Package 'modelsummary'

June 17, 2021

Type Package

Title Summary Tables and Plots for Statistical Models and Data: Beautiful, Customizable, and Publication-Ready

Description Create beautiful and customizable tables to summarize several statistical models side-by-side. Draw coefficient plots, multi-level cross-tabs, dataset summaries, balance tables (a.k.a. ``Table 1s''), and correlation matrices. This package supports dozens of statistical models, and it can produce tables in HTML, LaTeX, Word, Markdown, PDF, PowerPoint, Excel, RTF, JPG, or PNG. Tables can easily be embedded in 'Rmarkdown' or 'knitr' dynamic documents.

Version 0.8.1

URL https://vincentarelbundock.github.io/modelsummary/

BugReports https://github.com/vincentarelbundock/modelsummary/issues/

Depends R (>= 3.5.0)

Imports broom,
checkmate,
generics,
glue,
insight,
kableExtra (>= 1.2.1),
parameters,
performance,
rlang,
tables,
tidyr,
tidyselect

Suggests AER,
broom.mixed,
estimatr,
flextable,
fixest,
gamlss,
ggplot2,
gt (>= 0.3.0),
huxtable,
IRdisplay,
knitr,

lme4,
lmtest,
MASS,
nnet,
officer,
rmarkdown,
sandwich,
testthat,
vdiffr,
spelling,
covr

License GPL-3

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

Language en-US

RoxygenNote 7.1.1

R topics documented:

`coef_rename` *Rename model terms*

Description

A convenience function which can be passed to the `coef_rename` argument of the `modelsummary` function.

Usage

```
coef_rename(
  x,
  factor = TRUE,
  factor_name = TRUE,
  backticks = TRUE,
  titlecase = TRUE,
  underscore = TRUE,
  asis = TRUE
)
```

Arguments

`x`	character vector of term names to transform
`factor`	boolean remove the "factor()" label
`factor_name`	boolean remove the "factor()" label and the name of the variable
`backticks`	boolean remove backticks
`titlecase`	boolean convert to title case
`underscore`	boolean replace underscores by spaces
`asis`	boolean remove the `I` from as-is formula calls

`datasummary` *Summary tables using 2-sided formulae: crosstabs, frequencies, table 1s and more.*

Description

`datasummary` can use any summary function which produces one numeric or character value per variable. The examples section of this documentation shows how to define custom summary functions. The package also ships with several shortcut summary functions: Min, Max, Mean, Median, Var, SD, NPercent, NUnique, Ncol, P0, P25, P50, P75, P100.

Usage

```
datasummary(
  formula,
  data,
  output = "default",
  fmt = 2,
  title = NULL,
  notes = NULL,
  align = NULL,
  add_columns = NULL,
  add_rows = NULL,
  sparse_header = TRUE,
  ...
)
```

Arguments

formula	A two-sided formula to describe the table: rows ~ columns. See the Examples section for a mini-tutorial and the Details section for more resources.
data	A data.frame (or tibble)
output	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above. • Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages. • See the 'Details' section below for more information.
fmt	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)` • character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf` • function: returns a formatted character string.
title	string
notes	list or vector of notes to append to the bottom of the table.
align	A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.
add_columns	a data.frame (or tibble) with the same number of rows as your main table.
add_rows	a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.
sparse_header	TRUE or FALSE. TRUE eliminates column headers which have a unique label across all columns, except for the row immediately above the data. FALSE keeps all headers. The order in which terms are entered in the formula determines the

order in which headers appear. For example, `x~mean*z` will print the mean-related header above the z-related header.‘

`...` all other arguments are passed through to the table-making functions. This allows users to pass arguments directly to `datasummary` in order to affect the behavior of other functions behind the scenes, for instance:

- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.

Details

Visit the 'modelsummary' website for more usage examples: https://vincentarelbundock.github.io/modelsummary

The 'datasummary' function is a thin wrapper around the 'tabular' function from the 'tables' package. More details about table-making formulas can be found in the 'tables' package documentation: ?tables::tabular

Hierarchical or "nested" column labels are only available for these output formats: kableExtra, gt, html, rtf, and LaTeX. When saving tables to other formats, nested labels will be combined to a "flat" header.

Examples

```
## Not run:

# The left-hand side of the formula describes rows, and the right-hand side
# describes columns. This table uses the "mpg" variable as a row and the "mean"
# function as a column:

datasummary(mpg ~ mean, data = mtcars)

# This table uses the "mean" function as a row and the "mpg" variable as a column:

datasummary(mean ~ mpg, data = mtcars)

# Display several variables or functions of the data using the "+"
# concatenation operator. This table has 2 rows and 2 columns:

datasummary(hp + mpg ~ mean + sd, data = mtcars)

# Nest variables or statistics inside a "factor" variable using the "*" nesting
# operator. This table shows the mean of "hp" and "mpg" for each value of
# "cyl":

mtcars$cyl <- as.factor(mtcars$cyl)
datasummary(hp + mpg ~ cyl * mean, data = mtcars)

# If you don't want to convert your original data
# to factors, you can use the 'Factor()'
# function inside 'datasummary' to obtain an identical result:

datasummary(hp + mpg ~ Factor(cyl) * mean, data = mtcars)

# You can nest several variables or statistics inside a factor by using
# parentheses. This table shows the mean and the standard deviation for each
# subset of "cyl":

datasummary(hp + mpg ~ cyl * (mean + sd), data = mtcars)
```

```
# Summarize all numeric variables with 'All()'
datasummary(All(mtcars) ~ mean + sd, data = mtcars)

# Define custom summary statistics. Your custom function should accept a vector
# of numeric values and return a single numeric or string value:

minmax <- function(x) sprintf("[%.2f, %.2f]", min(x), max(x))
mean_na <- function(x) mean(x, na.rm = TRUE)

datasummary(hp + mpg ~ minmax + mean_na, data = mtcars)

# To handle missing values, you can pass arguments to your functions using
# '*Arguments()'

datasummary(hp + mpg ~ mean * Arguments(na.rm = TRUE), data = mtcars)

# For convenience, 'modelsummary' supplies several convenience functions
# with the argument `na.rm=TRUE` by default: Mean, Median, Min, Max, SD, Var,
# P0, P25, P50, P75, P100, NUnique, Histogram

datasummary(hp + mpg ~ Mean + SD + Histogram, data = mtcars)

# These functions also accept a 'fmt' argument which allows you to
# round/format the results

datasummary(hp + mpg ~ Mean * Arguments(fmt = "%.3f") + SD * Arguments(fmt = "%.1f"), data = mtcars)

# Save your tables to a variety of output formats:
f <- hp + mpg ~ Mean + SD
datasummary(f, data = mtcars, output = 'table.html')
datasummary(f, data = mtcars, output = 'table.tex')
datasummary(f, data = mtcars, output = 'table.md')
datasummary(f, data = mtcars, output = 'table.docx')
datasummary(f, data = mtcars, output = 'table.pptx')
datasummary(f, data = mtcars, output = 'table.jpg')
datasummary(f, data = mtcars, output = 'table.png')

# Display human-readable code
datasummary(f, data = mtcars, output = 'html')
datasummary(f, data = mtcars, output = 'markdown')
datasummary(f, data = mtcars, output = 'latex')

# Return a table object to customize using a table-making package
datasummary(f, data = mtcars, output = 'gt')
datasummary(f, data = mtcars, output = 'kableExtra')
datasummary(f, data = mtcars, output = 'flextable')
datasummary(f, data = mtcars, output = 'huxtable')

# add_rows
new_rows <- data.frame(a = 1:2, b = 2:3, c = 4:5)
attr(new_rows, 'position') <- c(1, 3)
datasummary(mpg + hp ~ mean + sd, data = mtcars, add_rows = new_rows)

## End(Not run)
```

