Package ‘table1’

January 6, 2023

Type Package

Version 1.4.3

Date 2023-01-05

Title Tables of Descriptive Statistics in HTML

Author Benjamin Rich [aut, cre, cph]

Maintainer Benjamin Rich `<mail@benjaminrich.net>`

URL `https://github.com/benjaminrich/table1`

BugReports `https://github.com/benjaminrich/table1/issues`

Description Create HTML tables of descriptive statistics, as one would expect to see as the first table (i.e. ``Table 1") in a medical/epidemiological journal article.

License GPL-3

Depends R (>= 3.5.0)

Imports stats,Formula,knitr,htmltools,yaml

Suggests boot,MatchIt,rmarkdown,printr,kableExtra,flextable

VignetteBuilder knitr

Language en-US

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-01-06 09:30:02 UTC

R topics documented:

`as.data.frame.table1` *Convert a* `table1` *object to a* `data.frame`.

Description

Convert a `table1` object to a `data.frame`.

Usage

```
## S3 method for class 'table1'
as.data.frame(x, ...)
```

Arguments

`x`	An object returned by `table1`.
`...`	Ignored.

Value

A `data.frame`.

eqcut *Cut a continuous variable into equal-sized groups.*

Description

Cut a continuous variable into equal-sized groups.

Usage

```
eqcut(
  x,
  ngroups,
  labeling = eqcut.default.labeling,
  withhold = NULL,
  varlabel = if (has.label(x)) label(x) else deparse(substitute(x)),
  quantile.type = 7,
  right = FALSE,
  ...
)

eqcut.default.labeling(x, xcat, which, what, from, to, ...)
```

Arguments

`x`	A numeric vector.
`ngroups`	The number of groups desired.
`labeling`	A function that produces the category labels (see Details).
`withhold`	A named list of logical vectors (see Details).
`varlabel`	A character string to be used as a label for `x`, or `NULL`.
`quantile.type`	An integer from 1 to 9, passed as the `type` argument to function `quantile`.
`right`	Should intervals be right-closed? (passed to `cut`).
`...`	Further arguments passed on to function `labeling`.
`xcat`	A factor returned by `cut`.
`which, what`	Character vectors for labeling the categories in an appropriate way (see Examples).
`from, to`	Numeric vectors giving the ranges covered by the categories of `x`.

Details

The function `labeling` must have the signature `function(x, xcat,which, what, from, to, ...)` and produces the character vector of factor levels. See below for an example.

The `withhold` list can be used when `x` contains special values that should not be considered in the calculation of the quantiles used to create the `ngroups` categories. The special values are given a label that corresponds to the name of the corresponding list element. See below for an example.

Value

A `factor` of the same length as `x`. There are `ngroups` levels plus one additional level for each element of `withhold`.

Functions

- `eqcut.default.labeling()`: The default labeling function.

See Also

`cut` `quantile`

Examples

```
x <- sample(100)
table(eqcut(x, 2))
table(eqcut(x, 3))
table(eqcut(x, 4))
table(eqcut(x, 5))
table(eqcut(x, 6))
table(eqcut(x, 7))
table(eqcut(x, 8))

# An example of using eqcut in a table with custom labeling function.
dat <- expand.grid(id=1:100, sex=c("Male", "Female"), treat=c("Treated", "Placebo"))
dat$age <- runif(nrow(dat), 18, 50)
dat$wt <- exp(rnorm(nrow(dat), log(75 + 10*(dat$sex=="Male")), 0.2))
dat$auc <- ifelse(dat$treat=="Placebo", NA, exp(rnorm(nrow(dat), log(1000), 0.34)))
dat$auc[3] <- NA  # Add a missing value

label(dat$sex) <- "Sex"
label(dat$age) <- "Age"
label(dat$wt)  <- "Weight"
label(dat$auc) <- "AUC"
units(dat$age) <- "y"
units(dat$wt)  <- "kg"
units(dat$auc) <- "ng.h/mL"

w <- list(Placebo=(dat$treat=="Placebo"), Excluded=is.na(dat$auc))
f <- function(x, xcat, which, what, from, to, ...) {
   what <- sub("of ", "of<br/>", what)
   sprintf("%s %s<br/>&ge;%s to &lt;%s",
       which, what, signif_pad(from, 3, FALSE), signif_pad(to, 3, FALSE))
}
table1(~ sex + age + wt | eqcut(auc, 3, f, w), data=dat)
```

