Package ‘qwraps2’

March 7, 2021

Title Quick Wraps 2

Version 0.5.2

Description A collection of (wrapper) functions the creator found useful for quickly placing data summaries and formatted regression results into '.Rnw' or '.Rmd' files. Functions for generating commonly used graphics, such as receiver operating curves or Bland-Altman plots, are also provided by 'qwraps2'. 'qwraps2' is a updated version of a package 'qwraps'. The original version 'qwraps' was never submitted to CRAN but can be found at <https://github.com/dewittpe/qwraps/>. The implementation and limited scope of the functions within 'qwraps2' <https://github.com/dewittpe/qwraps2/> is fundamentally different from 'qwraps'.

Depends R (>= 3.5.0)

License GPL-2

Encoding UTF-8

URL `https://github.com/dewittpe/qwraps2/`

Language en-us

LazyData true

Imports dplyr (>= 1.0.0), ggplot2, knitr, Rcpp (>= 0.12.11), rlang, utils, xfun

Suggests survival, testthat, covr, rbenchmark, rmarkdown

RoxygenNote 7.1.1

LinkingTo Rcpp (>= 0.12.11), RcppArmadillo

VignetteBuilder knitr

NeedsCompilation yes

Author Peter DeWitt [aut, cre] (<https://orcid.org/0000-0002-6391-0795>), Tell Bennett [ctb] (<https://orcid.org/0000-0003-1483-4236>)

Maintainer Peter DeWitt <`dewittpe@gmail.com`>

Repository CRAN

Date/Publication 2021-03-07 16:50:03 UTC

R topics documented:

`backtick` *Backtick*

Description

Encapsulate a string in backticks. Very helpful for in line code in knitr::spin scripts.

Usage

```
backtick(x, dequote = FALSE)
```

Arguments

`x`	the thing to be deparsed and encapsulated in backticks
`dequote`	remove the first and last double or signal quote form `x`

`check_comments` *Check Comments*

Description

A more robust check for open/close matching sets of comments in a spin file.

Usage

```
check_comments(c1, c2)
```

Arguments

`c1`	index (line numbers) for the start delimiter of comments
`c2`	index (line numbers) for the end delimiter of comments

`confusion_matrix` *Confusion Matrices (Contingency Tables)*

Description

Construction of confusion matrices, accuracy, sensitivity, specificity, confidence intervals (Wilson's method and (optional bootstrapping)).

Usage

```
confusion_matrix(x, ...)

## Default S3 method:
confusion_matrix(
  x,
  y,
  positive = NULL,
  boot = FALSE,
  boot_samples = 1000L,
  alpha = 0.05,
  ...
)

## S3 method for class 'formula'
confusion_matrix(
  formula,
  data = parent.frame(),
  positive = NULL,
  boot = FALSE,
  boot_samples = 1000L,
  alpha = 0.05,
  ...
)

is.confusion_matrix(x)

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

Arguments

`x`	prediction condition vector, a two level factor variable or a variable that can be converted to one.
`...`	not currently used
`y`	True Condition vector with the same possible values as `x`.
`positive`	the level of `x` and `y` which is the positive outcome. If `NULL` the first level of `factor(y)` will be used as the positive level.
`boot`	boolean, should bootstrapped confidence intervals for the sensitivity and specificity be computed? Defaults to FALSE.
`boot_samples`	number of bootstrapping sample to generate, defaults to 1000L. Ignored if `boot == FALSE`.
`alpha`	100(1-alpha) sensitivity. Ignored if `boot == FALSE`.
`formula`	column (known) ~ row (test) for building the confusion matrix
`data`	environment containing the variables listed in the formula

Details

Sensitivity and Specificity: For the sensitivity and specificity function we expect the 2-by-2 confusion matrix (contingency table) to be of the form:

		True	Condition
		+	-
Predicted Condition	+	TP	FP
Predicted Condition	-	FN	TN

where

- FN: False Negative, and
- FP: False Positive,
- TN: True Negative,

- TP: True Positive.

The statistics returned in the `stats` element are:

- accuracy = (TP + TN) / (TP + TN + FP + FN)
- sensitivity = TP / (TP + FN)
- specificity = TN / (TN + FP)
- positive predictive value (PPV) = TP / (TP + FP)
- negative predictive value (NPV) = TN / (TN + FN)
- false negative rate (FNR) = 1 - Sensitivity
- false positive rate (FPR) = 1 - Specificity
- false discovery rate (FDR) = 1 - PPV
- false omission rate (FOR) = 1 - NPV
- F1 score
- Matthews Correlation Coefficient (MCC) = ((TP * TN) - (FP * FN)) / sqrt((TP + FP) (TP+FN) (TN+FP) (TN+FN))

Synonyms for the statistics:

- Sensitivity: true positive rate (TPR), recall, hit rate
- Specificity: true negative rate (TNR), selectivity
- PPV: precision
- FNR: miss rate

Sensitivity and PPV could, in some cases, be indeterminate due to division by zero. To address this we will use the following rule based on the DICE group `https://github.com/dice-group/gerbil/wiki/Precision,-Recall-and-F1-measure`: If TP, FP, and FN are all 0, then PPV, sensitivity, and F1 will be defined to be 1. If TP are 0 and FP + FN > 0, then PPV, sensitivity, and F1 are all defined to be 0.

Value

The sensitivity and specificity functions return numeric values. `confusion_matrix` returns a list with elements:

- `tab` the confusion matrix,
- `cells`
- `stats` a matrix of summary statistics and confidence intervals.

Examples

```
################################################################################
## Example 1
test  <- c(rep(1, 53), rep(0, 47))
truth <- c(rep(1, 20), rep(0, 33), rep(1, 10), rep(0, 37))
con_mat <- confusion_matrix(x = test, y = truth, positive = "1")
str(con_mat)
```

```
con_mat

con_mat$cells$true_positives  # 20
con_mat$cells$true_negatives  # 37
con_mat$cells$false_positives # 33
con_mat$cells$false_negatives # 10

con_mat_with_boot <- confusion_matrix(test, truth, positive = "1", boot = TRUE)
con_mat_with_boot

# only one value in one of the vectors
a <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # all zeros
b <- c(1,0,1,0,1,0,0,0,0,0,0,0,0,1)  # some zeros and ones

confusion_matrix(a, b)
confusion_matrix(b, a)
confusion_matrix(a, b, positive = 1)
confusion_matrix(b, a, positive = 1)


################################################################################
## Example 2: based on an example from the wikipedia page:
# https://en.wikipedia.org/wiki/Confusion_matrix

animals <-
  data.frame(Predicted = c(rep("Cat",    5 + 2 +  0),
                           rep("Dog",    3 + 3 +  2),
                           rep("Rabbit", 0 + 1 + 11)),
             Actual    = c(rep(c("Cat", "Dog", "Rabbit"), times = c(5, 2,  0)),
                           rep(c("Cat", "Dog", "Rabbit"), times = c(3, 3,  2)),
                           rep(c("Cat", "Dog", "Rabbit"), times = c(0, 1, 11))),
             stringsAsFactors = FALSE)

table(animals)

cats <- apply(animals, 1:2, function(x) ifelse(x == "Cat", "Cat", "Non-Cat"))

# Default calls, note the difference based on what is set as the 'positive'
# value.
confusion_matrix(cats[, "Predicted"], cats[, "Actual"], positive = "Cat")
confusion_matrix(cats[, "Predicted"], cats[, "Actual"], positive = "Non-Cat")

# Using a Formula
confusion_matrix(formula = I(Actual == "Cat") ~ I(Predicted == "Cat"),
                 data = animals,
                 positive = "TRUE")

confusion_matrix(formula = I(Actual == "Cat") ~ I(Predicted == "Cat"),
                 data = animals,
                 positive = "TRUE",
                 boot = TRUE)
```

```
################################################################################
## Example 3
russell <-
  data.frame(Pred  = c(rep(0, 2295), rep(0, 118), rep(1, 1529), rep(1, 229)),
             Truth = c(rep(0, 2295), rep(1, 118), rep(0, 1529), rep(1, 229)))

