Package 'ggquickeda'

April 25, 2018

Title Quickly Explore Your Data Using 'ggplot2' and Summary Tables

Version 0.1.0

Description Quickly and easily perform exploratory data analysis by uploading your data as a 'csv' file. Start generating insights using 'ggplot2' plots and tables with descriptive stats, all using an easy-to-use point and click 'Shiny' interface.

URL https://github.com/smouksassi/ggquickeda

BugReports https://github.com/smouksassi/ggquickeda/issues

Depends R (>= 3.1.0)

Imports colourpicker, dplyr, DT, Formula, ggplot2, ggrepel (>= 0.7.0), grDevices, grid, gridExtra, Hmisc, lazyeval, markdown, methods, plotly, quantreg, rlang, scales, shiny (>= 1.0.4), shinyjs, stats, stringr, survival, tidyr, utils

Suggests knitr, rmarkdown

License MIT + file LICENSE

SystemRequirements pandoc with https support

LazyData true

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Samer Mouksassi [aut, cre], Dean Attali [aut], Michael Sachs [aut] (provided ggkm code), Benjamin Rich [aut] (provided table1 code)

Maintainer Samer Mouksassi <samermouksassi@gmail.com>

Repository CRAN

Date/Publication 2018-04-25 10:15:05 UTC

R topics documented:

`label` *Label attribute.*

Description

Label attribute.

Usage

```
label(x)

label(x) <- value

has.label(x)
```

Arguments

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

Functions

- `label<-`: Set label attribute.
- `has.label`: Check for label attribute.

Examples

```
x <- 1:10
label(x) <- "Foo"
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, 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)
```

`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, ...)
```

Arguments

`x`	A vector of type `factor`, `character` or `logical`.
`...`	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

```
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.
...	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)
```

run_ggquickeda *Run the ggquickeda application*

Description

Run the ggquickeda application.

Usage

```
run_ggquickeda(data = NULL)
```

Arguments

data	The initial data.frame to load into the application.

Examples

```
if (interactive()) {
  run_ggquickeda()
}
```

sample_data *Simulated Pharmacokinetic Concentration Data*

Description

A dataset containing concentration-time data with the given dose and some subject characteristics to help in the app exploration.

Usage

```
sample_data
```

Format

A data frame with 600 rows and 10 variables

ID Subject Identifier, an integer from 1 to 150

Time Time of dose given or drug sample measured, in hours

Amt dose given at the corresponding Time, in milligrams

Conc drug concentrations in the plasma sample, in mg/L

Age age of the subject, in years

Weight weight of the subject, in kg

Gender Sex of the subject, a factor with Female and Male levels

Race Race of the subject, a factor with Asian, Black, Caucasian, Hispanic and Other levels

Dose dose group of the subject, in milligrams

AGECAT age category of the subject, a variable cutting Age into two values 0/1

Source

"sd_oral_richpk" from 'PKPDmisc' R package with an additional AGECAT variable

---

`signif_pad` *Round numbers to specified significant digits with 0-padding.*

Description

A utility function to round numbers to a specified number of significant digits. Zeros are kept if they are significant.

Usage

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

Arguments

`x`	A numeric vector.
`digits`	An integer specifying the number of significant digits to keep.
`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 software (e.g. SAS, Excel) always round up.

Value

A character vector containing the rounded numbers.

See Also

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

Examples

```
x <- c(0.9001, 12345, 1.2, 1., 0.1)
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 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,
  ...)
```

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 software (e.g. SAS, Excel) always round up.
`...`	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, useNA = NULL)
```

Arguments

`x`	A vector or numeric, factor, character or logical values.
`useNA`	For categorical `x`, should missing values be treated as a category?

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
- `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
- `MAX`: the maximum of the non-missing values
- `Q25`: the lower quartile of the non-missing values
- `Q75`: the upper quartile of the non-missing values
- `IQR`: the inter-quartile range 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`
- `GCV`: the percent geometric coefficient of variation of the non-missing values if non-negative, or `NA`

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

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

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(100, 1, 1))), 5, 5)
table.data(x)
cat(table.rows(x, NULL))
cat(table.rows(x, LETTERS[1:10]))
cat(table.rows(LETTERS[1:3], "Headings", th=TRUE))
```

table1 *Generate an HTML table of descriptive statistics.*

Description

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.

Usage

```
table1(x, ...)

## Default S3 method:
table1(x, labels, groupspan = NULL, rowlabelhead = "",
  transpose = FALSE, topclass = "Rtable1", render = render.default,
  standalone = !isTRUE(getOption("knitr.in.progress")), ...)

## S3 method for class 'formula'
table1(x, data, overall = "Overall", rowlabelhead = "",
  transpose = FALSE, droplevels = TRUE, topclass = "Rtable1",
  render = render.default,
  standalone = !isTRUE(getOption("knitr.in.progress")), ...)
```

Arguments

x	An object, typically a formula or list of data.frames.
...	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.
render	A function to render the table cells (see Details).
standalone	Should a standalone HTML document be generated and immediately displayed? Otherwise an HTML fragment is printed to stdout.
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.
droplevels	Should empty factor levels be dropped?

Details

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.

Value

None (invisible NULL). Called for its side effects.

Methods (by class)

- default: The default interface, where x is a data.frame.
- 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<-`: Set units attribute.
- `has.units`: Check for attribute.

Examples

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

Index