`datasummary_balance` *Balance table: Summary statistics for different subsets of the data (e.g., control and treatment groups)*

Description

Balance table: Summary statistics for different subsets of the data (e.g., control and treatment groups)

Usage

```
datasummary_balance(
  formula,
  data,
  output = "default",
  fmt = 1,
  title = NULL,
  notes = NULL,
  align = NULL,
  add_columns = NULL,
  add_rows = NULL,
  dinm = TRUE,
  dinm_statistic = "std.error",
  ...
)
```

Arguments

`formula`	a one-sided formula with the "condition" or "column" variable on the right-hand side.
`data`	A data.frame (or tibble). If this data includes columns called "blocks", "clusters", and/or "weights", the "estimatr" package will consider them when calculating the difference in means.
`output`	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above. • Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages. • See the 'Details' section below for more information.
`fmt`	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)` • character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf` • function: returns a formatted character string.

title	string
notes	list or vector of notes to append to the bottom of the table.
align	A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.
add_columns	a data.frame (or tibble) with the same number of rows as your main table.
add_rows	a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.
dinm	TRUE calculates a difference in means with uncertainty estimates. This option is only available if the `estimatr` package is installed. If `data` includes columns named "blocks", "clusters", or "weights", this information will be taken into account automatically by `estimatr::difference_in_means`.
dinm_statistic	string: "std.error" or "p.value"
...	all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include: • `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression • `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables. • `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

Examples

```
## Not run:
datasummary_balance(~am, mtcars)

## End(Not run)
```

---

datasummary_correlation

Generate a correlation table for all numeric variables in your dataset.

Description

The names of the variables displayed in the correlation table are the names of the columns in the `data`. You can rename those columns (with or without spaces) to produce a table of human-readable variables.

Usage

```
datasummary_correlation(
  data,
  output = "default",
  fmt = 2,
  title = NULL,
```

```
  notes = NULL,
  method = "pearson",
  ...
)
```

Arguments

data	A data.frame (or tibble)
output	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above. • Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages. • See the 'Details' section below for more information.
fmt	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)` • character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf` • function: returns a formatted character string.
title	string
notes	list or vector of notes to append to the bottom of the table.
method	character or function • character: "pearson", "kendall", "spearman", or "pearspear" (Pearson correlations above and Spearman correlations below the diagonal) • function: takes a data.frame with numeric columns and returns a square matrix or data.frame with unique row.names and colnames corresponding to variable names. Note that the `datasummary_correlation_format` can often be useful for formatting the output of custom correlation functions.
...	other parameters are passed through to the table-making packages. This can be used, for example, to pass arguments such as `escape=FALSE` to `kableExtra`.

Examples

```
## Not run:
library(modelsummary)

# clean variable names (base R)
dat <- mtcars[, c("mpg", "hp")]
colnames(dat) <- c("Miles / Gallon", "Horse Power")
datasummary_correlation(dat)

# clean variable names (tidyverse)
library(tidyverse)
dat <- mtcars %>%
  select(`Miles / Gallon` = mpg,
         `Horse Power` = hp)
```

```
datasummary_correlation(dat)

# alternative methods
datasummary_correlation(dat, method = "pearspear")

# custom function
cor_fun <- function(x) cor(x, method = "kendall")
datasummary_correlation(dat, method = cor_fun)

# rename columns alphabetically and include a footnote for reference
note <- sprintf("(%s) %s", letters[1:ncol(dat)], colnames(dat))
note <- paste(note, collapse = "; ")

colnames(dat) <- sprintf("(%s)", letters[1:ncol(dat)])

datasummary_correlation(dat, notes = note)

# `datasummary_correlation_format`: custom function with formatting
dat <- mtcars[, c("mpg", "hp", "disp")]

cor_fun <- function(x) {
  out <- cor(x, method = "kendall")
  datasummary_correlation_format(
    out,
    fmt = 2,
    upper_triangle = "x",
    diagonal = ".")
}

datasummary_correlation(dat, method = cor_fun)

# use kableExtra and psych to color significant cells
library(psych)
library(kableExtra)

dat <- mtcars[, c("vs", "hp", "gear")]

cor_fun <- function(dat) {
  # compute correlations and format them
  correlations <- data.frame(cor(dat))
  correlations <- datasummary_correlation_format(correlations, fmt = 2)

  # calculate pvalues using the `psych` package
  pvalues <- psych::corr.test(dat)$p

  # use `kableExtra::cell_spec` to color significant cells
  for (i in 1:nrow(correlations)) {
    for (j in 1:ncol(correlations)) {
      if (pvalues[i, j] < 0.05 && i != j) {
        correlations[i, j] <- cell_spec(correlations[i, j], background = "pink")
      }
    }
  }
  return(correlations)
}

# The `escape=FALSE` is important here!
```

```
datasummary_correlation(dat, method = cor_fun, escape = FALSE)

## End(Not run)
```

`datasummary_correlation_format` *Format the content of a correlation table*

Description

Mostly for internal use, but can be useful when users supply a function to the `method` argument of `datasummary_correlation`.