# The values for Sensitivity, Specificity, PPV, and NPV are dependent on the
# "positive" level.  By default, the first level of y is used.
confusion_matrix(x = russell$Pred, y = russell$Truth, positive = "0")
confusion_matrix(x = russell$Pred, y = russell$Truth, positive = "1")

confusion_matrix(Truth ~ Pred, data = russell, positive = "0")
confusion_matrix(Truth ~ Pred, data = russell, positive = "1")
```

---

extract_fstat *Extract Summary stats from regression objects*

Description

A collection of functions for extracting summary statistics and reporting regression results from `lm`, `glm` and other regression objects.

Usage

```
extract_fstat(x)

extract_fpvalue(x)

## S3 method for class 'lm'
extract_fpvalue(x)
```

Arguments

x	a `lm` object

Value

a character vector of the formatted numbers

formatted p-value from the F-test

See Also

`lm`

Examples

```
fit <- lm(mpg ~ wt + hp + drat, data = mtcars)
summary(fit)
extract_fstat(fit)
extract_fpvalue(fit)
```

`file_check` *File and Working Directory Check*

Description

This check is three-fold: 1) verify the current working directory is as expected, 2) verify the user can access the file, and 3) verify the file contents are as expected (via md5sum).

Usage

```
file_check(paths, md5sums = NULL, stop = FALSE)
```

Arguments

`paths`	a character path to the target file
`md5sums`	a character string for the expected md5sum of the target file. If `NULL` then only a `file.exists` check will be done.
`stop`	if `TRUE` then an error is thrown if any of the checks fail. If `FALSE` (default) a logical is returned.

Details

The test for the file access is done to verify the file can be read by the current user.

The return of the function is `TRUE` if all the files in `paths` are accessible and all of requested md5sum checks pass. `FALSE` is any file is not accessible or any md5sum check fails. By default, if the return is `TRUE` then only `TRUE` will be printed to the console. If the return is `FALSE` then the `attr(,"checks")` is printed by default as well.

Good practice would be to use relative paths, a warning will be given if any of the `paths` are determined to be absolute paths.

Value

The function will return a single TRUE/FALSE value with attributes `attr(,"checks")`.

Examples

```
# create two example files in the working directory:
cat("example file.", file = "QWRAPS2_EXAMPLE_1.txt")
cat("Another example file.", file = "QWRAPS2_EXAMPLE_2.txt")

# Check that you have access to these files:  (Should return TRUE)
test1 <- file_check(c("QWRAPS2_EXAMPLE_1.txt", "QWRAPS2_EXAMPLE_2.txt"))
test1

# By default, when the checks return TRUE the details of the checks are not
# printed.  You can view the details of the checks as follows:
attr(test1, "checks")

# If one or more files is not accessable then return is FALSE and the meta data
# is printed by default.
test2 <- file_check(c("UNLIKELYFILENAME", "QWRAPS2_EXAMPLE_1.txt", "QWRAPS2_EXAMPLE_2.txt"))
test2

# Or have an error thrown:
## Not run:
file_check(c("UNLIKELYFILENAME", "QWRAPS2_EXAMPLE_1.txt", "QWRAPS2_EXAMPLE_2.txt"),
           stop = TRUE)

## End(Not run)

# Verify the md5sums as well as file access:
file_check("QWRAPS2_EXAMPLE_1.txt", "7a3409e17f9de067740e64448a86e708")

# If you only need to verify a subset of md5sums then use an NA in the md5sums
# argument:
file_check(c("QWRAPS2_EXAMPLE_1.txt", "QWRAPS2_EXAMPLE_2.txt"),
           c("7a3409e17f9de067740e64448a86e708", NA))

# Verify all the md5sums
file_check(c("QWRAPS2_EXAMPLE_1.txt", "QWRAPS2_EXAMPLE_2.txt"),
           c("7a3409e17f9de067740e64448a86e708", "798e52b92e0ae0e60f3f3db1273235d0"))


# clean up working directory
unlink("QWRAPS2_EXAMPLE_1.txt")
unlink("QWRAPS2_EXAMPLE_2.txt")
```

`frmt` *Format Wrappers*

Description

Functions for formatting numeric values for consistent display in reports.

Usage

```
frmt(x, digits = getOption("qwraps2_frmt_digits", 2))

frmtp(
  x,
  style = getOption("qwraps2_journal", "default"),
  digits = getOption("qwraps2_frmtp_digits", 4),
  markup = getOption("qwraps2_markup", "latex"),
  case = getOption("qwraps2_frmtp_case", "upper"),
  leading0 = getOption("qwraps2_frmtp_leading0", TRUE)
)

frmtci(
  x,
  est = 1,
  lcl = 2,
  ucl = 3,
  format = "est (lcl, ucl)",
  show_level = FALSE,
  ...
)
```

Arguments

`x`	a vector of numbers or a numeric matrix to format.
`digits`	number of digits, including trailing zeros, to the right of the decimal point. This option is ignored if `is.integer(x) == TRUE)`.
`style`	a character string indicating a specific journal requirements for p-value formatting.
`markup`	a character string indicating if the output should be latex or markup.
`case`	a character string indicating if the output should be upper case or lower case.
`leading0`	boolean, whether or not the p-value should be reported as 0.0123 (TRUE, default), or .0123 (FALSE).
`est`	the numeric index of the vector element or the matrix column containing the point estimate.
`lcl`	the numeric index of the vector element or the matrix column containing the lower confidence limit.
`ucl`	the numeric index of the vector element or the matrix column containing the upper confidence limit.
`format`	a string with "est" "lcl", and "ucl" to denote the location of the estimate, lower confidence limit, and upper confidence limit for the formatted string. Defaults to "est (lcl, ucl)".
`show_level`	defaults to FALSE. If TRUE and `format` is the default, then "100*(1-options()$qwraps2_alpha) parenthesis and the lcl. If set to a string, then the given string will be placed between the left parenthesis and the lcl. If the `format` is not the default, then this argument is ignored.

...	args passed to frmt

Details

'frmt' is really just a wrapper for the `formatC`.

'frmtp' formats P-values per journal requirements. As I work on papers aimed at different journals, the formatting functions will be extended to match.

Default settings are controlled through the function arguments but should be set via `options()`.

Default settings report the P-value exactly if P > `getOptions("qwraps2_frmtp_digits",4)` and reports P < `10^-(getOptions("qwraps2_frmtp_digits",2))` otherwise. By the leading zero is controlled via `getOptions("qwraps2_frmtp_leading0",TRUE)` and a upper or lower case P is controlled by `getOptions("qwraps2_frmtp_case","upper")`. These options are ignored if `style != "default"`.

Journals with predefined P-value formatting are noted in the **qwraps2** documentation.

'frmtci' takes a `matrix`, or `data.frame`, with a point estimate and the lcl and ucl and formats a string for reporting. est (lcl, ucl) is the default. The confidence level can be added to the string, e.g., "est (95 format.

'frmtcip' expects four values, est, lcl, ucl, and p-value. The resulting sting will be of the form "est (lcl, ucl; p-value)".

The 'Rpkg', 'CRANpkg', and 'Githubpkg' functions are used to help make documenting packages stylistically consistent and with valid urls. These functions were inspired by similar ones found in the BioConductor BiocStyle package.

Value

a character vector of the formatted numbers

See Also

`formatC`

Examples

```
# Formatting numbers
integers <- c(1234L, 9861230L)
numbers  <- c(1234,  9861230)
frmt(integers)  # no decimal point
frmt(numbers)   # decimal point and zeros to the right

numbers <- c(0.1234, 0.1, 1234.4321, 0.365, 0.375)
frmt(numbers)

# Formatting p-values
ps <- c(0.2, 0.001, 0.00092, 0.047, 0.034781, 0.0000872, 0.787, 0.05, 0.043)
# LaTeX is the default markup language
cbind("raw"      = ps,
      "default"  = frmtp(ps),
      "3lower"   = frmtp(ps, digits = 3, case = "lower"),
      "PediDent" = frmtp(ps, style = "pediatric_dentistry"))