knit_print.table1 — *Method for printing in a knitr context.*

Description

Method for printing in a knitr context.

Usage

```
## S3 method for class 'table1'
knit_print(x, ...)
```

Arguments

x	An object returned by table1.
...	Further arguments passed on to knitr::knit_print.

Details

If the target is HTML, the usual internal formatting will be applied; otherwise, fall back to a ‘data.frame‘.

label — *Label attribute.*

Description

Label attribute.

Usage

```
label(x)

label(x) <- value

setLabel(x, value)

has.label(x)
```

Arguments

x	An object.
value	A character specifying the label.

Functions

- `label(x) <- value`: Set label attribute.
- `setLabel()`: Set label attribute.
- `has.label()`: Check for label attribute.

Examples

```
x <- 1:10
label(x) <- "Foo"
x <- setLabel(x, "Foo") # Alternative syntax
has.label(x)
label(x)
```

`parse.abbrev.render.code` *Parse abbreviated code for rendering table output.*

Description

Parse abbreviated code for rendering table output.

Usage

```
parse.abbrev.render.code(code, ...)
```

Arguments

`code`	A `character` vector specifying the statistics to display in abbreviated code. See Details.
`...`	Further arguments, passed to `stats.apply.rounding`.

Details

In abbreviated code, the words N, NMISS, MEAN, SD, MIN, MEDIAN, MAX, IQR, CV, GMEAN, GSD, GCV, FREQ and PCT are substituted for their respective values (see `stats.default`). The substitution is case insensitive, and the substituted values are rounded appropriately (see `stats.apply.rounding`). Other text is left unchanged. The `code` can be a vector, in which case each element is displayed in its own row in the table. The `names` of `code` are used as row labels; if no names are present, then the `code` itself is used unless `code` is of length 1, in which case no label is used (for numeric variables only, categorical variables are always labeled by the class label). The special name '.' also indicates that `code` itself be is used as the row label.

Value

A function that takes a single argument and returns a `character` vector.

Examples

```
## Not run:
x <- round(exp(rnorm(100, log(20), 1)), 2)
stats.default(x)
f <- parse.abbrev.render.code(c("Mean (SD)", "Median [Min, Max]"), 3)
f(x)
f2 <- parse.abbrev.render.code(c("Geo. Mean (Geo. CV%)" = "GMean (GCV%)"), 3)
f2(x)
f3 <- parse.abbrev.render.code(c("Mean (SD)"), 3)
f3(x)

x <- sample(c("Male", "Female"), 30, replace=T)
stats.default(x)
f <- parse.abbrev.render.code("Freq (Pct%)")
f(x)

## End(Not run)
```

---

print.table1 *Print* `table1` *object.*

Description

Print `table1` object.

Usage

```
## S3 method for class 'table1'
print(x, ...)
```

Arguments

`x`	An object returned by `table1`.
`...`	Further arguments passed on to other `print` methods.

Details

In an interactive context, the rendered table will be displayed in a web browser. Otherwise, the HTML code will be printed as text.

Value

Returns x invisibly.

`render.categorical.default`
Render categorical values for table output.

Description

Called from `table1` by default to render categorical (i.e. `factor`, `character` or `logical`) values for displaying in the table.

Usage

```
render.categorical.default(x, ..., na.is.category = TRUE)
```

Arguments

`x`	A vector of type `factor`, `character` or `logical`.
`...`	Further arguments, passed to `stats.apply.rounding`.
`na.is.category`	Include missing values in the denominator for calculating percentages (the default) or omit them.

