Package 'qwraps2'

October 16, 2024

Title Quick Wraps 2

Version 0.6.1

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 (>= 3)

Encoding UTF-8

URL https://github.com/dewittpe/qwraps2/, http://www.peteredewitt.com/qwraps2/

BugReports https://github.com/dewittpe/qwraps2/issues

Language en-us

LazyData true

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

Suggests dplyr (>= 1.0.0), survival, covr, glmnet, rbenchmark, rmarkdown

RoxygenNote 7.3.2

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 2024-10-15 22:10:02 UTC

Contents

`backtick` *Backtick*

Description

Encapsulate a string in backticks. Very helpful for in line code in `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

Examples

```
backtick("a quoted string")
backtick(no-quote)
backtick(noquote)
```

---

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(
  ...,
  thresholds = NULL,
  confint_method = "logit",
  alpha = getOption("qwraps2_alpha", 0.05)
)

## Default S3 method:
confusion_matrix(
  truth,
  predicted,
  ...,
  thresholds = NULL,
  confint_method = "logit",
  alpha = getOption("qwraps2_alpha", 0.05)
)

## S3 method for class 'formula'
confusion_matrix(
  formula,
  data = parent.frame(),
  ...,
  thresholds = NULL,
  confint_method = "logit",
  alpha = getOption("qwraps2_alpha", 0.05)
)

## S3 method for class 'glm'
confusion_matrix(
  x,
  ...,
  thresholds = NULL,
  confint_method = "logit",
  alpha = getOption("qwraps2_alpha", 0.05)
)

## S3 method for class 'qwraps2_confusion_matrix'
print(x, ...)
```

Arguments

`...`	pass through
`thresholds`	a numeric vector of thresholds to be used to define the confusion matrix (one threshold) or matrices (two or more thresholds). If `NULL` the unique values of `predicted` will be used.
`confint_method`	character string denoting if the logit (default), binomial, or Wilson Score method for deriving confidence intervals

alpha	alpha level for 100 * (1 - alpha)% confidence intervals
truth	a integer vector with the values 0 and 1, or a logical vector. A value of 0 or FALSE is an indication of condition negative; 1 or TRUE is an indication of condition positive.
predicted	a numeric vector. See Details.
formula	column (known) ~ row (test) for building the confusion matrix
data	environment containing the variables listed in the formula
x	a glm object

Details

The confusion matrix:

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

where

- FN: False Negative = truth = 1 & prediction < threshold,
- FP: False Positive = truth = 0 & prediction >= threshold,
- TN: True Negative = truth = 0 & prediction < threshold, and
- TP: True Positive = truth = 1 & prediction >= threshold.

The statistics returned in the cm_stats element are:

- accuracy = (TP + TN) / (TP + TN + FP + FN)
- sensitivity, aka true positive rate or recall = TP / (TP + FN)
- specificity, aka true negative rate = TN / (TN + FP)
- positive predictive value (PPV), aka precision = 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

`confusion_matrix` returns a list with elements

- `cm_stats` a data.frame with columns:
- `auroc` numeric value for the area under the receiver operating curve
- `auroc_ci` a numeric vector of length two with the lower and upper bounds for a 100(1-alpha)% confidence interval about the auroc
- `auprc` numeric value for the area under the precision recall curve
- `auprc_ci` a numeric vector of length two with the lower and upper limits for a 100(1-alpha)% confidence interval about the auprc
- `confint_method` a character string reporting the method used to build the `auroc_ci` and `auprc_ci`
- `alpha` the alpha level of the confidence intervals
- `prevalence` the proportion of the input of positive cases, that is (TP + FN) / (TP + FN + FP + TN) = P / (P + N)

Examples

```
# Example 1: known truth and prediction status
df <-
  data.frame(
      truth = c(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0)
    , pred  = c(1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0)
  )

confusion_matrix(df$truth, df$pred, thresholds = 1)

# Example 2: Use with a logistic regression model
mod <- glm(
  formula = spam ~ word_freq_our + word_freq_over + capital_run_length_total
, data = spambase
, family = binomial()
)

confusion_matrix(mod)
confusion_matrix(mod, thresholds = 0.5)
```

deprecated *Deprecated Functions*

Description

Archive of deprecated functions. Some of these might be removed from the package in later releases.

Deprecated methods for building the data sets needed for plotting roc and prc plots. use `confusion_matrix` instead.

Usage

```
qroc_build_data_frame(fit, truth = NULL, n_threshold = 200, ...)