# Using markdown
cbind("raw"      = ps,
      "default"  = frmtp(ps, markup = "markdown"),
      "3lower"   = frmtp(ps, digits = 3, case = "lower", markup = "markdown"),
      "PediDent" = frmtp(ps, style = "pediatric_dentistry", markup = "markdown"))

# Formatting the point estimate and confidence interval
# for a set of three values
temp <- c(a = 1.23, b = .32, CC = 1.78)
frmtci(temp)
frmtci(temp, show_level = TRUE)

# note that the show_level will be ignored in the following
frmtci(temp, format = "est ***lcl, ucl***", show_level = TRUE)

# show_level as a character
frmtci(temp, show_level = "confidence between: ")

# For a matrix: the numbers in this example don't mean anything, but the
# formatting should.
temp2 <- matrix(rnorm(12), nrow = 4,
                dimnames = list(c("A", "B", "C", "D"), c("EST", "LOW", "HIGH")))
temp2
frmtci(temp2)
```

---

geometric_mean_var_sd *Geometric Mean, Variance, and Standard Deviation*

Description

Return the geometric mean, variance, and standard deviation,

Usage

```
gmean(x, na_rm = FALSE)

gvar(x, na_rm = FALSE)

gsd(x, na_rm = FALSE)
```

Arguments

x	a numeric vector
na_rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

Value

a numeric value

See Also

gmean_sd for easy formatting of the geometric mean and standard deviation. `vignette("summary-statistics",package = "qwraps2")`.

Examples

```
set.seed(42)
x <- runif(14, min = 4, max = 70)

# geometric mean - four equivalent ways to get the same result
prod(x) ^ (1 / length(x))
exp(mean(log(x)))
1.2 ^ mean(log(x, base = 1.2))
gmean(x)

# geometric variance
gvar(x)

# geometric sd
exp(sd(log(x)))                                  ## This is wrong (incorrect sample size)
exp(sqrt((length(x) - 1) / length(x)) * sd(log(x))) ## Correct calculation
gsd(x)

# Missing data will result in and NA being returned
x[c(2, 4, 7)] <- NA
gmean(x)
gmean(x, na_rm = TRUE)
gvar(x, na_rm = TRUE)
gsd(x, na_rm = TRUE)
```

`ggplot2_extract_legend` *ggplot2 tools*

Description

A few handy tools for working with ggplot2.

Usage

```
ggplot2_extract_legend(x)
```

Arguments

`x`	a ggplot

Details

The `ggplot2_extract_legend` function returns a list with the first element being the legend and the second the original plot with the legend omitted.

Value

a list with each element a ggplot

Examples

```
# a simple plot
my_plot <-
  ggplot2::ggplot(mtcars) +
  ggplot2::aes(x = wt, y = mpg, color = wt, shape = factor(cyl)) +
  ggplot2::geom_point()

my_plot

# extract the legend.  the return object is a list with two elements, the first
# element is the legend, the second is the original plot sans legend.
temp <- ggplot2_extract_legend(my_plot)

# view just the legend.  This can be done via a call to the object or using
# plot or print.
temp
plot(temp[[1]])

# the original plot without the legened
plot(temp[[2]])
```

`lazyload_cache_dir` *Lazyload Cache*

Description

Lazyload Cached label(s) or a whole directory.

Usage

```
lazyload_cache_dir(
  path = "./cache",
  envir = parent.frame(),
  ask = FALSE,
  verbose = TRUE,
  full.names = TRUE,
  ...
)

lazyload_cache_labels(
  labels,
  path = "./cache/",
  envir = parent.frame(),
  verbose = TRUE,
  filter,
  full.names = TRUE,
  ...
)
```

Arguments

`path`	the path to the cache directory.
`envir`	the environment to load the objects into
`ask`	if TRUE ask the user to confirm loading each database found in `path`
`verbose`	if TRUE display the chunk labels being loaded
`full.names`	use the full name, i.e., include the path, for the chunk label? This argument is passed to `list.files`.
`...`	additional arguments passed to `list.files`.
`labels`	a character vector of the chunk labels to load.
`filter`	an optional function passed to `lazyLoad`. when called on a character vector of object names returns a logical vector: only objects for which this is true will be loaded.

Details

These functions helpful for loading cached chunks into an interactive R session. Consider the following scenario: you use knitr and have cached chunks for lazyloading. You've created the document, close up your IDE and move on to the next project. Later, you revisit the initial project and need to retrieve the objects created in the cached chunks. One option is to reevaluate all the code, but this could be time consuming. The other option is to use `lazyload_cache_labels` or `lazyload_cache_dir` to quickly (lazy)load the chunks into an active R session.

Use `lazyload_cache_dir` to load a whole directory of cached objects.

Use `lazyload_cache_labels` to load and explicit set of cached chunks.

Examples

```
# this example is based on \url{https://stackoverflow.com/a/41439691/1104685}

# create a temp directory for a and place a .Rmd file within
tmpdr <- paste0(tempdir(), "/llcache_eg")
dir.create(tmpdr)
old_wd <- getwd()
```

```
setwd(tmpdr)

cat("---",
    "title: \"A Report\"",
    "output: html_document",
    "---",
    "",
    "```{r first-chunk, cache = TRUE}",
    "mpg_by_wt_hp <- lm(mpg ~ wt + hp, data = mtcars)",
    "x_is_pi <- pi",
    "```",
    "",
    "```{r second-chunk, cache = TRUE}",
    "mpg_by_wt_hp_am <- lm(mpg ~ wt + hp + am, data = mtcars)",
    "x_is_e <- exp(1)",
    "```",
    sep = "\n",
    file = paste0(tmpdr, "/report.Rmd"))

# knit the file.  evaluate the chuncks in a new environment so we can compare
# the objects after loading the cache.
kenv <- new.env()
knitr::knit(input = paste0(tmpdr, "/report.Rmd"), envir = kenv)

# The objects defined in the .Rmd file are now in kenv
ls(envir = kenv)

# view the cache
list.files(path = tmpdr, recursive = TRUE)

# create another environment, and load only the second chunk
lenv <- new.env()
ls(envir = lenv)

lazyload_cache_labels(labels = "second-chunk",
                      path = paste0(tmpdr, "/cache"),
                      envir = lenv)
lenv$x_is_e
lenv$mpg_by_wt_hp_am

# load all the chuncks
menv <- new.env()
lazyload_cache_dir(path = paste0(tmpdr, "/cache"), envir = menv)

ls(envir = menv)
menv$x_is_pi
menv$x_is_e

# cleanup
setwd(old_wd)
unlink(tmpdr, recursive = TRUE)
```

ll *List Object Aliases*

Description

Aliases for `ls` providing additional details.

Usage

```
ll(pos = 1, pattern, order_by = "Size", decreasing = TRUE, head = FALSE, n = 5)
```

Arguments

`pos`	specifies the environment as a position in the search list
`pattern`	an optional regular expression. Only names matching `pattern` are returned. `glob2rx` can be used to convert wildcard patterns to regular expressions.
`order_by`	a character, order the results by "Size" (default), "Type", "Rows", or "Columns".
`decreasing`	logical, defaults to `TRUE`, decreasing order? passed to `order`.
`head`	logical, if `TRUE` then only return the first `n` objects per `order_by` and `decreasing`.
`n`	number of rows to return, ignored if head = `FALSE`.

References

The basis for this work came from a Stack Overflow posting: https://stackoverflow.com/q/1358003/1104685

See Also

`ls`

Examples

```
# View your current workspace
## Not run:
ls()
ll()

## End(Not run)

# View another environment
e <- new.env()
ll(e)

e$fit <- lm(mpg ~ wt, mtcars)
e$fit2 <- lm(mpg ~ wt + am + vs, data = mtcars)
e$x <- rnorm(1e5)
e$y <- runif(1e4)
```

```
e$z <- with(e, x * y)
e$w <- sum(e$z)

ls(e)
ll(e)
ll(e, head = TRUE)
```

logit — *logit and inverse logit functions*

Description

transform `x` either via the logit, or inverse logit.

Usage

