Package 'jtools'

December 2, 2022

Type Package

Title Analysis and Presentation of Social Scientific Data

Version 2.2.1

Description This is a collection of tools for more efficiently understanding and sharing the results of (primarily) regression analyses. There are also a number of miscellaneous functions for statistical and programming purposes. Support for models produced by the survey and lme4 packages are points of emphasis.

URL https://jtools.jacob-long.com

BugReports https://github.com/jacob-long/jtools/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports crayon, generics, ggplot2 (>= 3.3.0), magrittr, pander, pkgconfig, rlang (>= 0.3.0), tibble

Suggests boot, broom, broom.mixed, huxtable (>= 3.0.0), kableExtra, lme4, lmerTest, MASS, methods, pbkrtest, RColorBrewer, sandwich, scales, survey, weights, knitr, rmarkdown, testthat, vdiffr

Enhances brms, quantreg, rstanarm

RoxygenNote 7.1.2

VignetteBuilder knitr

Depends R (>= 3.5.0)

NeedsCompilation no

Author Jacob A. Long [aut, cre] (<https://orcid.org/0000-0002-1582-6214>)

Maintainer Jacob A. Long <jacob.long@sc.edu>

Repository CRAN

Date/Publication 2022-12-02 00:10:02 UTC

R topics documented:

add_gridlines *Add and remove gridlines*

Description

These are convenience wrappers for editing `ggplot2::theme()`'s `panel.grid.major` and `panel.grid.minor` parameters with sensible defaults.

Usage

```
add_gridlines(x = TRUE, y = TRUE, minor = TRUE)

add_x_gridlines(minor = TRUE)

add_y_gridlines(minor = TRUE)

drop_gridlines(x = TRUE, y = TRUE, minor.only = FALSE)

drop_x_gridlines(minor.only = FALSE)

drop_y_gridlines(minor.only = FALSE)
```

Arguments

`x`	Apply changes to the x axis?
`y`	Apply changes to the y axis?
`minor`	Add minor gridlines in addition to major?
`minor.only`	Remove only the minor gridlines?

center *Mean-center vectors, data frames, and survey designs*

Description

This function is a wrapper around `gscale()` that is configured to mean-center variables without affecting the scaling of those variables.

Usage

```
center(
  data = NULL,
  vars = NULL,
  binary.inputs = "center",
  binary.factors = FALSE,
  weights = NULL
)
```

Arguments

data	A data frame or survey design. Only needed if you would like to rescale multiple variables at once. If x = NULL, all columns will be rescaled. Otherwise, x should be a vector of variable names. If x is a numeric vector, this argument is ignored.
vars	If data is a data.frame or similar, you can scale only select columns by providing a vector column names to this argument.
binary.inputs	Options for binary variables. Default is center; 0/1 keeps original scale; -0.5/0.5 rescales 0 as -0.5 and 1 as 0.5; center subtracts the mean; and full subtracts the mean and divides by 2 sd.
binary.factors	Coerce two-level factors to numeric and apply scaling functions to them? Default is FALSE.
weights	A vector of weights equal in length to x. If iterating over a data frame, the weights will need to be equal in length to all the columns to avoid errors. You may need to remove missing values before using the weights.

Details

Some more information can be found in the documentation for gscale()

Value

A transformed version of the data argument.

See Also

Other standardization: center_mod(), gscale(), scale_mod(), standardize()

Examples

```
# Standardize just the "qsec" variable in mtcars
standardize(mtcars, vars = "qsec")
```

---

center_mod *Center variables in fitted regression models*

Description

center_mod (previously known as center_lm) takes fitted regression models and mean-centers the continuous variables in the model to aid interpretation, especially in the case of models with interactions. It is a wrapper to scale_mod.

Usage

```
center_mod(
  model,
  binary.inputs = "0/1",
  center.response = FALSE,
  data = NULL,
  apply.weighted.contrasts = getOption("jtools-weighted.contrasts", FALSE),
  ...
)
```

Arguments

`model`	A regression model of type `lm`, `glm`, or `svyglm`; others may work as well but have not been tested.
`binary.inputs`	Options for binary variables. Default is `0/1`; `0/1` keeps original scale; `-0.5,0.5` rescales 0 as -0.5 and 1 as 0.5; `center` subtracts the mean; and `full` treats them like other continuous variables.
`center.response`	Should the response variable also be centered? Default is `FALSE`.
`data`	If you provide the data used to fit the model here, that data frame is used to re-fit the model instead of the `stats::model.frame()` of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.
`apply.weighted.contrasts`	Factor variables cannot be scaled, but you can set the contrasts such that the intercept in a regression model will reflect the true mean (assuming all other variables are centered). If set to TRUE, the argument will apply weighted effects coding to all factors. This is similar to the R default effects coding, but weights according to how many observations are at each level. An adapted version of `contr.wec()` from the `wec` package is used to do this. See that package's documentation and/or Grotenhuis et al. (2016) for more info.
`...`	Arguments passed on to `gscale()`.

Details

This function will mean-center all continuous variables in a regression model for ease of interpretation, especially for those models that have interaction terms. The mean for `svyglm` objects is calculated using `svymean`, so reflects the survey-weighted mean. The weight variables in `svyglm` are not centered, nor are they in other `lm` family models.

This function re-estimates the model, so for large models one should expect a runtime equal to the first run.

Value

The functions returns a `lm` or `glm` object, inheriting from whichever class was supplied.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

Bauer, D. J., & Curran, P. J. (2005). Probing interactions in fixed and multilevel regression: Inferential and graphical techniques. *Multivariate Behavioral Research*, *40*(3), 373-400.

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analyses for the behavioral sciences* (3rd ed.). Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

See Also

`sim_slopes` performs a simple slopes analysis.

`interact_plot` creates attractive, user-configurable plots of interaction models.

Other standardization: `center()`, `gscale()`, `scale_mod()`, `standardize()`

Examples

```
fit <- lm(formula = Murder ~ Income * Illiteracy,
          data = as.data.frame(state.x77))
fit_center <- center_mod(fit)

# With weights
fitw <- lm(formula = Murder ~ Income * Illiteracy,
           data = as.data.frame(state.x77),
           weights = Population)
fitw_center <- center_mod(fitw)

# With svyglm
if (requireNamespace("survey")) {
library(survey)
data(api)
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw,
                    data = apistrat, fpc =~ fpc)
regmodel <- svyglm(api00 ~ ell * meals, design = dstrat)
regmodel_center <- center_mod(regmodel)
}
```

`effect_plot` *Plot simple effects in regression models*

Description

`effect_plot()` plots regression paths. The plotting is done with `ggplot2` rather than base graphics, which some similar functions use.