## Default S3 method:
qroc_build_data_frame(fit, truth = NULL, n_threshold = 200, ...)

## S3 method for class 'glm'
qroc_build_data_frame(fit, truth = NULL, n_threshold = 200, ...)

qprc_build_data_frame(fit, n_threshold = 200, ...)
```

Arguments

`fit`	a `glm` fit with `family = binomial()`, or predicted values
`truth`	ignored if `fit` is a `glm` object. A vector of observations, 0/1 or FALSE/TRUE values, of equal length to `fit`
`n_threshold`	number of thresholds to use to estimate auroc or auprc
`...`	passed to `predict`

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,
  absolute_paths = c("warn", "stop", "silent"),
  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.
absolute_paths	a character string to set the behavior of warning (default), stopping, or silent if/when absolute file paths are used.
stop	if TRUE then an error is thrown if any of the checks fail, else returns a logical. 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, are case matched (optional), and all of requested md5sum checks pass. Windows and macOS are generally case-insensitive systems, but many Linux systems are case-sensitive. As such file.exists and file.access may return different values depending the OS that is active. file_check looks for a case match as part of its checks to hopefully prevent issues across operating systems.

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. That said, there are cases when an absolute path is needed, e.g., a common data file on a server with multiple users accessing the file(s). Set absolute_paths = c("silent") to silence the warnings.

Value

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

Examples

```
# create example files

relative_example_file1 <-
  basename(
    tempfile(
      pattern = "QWRAPS2_EXAMPLE_1"
      , fileext = ".txt"
      , tmpdir = getwd()
    )
  )

relative_example_file2 <-
  basename(
    tempfile(
      pattern = "QWRAPS2_EXAMPLE_2"
      , fileext = ".txt"
      , tmpdir = getwd()
    )
  )

absolute_example_file <- tempfile()

cat("example file.", file = relative_example_file1)
cat("Another example file.", file = relative_example_file2)
cat("Another example file.", file = absolute_example_file)

# Check that you have access to the files in the working directory.
test1 <- file_check(c(relative_example_file1, relative_example_file2))
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")

# access to absolute_example_file will generate a warning about
# absolute_paths by default
test2 <- file_check(absolute_example_file)
test2 <- file_check(absolute_example_file, absolute_paths = "silent")
test2

# Case Match
test_case_match <-
  file_check(
    c(relative_example_file1, tolower(relative_example_file1))
  )
test_case_match

# If one or more files is not accessable then return is FALSE and the meta data
# is printed by default.
test_non_existent_file <-
  file_check(
    c("UNLIKELYFILENAME", relative_example_file1, relative_example_file2)
  )
test_non_existent_file

# Or have an error thrown:
## Not run:
file_check(
  c("UNLIKELYFILENAME", relative_example_file1, relative_example_file2)
, stop = TRUE
)

## End(Not run)

# Verify the md5sums as well as file access:
md5_check1 <- file_check(relative_example_file1, "7a3409e17f9de067740e64448a86e708")
md5_check1

# If you only need to verify a subset of md5sums then use an NA in the md5sums
# argument:
md5_check2 <-
  file_check(c(relative_example_file1, relative_example_file2),
             c("7a3409e17f9de067740e64448a86e708", NA))
md5_check2

# Verify all the md5sums
md5_check3 <-
  file_check(c(relative_example_file1, relative_example_file2),
             c("7a3409e17f9de067740e64448a86e708", "798e52b92e0ae0e60f3f3db1273235d0"))
md5_check3

# clean up working directory
unlink(relative_example_file1)
```

```
unlink(relative_example_file2)
unlink(absolute_example_file)
```

---

frmt *Format Wrappers*

Description

Functions for formatting numeric values for consistent display in reports.

Usage

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

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`).
append	a character string to append to the formatted number. This is particularly useful for percentages or adding punctuation to the end of the formatted number. This should be a vector of length 1, or equal to the length of `x`.
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' was originally really just a wrapper for the `formatC`. It has extended functionality now as I have found common uses cases.

'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)

# reporting a percentage
frmt(17/19 * 100, digits = 2, append = "%")   # good for markdown
frmt(17/19 * 100, digits = 2, append = "\\%") # good for LaTeX

# append one character
frmt(c(1, 2, 3)/19 * 100, digits = 2, append = "%")

# append different characters
frmt(c(1, 2, 3)/19 * 100, digits = 2, append = c("%;", "%!", "%."))

### 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)

# show level uses getOption("qwraps2_alpha", 0.05)
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)

# similar for a data.frame
df2 <- as.data.frame(temp2)
frmtci(df2)
```

`ggplot2_extract_legend` *ggplot2 tools*

Description

A few handy tools for working with ggplot2.