```
logit(x)

invlogit(x)
```

Arguments

`x`	a numeric vector

Details

The logit and inverse logit functions are part of R via the logistic distribution functions in the stats package. Quoting from the documentation for the logistic distribution

"`qlogis(p)` is the same as the `logit` function, `logit(p) = log(p/1-p)`, and `plogis(x)` has consequently been called the 'inverse logit'."

See the examples for benchmarking these functions. The `logit` and `invlogit` functions are faster than the `qlogis` and `plogis` functions.

See Also

`qlogis`

Examples

```
library(rbenchmark)

# compare logit to qlogis
p <- runif(1e5)
identical(logit(p), qlogis(p))

## Not run:
rbenchmark::benchmark(logit(p), qlogis(p))
```

```
## End(Not run)

# compare invlogit to plogis
x <- runif(1e5, -1000, 1000)
identical(invlogit(x), plogis(x))

## Not run:
rbenchmark::benchmark(invlogit(x), plogis(x))

## End(Not run)
```

`mean_ci` *Means and Confidence Intervals*

Description

A function for calculating and formatting means and confidence interval.

Usage

```
mean_ci(
  x,
  na_rm = FALSE,
  alpha = getOption("qwraps2_alpha", 0.05),
  qdist = stats::qnorm,
  qdist.args = list(),
  ...
)

## S3 method for class 'qwraps2_mean_ci'
print(x, ...)
```

Arguments

`x`	a numeric vector
`na_rm`	if true, omit NA values
`alpha`	defaults to `getOption('qwraps2_alpha',0.05)`. The symmetric 100(1-alpha)% CI will be determined.
`qdist`	defaults to `qnorm`. use `qt` for a Student t intervals.
`qdist.args`	list of arguments passed to `qdist`
`...`	arguments passed to `frmtci`.

Details

Given a numeric vector, `mean_ci` will return a vector with the mean, LCL, and UCL. Using `frmtci` will be helpful for reporting the results in print.

Value

a vector with the mean, lower confidence limit (LCL), and the upper confidence limit (UCL).

See Also

frmtci

Examples

```
# using the standard normal for the CI
mean_ci(mtcars$mpg)

# print it nicely
qwraps2::frmtci(mean_ci(mtcars$mpg))
qwraps2::frmtci(mean_ci(mtcars$mpg), show_level = TRUE)
qwraps2::frmtci(mean_ci(mtcars$mpg, alpha = 0.01), show_level = TRUE)

# Compare to the ci that comes form t.test
t.test(mtcars$mpg)
t.test(mtcars$mpg)$conf.int
mean_ci(mtcars$mpg, qdist = stats::qt, qdist.args = list(df = 31))
```

---

mean_sd *Mean and Standard deviation*

Description

A function for calculating and formatting means and standard deviations.

Usage

```
mean_sd(
  x,
  digits = getOption("qwraps2_frmt_digits", 2),
  na_rm = FALSE,
  show_n = "ifNA",
  denote_sd = "pm",
  markup = getOption("qwraps2_markup", "latex")
)

gmean_sd(
  x,
  digits = getOption("qwraps2_frmt_digits", 2),
  na_rm = FALSE,
  show_n = "ifNA",
  denote_sd = "pm",
  markup = getOption("qwraps2_markup", "latex")
)
```

Arguments

`x`	a numeric vector
`digits`	digits to the right of the decimal point to return in the percentage estimate.
`na_rm`	if true, omit NA values
`show_n`	defaults to "ifNA". Other options are "always" or "never".
`denote_sd`	a character string set to either "pm" or "paren" for reporting 'mean $\pm$ sd' or 'mean (sd)'
`markup`	latex or markdown

Details

Given a numeric vector, `mean_sd` will return a character string with the mean and standard deviation. Formatting of the output will be extended in future versions.

`gmean_sd` returns the geometric mean and geometric standard deviation.

Value

a character vector of the formatted values

Examples

```
set.seed(42)
x <- rnorm(1000, 3, 4)
mean(x)
sd(x)
mean_sd(x)
mean_sd(x, show_n = "always")
mean_sd(x, show_n = "always", denote_sd = "paren")

x[187] <- NA
mean_sd(x, na_rm = TRUE)
```

`median_iqr` *Median and Inner Quartile Range*

Description

A function for calculating and formatting the median and inner quartile range of a data vector.

Usage

```
median_iqr(
  x,
  digits = getOption("qwraps2_frmt_digits", 2),
  na_rm = FALSE,
  show_n = "ifNA",
  markup = getOption("qwraps2_markup", "latex")
)
```

Arguments

`x`	a numeric vector
`digits`	digits to the right of the decimal point to return.
`na_rm`	if true, omit NA values
`show_n`	defaults to "ifNA". Other options are "always" or "never".
`markup`	latex or markdown

Details

Given a numeric vector, `median_iqr` will return a character string with the median and IQR. Formatting of the output will be extended in future versions.

Value

a character vector of the formatted values

Examples

```
set.seed(42)
x <- rnorm(1000, 3, 4)
median(x)
quantile(x, probs = c(1, 3)/4)
median_iqr(x)
median_iqr(x, show_n = "always")

x[187] <- NA
# median_iqr(x) ## Will error
median_iqr(x, na_rm = TRUE)
```

mtcars2 *mtcars2*

Description

An extended version of `mtcars` data set.

Usage

```
mtcars2
```

Format

An object of class `data.frame` with 32 rows and 19 columns.

See Also

`vignette("qwraps2-data-sets",package = "qwraps2")` for details on the construction of the data set.

---

n_perc *Count and Percentage*

Description

A function for calculating and formatting counts and percentages.

Usage

```
n_perc(
  x,
  digits = getOption("qwraps2_frmt_digits", 2),
  na_rm = FALSE,
  show_denom = "ifNA",
  show_symbol = TRUE,
  markup = getOption("qwraps2_markup", "latex")
)

perc_n(
  x,
  digits = getOption("qwraps2_frmt_digits", 2),
  na_rm = FALSE,
  show_denom = "ifNA",
  markup = getOption("qwraps2_markup", "latex")
)

n_perc0(
  x,
  digits = 0,
  na_rm = FALSE,
  show_denom = "never",
  show_symbol = FALSE,
  markup = getOption("qwraps2_markup", "latex")
)
```

Arguments

`x`	a 0:1 or boolean vector
`digits`	digits to the right of the decimal point to return in the percentage estimate.
`na_rm`	if true, omit NA values
`show_denom`	defaults to "ifNA". Other options are "always" or "never".
`show_symbol`	if TRUE (default) the percent symbol is shown, else it is suppressed.
`markup`	latex or markdown

Details

Default behavior will return the count of successes and the percentage as "N (pp can be controlled by setting `na.rm = TRUE`. In this case, the number of non-missing values will be reported by default. Omission of the non-missing values can be controlled by setting `show_denom = "never"`.

The function n_perc0 uses a set of default arguments which may be advantageous for use in building tables.

Value

a character vector of the formatted values

Examples

```
n_perc(c(0, 1,1, 1, 0, 0), show_denom = "always")
n_perc(c(0, 1,1, 1, 0, 0, NA), na_rm = TRUE)

n_perc(mtcars$cyl == 6)

set.seed(42)
x <- rbinom(4269, 1, 0.314)
n_perc(x)
n_perc(x, show_denom = "always")
n_perc(x, show_symbol = FALSE)

# n_perc0 examples
n_perc0(c(0, 1,1, 1, 0, 0))
n_perc0(mtcars$cyl == 6)
```

`pefr` *pefr*

Description

Peak expiratory flow rate data

Usage