Value

A `character` vector. Each element is to be displayed in a separate cell in the table. The `names` of the vector are the labels to use in the table. However, the first names should be empty as it will be replaced by the name of the variable. Empty strings are allowed and result in empty table cells.

Examples

```
y <- factor(sample(0:1, 99, replace=TRUE), labels=c("Female", "Male"))
y[1:10] <- NA
render.categorical.default(y)
```

`render.continuous.default`
Render continuous values for table output.

Description

Called from `table1` by default to render continuous (i.e. `numeric`) values for displaying in the table.

Usage

```
render.continuous.default(x, ...)
```

Arguments

`x`	A numeric vector.
`...`	Further arguments, passed to `stats.apply.rounding`.

Value

A `character` vector. Each element is to be displayed in a separate cell in the table. The `names` of the vector are the labels to use in the table. However, the first names should be empty as it will be replaced by the name of the variable. Empty strings are allowed and result in empty table cells.

Examples

```
x <- exp(rnorm(100, 1, 1))
render.continuous.default(x)
```

`render.default` *Render values for table output.*

Description

Called from `table1` by default to render values for displaying in the table. This function forwards the call to separate functions for rendering continuous, categorical and missing values. The idea is that each of these functions can be overridden to customize the table output.

Usage

```
render.default(
  x,
  name,
  missing = any(is.na(x)),
  transpose = F,
  render.empty = "NA",
  render.continuous = render.continuous.default,
  render.categorical = render.categorical.default,
  render.missing = render.missing.default,
  ...
)
```

Arguments

`x`	A vector or numeric, factor, character or logical values.
`name`	Name of the variable to be rendered (ignored).
`missing`	Should missing values be included?
`transpose`	Logical indicating whether on not the table is transposed.
`render.empty`	A `character` to return when `x` is empty.

`render.continuous`	A function to render continuous (i.e. `numeric`) values. Can also be a `character` string, in which case it is passed to `parse.abbrev.render.code`.
`render.categorical`	A function to render categorical (i.e. `factor`, `character` or `logical`) values. Can also be a `character` string, in which case it is passed to `parse.abbrev.render.code`.
`render.missing`	A function to render missing (i.e. `NA`) values. Can also be a `character` string, in which case it is passed to `parse.abbrev.render.code`. Set to `NULL` to ignore missing values.
`...`	Further arguments, passed to `stats.apply.rounding`.

Value

A `character` vector. Each element is to be displayed in a separate cell in the table. The `names` of the vector are the labels to use in the table. However, the first names should be empty as it will be replaced by the name of the variable. Empty strings are allowed and result in empty table cells.

Examples

```
x <- exp(rnorm(100, 1, 1))
render.default(x)
render.default(x, TRUE)

y <- factor(sample(0:1, 99, replace=TRUE), labels=c("Female", "Male"))
y[1:10] <- NA
render.default(y)
```

`render.missing.default` *Render missing values for table output.*

Description

Called from `table1` by default to render missing (i.e. `NA`) values for displaying in the table.

Usage

```
render.missing.default(x, ...)
```

Arguments

`x`	A vector.
`...`	Further arguments, passed to `stats.apply.rounding`.

Value

A `character` vector. Each element is to be displayed in a separate cell in the table. The `names` of the vector are the labels to use in the table. Empty strings are allowed and result in empty table cells.

Examples

```
y <- factor(sample(0:1, 99, replace=TRUE), labels=c("Female", "Male"))
y[1:10] <- NA
render.missing.default(y)
```

`render.varlabel` *Render variable labels for table output.*

Description

Called from `table1.formula` by default to render variable labels for displaying in the table.

Usage

```
render.varlabel(x, transpose = F)
```

Arguments

`x`	A vector, usually with the `label` and (if appropriate) `unit` attributes.
`transpose`	Logical indicating whether on not the table is transposed.