Usage

```
ggplot2_extract_legend(x, ...)
```

Arguments

x	a ggplot object
...	not currently used

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 elements

legend

plot the x

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]])
```

---

gmean *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

```
gmean(mtcars$mpg)
identical(gmean(mtcars$mpg), exp(mean(log(mtcars$mpg))))

gvar(mtcars$mpg)
identical(gvar(mtcars$mpg),
          exp(var(log(mtcars$mpg)) * (nrow(mtcars) - 1) / nrow(mtcars)))
```

```
gsd(mtcars$mpg)
identical(gsd(mtcars$mpg),
          exp(sqrt( var(log(mtcars$mpg)) * (nrow(mtcars) - 1) / nrow(mtcars))))

#############################################################################
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)
```

---

gmean_sd *Geometric Mean and Standard deviation*

Description

A function for calculating and formatting geometric means and standard deviations.

Usage

```
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 ± sd’ or ’mean (sd)’
markup	character string with value “latex” or “markdown”
...	pass through

Details

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

Value

a character vector of the formatted values

See Also

mean_sd, gmean, gsd

Examples

```
gmean_sd(mtcars$mpg, markup = "latex")
gmean_sd(mtcars$mpg, markup = "markdown")
```

---

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,
  ...
)
```

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

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
...	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
tmpdir <- normalizePath(paste0(tempdir(), "/llcache_eg"), mustWork = FALSE)
tmprmd <- tempfile(pattern = "report", tmpdir = tmpdir, fileext = "Rmd")
dir.create(tmpdir)
oldwd <- getwd()
setwd(tmpdir)

# build and example .Rmd file
# note that the variable x is created in the first chunck and then over
# written in the second chunk
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",
    "x <- pi",
    "```",
    "",
    "```{r second-chunk, cache = TRUE}",
    "mpg_by_wt_hp_am <- lm(mpg ~ wt + hp + am, data = mtcars)",
    "x_is_e <- exp(1)",
    "x <- exp(1)",
    "```",
    sep = "\n",
    file = tmprmd)

# 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 = tmprmd, envir = kenv)

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

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

# create three more environment, and load only the first chunk into the
# first, and the second chunck into the second, and then load all of the
# cache into the third
env1 <- new.env()
env2 <- new.env()
env3 <- new.env()

lazyload_cache_labels(labels = "first-chunk",
                      path = paste0(tmpdir, "/cache"),
                      envir = env1)

lazyload_cache_labels(labels = "second-chunk",
                      path = paste0(tmpdir, "/cache"),
                      envir = env2)

lazyload_cache_dir(path = paste0(tmpdir, "/cache"), envir = env3)

# Look at the conents of each of these environments
ls(envir = kenv)
ls(envir = env1)
ls(envir = env2)
ls(envir = env3)

# The regression models are only fitted once an should be the same in all the
# environments where they exist, as should the variables x_is_e and x_is_pi
```

```
all.equal(kenv$mpg_by_wt_hp, env1$mpg_by_wt_hp)
all.equal(env1$mpg_by_wt_hp, env3$mpg_by_wt_hp)

all.equal(kenv$mpg_by_wt_hp_am, env2$mpg_by_wt_hp_am)
all.equal(env2$mpg_by_wt_hp_am, env3$mpg_by_wt_hp_am)

# The value of x, however, should be different in the differnet
# environments.  For kenv, env2, and env3 the value should be exp(1) as that
# was the last assignment value.  In env1 the value should be pi as that is
# the only relevent assignment.

all.equal(kenv$x, exp(1))
all.equal(env1$x, pi)
all.equal(env2$x, exp(1))
all.equal(env3$x, exp(1))

# cleanup
setwd(oldwd)
unlink(tmpdir, recursive = TRUE)
```

---

ll *List Object Aliases*

Description

Aliases for ls providing additional details.

Usage

```
ll(
  pos = 1,
  pattern,
  order_by = "size",
  decreasing = order_by %in% c("size", "rows", "columns")
)
```

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 "object", "size" (default), "class", "rows", or "columns".
decreasing	logical, defaults to TRUE, decreasing order? passed to order.

Value

a data.frame with columns

- object: name of the object
- class: class, or mode if class is not present, of the object
- size: approximate size, in bytes, of the object in memory
- rows: number of rows for data.frames or matrices, or the number of elements for a list like structure
- columns: number of columns for data.frames or matrices

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)
```

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"),
  ...
)
```

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	character string with value "latex" or "markdown"
...	pass through

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.