```
pefr
```

Format

a data frame with four columns

[, 1] subject id number

[, 2] measurement first or second

[, 3] meter "Wright peak flow meter" or "Mini Write peak flow meter"

[, 4] pefr peak expiratory flow rate (liters / min)

Details

Peak expiratory flow rate (pefr) data is used for examples within the qwraps2 package. The data has been transcribed from Bland (1986).

"The sample comprised colleagues and family of J.M.B. chosen to give a wide range of PEFR but in no way representative of any defined population. Two measurements were made with a Wright peak flow meter and two with a mini Wright meter, in random order. All measurements were taken by J.M.B., using the same two instruments. (These data were collected to demonstrate the statistical method and provide no evidence on the comparability of these two instruments.) We did not repeat suspect readings and took a single reading as our measurement of PEFR. Only the first measurement by each method is used to illustrate the comparison of methods, the second measurements being used in the study of repeatability."

References

Bland, J. Martin, and Douglas G Altman. "Statistical methods for assessing agreement between two methods of clinical measurement." The lancet 327, no. 8476 (1986): 307-310.

See Also

`vignette("qwraps2-data-sets",package = "qwraps2")` for details on the construction of the data set.

`pkg_check` *Package Checks*

Description

Check if a package is available on the local machine and optionally verify a version.

Usage

```
pkg_check(pkgs, versions, stop = FALSE)
```

Arguments

pkgs	a character vector of package names to check for
versions	an optional character vector, of the same length of pkgs for the minimum version of the packages.
stop	if TRUE then an error is thrown if any of the checks fail. If FALSE (default) a logical is returned.

Details

When writing a script that will be shared it is very likely that the multiple authors/users will need to have a certain set of packages available to load. The pkg_check function will verify that the packages are available to load, this includes an optional version test, and attach the package to the search list if requested.

Testing for package versions will is done as `packageVersion(x) >= version`. If you need a specific version of a package you should explicitly use `packageVersion(x) == version` in your script.

Examples

```
# verify that the packages qwraps2, ggplot2, and BH are available
pkg_check(c("qwraps2", "ggplot2", "BH"))

# show that the return is FALSE if a package is not available
pkg_check(c("qwraps2", "ggplot2", "BH", "NOT a PCKG"))

# verify the version for just ggplot2
pkg_check(c("qwraps2", "ggplot2", "BH"),
          c(NA, "2.2.0", NA))

# verify the version for qwraps2 (this is expected to fail as we are looking for
# version 42.3.14 which is far too advanced for the actual package development.
pkg_check(c("qwraps2", "ggplot2", "BH"),
          c("42.3.14", "2.2.0", NA))


## Not run:
  # You can have the function throw an error is any of the checks fail
  pkg_check(c("qwraps2", "ggplot2", "BH"),
            c("42.3.14", "2.2.0", NA),
            stop = TRUE)

## End(Not run)

## Not run:
  # If you have missing packages that can be installed from CRAN you may find
  # the following helpful.  If this code, with the needed edits, were placed at
  # the top of a script, then if a package is missing then the current version
  # from a target repository will be installed.  Use this set up with
  # discretion, others may not want the automatic install of packages.
  pkgs <- pkg_check("<packages to install>")
  if (!pkgs) {
    install.packages(attr(pkgs, "checks")[!attr(pkgs, "checks")$available][["package"]])
  }

## End(Not run)
```

---

qable *Qable: an extended version of knitr::kable*

Description

Create a simple table via kable with row groups and rownames similar to those of `hmisc::latex` or `htmlTable::htmlTable`.

Usage

```
qable(
  x,
  rtitle,
  rgroup,
  rnames = rownames(x),
  cnames = colnames(x),
  markup = getOption("qwraps2_markup", "latex"),
  ...
)
```

Arguments

x	`matrix` or `data.frame` to be turned into a qable
rtitle	a row grouping title. See Details.
rgroup	a named numeric vector with the name of the row group and the number of rows within the group. `sum(rowgroup) == nrow(x)`.
rnames	a character vector of the row names
cnames	column names
markup	the markup language to use, passed to the `format` argument of `knitr::kable`.
...	additional arguments passed to `knitr::kable`

Details

`qable` is used as the printing method for `qwraps2_summary_table` objects. Check the vignettes for examples on building data summary tables.

`rtitle` can be used to add a title to the column constructed by the `rgroup` and `rnames`. The basic layout of a table generated by `qable` is:

rtitle	cnames[1]	cnames[2]

rgroup[1]		
rnames[1]	x[1, 1]	x[1, 2]
rnames[2]	x[2, 1]	x[2, 2]
rnames[3]	x[3, 1]	x[3, 2]
rgroup[2]		
rnames[4]	x[4, 1]	x[4, 1]
rnames[5]	x[5, 1]	x[5, 1]

It should be noted that `escape = !(markup == "latex")` is passed to `kable`.

Value

a character vector of the formatted numbers

See Also

`kable`

`summary_table`, for an example of build a data summary table, i.e., a "Table 1".

For more detail on arguments you can pass to `...` look at the non-exported functions form the knitr package `knitr:::kable_latex`, `knitr:::kable_markdown`, or others.

Examples

```
data(mtcars)
qable(mtcars)
qable(mtcars, markup = "markdown")

# by make
make <- sub("^(\\w+)\\s?(.*)$", "\\1", rownames(mtcars))
make <- c(table(make))

# A LaTeX table with a vertical bar between each column
qable(mtcars[sort(rownames(mtcars)), ], rgroup = make)

# A LaTeX table with no vertical bars between columns
qable(mtcars[sort(rownames(mtcars)), ], rgroup = make, vline = "")

# a markdown table
qable(mtcars[sort(rownames(mtcars)), ], rgroup = make, markup = "markdown")

# define your own column names
qable(mtcars[sort(rownames(mtcars)), ],
      rgroup = make,
      cnames = toupper(colnames(mtcars)),
      markup = "markdown")

# define your own column names and add a title
qable(mtcars[sort(rownames(mtcars)), ],
      rtitle = "Make & Model",
      rgroup = make,
      cnames = toupper(colnames(mtcars)),
      markup = "markdown")
```

qacf *Autocorrelation plot*

Description

ggplot2 style autocorrelation plot

Usage

```
qacf(x, conf_level = 0.95, show_sig = FALSE, ...)
```

Arguments

`x`	object
`conf_level`	confidence level for determining 'significant' correlations.
`show_sig`	logical, highlight significant correlations.
`...`	Other arguments passed to stats::acf

Value

a ggplot.

See Also

`vignette("qwraps2-graphics",pacakge = "qwraps2")`

Examples

```
# Generate a random data set
set.seed(42)
n <- 250
x1 <- x2 <- x3 <- x4 <- vector('numeric', length = n)
x1[1] <- runif(1)
x2[1] <- runif(1)
x3[1] <- runif(1)
x4[1] <- runif(1)

# white noise
Z.1 <- rnorm(n, 0, 1)
Z.2 <- rnorm(n, 0, 2)
Z.3 <- rnorm(n, 0, 5)

for(i in 2:n)
```

```
{
  x1[i] <- x1[i-1] + Z.1[i] - Z.1[i-1] + x4[i-1] - x2[i-1]
  x2[i] <- x2[i-1] - 2 * Z.2[i] + Z.2[i-1] - x4[i-1]
  x3[i] <- x3[i-1] + x2[i-1] + 0.2 * Z.3[i] + Z.3[i-1]
  x4[i] <- x4[i-1] + runif(1, 0.5, 1.5) * x4[i-1]
}
testdf <- data.frame(x1, x2, x3, x4)

# qacf plot for one variable
qacf(testdf$x1)
qacf(testdf$x1, show_sig = TRUE)

# more than one variable
qacf(testdf)
qacf(testdf, show_sig = TRUE)
```

qblandaltman *Bland Altman Plots*

Description

Construct and plot a Bland Altman plot in ggplot2.

Usage

