **manip**
 - By, 11
 - cases, 11
 - codebook, 13
 - collect, 15
 - dimrename, 22
 - items, 30
 - measurement, 34
 - query, 49
 - recode, 50
 - relabel, 53
 - rename, 54
 - reorder.array, 55
 - retain, 57
 - to.data.frame, 68
- *Topic **misc**
 - aggregate.formula, 2
 - applyTemplate, 5
 - getSummary, 24
 - Memisc, 35
 - mkHelpDir, 39
 - mtable, 40
 - Sapply, 58
 - Simulate, 59
 - styles, 63
 - toLatex, 69
- *Topic **programming**
 - as.symbols, 7
 - foreach, 23
 - include, 29
 - Substitute, 64
- *Topic **univar**
 - By, 11
 - percent, 46
 - Table, 65
- *Topic **utilities**
 - applyTemplate, 5
 - collect, 15
 - getSummary, 24
 - Sapply, 58
 - trimws, 71
 - [, data.set, atomic, atomic, ANY-method
(data.set), 19
 - [, data.set, atomic, missing, ANY-method
(data.set), 19
 - [, data.set, missing, atomic, ANY-method
(data.set), 19
 - [, data.set, missing, missing, ANY-method
(data.set), 19
 - [, importer, atomic, atomic, ANY-method
(importers), 25
 - [, importer, atomic, missing, ANY-method
(importers), 25
 - [, importer, missing, atomic, ANY-method
(importers), 25
 - [, importer, missing, missing, ANY-method
(importers), 25
 - [, item.vector, logical, missing, missing-method
(items), 30
 - [, item.vector, numeric, missing, missing-method
(items), 30
 - [, value.labels, logical, missing, missing-method
(labels), 33
 - [, value.labels, numeric, missing, missing-method
(labels), 33
 - [.bucket (bucket), 8
 - [<- , data.set-method (data.set), 19
 - %in%, numeric.item, character-method
(items), 30
 - %nin% (negative match), 44
- aggregate.formula, 2

- aggregate, 37
- aggregate.data.frame, 3
- aggregate.formula, 46, 65
- aggregate.formula
 - (*aggragate.formula*), 2
- agrep, 49
- annotation, 32
- annotation(*annotations*), 4
- annotation, ANY-method
 - (*annotations*), 4
- annotation, data.set-method
 - (*annotations*), 4
- annotation, item-method
 - (*annotations*), 4
- annotation-class(*annotations*), 4
- annotation<-(*annotations*), 4
- annotation<- , ANY, character-method
 - (*annotations*), 4
- annotation<- , item, annotation-method
 - (*annotations*), 4
- annotation<- , vector, annotation-method
 - (*annotations*), 4
- annotations, 4
- applyTemplate, 5
- Arith, missing.values, missing.values-method
 - (*value.filter*), 73
- Arith, numeric, numeric.item-method
 - (*items*), 30
- Arith, numeric.item, numeric-method
 - (*items*), 30
- Arith, numeric.item, numeric.item-method
 - (*items*), 30
- Arith, valid.range, valid.range-method
 - (*value.filter*), 73
- Arith, valid.values, valid.values-method
 - (*value.filter*), 73
- Arith, value.filter, vector-method
 - (*value.filter*), 73
- Arith, value.labels, ANY-method
 - (*labels*), 33
- as.array, 7
- as.array, data.frame-method
 - (*as.array*), 7
- as.character, item.vector-method
 - (*items*), 30
- as.data.frame, 19
- as.data.frame.character.item
 - (*items*), 30
- as.data.frame.data.set
 - (*data.set*), 19
- as.data.frame.default_bucket
 - (*bucket*), 8
- as.data.frame.double.item
 - (*items*), 30
- as.data.frame.integer.item
 - (*items*), 30
- as.data.frame.textfile_bucket
 - (*bucket*), 8
- as.data.set, 36
- as.data.set(*data.set*), 19
- as.data.set, importer-method
 - (*importers*), 25
- as.factor, item.vector-method
 - (*items*), 30
- as.integer, item-method(*items*), 30
- as.item(*items*), 30
- as.item, character-method(*items*), 30
- as.item, character.item-method
 - (*items*), 30
- as.item, double.item-method
 - (*items*), 30
- as.item, factor-method(*items*), 30
- as.item, integer.item-method
 - (*items*), 30
- as.item, logical-method(*items*), 30
- as.item, numeric-method(*items*), 30
- as.item, ordered-method(*items*), 30
- as.matrix.default_bucket
 - (*bucket*), 8
- as.matrix.textfile_bucket
 - (*bucket*), 8
- as.numeric, item-method(*items*), 30
- as.ordered, item.vector-method
 - (*items*), 30
- as.symbol, 8
- as.symbols, 7
- as.vector, item-method(*items*), 30
- as.vector, value.labels-method
 - (*labels*), 33
- atomic-class(*Utility classes*), 72
- attach, 37
- bucket, 8, 60
- By, 11, 37
- by, 37, 58