Usage

```
datasummary_correlation_format(
  x,
  fmt,
  leading_zero = FALSE,
  diagonal = NULL,
  upper_triangle = NULL
)
```

Arguments

`x`	square numeric matrix
`fmt`	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)` • character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf` • function: returns a formatted character string.
`leading_zero`	boolean. If FALSE, leading zeros are removed
`diagonal`	character or NULL. If character, all elements of the diagonal are replaced by the same character (e.g., "1").
`upper_triangle`	character or NULL. If character, all elements of the upper triangle are replaced by the same character (e.g., "" or ".").

Examples

```
## Not run:
library(modelsummary)

dat <- mtcars[, c("mpg", "hp", "disp")]

cor_fun <- function(x) {
  out <- cor(x, method = "kendall")
  datasummary_correlation_format(
    out,
    fmt = 2,
    upper_triangle = "x",
    diagonal = ".")
```

```
}

datasummary_correlation(dat, method = cor_fun)

## End(Not run)
```

`datasummary_crosstab` *Cross tabulations for categorical variables*

Description

Convenience function to tabulate counts, cell percentages, and row/column percentages for categorical variables. See the Details section for a description of the internal design. For more complex cross tabulations, use datasummary directly.

Usage

```
datasummary_crosstab(
  formula,
  statistic = 1 ~ 1 + N + Percent("row"),
  data,
  output = "default",
  fmt = 1,
  title = NULL,
  notes = NULL,
  align = NULL,
  add_columns = NULL,
  add_rows = NULL,
  sparse_header = TRUE,
  ...
)
```

Arguments

`formula`	A two-sided formula to describe the table: rows ~ columns, where rows and columns are variables in the data. Rows and columns may contain interactions, e.g., `var1 * var2 ~ var3`.
`statistic`	A formula of the form `1 ~ 1 + N + Percent("row")`. The left-hand side may only be empty or contain a `1` to include row totals. The right-hand side may contain: `1` for column totals, `N` for counts, `Percent()` for cell percentages, `Percent("row")` for row percentages, `Percent("col")` for column percentages.
`data`	A data.frame (or tibble)
`output`	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above.

- Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages.
- See the 'Details' section below for more information.

`fmt` determines how to format numeric values

- integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)`
- character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf`
- function: returns a formatted character string.

`title` string

`notes` list or vector of notes to append to the bottom of the table.

`align` A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.

`add_columns` a data.frame (or tibble) with the same number of rows as your main table.

`add_rows` a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.

`sparse_header` TRUE or FALSE. TRUE eliminates column headers which have a unique label across all columns, except for the row immediately above the data. FALSE keeps all headers. The order in which terms are entered in the formula determines the order in which headers appear. For example, `x~mean*z` will print the `mean`-related header above the `z`-related header.'

`...` all other arguments are passed through to the table-making functions. This allows users to pass arguments directly to `datasummary` in order to affect the behavior of other functions behind the scenes, for instance:

- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.

Details

`datasummary_crosstab` is a wrapper around the datasummary function. This wrapper works by creating a customized formula and by feeding it to `datasummary`. The customized formula comes in two parts.

First, we take a two-sided formula supplied by the `formula` argument. All variables of that formula are wrapped in a `Factor()` call to ensure that the variables are treated as categorical.

Second, the `statistic` argument gives a two-sided formula which specifies the statistics to include in the table. `datasummary_crosstab` modifies this formula automatically to include "clean" labels.

Finally, the `formula` and `statistic` formulas are combined into a single formula which is fed directly to the `datasummary` function to produce the table.

Variables in `formula` are automatically wrapped in `Factor()`.

Examples

```
## Not run:
  # crosstab of two variables, showing counts, row percentages, and row/column totals
  datasummary_crosstab(cyl ~ gear, data = mtcars)
```

```
  # crosstab of two variables, showing counts only and no totals
  datasummary_crosstab(cyl ~ gear, statistic = ~ N, data = mtcars)

  # crosstab of three variables
  datasummary_crosstab(am * cyl ~ gear, data = mtcars)

  # crosstab with two variables and column percentages
  datasummary_crosstab(am ~ gear, statistic = ~ Percentage("col"), data = mtcars)

## End(Not run)
```

`datasummary_df` *Draw a table from a data.frame*

Description

Draw a table from a data.frame

Usage

```
datasummary_df(
  data,
  output = "default",
  fmt = 2,
  align = NULL,
  hrule = NULL,
  title = NULL,
  notes = NULL,
  add_rows = NULL,
  add_columns = NULL,
  ...
)
```

Arguments

`data`	A data.frame (or tibble)
`output`	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above. • Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages. • See the 'Details' section below for more information.
`fmt`	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)`

- character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf`
- function: returns a formatted character string.

`align` A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.

`hrule` position of horizontal rules (integer vector)

`title` string

`notes` list or vector of notes to append to the bottom of the table.

`add_rows` a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.

`add_columns` a data.frame (or tibble) with the same number of rows as your main table.

`...` all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include:

- `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression
- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.
- `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

`datasummary_skim` *Quick overview of numeric or categorical variables*

Description

This function was inspired by the excellent `skimr` package for R.

Usage

```
datasummary_skim(
  data,
  type = "numeric",
  output = "default",
  fmt = "%.1f",
  histogram = TRUE,
  title = NULL,
  notes = NULL,
  align = NULL,
  ...
)
```

Arguments

`data`	A data.frame (or tibble)
`type`	of variables to summarize: "numeric" or "categorical" (character)
`output`	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above. • Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages. • See the 'Details' section below for more information.
`fmt`	determines how to format numeric values • integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)` • character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf` • function: returns a formatted character string.
`histogram`	include a histogram (TRUE/FALSE). Supported for: • type = "numeric" • output is "html", "default", "jpg", "png", or "kableExtra" • PDF and HTML documents compiled via Rmarkdown or knitr • See the examples section below for an example of how to use `datasummary` to include histograms in other formats such as markdown.
`title`	string
`notes`	list or vector of notes to append to the bottom of the table.
`align`	A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.
`...`	all other arguments are passed through to the table-making functions. This allows users to pass arguments directly to `datasummary` in order to affect the behavior of other functions behind the scenes, for instance: • `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.