```
qblandaltman(x, alpha = getOption("qwraps2_alpha", 0.05), generate_data = TRUE)

qblandaltman_build_data_frame(x, alpha = getOption("qwraps2_alpha", 0.05))
```

Arguments

`x`	a `data.frame` with two columns, or an object that can be coerced to a data frame. If a `data.frame` with more than two columns is used only the first two columns will be used.
`alpha`	(Defaults to 0.05) place (1 - alpha)*100 place on the plot.
`generate_data`	logical, defaults to TRUE. If TRUE, then the call to `qblandaltman_build_data_frame` is done automatically for you. If FALSE, then you should explicitly call `qblandaltman_build_data_fra` before calling `qblandaltman`.

Details

Providing a `data.frame` with two columns, the function returns a ggplot version of a Bland Altman plot with the specified confidence intervals.

Two ways to call the plotting function. If you submit a `data.frame` `qblandaltman` then the data needed to produce the Bland Altman plot is automatically generated by a call to `qblandaltman_build_data_frame`. Alternatively, you may call `qblandaltman_build_data_frame` directly and then call `qblandaltman`. This might be helpful if you are putting multiple Bland Altman plots together into one ggplot object. See Examples.

Value

a ggplot. Minimal aesthetics have been used so that the user may modify the graphic as desired with ease.

References

Altman, Douglas G., and J. Martin Bland. "Measurement in medicine: the analysis of method comparison studies." The statistician (1983): 307-317.

Bland, J. Martin, and Douglas G Altman. "Statistical methods for assessing agreement between two methods of clinical measurement." The lancet 327, no. 8476 (1986): 307-310.

See Also

`vignette("qwraps2-graphics",pacakge = "qwraps2")` for more examples and details.

Examples

```
data(pefr)
pefr_m1 <-
  cbind("Large" = pefr[pefr$measurement == 1 & pefr$meter == "Wright peak flow meter", "pefr"],
        "Mini"  = pefr[pefr$measurement == 1 & pefr$meter == "Mini Wright peak flow meter", "pefr"])

# The Bland Altman plot plots the average value on the x-axis and the
# difference in the measurements on the y-axis:
qblandaltman(pefr_m1) +
  ggplot2::xlim(0, 800) +
  ggplot2::ylim(-100, 100) +
  ggplot2::xlab("Average of two meters") +
  ggplot2::ylab("Difference in the measurements")
```

---

qkmplot *Kaplan-Meier Plot*

Description

A ggplot2 version of a Kaplan-Meier Plot

Usage

```
qkmplot(x, conf_int = FALSE, ...)

qkmplot_bulid_data_frame(x)
```

Arguments

`x`	object
`conf_int`	logical if TRUE show the CI
`...`	Other arguments passed to survival::plot.survfit

Details

Functions to build, explicitly or implicitly, data.frames and then creating a ggplot2 KM plot.

Value

a ggplot.

See Also

`vignette("qwraps2-graphics",package = "qwraps2")` for additional examples.

Examples

```
# create a survfit object
require(survival)
leukemia.surv <- survival::survfit(survival::Surv(time, status) ~ x, data = survival::aml)

qkmplot(leukemia.surv, conf_int = TRUE)
```

`qroc` *Receiver Operator Curves*

Description

Construction of ROC curves.

Usage

```
qroc(x, ...)

qroc_build_data_frame(fit, n_threshold = 200, ...)

auc(.data)
```

Arguments

`x`	a `glm` fit or `data.frame` generated by `qroc_build_data_frame`.
`...`	passed to `stats::predict`
`fit`	a `glm` fit with `family = binomial()`.
`n_threshold`	number of thresholds to test against.
`.data`	a `data.frame` generated by `qroc_build_data_frame`.

Details

Given a `glm` fit with `family = "binomial"` (either a log-link or logit-link should be fine, a data set will be constructed and ROC plots generated.

The area under the curve (AUC) is determined by a trapezoid approximation.

Value

a ggplot. Minimal aesthetics have been used so that the user may modify the graphic as desired with ease.

AUC for the data set generated by

See Also

`vignette("qwraps2-graphics",package = "qwraps2")` for more examples.

Examples

```
# load ggplot2 and the diamonds data set
data(diamonds, package = "ggplot2")

# Create a logistic regression models
fit <- glm(I(price > 2800) ~ cut * color, data = diamonds, family = binomial())

qroc(fit)
```

`qwraps2` *A collection of wrapper functions aimed at for aiding the authoring of reproducible reports.*

Description

qwraps2 is a collection of helpful functions when working on a varied collection of different analysis reports. There are two types of functions, helpful data summary functions, formatting results from regression models, and **ggplot2** wrappers.

Details

Several wrappers for **ggplot2** style graphics, such as ROC, AUC, Bland-Altman, and KM plots are provided. Named as `qroc`, `qacf`, `qblandaltman` and `qkmplot` to pay homage to `qplot` form **ggplot2** and the standard names for such plots.

Other functions are used to quickly generate meaningful character strings for outputting results in .Rnw, .Rmd, or other similar functions.

Options

There are several options which can be set via `options` and will be used via `getOption`. The following lists, in alphabetical order the different options which are available and what they control.

- `getOptions("qwraps2_alpha",0.05)` significance level, used for generating `(1 -getOptions("qwraps2_alpha",0.` `* 100%` confidence intervals, and determining significance for p-value < `getOptions("qwraps2_alpha",0.05)`.
- `getOptions("qwraps2_frmt_digits",2)` Number of digits to the right of the decimal point for any value other than p-values.
- `getOptions("qwraps2_frmtp_case","upper")` set to either 'upper' or 'lower' for the case of the 'P' for reporting p-values.
- `getOptions("qwraps2_frmtp_digits",4)` Number of digits to the right of the decimal point to report p-values too. If `log10(p-value)` < `getOptions("qwraps2_frmtp_digits",4)` then the output will be "P < 0.01", to however many digits are correct. Other options control other parts of the output p-value format.
- `getOptions("qwraps2_frmtp_leading0",TRUE)` to display or not to display the leading zero in p-values, i.e., if TRUE p-values are reported as 0.02 versus when FALSE p-values are reported as .02.
- `getOptions("qwraps2_journal","default")` if a journal has specific formatting for p-values or other statistics, this option will control the output. Many other options are ignored if this is any other than default. Check the github wiki, or this file, for current lists of implemented journal style methods.
- `getOptions("qwraps2_markup",latex)` value set to 'latex' or to 'markdown'. Output is formatted to meet requirements of either markup language.
- `getOptions("qwraps2_style","default")` By setting this option to a specific journal, p-values and other output, will be formatted to meet journal requirements.

Journals with predefined formatting

- Obstetrics \& Gynecology
  - https://www.editorialmanager.com/ong/default.aspx
  - `options(qwraps2_journal = "obstetrics_gynecology")`
  - P-value formatting as of April 2015:
    Express P values to no more than three decimal places.
    Based on observations of published work, leading 0 will be omitted.
- Pediatric Dentistry:
  - http://www.aapd.org/publications/
  - `options(qwraps2_journal = "pediatric_dentistry")`
  - P-value formatting as of March 2015.
    If P > .01, the actual value for P should be expressed to 2 digits. Non-significant values should not be expressed as "NS" whether or note P is significant, unless rounding a significant P-value expressed to 3 digits would make it non significant (i.e., P=.049, not P=.05). If P<.01, it should be express to 3 digits (e.g., P=.003, not P<.05). Actual P-values should be expressed unless P<.001, in which case they should be so designated.

Rpkg *Formatting Style on URLs for packages on CRAN, Github, and Gitlab.*

Description

Functions for controlling the look of package names in markdown created vignettes and easy curating of URLs for the packages.

Usage

```
Rpkg(pkg)

CRANpkg(pkg)

Githubpkg(pkg, username)

Gitlabpkg(pkg, username)
```

Arguments

pkg	The name of the package, will work as a quoted or raw name.
username	username for Github.com or Gitlab.com

Examples

```
Rpkg(qwraps2)
Rpkg("qwraps2")

CRANpkg(qwraps2)
CRANpkg("qwraps2")

Githubpkg(qwraps2, "dewittpe")
Githubpkg("qwraps2", dewittpe)

Gitlabpkg(qwraps2, "dewittpe")
Gitlabpkg("qwraps2", dewittpe)
```

set_diff *Set Differences*

Description

function for testing for unique values between two vectors, specifically, which values are in vector1, and not in vector2, which values are not in vector1 and in vector2, which values are in both vector1 and vector2.

Usage

```
set_diff(x, y)
```

Arguments

`x, y`	vectors (of the same mode)

Details

Wrapper to call the `union`,

Examples