Usage

```
effect_plot(
  model,
  pred,
  pred.values = NULL,
  centered = "all",
  plot.points = FALSE,
  interval = FALSE,
  data = NULL,
  at = NULL,
  int.type = c("confidence", "prediction"),
  int.width = 0.95,
  outcome.scale = "response",
  robust = FALSE,
  cluster = NULL,
  vcov = NULL,
  set.offset = 1,
  x.label = NULL,
  y.label = NULL,
  pred.labels = NULL,
  main.title = NULL,
  colors = "black",
  line.colors = colors,
  line.thickness = 1.1,
  point.size = 1.5,
  point.alpha = 0.6,
  jitter = 0,
  rug = FALSE,
  rug.sides = "lb",
  force.cat = FALSE,
  cat.geom = c("point", "line", "bar"),
  cat.interval.geom = c("errorbar", "linerange"),
  cat.pred.point.size = 3.5,
  partial.residuals = FALSE,
  color.class = colors,
  ...
)
```

Arguments

model	A regression model. The function is tested with lm, glm, svyglm, merMod, rq, brmsfit, stanreg models. Models from other classes may work as well but are not officially supported. The model should include the interaction of interest.
pred	The name of the predictor variable involved in the interaction. This can be a bare name or string. Note that it is evaluated using rlang, so programmers can use the !! syntax to pass variables instead of the verbatim names.
pred.values	Values of pred to use instead of the equi-spaced series by default (for continuous variables) or all unique values (for non-continuous variables).

centered	A vector of quoted variable names that are to be mean-centered. If "all", all non-focal predictors are centered. You may instead pass a character vector of variables to center. User can also use "none" to base all predictions on variables set at 0. The response variable, pred, weights, and offset variables are never centered.
plot.points	Logical. If TRUE, plots the actual data points as a scatterplot on top of the interaction lines. The color of the dots will be based on their moderator value.
interval	Logical. If TRUE, plots confidence/prediction intervals around the line using geom_ribbon.
data	Optional, default is NULL. You may provide the data used to fit the model. This can be a better way to get mean values for centering and can be crucial for models with variable transformations in the formula (e.g., log(x)) or polynomial terms (e.g., poly(x, 2)). You will see a warning if the function detects problems that would likely be solved by providing the data with this argument and the function will attempt to retrieve the original data from the global environment.
at	If you want to manually set the values of other variables in the model, do so by providing a named list where the names are the variables and the list values are vectors of the values. This can be useful especially when you are exploring interactions or other conditional predictions.
int.type	Type of interval to plot. Options are "confidence" or "prediction". Default is confidence interval.
int.width	How large should the interval be, relative to the standard error? The default, .95, corresponds to roughly 1.96 standard errors and a .05 alpha level for values outside the range. In other words, for a confidence interval, .95 is analogous to a 95% confidence interval.
outcome.scale	For nonlinear models (i.e., GLMs), should the outcome variable be plotted on the link scale (e.g., log odds for logit models) or the original scale (e.g., predicted probabilities for logit models)? The default is "response", which is the original scale. For the link scale, which will show straight lines rather than curves, use "link".
robust	Should robust standard errors be used to find confidence intervals for supported models? Default is FALSE, but you should specify the type of sandwich standard errors if you'd like to use them (i.e., "HC0", "HC1", and so on). If TRUE, defaults to "HC3" standard errors.
cluster	For clustered standard errors, provide the column name of the cluster variable in the input data frame (as a string). Alternately, provide a vector of clusters.
vcov	Optional. You may supply the variance-covariance matrix of the coefficients yourself. This is useful if you are using some method for robust standard error calculation not supported by the **sandwich** package.
set.offset	For models with an offset (e.g., Poisson models), sets an offset for the predicted values. All predicted values will have the same offset. By default, this is set to 1, which makes the predicted values a proportion. See details for more about offset support.
x.label	A character object specifying the desired x-axis label. If NULL, the variable name is used.

`y.label` A character object specifying the desired x-axis label. If `NULL`, the variable name is used.

`pred.labels` A character vector of labels for categorical predictors. If `NULL`, the default, the factor labels are used.

`main.title` A character object that will be used as an overall title for the plot. If `NULL`, no main title is used.

`colors` See jtools_colors for details on the types of arguments accepted. Default is "black". This affects the coloration of the line as well as confidence intervals and points unless you give a different argument to `line.color`.

`line.colors` See jtools_colors for details on the types of arguments accepted. Default is `colors`. This affects the coloration of the line as well as confidence intervals, but not the points.

`line.thickness` How thick should the plotted lines be? Default is 1.1; ggplot's default is 1.

`point.size` What size should be used for observed data when `plot.points` is TRUE? Default is 1.5.

`point.alpha` What should the `alpha` aesthetic for plotted points of observed data be? Default is 0.6, and it can range from 0 (transparent) to 1 (opaque).

`jitter` How much should `plot.points` observed values be "jittered" via `ggplot2::position_jitter()`? When there are many points near each other, jittering moves them a small amount to keep them from totally overlapping. In some cases, though, it can add confusion since it may make points appear to be outside the boundaries of observed values or cause other visual issues. Default is 0, but try various small values (e.g., 0.1) and increase as needed if your points are overlapping too much. If the argument is a vector with two values, then the first is assumed to be the jitter for width and the second for the height.

`rug` Show a rug plot in the margins? This uses `ggplot2::geom_rug()` to show the distribution of the predictor (top/bottom) and/or response variable (left/right) in the original data. Default is FALSE.

`rug.sides` On which sides should rug plots appear? Default is "lb", meaning both left and bottom. "t" and/or "b" show the distribution of the predictor while "l" and/or "r" show the distribution of the response.

`force.cat` Force the continuous `pred` to be treated as categorical? default is FALSE, but this can be useful for things like dummy 0/1 variables.

`cat.geom` If `pred` is categorical (or `force.cat` is TRUE), what type of plot should this be? There are several options here since the best way to visualize categorical interactions varies by context. Here are the options:

- `"point"`: The default. Simply plot the point estimates. You may want to use `point.shape = TRUE` with this and you should also consider `interval = TRUE` to visualize uncertainty.
- `"line"`: This connects observations across levels of the `pred` variable with a line. This is a good option when the `pred` variable is ordinal (ordered). You may still consider `point.shape = TRUE` and `interval = TRUE` is still a good idea.

- "bar": A bar chart. Some call this a "dynamite plot." Many applied researchers advise against this type of plot because it does not represent the distribution of the observed data or the uncertainty of the predictions very well. It is best to at least use the interval = TRUE argument with this geom.

cat.interval.geom
: For categorical by categorical interactions. One of "errorbar" or "linerange". If the former, ggplot2::geom_errorbar() is used. If the latter, ggplot2::geom_linerange() is used.

cat.pred.point.size
: (for categorical pred) If TRUE and geom is "point" or "line", sets the size of the predicted points. Default is 3.5. Note the distinction from point.size, which refers to the observed data points.

partial.residuals
: Instead of plotting the observed data, you may plot the partial residuals (controlling for the effects of variables besides pred).

color.class
: Deprecated. Now known as colors.