Value

a character vector of the formatted values

See Also

gmean_sd, n_perc, median_iqr

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)
```

---

mean_se *Mean and Standard Error (of the mean)*

Description

A function for calculating and formatting means and standard deviations.

Usage

```
mean_se(
  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 ± sd' or 'mean (sd)'
markup	latex or markdown
...	pass through

Details

Given a numeric vector, mean_se will return a character string with the mean and standard error of the mean. 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)
mean(x)
sd(x) / sqrt(length(x)) # standard error
mean_se(x)
mean_se(x, show_n = "always")
mean_se(x, show_n = "always", denote_sd = "paren")

x[187] <- NA
mean_se(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
...	pass through

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

a data.frame with 32 rows and 19 columns

[, 1]	make	Manufacturer name	parted out from `rownames(mtcars)`
[, 2]	model		parted out from `rownames(mtcars)`
[, 3]	mpg	miles per (US) gallon	identical to mtcars$mpg
[, 4]	disp	Displacement (cu.in.)	identical to mtcars$disp
[, 5]	hp	Gross horsepower	identical to mtcars$hp
[, 6]	drat	Rear axle ratio	identical to mtcars$drat
[, 7]	wt	weight (1000 lbs)	identical to mtcars$wt
[, 8]	qsec	1/4 mile time	identical to mtcars$qsec
[, 9]	cyl	number of cylinders	identical to mtcars$cyl
[, 10]	cyl_character		
[, 11]	cyl_factor		
[, 12]	vs	Engine (0 = V-shaped, 1 = straight)	identical to mtcars$vs
[, 13]	engine		
[, 14]	am	Transmission (0 = automatic, 1 = manual)	identical to mtcars$am
[, 15]	transmission		
[, 16]	gear	Number of forward gears	identical to mtcars$gear
[, 17]	gear_factor		
[, 18]	carb	Number of carburetors	identical to mtcars$carb
[, 19]	test_date	fictitious testing date	

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",
  show_symbol = FALSE,
  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
...	pass through

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. In general, pkg_check is a handy tool in interactive sessions. For a package you should have package version documentation in the DESCRIPTION file. For a script a base R solution of `stopifnot(packageVersion("pkg") >= "x.y.z")`

Examples

```
# verify that the packages qwraps2, and ggplot2 are available (this should be
# TRUE if you have qwraps2 installed since ggplot2 is imported by qwraps2)
pkg_check(c("qwraps2", "ggplot2"))

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

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

# 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"), c("42.3.14", "2.2.0"))

## Not run:
  # You can have the function throw an error is any of the checks fail
  pkg_check(c("qwraps2", "ggplot2"),
            c("42.3.14", "2.2.0"),
            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 latex from the Hmisc package or htmlTable from the htmlTable package.

Usage

```
qable(
  x,
  rtitle = "",
  rgroup = numeric(0),
  rnames = rownames(x),
  cnames = colnames(x),
  markup = getOption("qwraps2_markup", "latex"),
  kable_args = list(),
  ...
)
```

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 expected to be either "markdown" or "latex"
kable_args	a list of named arguments to send to kable. See Details.
...	pass through

Details

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]

Passing arguments to link[knitr]{kable} is done via the list kable_args. This is an improvement in 0.6.0 to address arguments with different use between qable and kable but the same name, notably format. Within the print method for qwraps2_qable objects, some default arguments for knitr::kable are created.

Defaults if the named element of kable_args is missing: kable_args$format will be "latex" if markup = "latex" and will be "pipe" if markup = "markdown".

kable_args$escape = !(markup = "latex")

kable_args$row.names defaults to FALSE

kable_args$col.names defaults to colnames(x)

Value

qable returns a qwraps2_qable object that is just a character matrix with some additional attributes and the print method returns, invisibly, the object passed to print.

See Also

kable

summary_table, for an example of build a data summary table.

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

Examples

```
data(mtcars)
x <- qable(mtcars)
x
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, kable_args = list(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 = 1 - getOption("qwraps2_alpha", 0.05),
  show_sig = FALSE,
  ...
)
```

Arguments

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

Details

qacf calls acf to generate a data set which is then plotted via ggplot2.

More details and examples for graphics within qwraps2 are in the vignette("qwraps2-graphics", package = "qwraps2")

Value

a ggplot.

See Also

acf.

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.

More details and examples for graphics within qwraps2 are in the vignette("qwraps2-graphics", package = "qwraps2")

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.

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)

## S3 method for class 'survfit'
qkmplot_bulid_data_frame(x)

qrmst(x, tau = Inf)

## S3 method for class 'survfit'
qrmst(x, tau = Inf)

## S3 method for class 'qkmplot_data'
qrmst(x, tau = Inf)
```

Arguments

x	object
conf_int	logical if TRUE show the CI
...	Other arguments passed to survival::plot.survfit
tau	upper bound on time for restricted mean survival time estimate

Details

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

More details and examples for graphics within qwraps2 are in the vignette("qwraps2-graphics", package = "qwraps2")

Value

a ggplot.