```
# example with two sets which as a union are the upper and lower case vowels.
set_a <- c("A", "a", "E",      "I", "i", "O", "o", "U", "u", "E", "I")
set_b <- c("A", "a", "E", "e",      "i",      "o", "U", "u", "u", "a", "e")
set_diff(set_a, set_b)
set_diff(set_b, set_a)

# example
set_a <- 1:90
set_b <- set_a[-c(23, 48)]
set_diff(set_a, set_b)
set_diff(set_b, set_a)

# example
set_a <- c("A", "A", "B")
set_b <- c("B", "A")
set_diff(set_a, set_b)
```

`spin_comments` *Spin Comment Check*

Description

A tool to help identify the opening and closing of comments in a spin document. This function is designed to help the user resolve the error "comments must be put in pairs of start and end delimiters."

Usage

```
spin_comments(hair, comment = c("^[# ]*/[*]", "^.*[*]/ *$"), text = NULL, ...)
```

Arguments

`hair`	Path to the R script. The script must be encoded in UTF-8 if it contains multi-byte characters.
`comment`	A pair of regular expressions for the start and end delimiters of comments; the lines between a start and an end delimiter will be ignored. By default, the delimiters are /* at the beginning of a line, and */ at the end, following the convention of C-style comments.
`text`	A character vector of code, as an alternative way to provide the R source. If `text` is not `NULL`, `hair` will be ignored.
`...`	additional arguments (not currently used.)

Examples

```
spin_comments(hair = system.file("examples/spinner1.R", package = "qwraps2"))
```

StatStepribbon *Stat Step Ribbon*

Description

Provides stair step values for ribbon plots (Copied this from the https://github.com/hrbrmstr/ggalt version 0.6.0, which is not yet on CRAN. Some minor modifications to the file have been made).

References

https://groups.google.com/forum/?fromgroups=#!topic/ggplot2/9cFWHaH1CPs

stat_stepribbon *Step ribbon statistic*

Description

Provides stair step values for ribbon plots (Copied this from the https://github.com/hrbrmstr/ggalt version 0.6.0, which is not yet on CRAN. Some minor modifications to the file have been made).

Usage

```
stat_stepribbon(
  mapping = NULL,
  data = NULL,
  geom = "ribbon",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  direction = "hv",
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).
`geom`	which geom to use; defaults to coderibbon
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.
`direction`	`hv` for horizontal-vertical steps, `vh` for vertical-horizontal steps
`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

References

https://groups.google.com/forum/?fromgroups=#!topic/ggplot2/9cFWHaH1CPs

Examples

```
x <- 1:10
df <- data.frame(x=x, y=x+10, ymin=x+7, ymax=x+12)

# horizontal-vertical steps (default)
gg <- ggplot2::ggplot(df, ggplot2::aes(x, y))
gg <- gg + ggplot2::geom_ribbon(ggplot2::aes(ymin=ymin, ymax=ymax),
                                stat="stepribbon", fill="#b2b2b2",
                                direction="hv")
gg <- gg + ggplot2::geom_step(color="#2b2b2b")
gg

# vertical-horizontal steps (default)
gg <- ggplot2::ggplot(df, ggplot2::aes(x, y))
gg <- gg + ggplot2::geom_ribbon(ggplot2::aes(ymin=ymin, ymax=ymax),
                                stat="stepribbon", fill="#b2b2b2",
                                direction="vh")
gg <- gg + ggplot2::geom_step(color="#2b2b2b")
gg

# The same plot calling stat_stepribbon directly
gg <- ggplot2::ggplot(df, ggplot2::aes(x, y))
gg <- gg + stat_stepribbon(mapping = ggplot2::aes(ymin=ymin, ymax=ymax),
                           fill="#b2b2b2", direction="vh")
gg <- gg + ggplot2::geom_step(color="#2b2b2b")
gg
```

`summary_table` *Data Summary Tables*

Description

Tools useful for building data summary tables.

Usage

```
summary_table(x, summaries = qsummary(x), by = NULL)

qsummary(x, numeric_summaries, n_perc_args, env = parent.frame())

## S3 method for class 'qwraps2_summary_table'
cbind(..., deparse.level = 1)

## S3 method for class 'qwraps2_summary_table'
rbind(..., deparse.level = 1)
```

Arguments

`x`	a `data.frame` or `grouped_df`.
`summaries`	a list of lists of formulea for summarizing the data set. See Details and examples.
`by`	a character vector of variable names to generate the summary by, that is one column for each unique values of the variables specified.
`numeric_summaries`	a list of functions to use for summarizing numeric variables. The functions need to be provided as character strings with the single argument defined by the `%s` symbol.
`n_perc_args`	a list of arguments to pass to `n_perc` to be used with `character` or `factor` variables in `.data`.
`env`	environment to assign to the resulting formulae
`...`	`qwraps2_summary_table` objects to bind together
`deparse.level`	integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method): `deparse.level = 0` constructs no labels; the default, `deparse.level = 1` or `deparse.level = 2` constructs labels from the argument names.

Details

`summary_table` can be used to generate good looking, simple tables in LaTeX or markdown. Functions like xtables::print.xtable and Hmisc::latex provide many more tools for formatting tables. The purpose of `summary_table` is to generate good looking tables quickly within workflow for summarizing a data set.

Creating a list-of-lists of summary functions to apply to a data set will allow the exploration of the whole data set and grouped data sets. In the example provided on this page we see a set of summary measures for the `mtcars` data set and the construction of a table for the whole data set and for a grouped data set.

The list-of-lists should be thought of as follows: the outer list defines row groups, the inner lists define the rows within each row group.

More detailed use of these functions can be found the "summary-statistics" vignette.

The `print` method for the `qwraps2_summary_table` objects is just a simple wrapper for `qable`.

Value

a `qwraps2_summary_table` object.

See Also

`qsummary` for generating the summaries, `qable` for marking up `qwraps2_data_summary` objects. The `vignette("summary-statistics",package = "qwraps2")` for detailed use of these functions and caveats.