Value

A `character`, which may contain HTML markup.

Examples

```
x <- exp(rnorm(100, 1, 1))
label(x) <- "Weight"
units(x) <- "kg"
render.varlabel(x)

y <- factor(sample(0:1, 99, replace=TRUE), labels=c("Female", "Male"))
y[1:10] <- NA
label(y) <- "Sex"
render.varlabel(y)
```

signif_pad — *Round numbers with 0-padding.*

Description

Utility functions to round numbers, similar the the base functions `signif` and `round`, but resulting in character representations that keep zeros at the right edge if they are significant.

Usage

```
signif_pad(x, digits = 3, round.integers = TRUE, round5up = TRUE, dec, ...)

round_pad(x, digits = 2, round5up = TRUE, dec, ...)
```

Arguments

`x`	A numeric vector.
`digits`	An integer specifying the number of significant digits to keep (for `signif_pad`) or the number of digits after the decimal point (for `round_pad`).
`round.integers`	Should rounding be limited to digits to the right of the decimal point?
`round5up`	Should numbers with 5 as the last digit always be rounded up? The standard R approach is "go to the even digit" (IEC 60559 standard, see `round`), while some other softwares (e.g. SAS, Excel) always round up.
`dec`	The character symbol to use as decimal mark (locale specific). [Deprecated; use `decimal.mark` instead]
`...`	Further options, passed to `formatC` (which is used internally). Not all options will work, but some might be useful (e.g. `big.mark`, `decimal.mark`).

Value

A character vector containing the rounded numbers.

See Also

`signif` `round` `formatC` `prettyNum` `format`

Examples

```
x <- c(0.9001, 12345, 1.2, 1., 0.1, 0.00001 , 1e5)
signif_pad(x, digits=3)
signif_pad(x, digits=3, round.integers=TRUE)

# Compare:
as.character(signif(x, digits=3))
format(x, digits=3, nsmall=3)
prettyNum(x, digits=3, drop0trailing=TRUE)
prettyNum(x, digits=3, drop0trailing=FALSE)
```

```
# This is very close.
formatC(x, format="fg", flag="#", digits=3)
formatC(signif(x, 3), format="fg", flag="#", digits=3)

# Could always remove the trailing "."
sub("[.]$", "", formatC(x, format="fg", flag="#", digits=3))
```

stats.apply.rounding *Apply rounding to basic descriptive statistics.*

Description

Not all statistics should be rounded in the same way, or at all. This function will apply rounding selectively to a list of statistics as returned by `stats.default`. In particular we don't round counts (N, NMISS and FREQ), and for MIN, MAX and MEDIAN the `digits` is interpreted as the *minimum* number of significant digits, so that we don't loose any precision. Percentages are rounded to a fixed number of decimal places (default 1) rather than a specific number of significant digits.

Usage

```
stats.apply.rounding(
  x,
  digits = 3,
  digits.pct = 1,
  round.median.min.max = TRUE,
  round.integers = TRUE,
  round5up = TRUE,
  rounding.fn = signif_pad,
  ...
)
```

Arguments

`x`	A list, such as that returned by `stats.default`.
`digits`	An integer specifying the number of significant digits to keep.
`digits.pct`	An integer specifying the number of digits after the decimal place for percentages.
`round.median.min.max`	Should rounding applied to median, min and max?
`round.integers`	Should rounding be limited to digits to the right of the decimal point?
`round5up`	Should numbers with 5 as the last digit always be rounded up? The standard R approach is "go to the even digit" (IEC 60559 standard, see `round`), while some other softwares (e.g. SAS, Excel) always round up.
`rounding.fn`	The function to use to do the rounding. Defaults to `signif_pad`.
`...`	Further arguments.

Value

A list with the same number of elements as `x`. The rounded values will be `character` (not `numeric`) and will have 0 padding to ensure consistent number of significant digits.