Examples

```
## Not run:
dat <- mtcars
dat$vs <- as.logical(dat$vs)
dat$cyl <- as.factor(dat$cyl)
datasummary_skim(dat)
datasummary_skim(dat, "categorical")

# You can use `datasummary` to produce a similar table in different formats.
# Note that the `Histogram` function relies on unicode characters. These
# characters will only display correctly in some operating systems, under some
# locales, using some fonts. Displaying such histograms on Windows computers
```

```
# is notoriously tricky. The `modelsummary` authors cannot provide support to
# display these unicode histograms.

f <- All(mtcars) ~ Mean + SD + Min + Median + Max + Histogram
datasummary(f, mtcars, output="markdown")


## End(Not run)
```

---

dvnames *Title models with their dependent variables*

Description

A convenience function for use with a regression model or list of regression models. Returns a named list of models, where the names are the models' respective dependent variables. Pass your list of models to dvnames before sending to modelsummary to automatically get dependent variable-titled columns.

Usage

```
dvnames(models, number = FALSE, fill = "Model")
```

Arguments

models	A regression model or list of regression models
number	Should the models be numbered (1), (2), etc., in addition to their dependent variable names?
fill	If insight::find_response() cannot find a response, the column title to use in its place. Set to ' ' to leave blank.

Examples

```
m1 <- lm(mpg ~ hp, data = mtcars)
m2 <- lm(mpg ~ hp + wt, data = mtcars)

# Without dvnames, column names are Model 1 and Model 2
modelsummary(list(m1, m2))

# With dvnames, they are "mpg" and "mpg"
modelsummary(dvnames(list(m1,m2)))
```

`get_estimates` *Extract model estimates. A mostly internal function with some potential uses outside.*

Description

Extract model estimates. A mostly internal function with some potential uses outside.

Usage

```
get_estimates(model, conf_level = 0.95, vcov = NULL, ...)
```

Arguments

`model`	a single model object
`conf_level`	confidence level to use for confidence intervals
`vcov`	robust standard errors and other manual statistics. The `vcov` argument accepts six types of input (see the 'Details' and 'Examples' sections below): • NULL returns the default uncertainty estimates of the model object • string, vector, or (named) list of strings. The strings "classical", "iid" and "constant" are aliases for `NULL`, and they return the model's default uncertainty estimates. The strings "robust", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "stata", "HAC", "NeweyWest", "Andrews", "panel-corrected", "outer-product", "weave" use variance-covariance matrices computed using functions from the `sandwich` package. The behavior of those functions can (and sometimes *must*) be altered by passing arguments to `sandwich` directly from `modelsummary` through the ellipsis (`...`), but it is safer to define your own custom functions as described in the next bullet. • function or (named) list of functions which return variance-covariance matrices with row and column names equal to the names of your coefficient estimates (e.g., `stats::vcov`, `sandwich::vcovHC`, `function(x) vcovPC(x,cluster="country"` • formula or (named) list of formulas with the cluster variable(s) on the right-hand side (e.g., ~clusterid). • (named) list of `length(models)` variance-covariance matrices with row and column names equal to the names of your coefficient estimates. • a (named) list of length(models) vectors with names equal to the names of your coefficient estimates. See 'Examples' section below. Warning: since this list of vectors can include arbitrary strings or numbers, `modelsummary` cannot automatically calculate p values. The `stars` argument may thus use incorrect significance thresholds when `vcov` is a list of vectors.
`...`	all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include: • `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression • `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables. • `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

get_gof — *Extract model gof A mostly internal function with some potential uses outside.*

Description

Extract model gof A mostly internal function with some potential uses outside.

Usage

```
get_gof(model, vcov_type = NULL, ...)
```

Arguments

model	a single model object
vcov_type	string vcov type to add at the bottom of the table
...	all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include: • `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression • `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables. • `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

glance_custom — *Extract custom information from a model object and turn it into a tidy data.frame or tibble with a single row.*

Description

To customize the output of a model of class `lm`, you can define a new method called `glance_custom.lm` which returns a one-row data.frame.

Usage

```
glance_custom(x, ...)
```

Arguments

x	model or other R object to convert to single-row data frame
...	ellipsis

Methods

No methods found in currently loaded packages.

gof_map *Data.frame used to clean up and format goodness-of-fit statistics*

Description

By default, this data frame is passed to the 'gof_map' argument of the 'modelsummary' function. Users can modify this data frame to customize the list of statistics to display and their format. See example below.

Usage

```
gof_map
```

Format

data.frame with 4 columns of character data: raw, clean, fmt, omit

Examples

```
## Not run:

library(modelsummary)
mod <- lm(wt ~ drat, data = mtcars)
gm <- modelsummary::gof_map
gm$omit[gm$raw == 'deviance'] <- FALSE
gm$fmt[gm$raw == 'r.squared'] <- "%.5f"
modelsummary(mod, gof_map = gm)

## End(Not run)
```

Histogram *datasummary statistic shortcut*

Description

This function uses Unicode characters to create a histogram. This can sometimes be useful, but is generally discouraged. Unicode characters can only display a limited number of heights for bars, and the accuracy of output is highly dependent on the platform (typeface, output type, windows vs. mac, etc.). We recommend you use the `kableExtra::spec_hist` function instead.

Usage

```
Histogram(x, bins = 10)
```

Arguments

x	variable to summarize
bins	number of histogram bars

Max *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Max(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

Mean *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Mean(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

Median *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Median(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

Min *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Min(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

---

modelplot *Model Summary Plots with Estimates and Confidence Intervals*

Description

Model Summary Plots with Estimates and Confidence Intervals

Usage

```
modelplot(
  models,
  conf_level = 0.95,
  coef_map = NULL,
  coef_omit = NULL,
  coef_rename = NULL,
  vcov = NULL,
  add_rows = NULL,
  facet = FALSE,
  draw = TRUE,
  background = NULL,
  ...
)
```

Arguments

models	a model or (optionally named) list of models
conf_level	confidence level to use for confidence intervals
coef_map	character vector. Subset, rename, and reorder coefficients. Coefficients omitted from this vector are omitted from the table. The order of the vector determines the order of the table. coef_map can be a named or an unnamed character vector (see the Examples section below). If coef_map is a named vector, its values define the labels that must appear in the table, and its names identify the original term names stored in the model object: c("hp:mpg"="HPxM/G").