...
: extra arguments passed to make_predictions()

Details

This function provides a means for plotting effects for the purpose of exploring regression estimates. You must have the package ggplot2 installed to benefit from these plotting functions.

By default, all numeric predictors other than the one specified in the pred argument are mean-centered, which usually produces more intuitive plots. This only affects the y-axis in linear models, but may be especially important/influential in non-linear/generalized linear models.

This function supports nonlinear and generalized linear models and by default will plot them on their original scale (outcome.scale = "response").

While mixed effects models from lme4 are supported, only the fixed effects are plotted. lme4 does not provide confidence intervals, so they are not supported with this function either.

Note: to use transformed predictors, e.g., log(x), or polynomials, e.g., poly(x, 2), provide the raw variable name (x) to the pred = argument. You will need to input the data frame used to fit the model with the data = argument.

Value

The functions returns a ggplot object, which can be treated like a user-created plot and expanded upon as such.

Author(s)

Jacob Long <jacob.long@sc.edu>

See Also

interact_plot from the interactions package plots interaction effects, producing plots like this function but with separate lines for different levels of a moderator. cat_plot from interactions does the same for categorical by categorical interactions.

Examples

```
# Using a fitted lm model
states <- as.data.frame(state.x77)
states$HSGrad <- states$`HS Grad`
fit <- lm(Income ~ HSGrad + Murder,
  data = states)
effect_plot(model = fit, pred = Murder)

# Using polynomial predictor, plus intervals
fit <- lm(accel ~ poly(mag,3) + dist, data = attenu)
effect_plot(fit, pred = mag, interval = TRUE,
  int.type = "confidence", int.width = .8, data = attenu) # note data arg.

# With svyglm
if (requireNamespace("survey")) {
library(survey)
data(api)
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw,
                    data = apistrat, fpc = ~fpc)
regmodel <- svyglm(api00 ~ ell + meals, design = dstrat)
effect_plot(regmodel, pred = ell, interval = TRUE)
}

# With lme4
## Not run:
library(lme4)
data(VerbAgg)
mv <- glmer(r2 ~ Anger + mode + (1 | item), data = VerbAgg,
            family = binomial,
            control = glmerControl("bobyqa"))
effect_plot(mv, pred = Anger)

## End(Not run)
```

`export_summs` *Export regression summaries to tables*

Description

This function allows users to use the features of `summ()` (e.g., standardization, robust standard errors) in the context of shareable HTML, LaTeX, and Microsoft Word tables. It relies heavily on `huxtable::huxreg()` to do the table formatting. This is particularly useful for putting the results of multiple models into a single table.

Usage

```
export_summs(
  ...,
```

```
  error_format = "({std.error})",
  error_pos = c("below", "right", "same"),
  ci_level = 0.95,
  statistics = NULL,
  model.names = NULL,
  coefs = NULL,
  to.file = NULL,
  file.name = NULL
)
```

Arguments

`...`	At minimum, a regression object(s). See details for more arguments.
`error_format`	Which of standard error, confidence intervals, test statistics, or p values should be used to express uncertainty of estimates for regression coefficients? See details for more info. Default: "(std.error)"
`error_pos`	Where should the error statistic defined in `error_style` be placed relative to the coefficient estimate? Default: "below"
`ci_level`	If reporting confidence intervals, what should the confidence level be? By default, it is .95 if confidence intervals are requested in `error_format`.
`statistics`	Which model summary statistics should be included? See `huxreg` for more on usage. The default for this function depends on the model type. See details for more on the defaults by model type.
`model.names`	If you want to give your model(s) names at the top of each column, provide them here as a character vector. Otherwise, they will just be labeled by number. Default: NULL
`coefs`	If you want to include only a subset of the coefficients in the table, specify them here in a character vector. If you want the table to show different names for the coefficients, give a named vector where the names are the preferred coefficient names. See details for more.
`to.file`	Export the table to a Microsoft Word, PDF, or HTML document? This functionality relies on `huxtable`'s `quick_` functions (`huxtable::quick_docx()`, `huxtable::quick_pdf()`, `huxtable::quick_html()`, `huxtable::quick_xlsx()`). Acceptable arguments are "Word" or "docx" (equivalent), "pdf", "html", or "xlsx". All are case insensitive. Default is NULL, meaning the table is not saved.
`file.name`	File name with (optionally) file path to save the file. Ignored if `to.file` is FALSE. Default: NULL

Details

There are many optional parameters not documented above. Any argument that you would want to pass to `summ()`, for instance, will be used. Of particular interest may be the robust and scale arguments. Note that some `summ` arguments may not have any bearing on the table output.

The default model summary statistics reporting follows this logic:

- summ.lm = `c(N = "nobs", R2 = "r.squared")`,

- summ.glm = c(N = "nobs", AIC = "AIC", BIC = "BIC", `Pseudo R2` = "pseudo.r.squared"),
- summ.svyglm = c(N = "nobs", R2 = "r.squared"),
- summ.merMod = c(N = "nobs", AIC = "AIC", BIC = "BIC", `R2 (fixed)` = "r.squared.fixed", `R2 (total)` = "r.squared")
- summ.rq = c(N = "nobs", tau = "tau", R1 = "r.1", AIC = "AIC", BIC = "BIC")

Be sure to look at the summ() documentation for more on the calculation of these and other statistics, especially for mixed models.

If you set statistics = "all", then the statistics argument passed to huxreg will be NULL, which reports whichever model statistics are available via glance. If you want no model summary statistics, set the argument to character(0).

You have a few options for the error_format argument. You can include anything returned by broom::tidy() (see also tidy.summ()). For the most part, you will be interested in std.error (standard error), statistic (test statistic, e.g. t-value or z-value), p.value, or conf.high and conf.low, which correspond to the upper and lower bounds of the confidence interval for the estimate. Note that the default ci_level argument is .95, but you can alter that as desired.

To format the error statistics, simply put the statistics desired in curly braces wherever you want them in a character string. For example, if you want the standard error in parentheses, the argument would be "({std.error})", which is the default. Some other ideas:

- "({statistic})", which gives you the test statistic in parentheses.
- "({statistic}, p = {p.value})", which gives the test statistic followed by a "p =" p value all in parentheses. Note that you'll have to pay special attention to rounding if you do this to keep cells sufficiently narrow.
- "[{conf.low}, {conf.high}]", which gives the confidence interval in the standard bracket notation. You could also explicitly write the confidence level, e.g., "CI [{conf.low}, {conf.high}]".