See Also

`signif_pad` `stats.default`

Examples

```
x <- round(exp(rnorm(100, 1, 1)), 6)
stats.default(x)
stats.apply.rounding(stats.default(x), digits=3)
stats.apply.rounding(stats.default(round(x, 1)), digits=3)
```

`stats.default` *Compute some basic descriptive statistics.*

Description

Values of type `factor`, `character` and `logical` are treated as categorical. For logicals, the two categories are given the labels ‘Yes‘ for `TRUE`, and ‘No‘ for `FALSE`. Factor levels with zero counts are retained.

Usage

```
stats.default(x, quantile.type = 7, ...)
```

Arguments

`x`	A vector or numeric, factor, character or logical values.
`quantile.type`	An integer from 1 to 9, passed as the `type` argument to function `quantile`.
`...`	Further arguments (ignored).

Value

A list. For numeric `x`, the list contains the numeric elements:

- `N`: the number of non-missing values
- `NMISS`: the number of missing values
- `SUM`: the sum of the non-missing values
- `MEAN`: the mean of the non-missing values
- `SD`: the standard deviation of the non-missing values
- `MIN`: the minimum of the non-missing values
- `MEDIAN`: the median of the non-missing values

- `CV`: the percent coefficient of variation of the non-missing values
- `GMEAN`: the geometric mean of the non-missing values if non-negative, or `NA`
- `GSD`: the geometric standard deviation of the non-missing values if non-negative, or `NA`
- `GCV`: the percent geometric coefficient of variation of the non-missing values if non-negative, or `NA`
- `qXX`: various quantiles (percentiles) of the non-missing values (q01: 1%, q02.5: 2.5%, q05: 5%, q10: 10%, q25: 25% (first quartile), q33.3: 33.33333% (first tertile), q50: 50% (median, or second quartile), q66.7: 66.66667% (second tertile), q75: 75% (third quartile), q90: 90%, q95: 95%, q97.5: 97.5%, q99: 99%)
- `Q1`: the first quartile of the non-missing values (alias `q25`)
- `Q2`: the second quartile of the non-missing values (alias `q50` or `Median`)
- `Q3`: the third quartile of the non-missing values (alias `q75`)
- `IQR`: the inter-quartile range of the non-missing values (i.e., `Q3 - Q1`)
- `T1`: the first tertile of the non-missing values (alias `q33.3`)
- `T2`: the second tertile of the non-missing values (alias `q66.7`)

If `x` is categorical (i.e. factor, character or logical), the list contains a sublist for each category, where each sublist contains the numeric elements:

- `FREQ`: the frequency count
- `PCT`: the percent relative frequency, including NA in the denominator
- `PCTnoNA`: the percent relative frequency, excluding NA from the denominator
- `NMISS`: the number of missing values

Examples

```
x <- exp(rnorm(100, 1, 1))
stats.default(x)

y <- factor(sample(0:1, 99, replace=TRUE), labels=c("Female", "Male"))
y[1:10] <- NA
stats.default(y)
stats.default(is.na(y))
```

`subsetp` *Subset function that preserves column attributes.*

Description

Subset function that preserves column attributes.

Usage

```
subsetp(x, ..., droplevels = TRUE)
```

Arguments

`x`	An object to be subsetted (usually a `data.frame`).
`...`	Further arguments passed to `subset`.
`droplevels`	If `TRUE` (the default), then unused factor levels are dropped (see `droplevels`).

Value

An object similar to `x` containing just the selected elements. In the case of a `data.frame`, attributes of columns (such as `label` and `units`) are preserved.

See Also

`subset` `droplevels`

`t1flex` *Convert a* `table1` *object to* `flextable`.

Description

Convert a `table1` object to `flextable`.

Usage

```
t1flex(x, tablefn = c("qflextable", "flextable", "regulartable"), ...)
```

Arguments

`x`	An object returned by `table1`.
`tablefn`	Choose a function from the `flextable` package to use as the basis for the table.
`...`	Further options passed to `tablefn`.

Value

A `flextable` object.