- cases, 11, 36
- character.item-class (*items*), 30
- codebook, 13, 21, 27, 37, 65
- codebook, data.set-method (*codebook*), 13
- codebook, importer-method (*codebook*), 13
- codebook, item-method (*codebook*), 13
- codebook-class (*codebook*), 13
- coef.style, 41
- coef.style (*styles*), 63
- coerce, atomic, missing.values-method (*value.filter*), 73
- coerce, atomic, valid.range-method (*value.filter*), 73
- coerce, atomic, valid.values-method (*value.filter*), 73
- coerce, character, value.labels-method (*labels*), 33
- coerce, list, missing.values-method (*value.filter*), 73
- coerce, numeric, value.labels-method (*labels*), 33
- coerce, value.labels, character-method (*labels*), 33
- coerce, value.labels, numeric-method (*labels*), 33
- collect, 15, 39
- colrename, 38
- colrename (*dimrename*), 22
- Compare, character, numeric.item-method (*items*), 30
- Compare, numeric.item, character-method (*items*), 30
- contr, 17
- contr.sum, 17
- contr.treatment, 17, 18
- contrasts, 18
- contrasts (*contr*), 17
- contrasts, ANY-method (*contr*), 17
- contrasts, item-method (*contr*), 17
- contrasts<- (*contr*), 17
- contrasts<-, ANY-method (*contr*), 17
- contrasts<-, item-method (*contr*), 17
- data.frame, 19, 34
- data.set, 5, 13, 19, 25, 34–36, 49
- data.set-class (*data.set*), 19
- default_bucket (*bucket*), 8
- description, 25, 27
- description (*annotations*), 4
- description, data.set-method (*annotations*), 4
- description, importer-method (*annotations*), 4
- description<- (*annotations*), 4
- Descriptives, 21
- Descriptives, atomic-method (*Descriptives*), 21
- Descriptives, item.vector-method (*Descriptives*), 21
- detach.sources (*include*), 29
- dim, data.set-method (*data.set*), 19
- dim, importer-method (*importers*), 25
- dim.default_bucket (*bucket*), 8
- dim.textfile_bucket (*bucket*), 8
- dimnames, 15, 38
- dimnames, data.set-method (*data.set*), 19
- dimnames<- , data.set-method (*data.set*), 19
- dimrename, 22, 38, 41
- double-class (*Utility classes*), 72
- double.item-class (*items*), 30
- factor, 34
- factor.style, 41
- factor.style (*styles*), 63
- fapply (*aggregate.formula*), 2
- foreach, 23, 38
- format, 70
- format, codebookEntry-method (*codebook*), 13
- format, item.vector-method (*items*), 30
- format, missing.values-method (*value.filter*), 73
- format, valid.range-method (*value.filter*), 73
- format, valid.values-method (*value.filter*), 73
- format.mtable (*mtable*), 40
- formatC, 6
- formula, 37
- ftable, 38, 69, 70

- genTable, 37, 38, 65
- genTable (*aggragate.formula*), 2
- getCoefTemplate, 41
- getCoefTemplate (*styles*), 63
- getSummary, 24, 41
- getSummaryTemplate, 41
- getSummaryTemplate (*styles*), 63
- glm, 25
- grep, 49
- gsub, 22, 53, 55