For coefs, the argument is slightly different than what is default in huxreg. If you provide a named vector of coefficients, then the table will refer to the selected coefficients by the names of the vector rather than the coefficient names. For instance, if I want to include only the coefficients for the hp and mpg but have the table refer to them as "Horsepower" and "Miles/gallon", I'd provide the argument like this: c("Horsepower" = "hp", "Miles/gallon" = "mpg")

You can also pass any argument accepted by the huxtable::huxreg() function. A few that are likely to be oft-used are documented above, but visit huxreg's documentation for more info.

For info on converting the huxtable::huxtable() object to HTML or LaTeX, see huxtable's documentation.

Value

A huxtable.

See Also

summ

huxreg

Examples

```
states <- as.data.frame(state.x77)
fit1 <- lm(Income ~ Frost, data = states)
fit2 <- lm(Income ~ Frost + Illiteracy, data = states)
fit3 <- lm(Income ~ Frost + Illiteracy + Murder, data = states)

if (requireNamespace("huxtable")) {
  # Export all 3 regressions with "Model #" labels,
  # standardized coefficients, and robust standard errors
  export_summs(fit1, fit2, fit3,
               model.names = c("Model 1","Model 2","Model 3"),
               coefs = c("Frost Days" = "Frost",
                         "% Illiterate" = "Illiteracy",
                         "Murder Rate" = "Murder"),
               scale = TRUE, robust = TRUE)
}
```

---

get_colors *Get colors for plotting functions*

Description

This is a helper function that provides hex color codes for `jtools`, `interactions`, and perhaps other packages.

Usage

```
get_colors(colors, num.colors = 1, reverse = FALSE, gradient = FALSE)
```

Arguments

`colors`	The name of the desired color class or a vector of colors. See details of jtools_colors.
`num.colors`	How many colors should be returned? Default is 1.
`reverse`	Should the colors be returned in reverse order, compared to normal? Default is FALSE.
`gradient`	Return endpoints for a gradient? Default is FALSE. If TRUE, `num.colors` is ignored.

get_formula *Retrieve formulas from model objects*

Description

This function is primarily an internal helper function in `jtools` and related packages to standardize the different types of formula objects used by different types of models.

Usage

```
get_formula(model, ...)

## Default S3 method:
get_formula(model, ...)

## S3 method for class 'brmsfit'
get_formula(model, resp = NULL, dpar = NULL, ...)

## S3 method for class 'panelmodel'
get_formula(model, ...)
```

Arguments

`model`	The fitted model object.
`...`	Ignored.
`resp`	For `brmsfit` objects, the response variable for which the formula is desired. `brmsfit` objects may have multiple formulas, so this selects a particular one. If `NULL`, the first formula is chosen (unless `dpar` is specified).
`dpar`	For `brmsfit` objects, the distributional variable for which the formula is desired. If `NULL`, no distributional parameter is used. If there are multiple responses with distributional parameters, then `resp` should be specified or else the first formula will be used by default.

Value

A `formula` object.

Examples

```
data(mtcars)
fit <- lm(mpg ~ cyl, data = mtcars)
get_formula(fit)
```

get_offset_name *Utility functions for generating model predictions*

Description

These functions get information and data from regression models.

Usage

```
get_offset_name(model)

get_weights(model, data)

get_data(model, formula = NULL, warn = TRUE, ...)

get_response_name(model, ...)
```

Arguments

model	The model (e.g., lm, glm, merMod, svyglm)
data	For get_weights(), the data used to fit the model.
formula	The formula for model, if desired. Otherwise get_formula() is called.
warn	For get_data(), should there be a warning when model.frame() won't work because of variable transformations? Default is TRUE but this may not be desired when get_data() is used inside of another function or used multiple times.
...	Arguments passed to get_formula()

Value

- get_data(): The data used to fit the model.
- get_response_name(): The name of the response variable.
- get_offset_name(): The name of the offset variable.
- get_weights(): A list with weights_name, the name of the weighting variable, and weights, the weights themselves (or all 1 when there are no weights).

`get_robust_se` *Calculate robust standard errors and produce coefficient tables*

Description

This function wraps around several **sandwich** and **lmtest** functions to calculate robust standard errors and returns them in a useful format.

Usage

```
get_robust_se(
  model,
  type = "HC3",
  cluster = NULL,
  data = model.frame(model),
  vcov = NULL
)
```

Arguments

`model`	A regression model, preferably of class `lm` or `glm`
`type`	One of `"HC3"`, `"const"`, `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `sandwich::vcovHC()` for some more details on these choices. Note that some of these do not work for clustered standard errors (see sandwich::vcovCL()]).
`cluster`	If you want clustered standard errors, either a string naming the column in `data` that represents the clusters or a vector of clusters that is the same length as the number of rows in `data`.
`data`	The data used to fit the model. Default is to just get the `model.frame` from `model`.
`vcov`	You may provide the variance-covariance matrix yourself and this function will just calculate standard errors, etc. based on that. Default is NULL.

Value

A list with the following:

- `coefs`: a coefficient table with the estimates, standard errors, t-statistics, and p-values from `lmtest`.
- `ses`: The standard errors from `coefs`.
- `ts`: The t-statistics from `coefs`.
- `ps`: The p-values from `coefs`.
- `type`: The argument to `robust`.
- `use_cluster`: `TRUE` or `FALSE` indicator of whether clusters were used.
- `cluster`: The clusters or name of cluster variable used, if any.
- `vcov`: The robust variance-covariance matrix.

gscale *Scale and/or center data, including survey designs*

Description

gscale standardizes variables by dividing them by 2 standard deviations and mean-centering them by default. It contains options for handling binary variables separately. gscale() is a fork of rescale from the arm package—the key feature difference is that gscale() will perform the same functions for variables in svydesign objects. gscale() is also more user-friendly in that it is more flexible in how it accepts input.

Usage

```
gscale(
  data = NULL,
  vars = NULL,
  binary.inputs = "center",
  binary.factors = FALSE,
  n.sd = 2,
  center.only = FALSE,
  scale.only = FALSE,
  weights = NULL,
  apply.weighted.contrasts = getOption("jtools-weighted.contrasts", FALSE),
  x = NULL,
  messages = FALSE
)
```

Arguments

data	A data frame or survey design. Only needed if you would like to rescale multiple variables at once. If x = NULL, all columns will be rescaled. Otherwise, x should be a vector of variable names. If x is a numeric vector, this argument is ignored.
vars	If data is a data.frame or similar, you can scale only select columns by providing a vector column names to this argument.
binary.inputs	Options for binary variables. Default is center; 0/1 keeps original scale; -0.5/0.5 rescales 0 as -0.5 and 1 as 0.5; center subtracts the mean; and full subtracts the mean and divides by 2 sd.
binary.factors	Coerce two-level factors to numeric and apply scaling functions to them? Default is FALSE.
n.sd	By how many standard deviations should the variables be divided by? Default for gscale is 2, like arm's rescale. 1 is the more typical standardization scheme.
center.only	A logical value indicating whether you would like to mean -center the values, but not scale them.
scale.only	A logical value indicating whether you would like to scale the values, but not mean-center them.