Note

The `flextable` package needs to be installed for this to work.

`t1kable` *Convert a* `table1` *object to* `kabelExtra`.

Description

Convert a `table1` object to `kabelExtra`.

Usage

```
t1kable(x, booktabs = TRUE, ..., format)
```

Arguments

`x`	An object returned by `table1`.
`booktabs`	Passed to `kbl` (default `TRUE`).
`...`	Other options passed to `kbl`.
`format`	Passed to `kbl` (optional).

Value

A `kabelExtra` object.

Note

The `kableExtra` package needs to be installed for this to work.

`t1read` *Read and augment data with extended metadata attributes*

Description

Read and augment data with extended metadata attributes

Usage

```
t1read(data, metadata = NULL, read.fun = read.csv, ..., escape.html = TRUE)
```

Arguments

data	Either a file name (character) or a data.frame. If a file name it will be read using the function read.fun.
metadata	Either a file name (character) or a list. If a file name it will be read using the function read_yaml (so it should be a file the contains valid YAML text), and a list results. See Details regarding the list contents.
read.fun	A function to read files. It should accept a file name as its first argument and return a data.frame.
...	Further optional arguments, passed to read.fun.
escape.html	Logical. Should strings (labels, units) be converted to valid HTML by escaping special symbols?

Details

The metadata list may contain the following 3 named elements (other elements are ignored):

- labels: a named list, with names corresponding to columns in data and values the associated label attribute.
- units: a named list, with names corresponding to columns in data and values the associated units attribute.
- categoricals: a named list, with names corresponding to columns in data and values are themselves lists, used to convert the column to a factor: the list names are the levels, and the values are the associated labels. The names can also be omitted if the goal is just to specify the order of the factor levels.

Value

A data.frame (as returned by read.fun).

Examples

```
# Simulate some data
set.seed(123)
data <- expand.grid(sex=0:1, cohort=1:3)[rep(1:6, times=c(7, 9, 21, 22, 11, 14)),]
data$age <- runif(nrow(data), 18, 80)
data$agecat <- 1*(data$age >= 65)
data$wgt <- rnorm(nrow(data), 75, 15)

metadata <- list(
  labels=list(
    cohort = "Cohort",
    sex = "Sex",
    age = "Age",
    agecat  = "Age category",
    wgt = "Weight"),
  units=list(
    age = "years",
    wgt = "kg"),
  categoricals=list(
    cohort = list(
      `1` = "Cohort A",
      `2` = "Cohort B",
      `3` = "Cohort C"),
    sex = list(
      `0` = "Female",
      `1` = "Male"),
    agecat = list(
      `0` = "< 65",
      `1` = "\U{2265} 65")))

data <- t1read(data, metadata)
table1(~ sex + age + agecat + wgt | cohort, data=data)
```

`table.rows` *Convert to HTML table rows.*

Description

Many functions exist in R to generate HTML tables. These functions are useful for generating HTML table fragments (rather than whole tables), which can then be used to build up complete tables. The first column my be used to label the rows of the table. Row labels, if specified, can have a special HTML `class` designated, which can be useful as a hook to customize their appearance using CSS. The same is true for the the first and last row of cells.

Usage

```
table.rows(
  x,
  row.labels = rownames(x),
  th = FALSE,
  class = NULL,
  rowlabelclass = "rowlabel",
  firstrowclass = "firstrow",
  lastrowclass = "lastrow",
  ...
)

table.data(
  x,
  row.labels = rownames(x),
  th = FALSE,
  class = NULL,
  rowlabelclass = "rowlabel",
  firstrowclass = "firstrow",
  lastrowclass = "lastrow",
  ...
)
```

Arguments

`x`	A vector or table-like structure (e.g. a `data.frame` or `matrix`).
`row.labels`	Values for the first column, typically used to label the row, or `NULL` to omit.
`th`	A logical. Should `th` tags be used rather than `td`?
`class`	HTML class attribute. Can be a single `character`, a vector or a matrix.
`rowlabelclass`	HTML class attribute for the row labels (i.e. first column).
`firstrowclass`	HTML class attribute for the first row of cells.
`lastrowclass`	HTML class attribute for the last row of cells.
`...`	Additional arguments.