`cbind`

`rbind`

Examples

```
# A list-of-lists for the summaries arg.  This object is of the basic form:
# It is recommended that you use the .data pronoun in the functions, see
# help(topic = ".data", package = "rlang") for details on this pronoun.
# list("row group A" =
#      list("row 1A" = ~ <summary function>,
#           "row 2A" = ~ <summary function>),
#      "row group B" =
#      list("row 1B" = ~ <summary function>,
#           "row 2B" = ~ <summary function>,
#           "row 3B" = ~ <summary function>))

our_summaries <-
  list("Miles Per Gallon" =
         list("min"  = ~ min(mpg),
              "mean" = ~ mean(mpg),
              "mean &plusmn; sd" = ~ qwraps2::mean_sd(mpg),
              "max"  = ~ max(mpg)),
       "Weight" =
         list("median" = ~ median(wt)),
       "Cylinders" =
         list("4 cyl: n (%)" = ~ qwraps2::n_perc0(cyl == 4),
              "6 cyl: n (%)" = ~ qwraps2::n_perc0(cyl == 6),
              "8 cyl: n (%)" = ~ qwraps2::n_perc0(cyl == 8)))

# Going to use markdown for the markup language in this example,  the original
# option will be reset at the end of the example.
orig_opt <- options()$qwraps2_markup
options(qwraps2_markup = "markdown")

# The summary table for the whole mtcars data set
# whole_table <- summary_table_042(mtcars, our_summaries)
whole_table <- summary_table(mtcars, our_summaries)
whole_table

# The summary table for mtcars grouped by am (automatic or manual transmission)
# This will generate one column for each level of mtcars$am
grouped_by_table <-
  summary_table(mtcars, our_summaries, by = "am")
grouped_by_table

# an equivalent call if you are using the tidyverse:
summary_table(dplyr::group_by(mtcars, am), our_summaries)

# To build a table with a column for the whole data set and each of the am
# levels
cbind(whole_table, grouped_by_table)

# Adding a caption for a LaTeX table
print(whole_table, caption = "Hello world", markup = "latex")

# A **warning** about grouped_df objects.
# If you use dplyr::group_by or
# dplyr::rowwise to manipulate a data set and fail to use dplyr::ungroup you
# might find a table that takes a long time to create and does not summarize the
# data as expected.  For example, let's build a data set with twenty subjects
# and injury severity scores for head and face injuries.  We'll clean the data
# by finding the max ISS score for each subject and then reporting summary
# statistics there of.
set.seed(42)
dat <- data.frame(id = letters[1:20],
                  head_iss = sample(1:6, 20, replace = TRUE, prob = 10 * (6:1)),
                  face_iss = sample(1:6, 20, replace = TRUE, prob = 10 * (6:1)))
dat <- dplyr::group_by(dat, id)
dat <- dplyr::mutate(dat, iss = max(head_iss, face_iss))

iss_summary <-
  list("Head ISS" =
       list("min"    = ~ min(head_iss),
            "median" = ~ median(head_iss),
            "max"    = ~ max(head_iss)),
       "Face ISS" =
       list("min"    = ~ min(face_iss),
            "median" = ~ median(face_iss),
            "max"    = ~ max(face_iss)),
       "Max ISS" =
       list("min"    = ~ min(iss),
            "median" = ~ median(iss),
            "max"    = ~ max(iss)))

# Want: a table with one column for all subjects with nine rows divided up into
# three row groups.  However, the following call will create a table with 20
# columns, one for each subject because dat is a grouped_df
summary_table(dat, iss_summary)

# Ungroup the data.frame to get the correct output
summary_table(dplyr::ungroup(dat), iss_summary)


################################################################################
# The Default call will work with non-syntactically valid names and will
# generate a table with statistics defined by the qsummary call.
summary_table(mtcars, by = "cyl")

# Another example from the diamonds data
data("diamonds", package = "ggplot2")
diamonds["The Price"] <- diamonds$price
diamonds["A Logical"] <- sample(c(TRUE, FALSE), size = nrow(diamonds), replace = TRUE)

# the next two lines are equivalent.
summary_table(diamonds)
summary_table(diamonds, qsummary(diamonds))

summary_table(diamonds, by = "cut")
summary_table(diamonds,
              summaries =
              list("My Summary of Price" =
                   list("min price" = ~ min(price),
                        "IQR"       = ~ stats::IQR(price))),
              by = "cut")

################################################################################
# Data sets with missing values
temp <- mtcars
temp$cyl[5] <- NA
temp$am[c(1, 5, 10)] <- NA
temp$am <- factor(temp$am, levels = 0:1, labels = c("Automatic", "Manual"))
temp$vs <- as.logical(temp$vs)
temp$vs[c(2, 6)] <- NA
qsummary(dplyr::select(temp, cyl, am, vs))
summary_table(dplyr::select(temp, cyl, am, vs))

################################################################################
# binding tables together.  The original design and expected use of
# summary_table did not require a rbind, as all rows are defined in the
# summaries argument.  That said, here are examples of using cbind and rbind to
# build several different tables.
our_summary1 <-
  list("Miles Per Gallon" =
       list("min" = ~ min(mpg),
            "max" = ~ max(mpg),
            "mean (sd)" = ~ qwraps2::mean_sd(mpg)),
       "Displacement" =
       list("min" = ~ min(disp),
            "max" = ~ max(disp),
            "mean (sd)" = ~ qwraps2::mean_sd(disp)))

our_summary2 <-
  list(
       "Weight (1000 lbs)" =
       list("min" = ~ min(wt),
            "max" = ~ max(wt),
            "mean (sd)" = ~ qwraps2::mean_sd(wt)),
       "Forward Gears" =
       list("Three" = ~ qwraps2::n_perc0(gear == 3),
            "Four"  = ~ qwraps2::n_perc0(gear == 4),
            "Five"  = ~ qwraps2::n_perc0(gear == 5))
       )

tab1 <- summary_table(mtcars, our_summary1)
tab2 <- summary_table(dplyr::group_by(mtcars, am), our_summary1)
tab3 <- summary_table(dplyr::group_by(mtcars, vs), our_summary1)

tab4 <- summary_table(mtcars, our_summary2)
tab5 <- summary_table(dplyr::group_by(mtcars, am), our_summary2)
tab6 <- summary_table(dplyr::group_by(mtcars, vs), our_summary2)
```

```
cbind(tab1, tab2, tab3)
cbind(tab4, tab5, tab6)

# row bind is possible, but it is recommended to extend the summary instead.
rbind(tab1, tab4)
summary_table(mtcars, summaries = c(our_summary1, our_summary2))

## Not run:
  cbind(tab1, tab4) # error because rows are not the same
  rbind(tab1, tab2) # error because columns are not the same

## End(Not run)

################################################################################
# reset the original markup option that was used before this example was
# evaluated.
options(qwraps2_markup = orig_opt)

# Detailed examples in the vignette
# vignette("summary-statistics", package = "qwraps2")
```

`summary_table_042` *Deprecated functions*

Description

Functions listed here are deprecated. Replacement methods have been developed to replace these methods.

Usage

```
summary_table_042(x, summaries = qsummary_042(x))

## S3 method for class 'qwraps2_summary_table_042'
cbind(..., deparse.level = 1)

## S3 method for class 'qwraps2_summary_table_042'
rbind(..., deparse.level = 1)

qsummary_042(.data, numeric_summaries, n_perc_args, env)

tab_summary(
  x,
  n_perc_args = list(digits = 0, show_symbol = FALSE),
  envir = parent.frame()
)
```

Arguments

`x`	a variable to summarize
`summaries`	a list of lists of formulea for summarizing the data set. See Details and examples.
`...`	`qwraps2_summary_table_042` objects to bind together
`deparse.level`	integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method): `deparse.level = 0` constructs no labels; the default, `deparse.level = 1` or `deparse.level = 2` constructs labels from the argument names.
`.data`	a `data.frame`
`numeric_summaries`	a list of functions to use for summarizing numeric variables. The functions need to be provided as character strings with the single argument defined by the `%s` symbol.
`n_perc_args`	a list of arguments to pass to `n_perc`
`env`	environment to assign to the resulting formulae
`envir`	the environment to attach to the resulting formulea

Details

`summary_table_042` and `qsummary_042` have been redesigned with some changes to the api for version 0.5.0 of qwraps2. The version released up through version 0.4.2 have been placed here with the appended _042 on the function names. This will will for soft deprecation in 0.5.0 (warning), hard deprecation in later (error), and eventual removal from the package.

See Also

`qable` for marking up `qwraps2_data_summary` objects. `group_by` for `grouped_df` objects. The `vignette("summary-statistics",package = "qwraps2")` for detailed use of these functions and caveats.

`cbind`

`rbind`

Examples

```
data(mtcars2)
st <- summary_table_042(mtcars2[, c("mpg", "wt", "cyl_factor")])
print(st, markup = "markdown")

# build the summaries quickly
qs <- qsummary_042(mtcars2)

summary_table_042(mtcars2, summaries = qs[c("mpg", "cyl", "wt", "gear_factor")])

# the _042 method would only allow for summary by a variable is
# dplyr::group_by was used.  The updated version for qwraps2 version 0.5.0
# has a improved api to summarize by a variable much easier.
```

```
st <- summary_table_042(dplyr::group_by(mtcars2, transmission),
                  summaries = qs[c("mpg", "wt", "cyl", "cyl_character", "cyl_factor")])
print(st, markup = "markdown")
```

traprule *Trapezoid Rule Numeric Integration*

Description

Compute the integral of y with respect to x via trapezoid rule.

Usage

```
traprule(x, y)
```

Arguments

`x, y`	numeric vectors of equal length

Value

a numeric value, the estimated integral

Examples

```
xvec <- seq(-2 * pi, 3 * pi, length = 560)
foo  <- function(x) { sin(x) + x * cos(x) + 12 }
yvec <- foo(xvec)
plot(xvec, yvec, type = "l")

integrate(f = foo, lower = -2 * pi, upper = 3 * pi)
traprule(xvec, yvec)
```

Index