`weights` A vector of weights equal in length to `x`. If iterating over a data frame, the weights will need to be equal in length to all the columns to avoid errors. You may need to remove missing values before using the weights.

`apply.weighted.contrasts` Factor variables cannot be scaled, but you can set the contrasts such that the intercept in a regression model will reflect the true mean (assuming all other variables are centered). If set to TRUE, the argument will apply weighted effects coding to all factors. This is similar to the R default effects coding, but weights according to how many observations are at each level. An adapted version of `contr.wec()` from the `wec` package is used to do this. See that package's documentation and/or Grotenhuis et al. (2016) for more info.

`x` Deprecated. Pass numeric vectors to `data`. Pass vectors of column names to `vars`.

`messages` Print messages when variables are not processed due to being non-numeric or all missing? Default is FALSE.

Details

This function is adapted from the `rescale` function of the `arm` package. It is named `gscale()` after the popularizer of this scaling method, Andrew **G**elman. By default, it works just like `rescale`. But it contains many additional options and can also accept multiple types of input without breaking a sweat.

Only numeric variables are altered when in a data.frame or survey design. Character variables, factors, etc. are skipped.

For those dealing with survey data, if you provide a `survey.design` object you can rest assured that the mean-centering and scaling is performed with help from the `svymean()` and `svyvar()` functions, respectively. It was among the primary motivations for creating this function. `gscale()` will not center or scale the weights variables defined in the survey design unless the user specifically requests them in the `x =` argument.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

Gelman, A. (2008). Scaling regression inputs by dividing by two standard deviations. *Statistics in Medicine*, *27*, 2865–2873. http://www.stat.columbia.edu/~gelman/research/published/standardizing7.pdf

Grotenhuis, M. te, Pelzer, B., Eisinga, R., Nieuwenhuis, R., Schmidt-Catran, A., & Konig, R. (2017). When size matters: Advantages of weighted effect coding in observational studies. *International Journal of Public Health*, *62*, 163–167. https://doi.org/10.1007/s00038-016-0901-1 (open access)

See Also

`j_summ` is a replacement for the `summary` function for regression models. On request, it will center and/or standardize variables before printing its output.

Other standardization: center_mod(), center(), scale_mod(), standardize()

Examples

```
x <- rnorm(10, 2, 1)
x2 <- rbinom(10, 1, .5)

# Basic use
gscale(x)
# Normal standardization
gscale(x, n.sd = 1)
# Scale only
gscale(x, scale.only = TRUE)
# Center only
gscale(x, center.only = TRUE)
# Binary inputs
gscale(x2, binary.inputs = "0/1")
gscale(x2, binary.inputs = "full") # treats it like a continous var
gscale(x2, binary.inputs = "-0.5/0.5") # keep scale, center at zero
gscale(x2, binary.inputs = "center") # mean center it

# Data frame as input
# loops through each numeric column
gscale(data = mtcars, binary.inputs = "-0.5/0.5")

# Specified vars in data frame
gscale(mtcars, vars = c("hp", "wt", "vs"), binary.inputs = "center")

# Weighted inputs

wts <- runif(10, 0, 1)
gscale(x, weights = wts)
# If using a weights column of data frame, give its name
mtcars$weights <- runif(32, 0, 1)
gscale(mtcars, weights = weights) # will skip over mtcars$weights
# If using a weights column of data frame, can still select variables
gscale(mtcars, vars = c("hp", "wt", "vs"), weights = weights)

# Survey designs
if (requireNamespace("survey")) {
  library(survey)
  data(api)
  ## Create survey design object
  dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw,
                      data = apistrat, fpc=~fpc)
  # Creating test binary variable
  dstrat$variables$binary <- rbinom(200, 1, 0.5)

  gscale(data = dstrat, binary.inputs = "-0.5/0.5")
  gscale(data = dstrat, vars = c("api00","meals","binary"),
         binary.inputs = "-0.5/0.5")
}
```

interact_plot *Deprecated interaction functions*

Description

These functions are now part of the `interactions` package.

Usage

```
interact_plot(...)

cat_plot(...)

sim_slopes(...)

johnson_neyman(...)

probe_interaction(...)
```

Arguments

`...`	arguments are ignored

jtools_colors *Color palettes in* `jtools` *functions*

Description

`jtools` combines several options into the `colors` argument in plotting functions.

Details

The argument to `colors` in functions like `effect_plot`, `plot_coefs`, and others is very flexible but may also cause confusion.

If you provide an argument of length 1, it is assumed that you are naming a palette. `jtools` provides 6 color palettes design for qualitative data. 4 of the 6 are based on Paul Tol's suggestions (see references) and are meant to both optimize your ability to quickly differentiate the colors and to be distinguishable to colorblind people. These are called `"Qual1"`, `"Qual2"`, `"Qual3"`, `"CUD"`, `"CUD Bright"`, and `"Rainbow"`. Each of the "Qual" schemes comes from Paul Tol. "Rainbow" is Paul Tol's compromise rainbow color scheme that is fairly differentiable for colorblind people and when rendered in grayscale. `"CUD Bright"` is a brightened and reordered version of Okabe and Ito's

suggestions for 'Color Universal Design' while "CUD" is their exact scheme (see references). "CUD Bright" is the default for qualitative scales in jtools functions.

You may also provide any color palette supported by RColorBrewer. See all of those options at RColorBrewer::brewer.pal()'s documentation. If you provide one of RColorBrewer's sequential palettes, like "Blues", jtools automatically requests one more color than needed from brewer.pal and then drops the lightest color. My experience is that those scales tend to give one color that is too light to easily differentiate against a white background.

For **gradients**, you can use any of the RColorBrewer sequential palette names and get comparable results on a continuous scale. There are also some jtools-specific gradient schemes: "blue", "blue2", "green", "red", "purple", "seagreen". If you want something a little non-standard, I'd suggest taking a look at "blue2" or "seagreen".

Lastly, you may provide colors by name. This must be a vector of the same length as whatever it is the colors will correspond to. The format must be one understood by ggplot2's manual scale functions. This basically means it needs to be in hex format (e.g., "#000000") or one of the many names R understands (e.g., "red"; use colors() to see all of those options).

References

Paul Tol's site is what is used to derive 4 of the 6 jtools-specific qualitative palettes: https://personal.sron.nl/~pault/

Okabe and Ito's palette inspired "CUD Bright", though "CUD Bright" is not exactly the same. "CUD" is the same. See https://web.archive.org/web/20190216090108/jfly.iam.u-tokyo.ac.jp/color/ for more.

---

knit_print.summ.lm *knitr methods for summ*

Description

There's no reason for end users to utilize these functions, but CRAN requires it to be documented.