`coef_omit` string regular expression. Omits all matching coefficients from the table using `grepl(perl=TRUE)`.

`coef_rename` named character vector or function which returns a named vector. Values of the vector refer to the variable names that will appear in the table. Names refer to the original term names stored in the model object, e.g. c("hp:mpg"="hp X mpg") for an interaction term.

`vcov` robust standard errors and other manual statistics. The `vcov` argument accepts six types of input (see the 'Details' and 'Examples' sections below):

- NULL returns the default uncertainty estimates of the model object
- string, vector, or (named) list of strings. The strings "classical", "iid" and "constant" are aliases for `NULL`, and they return the model's default uncertainty estimates. The strings "robust", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "stata", "HAC", "NeweyWest", "Andrews", "panel-corrected", "outer-product", "weave" use variance-covariance matrices computed using functions from the `sandwich` package. The behavior of those functions can (and sometimes *must*) be altered by passing arguments to `sandwich` directly from `modelsummary` through the ellipsis (`...`), but it is safer to define your own custom functions as described in the next bullet.
- function or (named) list of functions which return variance-covariance matrices with row and column names equal to the names of your coefficient estimates (e.g., `stats::vcov`, `sandwich::vcovHC`, `function(x) vcovPC(x,cluster="country"`
- formula or (named) list of formulas with the cluster variable(s) on the right-hand side (e.g., ~clusterid).
- (named) list of `length(models)` variance-covariance matrices with row and column names equal to the names of your coefficient estimates.
- a (named) list of length(models) vectors with names equal to the names of your coefficient estimates. See 'Examples' section below. Warning: since this list of vectors can include arbitrary strings or numbers, `modelsummary` cannot automatically calculate p values. The `stars` argument may thus use incorrect significance thresholds when `vcov` is a list of vectors.

`add_rows` a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.

`facet` TRUE or FALSE. When the 'models' argument includes several model objects, TRUE draws terms in separate facets, and FALSE draws terms side-by-side (dodged).

`draw` TRUE returns a 'ggplot2' object, FALSE returns the data.frame used to draw the plot.

`background` A list of 'ggplot2' geoms to add to the background of the plot. This is especially useful to display annotations "behind" the 'geom_pointrange' that 'modelplot' draws.

`...` all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include:

- `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression
- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.

- `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

Examples

```
## Not run:

library(modelsummary)

# single model
mod <- lm(hp ~ vs + drat, mtcars)
modelplot(mod)

# omit terms with string matches or regexes
modelplot(mod, coef_omit = 'Interc')

# rename, reorder and subset with 'coef_map'
cm <- c('vs' = 'V-shape engine',
  'drat' = 'Rear axle ratio')
modelplot(mod, coef_map = cm)

# several models
models <- list()
models[['Small model']] <- lm(hp ~ vs, mtcars)
models[['Medium model']] <- lm(hp ~ vs + factor(cyl), mtcars)
models[['Large model']] <- lm(hp ~ vs + drat + factor(cyl), mtcars)
modelplot(models)

# add_rows: add an empty reference category

mod <- lm(hp ~ factor(cyl), mtcars)

add_rows = data.frame(
  term = "factory(cyl)4",
  model = "Model 1",
  estimate = NA)
attr(add_rows, "position") = 3
modelplot(mod, add_rows = add_rows)


# customize your plots with 'ggplot2' functions
library(ggplot2)

modelplot(models) +
  scale_color_brewer(type = 'qual') +
  theme_classic()

# pass arguments to 'geom_pointrange' through the ... ellipsis
modelplot(mod, color = 'red', size = 1, fatten = .5)

# add geoms to the background, behind geom_pointrange
b <- list(geom_vline(xintercept = 0, color = 'orange'),
  annotate("rect", alpha = .1,
    xmin = -.5, xmax = .5,
    ymin = -Inf, ymax = Inf),
  geom_point(aes(y = term, x = estimate), alpha = .3,
```

```
    size = 10, color = 'red', shape = 'square'))
modelplot(mod, background = b)

## End(Not run)
```

`modelsummary` *Model Summary Tables*

Description

The content of the tables can be altered with the function's arguments, or by calling `options`, as described in the *Details* section below. The look of the tables can be customized by specifying the `output` argument, and by using functions from one of the supported table customization packages: `kableExtra`, `gt`, `flextable`, `huxtable`.

Usage

```
modelsummary(
  models,
  output = "default",
  fmt = 3,
  estimate = "estimate",
  statistic = "std.error",
  vcov = NULL,
  conf_level = 0.95,
  stars = FALSE,
  coef_map = NULL,
  coef_omit = NULL,
  coef_rename = NULL,
  gof_map = NULL,
  gof_omit = NULL,
  group = term ~ model,
  group_map = NULL,
  add_rows = NULL,
  align = NULL,
  notes = NULL,
  title = NULL,
  ...
)
```

Arguments

`models`	a model or (optionally named) list of models
`output`	filename or object type (character string) • Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg. • Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter". • To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above.

- Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages.
- See the 'Details' section below for more information.

`fmt` determines how to format numeric values

- integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)`
- character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf`
- function: returns a formatted character string.

`estimate` string or `glue` string of the estimate to display (or a vector with one string per model). Valid entries include any column name of the data.frame produced by `get_estimates(model)`. Examples:

- `"estimate"`
- `"{estimate} ({std.error}){stars}"`
- `"{estimate} [{conf.low},{conf.high}]"`

`statistic` vector of strings or `glue` strings which select uncertainty statistics to report vertically below the estimate. NULL omits all uncertainty statistics.

- "conf.int", "std.error", "statistic", "p.value", "conf.low", "conf.high", or any column name produced by: `get_estimates(model)`
- `glue` package strings with braces, such as:
  - `"{p.value} [{conf.low},{conf.high}]"`
  - `"Std.Error: {std.error}"`
- Note: Parentheses are added automatically unless the string includes `glue` curly braces `{}`.
- Note: To report uncertainty statistics *next* to coefficients, you can #' supply a `glue` string to the `estimate` argument.