Value

A `character` which contains an HTML table fragment.

Functions

- `table.data()`: Convert to HTML table data (cells).

Examples

```
x <- matrix(signif_pad(exp(rnorm(5*5, 1, 1))), 5, 5)
table.data(x)
cat(table.rows(x, NULL))
cat(table.rows(x, LETTERS[1:nrow(x)]))
cat(table.rows(LETTERS[1:3], "Headings", th=TRUE))
```

`table1` *Generate an HTML table of descriptive statistics.*

Description

Produces a nicely formatted table of descriptive statistics for any number of numeric or categorical variables, optionally stratified by a factor.

Usage

```
table1(x, ...)

## Default S3 method:
table1(
  x,
  labels,
  groupspan = NULL,
  rowlabelhead = "",
  transpose = FALSE,
  topclass = "Rtable1",
  footnote = NULL,
  caption = NULL,
  render = render.default,
  render.strat = render.strat.default,
  extra.col = NULL,
  extra.col.pos = NULL,
  ...
)

## S3 method for class 'formula'
table1(
  x,
  data,
  overall = "Overall",
  rowlabelhead = "",
  transpose = FALSE,
  droplevels = TRUE,
  topclass = "Rtable1",
  footnote = NULL,
  caption = NULL,
  render = render.default,
  render.strat = render.strat.default,
  extra.col = NULL,
  extra.col.pos = NULL,
  ...
)
```

Arguments

`x`	An object, typically a `formula` or list of `data.frames` (see Details).
`...`	Further arguments, passed to `render`.
`labels`	A list containing labels for variables, strata and groups (see Details).
`groupspan`	A vector of integers specifying the number of strata to group together.
`rowlabelhead`	A heading for the first column of the table, which contains the row labels.
`transpose`	Logical. Should the table be transposed (i.e. strata as rows and variables as columns)?
`topclass`	A class attribute for the outermost (i.e. `<table>`) tag.
`footnote`	A character string to be added as a footnote to the table. Can also be a vector which results in multiple lines of footnotes. The default `NULL` causes the footnote to be omitted.
`caption`	A character string to be added as a caption to the table. The default `NULL` causes the caption to be omitted.
`render`	A function to render the table cells (see Details).
`render.strat`	A function to render the stratum labels. Accepts 3 arguments: the stratum label, the stratum size (number of observations), and a flag indicating whether we are in transpose mode or not. See `render.strat.default` for an example.

`extra.col`	An optional names list of functions that produce extra columns in the table (see Details).
`extra.col.pos`	An optional integer vector given the positions of extra columns (see Details).
`data`	For the formula interface, a `data.frame` from which the variables in `x` should be taken.
`overall`	A label for the "Overall" column. Specify `NULL` or `FALSE` to omit the column altogether. By default, the "Overall" column appears at the right end of the table; to place it on the left instead use a named `character` with the name "left", e.g. `c(left="Overall")`.
`droplevels`	Should empty factor levels be dropped?

Details

There are two interfaces, the default, which typically takes a list of `data.frames` for `x`, and the formula interface. The formula interface is less flexible, but simpler to use and designed to handle the most common use cases. It is important to use factors appropriately for categorical variables (i.e. have the levels labeled properly and in the desired order). The contents of the table can be customized by providing user-defined 'renderer' functions. Customization of the table appearance is deliberately not attempted, as this is best accomplished with CSS. To facilitate this, some tags (such as row labels) are given specific classes for easy CSS selection.