Usage

```
knit_print.summ.lm(x, options = NULL, ...)

knit_print.summ.glm(x, options = NULL, ...)

knit_print.summ.svyglm(x, options = NULL, ...)

knit_print.summ.merMod(x, options = NULL, ...)

knit_print.summ.rq(x, options = NULL, ...)
```

Arguments

x	The summ object
options	Chunk options.
...	Ignored.

---

make_new_data *Make new data for generating predicted data from regression models.*

Description

This is a convenience function that helps automate the process of generating predicted data from regression model from a predictor(s). It is designed to give you the data frame for the predict method's newdata argument.

Usage

```
make_new_data(
  model,
  pred,
  pred.values = NULL,
  at = NULL,
  data = NULL,
  center = TRUE,
  set.offset = NULL,
  num.preds = 100,
  ...
)
```

Arguments

model	The model (e.g., lm, glm, merMod, svyglm)
pred	The name of the focal predictor as a string. This is the variable for which, if you are plotting, you'd likely have along the x-axis (with the dependent variable as the y-axis).
pred.values	The values of pred you want to include. Default is NULL, which means a sequence of equi-spaced values over the range of a numeric predictor or each level of a non-numeric predictor.
at	If you want to manually set the values of other variables in the model, do so by providing a named list where the names are the variables and the list values are vectors of the values. This can be useful especially when you are exploring interactions or other conditional predictions.
data	The data frame used in fitting the model. Default is NULL, in which case the data will be retrieved via model.frame or, if there are variable transformations in the formula, by looking in the environment for the data.

center	Set numeric covariates to their mean? Default is TRUE. You may also just provide a vector of names (as strings) of covariates to center. Note that for svyglm models, the survey-weighted means are used. For models with weights, these are weighted means.
set.offset	If the model has an offset, the value to use for the offset variable. Default is NULL, in which case the median of the offset variable is used.
num.preds	The number of predictions to generate. Default is 100. Ignored if pred.values is not NULL.
...	Extra arguments passed to get_formula()

Details

Please bear in mind that this does not generate the predictions. You will need to do that with a predict function for your model or another interface, such as the prediction package's titular function.

Value

A data frame.

Examples

```
fit <- lm(Income ~ Frost + Illiteracy + Murder, data = as.data.frame(state.x77))
# Basic use
new_data <- make_new_data(fit, pred = "Frost")
# Set covariate to specific value
new_data <- make_new_data(fit, pred = "Frost", at = list(Murder = 5))
# Set covariate to several specific values
new_data <- make_new_data(fit, pred = "Frost", at = list(Murder = c(5, 10, 15)))
```

---

make_predictions *Generate predicted data for plotting results of regression models*

Description

This is an alternate interface to the underlying tools that make up effect_plot() as well as interactions::interact_plot() and interactions::cat_plot() from the interactions package. make_predictions() creates the data to be plotted and adds information to the original data to make it more amenable for plotting with the predicted data.

Usage

```
make_predictions(model, ...)

## Default S3 method:
make_predictions(
  model,
  pred,
  pred.values = NULL,
  at = NULL,
  data = NULL,
  center = TRUE,
  interval = TRUE,
  int.type = c("confidence", "prediction"),
  int.width = 0.95,
  outcome.scale = "response",
  robust = FALSE,
  cluster = NULL,
  vcov = NULL,
  set.offset = NULL,
  new_data = NULL,
  return.orig.data = FALSE,
  partial.residuals = FALSE,
  ...
)
```

Arguments

model	The model (e.g., lm, glm, merMod, svyglm)
...	Ignored.
pred	The name of the focal predictor as a string. This is the variable for which, if you are plotting, you'd likely have along the x-axis (with the dependent variable as the y-axis).
pred.values	The values of pred you want to include. Default is NULL, which means a sequence of equi-spaced values over the range of a numeric predictor or each level of a non-numeric predictor.
at	If you want to manually set the values of other variables in the model, do so by providing a named list where the names are the variables and the list values are vectors of the values. This can be useful especially when you are exploring interactions or other conditional predictions.
data	Optional, default is NULL. You may provide the data used to fit the model. This can be a better way to get mean values for centering and can be crucial for models with variable transformations in the formula (e.g., log(x)) or polynomial terms (e.g., poly(x, 2)). You will see a warning if the function detects problems that would likely be solved by providing the data with this argument and the function will attempt to retrieve the original data from the global environment.

`center` Set numeric covariates to their mean? Default is TRUE. You may also just provide a vector of names (as strings) of covariates to center. Note that for `svyglm` models, the survey-weighted means are used. For models with weights, these are weighted means.

`interval` Logical. If `TRUE`, plots confidence/prediction intervals around the line using `geom_ribbon`.

`int.type` Type of interval to plot. Options are "confidence" or "prediction". Default is confidence interval.

`int.width` How large should the interval be, relative to the standard error? The default, .95, corresponds to roughly 1.96 standard errors and a .05 alpha level for values outside the range. In other words, for a confidence interval, .95 is analogous to a 95% confidence interval.

`outcome.scale` For nonlinear models (i.e., GLMs), should the outcome variable be plotted on the link scale (e.g., log odds for logit models) or the original scale (e.g., predicted probabilities for logit models)? The default is `"response"`, which is the original scale. For the link scale, which will show straight lines rather than curves, use `"link"`.

`robust` Should robust standard errors be used to find confidence intervals for supported models? Default is FALSE, but you should specify the type of sandwich standard errors if you'd like to use them (i.e., `"HC0"`, `"HC1"`, and so on). If `TRUE`, defaults to `"HC3"` standard errors.

`cluster` For clustered standard errors, provide the column name of the cluster variable in the input data frame (as a string). Alternately, provide a vector of clusters.

`vcov` Optional. You may supply the variance-covariance matrix of the coefficients yourself. This is useful if you are using some method for robust standard error calculation not supported by the **sandwich** package.

`set.offset` For models with an offset (e.g., Poisson models), sets an offset for the predicted values. All predicted values will have the same offset. By default, this is set to 1, which makes the predicted values a proportion. See details for more about offset support.

`new_data` If you would prefer to generate your own hypothetical (or not hypothetical) data rather than have the function make a call to `make_new_data()`, you can provide it.

`return.orig.data` Instead of returning a just the predicted data frame, should the original data be returned as well? If so, then a list will be return with both the predicted data (as the first element) and the original data (as the second element). Default is FALSE.

`partial.residuals` If `return.orig.data` is TRUE, should the observed dependent variable be replaced with the partial residual? This makes a call to `partialize()`, where you can find more details.

`md_table` *Print attractive data frames in the console*

Description

This function takes data frame input and prints to the console as an ASCII/markdown table for better readability.