`vcov` robust standard errors and other manual statistics. The `vcov` argument accepts six types of input (see the 'Details' and 'Examples' sections below):

- NULL returns the default uncertainty estimates of the model object
- string, vector, or (named) list of strings. The strings "classical", "iid" and "constant" are aliases for `NULL`, and they return the model's default uncertainty estimates. The strings "robust", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "stata", "HAC", "NeweyWest", "Andrews", "panel-corrected", "outer-product", "weave" use variance-covariance matrices computed using functions from the `sandwich` package. The behavior of those functions can (and sometimes *must*) be altered by passing arguments to `sandwich` directly from `modelsummary` through the ellipsis (`...`), but it is safer to define your own custom functions as described in the next bullet.
- function or (named) list of functions which return variance-covariance matrices with row and column names equal to the names of your coefficient estimates (e.g., `stats::vcov`, `sandwich::vcovHC`, `function(x) vcovPC(x,cluster="country"`
- formula or (named) list of formulas with the cluster variable(s) on the right-hand side (e.g., ~clusterid).
- (named) list of `length(models)` variance-covariance matrices with row and column names equal to the names of your coefficient estimates.
- a (named) list of length(models) vectors with names equal to the names of your coefficient estimates. See 'Examples' section below. Warning: since this list of vectors can include arbitrary strings or numbers, `modelsummary` cannot automatically calculate p values. The `stars` argument may thus use incorrect significance thresholds when `vcov` is a list of vectors.

`conf_level` confidence level to use for confidence intervals

`stars` to indicate statistical significance

- FALSE (default): no significance stars.
- TRUE: +=.1, *=.05, **=.01, ***=0.001
- Named numeric vector for custom stars such as `c('*' = .1, '+' = .05)`
- Note: a legend will not be inserted at the bottom of the table when the `estimate` or `statistic` arguments use "glue strings" with `{stars}`.

`coef_map` character vector. Subset, rename, and reorder coefficients. Coefficients omitted from this vector are omitted from the table. The order of the vector determines the order of the table. `coef_map` can be a named or an unnamed character vector (see the Examples section below). If `coef_map` is a named vector, its values define the labels that must appear in the table, and its names identify the original term names stored in the model object: `c("hp:mpg"="HPxM/G")`.

`coef_omit` string regular expression. Omits all matching coefficients from the table using `grepl(perl=TRUE)`.

`coef_rename` named character vector or function which returns a named vector. Values of the vector refer to the variable names that will appear in the table. Names refer to the original term names stored in the model object, e.g. c("hp:mpg"="hp X mpg") for an interaction term.

`gof_map` rename, reorder, and omit goodness-of-fit statistics and other model information. This argument accepts 3 types of values:

- NULL (default): the `modelsummary::gof_map` dictionary is used for formatting, and all unknown statistic are included.
- data.frame with 3 columns named "raw", "clean", "fmt". Unknown statistics are omitted. See the 'Examples' section below.
- list of lists, each of which includes 3 elements named "raw", "clean", "fmt". Unknown statistics are omitted. See the 'Examples section below'.

`gof_omit` string regular expression. Omits all matching gof statistics from the table (using `grepl(perl=TRUE)`).

`group` a two-sided formula with two or three components which describes how groups of parameters should be displayed. The formula must include both a "term" and a "model" component. In addition, a component can be used to identify groups of parameters (e.g., outcome levels of a multinomial logit model). This group identifier must be the name of a column in the data.frame produced by `get_estimates(model)`.

- `term ~ model` displays coefficients as rows and models as columns
- `model ~ term` displays models as rows and coefficients as columns
- `response + term ~ model` displays response levels and coefficients as rows and models as columns.

`group_map` named or unnamed character vector. Subset, rename, and reorder coefficient groups specified in the `group` argument. See `coef_map`.

`add_rows` a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.

`align` A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.

notes — list or vector of notes to append to the bottom of the table.

title — string

... — all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include:

- `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression
- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.
- `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

Details

`options`

`modelsummary` supports 4 table-making packages: `kableExtra`, `gt`, `flextable`, and `huxtable`. Some of these packages have overlapping functionalities. For example, 3 of those packages can export to LaTeX. To change the default backend used for a specific file format, you can use the `options` function:

`options(modelsummary_html = 'kableExtra')` `options(modelsummary_latex = 'gt')` `options(modelsummary_w = 'huxtable')` `options(modelsummary_png = 'gt')`

`modelsummary` can use two sets of packages to extract information from statistical models: `broom` and the `easystats` family (`performance` and `parameters`). By default, it uses `broom` first and `easystats` as a fallback if `broom` fails. You can change the order of priorities or include goodness-of-fit extracted by *both* packages by setting:

`options(modelsummary_get = "broom")` `options(modelsummary_get = "easystats")` `options(modelsummary_ge = "all")`

`output` argument:

The `modelsummary_list` output type is a lightweight representation of the model results. The `modelsummary` function can export to this format by setting the `output` argument, and it can accept objects of this format as input models to create a table. This can be useful to save raw results, in order to print a table later, without having to save and extract from the entire model object. Note that the confidence intervals are only stored in a `modelsummary_list` if explicitly requested:
backup <- modelsummary(models, output = "modelsummary_list" statistic = "conf.int")
`modelsummary(backup)`

When a file name with a valid extension is supplied to the `output` argument, the table is written immediately to file. If you want to customize your table by post-processing it with an external package, you need to choose a different output format and saving mechanism. Unfortunately, the approach differs from package to package:

- `gt`: set `output="gt"`, post-process your table, and use the `gt::gtsave` function.
- `kableExtra`: set `output` to your destination format (e.g., "latex", "html", "markdown"), post-process your table, and use `kableExtra::save_kable` function.

`vcov` argument:

To use a string such as "robust" or "HC0", your model must be supported by the `sandwich` package. This includes objects such as: lm, glm, survreg, coxph, mlogit, polr, hurdle, zeroinfl, and more.

NULL, "classical", "iid", and "constant" are aliases which do not modify uncertainty estimates and simply report the default standard errors stored in the model object.

One-sided formulas such as ~clusterid are passed to the sandwich::vcovCL function.

Matrices and functions producing variance-covariance matrices are first passed to lmtest. If this does not work, modelsummary attempts to take the square root of the diagonal to adjust "std.error", but the other uncertainty estimates are not be adjusted.

Numeric vectors are formatted according to fmt and placed in brackets. Character vectors printed as given, without parentheses.