- help.start, 39

- ifelse, 11, 12, 36
- importer, 5, 13, 36, 49
- importer (*importers*), 25
- importer-class (*importers*), 25
- importers, 25
- include, 29
- include.missings (*value.filter*), 73
- include.missings, item-method (*value.filter*), 73
- initialize, data.set-method (*data.set*), 19
- initialize, item.list-method (*Utility classes*), 72
- initialize, named.list-method (*Utility classes*), 72
- initialize, spss.fixed.importer-method (*importers*), 25
- initialize, spss.portable.importer-method (*importers*), 25
- initialize, spss.system.importer-method (*importers*), 25
- initialize, Stata.importer-method (*importers*), 25
- initialize, value.labels-method (*labels*), 33
- integer.item-class (*items*), 30
- interrupt (*Simulate*), 59
- is.data.set (*data.set*), 19
- is.interval (*measurement*), 34
- is.missing (*value.filter*), 73
- is.missing, atomic-method (*value.filter*), 73
- is.missing, factor-method (*value.filter*), 73
- is.missing, item.vector-method (*value.filter*), 73
- is.nominal (*measurement*), 34
- is.ordinal (*measurement*), 34
- is.ratio (*measurement*), 34
- is.valid (*value.filter*), 73
- item, 13, 19, 35, 72
- item (*items*), 30
- item-class (*items*), 30
- item.list (*Utility classes*), 72
- item.list-class (*Utility classes*), 72
- item.vector-class (*items*), 30
- items, 30

- labels, 25, 32, 33, 36
- labels, item-method (*labels*), 33
- labels<- (*labels*), 33
- labels<- , item, ANY-method (*labels*), 33
- labels<- , item, NULL-method (*labels*), 33
- labels<- , vector, ANY-method (*labels*), 33
- Lapply, 38
- Lapply (*Sapply*), 58
- lapply, 38
- lattice, 37
- lm, 25, 41

- make.packages.html, 39
- Math, numeric.item-method (*items*), 30
- Math2, numeric.item-method (*items*), 30
- measurement, 19, 34
- measurement, ANY-method (*measurement*), 34
- measurement, item-method (*measurement*), 34
- measurement<- (*measurement*), 34
- measurement<- , item-method (*measurement*), 34
- Memisc, 35
- memisc (*Memisc*), 35
- memisc-package (*Memisc*), 35
- missing.values, 25, 36
- missing.values (*value.filter*), 73

- missing.values, item.vector-method
(*value.filter*), 73
- missing.values-class
(*value.filter*), 73
- missing.values<- (*value.filter*),
73
- missing.values<- , ANY, atomic-method
(*value.filter*), 73
- missing.values<- , ANY, list-method
(*value.filter*), 73
- missing.values<- , atomic, missing.values-method
(*value.filter*), 73
- missing.values<- , item, ANY-method
(*value.filter*), 73
- missing.values<- , item, missing.values-method
(*value.filter*), 73
- missing.values<- , item, NULL-method
(*value.filter*), 73
- mkHelpDir, 39, 39
- mtable, 5, 24, 25, 38, 40, 63, 64

- named.list (*Utility classes*), 72
- named.list-class (*Utility classes*), 72
- names, 15, 38
- names, importer-method
(*importers*), 25
- negative match, 44
- numeric.item-class (*items*), 30
- nvalid, 3
- nvalid(*value.filter*), 73

- options, 6, 63, 64
- ordered, 34

- panel.errbars, 45
- panel.xyplot, 45
- paste, 8
- percent, 3, 46
- pour_out (*bucket*), 8
- predict, 48
- prediction.frame, 47
- print, data.set-method (*data.set*),
19
- print, item.vector-method (*items*),
30
- print.bucket (*bucket*), 8
- print.mtable (*mtable*), 40
- put_into (*bucket*), 8

- query, 49
- query, data.set-method (*query*), 49
- query, importer-method (*query*), 49
- query, item-method (*query*), 49

- read.dta, 25
- read.spss, 25, 27
- recode, 36, 50, 51
- recode, factor-method (*recode*), 50
- recode, item-method (*recode*), 50
- recode, vector-method (*recode*), 50
- relabel, 41, 53
- relabel.mtable (*mtable*), 40
- relabel4 (*relabel*), 53
- relabel4, item-method (*relabel*), 53
- rename, 38, 54
- reorder, 39, 56
- reorder (*reorder.array*), 55
- reorder.array, 55
- retain, 57
- row.names, data.set-method
(*data.set*), 19
- rowrename, 38
- rowrename (*dimrename*), 22