Usage

```
md_table(
  x,
  format = getOption("md_table_format", "grid"),
  digits = getOption("jtools-digits", 2),
  sig.digits = TRUE,
  row.names = rownames(x),
  col.names = colnames(x),
  align = NULL
)
```

Arguments

`x`	A data frame or matrix.
`format`	The style, which can be one of the following: "multiline", "grid", "simple" (also "pandoc"), "rmarkdown" (also "markdown"). Default: "markdown"
`digits`	How many digits to print for numbers. Default: 2
`sig.digits`	Should each number be printed with `digits` number of digits or only when there are at least that many significant digits? Default is TRUE, meaning only print `digits` number of *significant* digits.
`row.names`	if FALSE, row names are suppressed. A character vector of row names can also be specified here. By default, row names are included if rownames(t) is neither NULL nor identical to 1:nrow(x).
`col.names`	a character vector of column names to be used in the table
`align`	Column alignment: a character vector consisting of ''l" (left), ''c" (center) and/or ''r" (right). By default or if 'align = NULL', numeric columns are right-aligned, and other columns are left-aligned.

`movies` *Data about movies*

Description

A dataset containing information about films, how popular they were, and the extent to which they feature women.

Usage

```
movies
```

Format

A data frame with 841 rows and 24 variables:

title The movie's title

year The year of the movie's US theatrical release

release_date The exact date of the movie's US theatrical release

runtime The length of the movie in hours

genre5 The movie's primary genre per IMDB, fit into one of 5 broad categories

genre_detailed The verbatim genre description per IMDB

rated The movie's MPA rating (G, PG, PG-13, R, or NC-17) as an ordered factor

director The name of the movie's director(s)

writer The name of the movie's screenwriter(s)

actors A comma-separated string of leading actors in the film

language The movie's language(s), per IMDB

country The country(ies) in which the movie was produced

metascore The movie's score on MetaCritic, ranging from 0 to 100

imdb_rating The movie's rating on IMDB, ranging from 0 to 10

imdb_votes The number of users who submitted a rating on IMDB

imdb_id The unique identifier for the movie at IMDB

studio The studio(s) who produced the movie

bechdel_binary A logical indicating whether the movie passed the Bechdel test

bechdel_ordinal A more granular measure of the bechdel test, indicating not just whether the movie passed or failed but how close it got to passing if it did fail

us_gross The movie's US gross in 2013 US dollars

int_gross The movie's international gross in 2013 US dollars

budget The movie's budget in 2013 US dollars

men_lines The proportion of spoken lines that were spoken by male characters

lines_data The raw data used to calculate `men_lines`; see Source for more information

Source

These data are aggregated from several sources. Metadata is gathered from IMDB. Other information, particularly about the lines, is collected from The Pudding. The data regarding the Bechdel Test, as well as about finances, comes from FiveThirtyEight and its associated R package (`fivethirtyeight` and its dataset, `bechdel`).

---

num_print *Numbering printing with signed zeroes and trailing zeroes*

Description

This function will print exactly the amount of digits requested as well as signed zeroes when appropriate (e.g, `-0.00`).

Usage

```
num_print(x, digits = getOption("jtools-digits", 2), format = "f")
```

Arguments

`x`	The number(s) to print
`digits`	Number of digits past the decimal to print
`format`	equal to `"d"` (for integers), `"f"`, `"e"`, `"E"`, `"g"`, `"G"`, `"fg"` (for reals). Default is `"f"`

---

partialize *Adjust observed data for partial residuals plots*

Description

This function is designed to facilitate the creation of partial residual plots, in which you can plot observed data alongside model predictions. The difference is instead of the *actual* observed data, the outcome variable is adjusted for the effects of the covariates.

Usage

```
partialize(model, ...)

## Default S3 method:
partialize(
  model,
  vars = NULL,
  data = NULL,
  at = NULL,
```

```
  center = TRUE,
  scale = c("response", "link"),
  set.offset = 1,
  ...
)
```

Arguments

model	A regression model.
...	Ignored.
vars	The variable(s) to *not* adjust for, as a string (or vector of strings). If I want to show the effect of x adjusting for the effect of z, then I would make "x" the vars argument.
data	Optionally, provide the data used to fit the model (or some other data frame with the same variables). Otherwise, it will be retrieved from the model or the global environment.
at	If you want to manually set the values of other variables in the model, do so by providing a named list where the names are the variables and the list values are vectors of the values. This can be useful especially when you are exploring interactions or other conditional predictions.
center	Set numeric covariates to their mean? Default is TRUE. You may also just provide a vector of names (as strings) of covariates to center. Note that for svyglm models, the survey-weighted means are used. For models with weights, these are weighted means.
scale	For GLMs, should the outcome variable be returned on the link scale or response scale? Default is "response".
set.offset	For models with an offset (e.g., Poisson models), sets an offset for the predicted values. All predicted values will have the same offset. By default, this is set to 1, which makes the predicted values a proportion. See details for more about offset support.

Details

The main use for working with partial residuals rather than the observed values is to explore patterns in the model fit with respect to one or more variables while "controlling out" the effects of others. Plotting a predicted line along with observed data may make a very well-fitting model look as if it is a poor fit if a lot of variation is accounted for by variables other than the one on the x-axis.

I advise consulting Fox and Weisberg (available free) for more details on what partial residuals are. This function is designed to produce data in a similar format to effects::Effect() when that function has residuals set to TRUE and is plotted. I wanted a more modular function to produce the data separately. To be clear, the developers of the effects package have nothing to do with this function; ‘partialize“ is merely designed to replicate some of that functionality.

Value

data plus the residualized outcome variable.

References

Fox, J., & Weisberg, S. (2018). Visualizing fit and lack of fit in complex regression models with predictor effect plots and partial residuals. *Journal of Statistical Software*, *87*(9), 1–27. https://doi.org/10.18637/jss.v087.i09

`pf_sv_test` *Test whether sampling weights are needed*

Description

Use the test proposed in Pfeffermann and Sverchkov (1999) to check whether a regression model is specified correctly without weights.

Usage

```
pf_sv_test(
  model,
  data = NULL,
  weights,
  sims = 1000,
  digits = getOption("jtools-digits", default = 3)
)
```

Arguments

`model`	The fitted model, without weights
`data`	The data frame with the data fed to the fitted model and the weights
`weights`	The name of the weights column in `model`'s data frame or a vector of weights equal in length to the number of observations included in `model`.
`sims`	The number of bootstrap simulations to use in estimating the variance of the residual correlation. Default is 1000, but for publications or when computing power/time is sufficient, a higher number is better.
`digits`	An integer specifying the number of digits past the decimal to report in the output. Default is 3. You can change the default number of digits for all jtools functions with `options("jtools-digits" = digits)` where digits is the desired number.

Details

This is a test described by Pfeffermann and Sverchkov (1999) that is designed to help analysts decide whether they need to use sample weights in their regressions to avoid biased parameter estimation.