If your model type is supported by the lmtest package, the vcov argument will try to use that package to adjust all the uncertainty estimates, including "std.error", "statistic", "p.value", and "conf.int". If your model is not supported by lmtest, only the "std.error" will be adjusted by, for example, taking the square root of the matrix's diagonal.

Value

a regression table in a format determined by the output argument.

Examples

```
## Not run:

# The `modelsummary` website includes \emph{many} examples and tutorials:
# https://vincentarelbundock.github.io/modelsummary

library(modelsummary)

# load data and estimate models
data(trees)
models <- list()
models[['Bivariate']] <- lm(Girth ~ Height, data = trees)
models[['Multivariate']] <- lm(Girth ~ Height + Volume, data = trees)

# simple table
modelsummary(models)

# statistic
modelsummary(models, statistic = NULL)
modelsummary(models, statistic = 'p.value')
modelsummary(models, statistic = 'statistic')
modelsummary(models, statistic = 'conf.int', conf_level = 0.99)
modelsummary(models, statistic = c("t = {statistic}",
                                   "se = {std.error}",
                                   "conf.int"))

# estimate
modelsummary(models,
  statistic = NULL,
  estimate = "{estimate} [{conf.low}, {conf.high}]")
modelsummary(models,
  estimate = c("{estimate}{stars}",
               "{estimate} ({std.error})"))

# vcov
modelsummary(models, vcov = "robust")
modelsummary(models, vcov = list("classical", "stata"))
modelsummary(models, vcov = sandwich::vcovHC)
modelsummary(models,
```

```
  vcov = list(stats::vcov, sandwich::vcovHC))
modelsummary(models,
  vcov = list(c("(Intercept)"="", "Height"="!"),
              c("(Intercept)"="", "Height"="!", "Volume"="!!")))

# vcov with custom names
modelsummary(
  models,
  vcov = list("Stata Corp" = "stata",
              "Newey Lewis & the News" = "NeweyWest"))

# coef_rename
modelsummary(models, coef_map = c('Volume' = 'Large', 'Height' = 'Tall'))

# coef_map
modelsummary(models, coef_map = c('Volume' = 'Large', 'Height' = 'Tall'))
modelsummary(models, coef_map = c('Volume', 'Height'))

# title
modelsummary(models, title = 'This is the title')

# title with LaTeX label (for numbering and referencing)
modelsummary(models, title = 'This is the title \\label{tab:description}')

# add_rows
rows <- tibble::tribble(~term, ~Bivariate, ~Multivariate,
  'Empty row', '-', '-',
  'Another empty row', '?', '?')
attr(rows, 'position') <- c(1, 3)
modelsummary(models, add_rows = rows)

# notes
modelsummary(models, notes = list('A first note', 'A second note'))

# gof_map: data.frame
gm <- modelsummary::gof_map
gof_custom$omit[gof_custom$raw == 'deviance'] <- FALSE
gof_custom$fmt[gof_custom$raw == 'r.squared'] <- "%.5f"
modelsummary(models, gof_map = gof_custom)

# gof_map: list of lists
f1 <- function(x) format(round(x, 3), big.mark=",")
f2 <- function(x) format(round(x, 0), big.mark=",")
gm <- list(
  list("raw" = "nobs", "clean" = "N", "fmt" = f2),
  list("raw" = "AIC", "clean" = "aic", "fmt" = f1))
modelsummary(models,
  fmt = f1,
  gof_map = gm)


## End(Not run)
```

`modelsummary_wide` *Summary Tables for Models with Grouped Coefficients*

Description

`modelsummary_wide` summarizes models with grouped coefficients. For example, these groups could correspond to levels of a multinomial logit outcome variable, or to parameters of a GAMLSS model. This function's arguments are the same as in `modelsummary`, except for the `coef_group` and the `stacking` arguments.

Usage

```
modelsummary_wide(
  models,
  output = "default",
  fmt = 3,
  estimate = "estimate",
  statistic = "std.error",
  vcov = NULL,
  conf_level = 0.95,
  stars = FALSE,
  coef_group = NULL,
  coef_map = NULL,
  coef_omit = NULL,
  coef_rename = NULL,
  gof_map = NULL,
  gof_omit = NULL,
  add_rows = NULL,
  align = NULL,
  notes = NULL,
  title = NULL,
  stacking = "horizontal",
  ...
)
```

Arguments

`models` a model or (optionally named) list of models

`output` filename or object type (character string)

- Supported filename extensions: .html, .tex, .md, .txt, .png, .jpg.
- Supported object types: "default", "html", "markdown", "latex", "latex_tabular", "data.frame", "modelsummary_list", "gt", "kableExtra", "huxtable", "flextable", "jupyter".
- To change the default output format, type `options(modelsummary_default = "latex")`, where `latex` can be any of the valid object types listed above.
- Warning: users should not supply a file name to the `output` argument if they intend to customize the table with external packages.
- See the 'Details' section below for more information.

`fmt` determines how to format numeric values

- integer: the number of digits to keep after the period `format(round(x,fmt),nsmall=fmt)`
- character: passed to the `sprintf` function (e.g., '%.3f' keeps 3 digits with trailing zero). See `?sprintf`
- function: returns a formatted character string.

`estimate` string or `glue` string of the estimate to display (or a vector with one string per model). Valid entries include any column name of the data.frame produced by `get_estimates(model)`. Examples:

- `"estimate"`
- `"{estimate} ({std.error}){stars}"`
- `"{estimate} [{conf.low},{conf.high}]"`

`statistic` vector of strings or `glue` strings which select uncertainty statistics to report vertically below the estimate. NULL omits all uncertainty statistics.

- "conf.int", "std.error", "statistic", "p.value", "conf.low", "conf.high", or any column name produced by: `get_estimates(model)`
- `glue` package strings with braces, such as:
  - `"{p.value} [{conf.low},{conf.high}]"`
  - `"Std.Error: {std.error}"`
- Note: Parentheses are added automatically unless the string includes `glue` curly braces `{}`.
- Note: To report uncertainty statistics *next* to coefficients, you can #' supply a `glue` string to the `estimate` argument.