- sample-methods, 57
- sample.data.frame
(*sample-methods*), 57
- sample.data.set (*sample-methods*),
57
- Sapply, 38, 58
- sapply, 38, 58
- setCoefTemplate (*styles*), 63
- setSummaryTemplate, 25
- setSummaryTemplate (*styles*), 63
- show, 4, 14
- show, annotation-method
(*annotations*), 4
- show, codebook-method (*codebook*),
13
- show, data.set-method (*data.set*),
19
- show, item.vector-method (*items*),
30
- show, named.list-method (*Utility classes*), 72
- show, spss.fixed.importer-method
(*importers*), 25

- show, spss.portable.importer-method
(*importers*), 25
- show, spss.system.importer-method
(*importers*), 25
- show, Stata.importer-method
(*importers*), 25
- show, value.filter-method
(*value.filter*), 73
- show, value.labels-method
(*labels*), 33
- Simulate, 9, 38, 59
- sort-methods, 62
- sort.data.frame (*sort-methods*), 62
- sort.data.set (*sort-methods*), 62
- spss.fixed.file, 73
- spss.fixed.file (*importers*), 25
- spss.fixed.importer-class
(*importers*), 25
- spss.portable.file, 73
- spss.portable.file (*importers*), 25
- spss.portable.importer-class
(*importers*), 25
- spss.system.file, 73
- spss.system.file (*importers*), 25
- spss.system.importer-class
(*importers*), 25
- Stata.file (*importers*), 25
- Stata.importer-class (*importers*),
25
- str.character.item (*items*), 30
- str.data.set (*data.set*), 19
- str.double.item (*items*), 30
- str.integer.item (*items*), 30
- styles, 63
- sub, 71
- subset, 36
- subset, data.set-method
(*data.set*), 19
- subset, importer-method
(*importers*), 25
- Substitute, 64
- substitute, 64
- summary, data.set-method
(*data.set*), 19
- summary, item.vector-method
(*items*), 30
- Summary, numeric.item-method
(*items*), 30
- syms (*as.symbols*), 7
- Table, 3, 65
- table, 3, 11
- Table, atomic-method (*Table*), 65
- Table, factor-method (*Table*), 65
- Table, item.vector-method (*Table*),
65
- template, 63
- template (*applyTemplate*), 5
- Termplot, 37, 66
- termplot, 37, 67
- textfile_bucket (*bucket*), 8
- to.data.frame, 68
- toLatex, 38, 69, 69
- toLatex.mtable (*mtable*), 40
- trimws, 71
- uninclude (*include*), 29
- unique, data.set-method
(*data.set*), 19
- unique, item.vector-method
(*items*), 30
- UnZip, 72
- Utility classes, 72
- utils, 69
- valid.range (*value.filter*), 73
- valid.range, item.vector-method
(*value.filter*), 73
- valid.range-class (*value.filter*),
73
- valid.range<- (*value.filter*), 73
- valid.range<-, ANY, atomic-method
(*value.filter*), 73
- valid.range<-, atomic, valid.range-method
(*value.filter*), 73
- valid.range<-, item, valid.range-method
(*value.filter*), 73
- valid.values (*value.filter*), 73
- valid.values, item.vector-method
(*value.filter*), 73
- valid.values-class
(*value.filter*), 73
- valid.values<- (*value.filter*), 73
- valid.values<-, ANY, atomic-method
(*value.filter*), 73
- valid.values<-, atomic, valid.values-method
(*value.filter*), 73

`valid.values<-`, `item`, `valid.values-method`
 (`value.filter`), 73
`value.filter`, 32, 73
`value.filter`, `item-method`
 (`value.filter`), 73
`value.filter-class`
 (`value.filter`), 73
`value.labels-class` (`labels`), 33

`within`, 19
`within`, `data.set-method`
 (`data.set`), 19
`wording` (`annotations`), 4
`wording<-` (`annotations`), 4
`write.mtable` (`mtable`), 40

`xtabs`, 3, 37
`xyplot`, 67