For the formula version, the formula is expected to be a one-sided formula, optionally with a vertical bar separating the variables that are to appear as data in the table (as rows) from those used for stratification (i.e. columns). There can be at most 2 variables for stratification (and only one if `transpose = TRUE` is specified), and if 2 are specified, the second is nested within the first. Stratification variables may not contain missing values. The formula may contain a dot (".") to refer to "all variables in `data` other than those that appear elsewhere in the formula". It is legitimate to use functions inside the formula to create new variables.

For the default version, is is expected that `x` is a named list of `data.frames`, one for each stratum, with names corresponding to strata labels.

Extra columns can be added to the table using the `extra.col` argument. This is an optional named list of functions, with the names corresponding to the column headings. Each function will be called once for each variable included in the table. Each function should expect 2 arguments, the first being a list, the second the name of the variable. The contents of the list passed in as the first argument will be the data associated with each stratum in the table; i.e., one element for each normal column (not extra column). It is then up the function to compute the value to appear in the extra column and return it as a string. By default, extra columns will be placed to the far right, after the normal columns, in the order they are specified in. This can be overridden, however, using the `extra.col.pos` vector of integer positions. For example, to place the first extra column in position 1 (far left), and the second extra column in position 3, use `extra.col.pos = c(1, 3)`; any extra columns that are not assigned positions will be placed to the far right. A typical use case for extra columns would be a column of p-values for differences between strata. Note that this feature is not available when the option `transpose = TRUE` is specified.

Value

An object of class "table1".

Methods (by class)

- `table1(default)`: The default interface, where `x` is a `data.frame`.
- `table1(formula)`: The `formula` interface.

Examples

```
dat <- expand.grid(id=1:10, sex=c("Male", "Female"), treat=c("Treated", "Placebo"))
dat$age <- runif(nrow(dat), 10, 50)
dat$age[3] <- NA  # Add a missing value
dat$wt <- exp(rnorm(nrow(dat), log(70), 0.2))

label(dat$sex) <- "Sex"
label(dat$age) <- "Age"
label(dat$treat) <- "Treatment Group"
label(dat$wt) <- "Weight"

units(dat$age) <- "years"
units(dat$wt) <- "kg"

# One level of stratification
table1(~ sex + age + wt | treat, data=dat)

# Two levels of stratification (nesting)
table1(~ age + wt | treat*sex, data=dat)

# Switch the order or nesting
table1(~ age + wt | sex*treat, data=dat)

# No stratification
table1(~ treat + sex + age + wt, data=dat)

# Something more complicated

dat$dose <- ifelse(dat$treat=="Placebo", "Placebo",
                   sample(c("5 mg", "10 mg"), nrow(dat), replace=TRUE))
dat$dose <- factor(dat$dose, levels=c("Placebo", "5 mg", "10 mg"))

strata <- c(split(dat, dat$dose),
            list("All treated"=subset(dat, treat=="Treated")),
            list(Overall=dat))

labels <- list(
    variables=list(sex=render.varlabel(dat$sex),
                   age=render.varlabel(dat$age),
                   wt=render.varlabel(dat$wt)),
    groups=list("", "Treated", ""))

my.render.cont <- function(x) {
    with(stats.default(x),
        sprintf("%0.2f (%0.1f)", MEAN, SD))
}
```

```
table1(strata, labels, groupspan=c(1, 3, 1), render.continuous=my.render.cont)

# Transposed table
table1(~ age + wt | treat, data=dat, transpose=TRUE)
```

units *Units attribute.*

Description

Units attribute.

Usage

```
units(x)

units(x) <- value

has.units(x)
```

Arguments

x	An object.
value	A `character` specifying the units

Functions

- `units(x) <- value`: Set units attribute.
- `has.units()`: Check for attribute.

Examples

```
x <- 1:10
units(x) <- "cm"
has.units(x)
units(x)
```

`update_html` *Update HTML.*

Description

Used to (re-)generate the HTML code for a `link{table1}` object. In most cases, this should not be used direction, unless you know what you are doing.

Usage

```
update_html(x)
```

Arguments

`x`	An object returned by `table1`.

Value

An object of class "table1" which contains the updated HTML.

Index