`vcov` robust standard errors and other manual statistics. The `vcov` argument accepts six types of input (see the 'Details' and 'Examples' sections below):

- NULL returns the default uncertainty estimates of the model object
- string, vector, or (named) list of strings. The strings "classical", "iid" and "constant" are aliases for `NULL`, and they return the model's default uncertainty estimates. The strings "robust", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "stata", "HAC", "NeweyWest", "Andrews", "panel-corrected", "outer-product", "weave" use variance-covariance matrices computed using functions from the `sandwich` package. The behavior of those functions can (and sometimes *must*) be altered by passing arguments to `sandwich` directly from `modelsummary` through the ellipsis (`...`), but it is safer to define your own custom functions as described in the next bullet.
- function or (named) list of functions which return variance-covariance matrices with row and column names equal to the names of your coefficient estimates (e.g., `stats::vcov`, `sandwich::vcovHC`, `function(x) vcovPC(x,cluster="country"`
- formula or (named) list of formulas with the cluster variable(s) on the right-hand side (e.g., ~clusterid).
- (named) list of `length(models)` variance-covariance matrices with row and column names equal to the names of your coefficient estimates.
- a (named) list of length(models) vectors with names equal to the names of your coefficient estimates. See 'Examples' section below. Warning: since this list of vectors can include arbitrary strings or numbers, `modelsummary` cannot automatically calculate p values. The `stars` argument may thus use incorrect significance thresholds when `vcov` is a list of vectors.

`conf_level` confidence level to use for confidence intervals

`stars` to indicate statistical significance

- FALSE (default): no significance stars.
- TRUE: +=.1, *=.05, **=.01, ***=0.001
- Named numeric vector for custom stars such as `c('*' = .1, '+' = .05)`
- Note: a legend will not be inserted at the bottom of the table when the `estimate` or `statistic` arguments use "glue strings" with `{stars}`.

`coef_group` the name of the coefficient groups to use as columns (NULL or character). If `coef_group` is NULL, `modelsummary` tries to guess the correct coefficient group identifier. To be valid, this identifier must be a column in the data.frame produced by `get_estimates(model)`.

`coef_map` character vector. Subset, rename, and reorder coefficients. Coefficients omitted from this vector are omitted from the table. The order of the vector determines the order of the table. `coef_map` can be a named or an unnamed character vector (see the Examples section below). If `coef_map` is a named vector, its values define the labels that must appear in the table, and its names identify the original term names stored in the model object: `c("hp:mpg"="HPxM/G")`.

`coef_omit` string regular expression. Omits all matching coefficients from the table using `grepl(perl=TRUE)`.

`coef_rename` named character vector or function which returns a named vector. Values of the vector refer to the variable names that will appear in the table. Names refer to the original term names stored in the model object, e.g. c("hp:mpg"="hp X mpg") for an interaction term.

`gof_map` rename, reorder, and omit goodness-of-fit statistics and other model information. This argument accepts 3 types of values:

- NULL (default): the `modelsummary::gof_map` dictionary is used for formatting, and all unknown statistic are included.
- data.frame with 3 columns named "raw", "clean", "fmt". Unknown statistics are omitted. See the 'Examples' section below.
- list of lists, each of which includes 3 elements named "raw", "clean", "fmt". Unknown statistics are omitted. See the 'Examples section below'.

`gof_omit` string regular expression. Omits all matching gof statistics from the table (using `grepl(perl=TRUE)`).

`add_rows` a data.frame (or tibble) with the same number of columns as your main table. By default, rows are appended to the bottom of the table. You can define a "position" attribute of integers to set the row positions. See Examples section below.

`align` A character string of length equal to the number of columns in the table. "lcr" means that the first column will be left-aligned, the 2nd column center-aligned, and the 3rd column right-aligned.

`notes` list or vector of notes to append to the bottom of the table.

`title` string

`stacking` direction in which models are stacked: "horizontal" or "vertical"

`...` all other arguments are passed through to the extractor and table-making functions. This allows users to pass arguments directly to `modelsummary` in order to affect the behavior of other functions behind the scenes. Examples include:

- `broom::tidy(exponentiate=TRUE)` to exponentiate logistic regression
- `kableExtra::kbl(escape=FALSE)` to avoid escaping math characters in `kableExtra` tables.
- `performance::model_performance(metrics="RMSE")` to select goodness-of-fit statistics to extract using the `performance` package (must have set `options(modelsummary_get="easystats")` first).

Value

a regression table in a format determined by the `output` argument.

N *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
N(x)
```

Arguments

x	variable to summarize

Examples

```
## Not run:
datasummary(Factor(cyl) ~ N, data = mtcars)

## End(Not run)
```

Ncol *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Ncol(x, ...)
```

Arguments

x	variable to summarize
...	unused

NPercent *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
NPercent(x, y)
```

Arguments

x	variable to summarize
y	denominator variable

NUnique *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
NUnique(x, ...)
```

Arguments

x	variable to summarize
...	unused

Examples

```
## Not run:
datasummary(cyl + hp ~ NUnique, data = mtcars)

## End(Not run)
```

P0 *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
P0(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

P100 *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
P100(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

P25 *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
P25(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

P50 *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
P50(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

P75 *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
P75(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

PercentMissing *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
PercentMissing(x)
```

Arguments

x	variable to summarize

SD *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
SD(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the `modelsummary:::rounding` function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

supported_models *List of model objects from which* modelsummary *can extract estimates and statistics*

Description

List of model objects from which modelsummary can extract estimates and statistics

Usage

```
supported_models()
```

tidy_custom *Extract custom information from a model object and turn it into a tidy data.frame or tibble*

Description

To customize the output of a model of class lm, you can define a method called tidy_custom.lm which returns a data.frame with a column called "term", and the other columns you want to use as "estimate" or "statistic" in your modelsummary() call. The output of this method must be similar to the result of tidy(model).

Usage

```
tidy_custom(x)
```

Arguments

x	An object to be converted into a tidy data.frame or tibble.

Value

A data.frame or tibble with information about model components.

Var *datasummary statistic shortcut*

Description

datasummary statistic shortcut

Usage

```
Var(x, fmt = NULL, na.rm = TRUE, ...)
```

Arguments

x	variable to summarize
fmt	passed to the modelsummary:::rounding function
na.rm	a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
...	unused

Examples

```
## Not run:
datasummary(mpg + hp ~ Mean + Median + P0 + P25 + P50 + P75 + P100 +
            Min + Max + SD + Var,
            data = mtcars)

## End(Not run)
```

Index