Examples

```
require(survival)

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

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

qkmplot_bulid_data_frame(leukemia.surv)

qrmst(leukemia.surv) # NaN for rmst.se in Nonmaintained strata as last observation is an event
qrmst(leukemia.surv, 44)

# pbc examples
pbc_fit <-
  survival::survfit(
      formula = survival::Surv(time, status > 0) ~ trt
    , data = pbc
    , subset = !is.na(trt)
  )

qkmplot(pbc_fit)
qkmplot(pbc_fit, conf_int = TRUE)

qrmst(pbc_fit)
qrmst(pbc_fit)
```

---

qroc-qprc *Receiver-Operator and Precision-Recall Curves*

Description

Construction of ROC and PRC data and plots.

Usage

```
qroc(x, ...)

## Default S3 method:
qroc(x, ...)

## S3 method for class 'qwraps2_confusion_matrix'
qroc(x, ...)

## S3 method for class 'glm'
qroc(x, ...)

qprc(x, ...)

## Default S3 method:
qprc(x, ...)

## S3 method for class 'qwraps2_confusion_matrix'
qprc(x, ...)
```

```
## S3 method for class 'glm'
qprc(x, ...)
```

Arguments

x	an object
...	pass through

Details

The area under the curve (AUC) is determined by a trapezoid approximation for both the AUROC and AUPRC.

More details and examples for graphics within qwraps2 are in the vignette("qwraps2-graphics", package = "qwraps2")

Value

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

Examples

```
##########################################################
# Example 1

df <-
  data.frame(
      truth = c(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0)
    , pred  = c(1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0)
  )

cm <- confusion_matrix(df$truth, df$pred)
qroc(cm)
qprc(cm)

##########################################################
# Getting a ROC or PRC plot from a glm object:

mod <- glm(
  formula = spam ~ word_freq_our + word_freq_over + capital_run_length_total
, data = spambase
, family = binomial()
)

qroc(mod)
qprc(mod)

##########################################################
# View the vignette for more examples
## Not run:
```

```
vignette("qwraps2-graphics")

## End(Not run)
```

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)

Value

a qwraps2_set_diff object, a list of set comparisons

- all_values = union(x, y)
- x_only = setdiff(x, y)
- y_only = setdiff(y, x)
- both = intersect(x, y)
- equal = setequal(x, y)

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)
str(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)
```

spambase *Spambase*

Description

Classifying Email as Spam or Non-Spam

Usage

```
spambase
```

Format

a data.frame with 4601 rows, 58 columns; 57 features and 0/1 indicator for spam

Used under CC BY 4.0 license.

References

Hopkins,Mark, Reeber,Erik, Forman,George, and Suermondt,Jaap. (1999). Spambase. UCI Machine Learning Repository. https://doi.org/10.24432/C53G6X.

See Also

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

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(). 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 ribbon
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
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()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

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, qable_args = list(), ...)

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

Arguments

x	a data.frame.
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.
qable_args	additional values passed to qable
...	pass through
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 within x.
env	environment to assign to the resulting formulae

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.

Examples

```
# A list-of-lists for the summaries arg.  This object is of the basic form:
#
# 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(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(temp[, c("cyl", "am", "vs")])
summary_table(temp[, c("cyl", "am", "vs")])

################################################################################
# Group by Multiple Variables
temp <- mtcars
temp$trans <- factor(temp$am, 0:1, c("Manual", "Auto"))
temp$engine <- factor(temp$vs, 0:1, c("V-Shaped", "Straight"))
summary_table(temp, our_summaries, by = c("trans", "engine"))

################################################################################
# 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")
```

`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)
```

%s% *Operators*

Description

A set of helpful operators to make writing and basic data analysis easier.

Usage

```
e1 %s% e2
```

Arguments

e1	a character string
e2	a character string

Examples

```
# base R
paste0("A longer string ", "can be ", "built")

# with the %s% operator
"A longer string " %s% "can be " %s% "built"
```

Index