It first checks the correlation of the residuals of the model with the weights. It then uses bootstrapping to estimate the variance of the correlation, ending with a t-test of whether the correlation differs from zero. This is done for the squared residuals and cubed residuals as well. If anyone of

them are statistically significant (at whatever level you feel appropriate), it is best to do a weighted regression. Note that in large samples, a very small correlation may have a low p-value without a large bias in the unweighted regression.

References

Pfeffermann, D., & Sverchkov, M. (1999). Parametric and semi-parametric estimation of regression models fitted to survey data. *Sankhya: The Indian Journal of Statistics*, *61*. 166-186.

See Also

Other survey tools: `svycor()`, `svysd()`, `weights_tests()`, `wgttest()`

Examples

```
# Note: This is a contrived example to show how the function works,
# not a case with actual sammpling weights from a survey vendor
if (requireNamespace("boot")) {
  states <- as.data.frame(state.x77)
  set.seed(100)
  states$wts <- runif(50, 0, 3)
  fit <- lm(Murder ~ Illiteracy + Frost, data = states)
  pf_sv_test(model = fit, data = states, weights = wts, sims = 100)
}
```

---

plot_summs *Plot Regression Summaries*

Description

`plot_summs` and `plot_coefs` create regression coefficient plots with ggplot2.

Usage

```
plot_summs(
  ...,
  ci_level = 0.95,
  model.names = NULL,
  coefs = NULL,
  omit.coefs = "(Intercept)",
  inner_ci_level = NULL,
  colors = "CUD Bright",
  plot.distributions = FALSE,
  rescale.distributions = FALSE,
  exp = FALSE,
  point.shape = TRUE,
```

```
  point.size = 5,
  legend.title = "Model",
  groups = NULL,
  facet.rows = NULL,
  facet.cols = NULL,
  facet.label.pos = "top",
  color.class = colors,
  resp = NULL,
  dpar = NULL
)

plot_coefs(
  ...,
  ci_level = 0.95,
  inner_ci_level = NULL,
  model.names = NULL,
  coefs = NULL,
  omit.coefs = c("(Intercept)", "Intercept"),
  colors = "CUD Bright",
  plot.distributions = FALSE,
  rescale.distributions = FALSE,
  exp = FALSE,
  point.shape = TRUE,
  point.size = 5,
  legend.title = "Model",
  groups = NULL,
  facet.rows = NULL,
  facet.cols = NULL,
  facet.label.pos = "top",
  color.class = colors,
  resp = NULL,
  dpar = NULL
)
```

Arguments

`...`	regression model(s). You may also include arguments to be passed to `tidy()`.
`ci_level`	The desired width of confidence intervals for the coefficients. Default: 0.95
`model.names`	If plotting multiple models simultaneously, you can provide a vector of names here. If NULL, they will be named sequentially as "Model 1", "Model 2", and so on. Default: NULL
`coefs`	If you'd like to include only certain coefficients, provide them as a vector. If it is a named vector, then the names will be used in place of the variable names. See details for examples. Default: NULL
`omit.coefs`	If you'd like to specify some coefficients to not include in the plot, provide them as a vector. This argument is overridden by `coefs` if both are provided. By default, the intercept term is omitted. To include the intercept term, just set omit.coefs to NULL.

`inner_ci_level` Plot a thicker line representing some narrower span than `ci_level`. Default is NULL, but good options are .9, .8, or .5.

`colors` See jtools_colors for more on your color options. Default: 'CUD Bright'

`plot.distributions`
Instead of just plotting the ranges, you may plot normal distributions representing the width of each estimate. Note that these are completely theoretical and not based on a bootstrapping or MCMC procedure, even if the source model was fit that way. Default is FALSE.

`rescale.distributions`
If `plot.distributions` is TRUE, the default behavior is to plot each normal density curve on the same scale. If some of the uncertainty intervals are much wider/narrower than others, that means the wide ones will have such a low height that you won't be able to see the curve. If you set this parameter to TRUE, each curve will have the same maximum height regardless of their width.

`exp` If TRUE, all coefficients are exponentiated (e.g., transforms logit coefficents from log odds scale to odds). The reference line is also moved to 1 instead of 0.

`point.shape` When using multiple models, should each model's point estimates use a different point shape to visually differentiate each model from the others? Default is TRUE. You may also pass a vector of shapes to specify shapes yourself.

`point.size` Change the size of the points. Default is 3.

`legend.title` What should the title for the legend be? Default is "Model", but you can specify it here since it is rather difficult to change later via `ggplot2`'s typical methods.

`groups` If you would like to have facets (i.e., separate panes) for different groups of coefficients, you can specify those groups with a list here. See details for more on how to do this.

`facet.rows` The number of rows in the facet grid (the `nrow` argument to `ggplot2::facet_wrap()`).

`facet.cols` The number of columns in the facet grid (the `nrow` argument to `ggplot2::facet_wrap()`).

`facet.label.pos`
Where to put the facet labels. One of "top" (the default), "bottom", "left", or "right".

`color.class` Deprecated. Now known as `colors`.

`resp` For any models that are `brmsfit` and have multiple response variables, specify them with a vector here. If the model list includes other types of models, you do not need to enter `resp` for those models. For instance, if I want to plot a `lm` object and two `brmsfit` objects, you only need to provide a vector of length 2 for `resp`.

`dpar` For any models that are `brmsfit` and have a distributional dependent variable, that can be specified here. If NULL, it is assumed you want coefficients for the location/mean parameter, not the distributional parameter(s).

Details

A note on the distinction between `plot_summs` and `plot_coefs`: `plot_summs` only accepts models supported by `summ()` and allows users to take advantage of the standardization and robust standard error features (among others as may be relevant). `plot_coefs` supports any models that have a

broom::tidy() method defined in the broom package, but of course lacks any additional features like robust standard errors. To get a mix of the two, you can pass summ objects to plot_coefs too.

For coefs, if you provide a named vector of coefficients, then the plot will refer to the selected coefficients by the names of the vector rather than the coefficient names. For instance, if I want to include only the coefficients for the hp and mpg but have the plot refer to them as "Horsepower" and "Miles/gallon", I'd provide the argument like this: c("Horsepower" = "hp", "Miles/gallon" = "mpg")

To use the groups argument, provide a (preferably named) list of character vectors. If I want separate panes with "Frost" and "Illiteracy" in one and "Population" and "Area" in the other, I'd make a list like this:

list(pane_1 = c("Frost", "Illiteracy"), pane_2 = c("Population", "Area"))

Value

A ggplot object.

Examples

```
states <- as.data.frame(state.x77)
fit1 <- lm(Income ~ Frost + Illiteracy + Murder +
           Population + Area + `Life Exp` + `HS Grad`,
           data = states, weights = runif(50, 0.1, 3))
fit2 <- lm(Income ~ Frost + Illiteracy + Murder +
           Population + Area + `Life Exp` + `HS Grad`,
           data = states, weights = runif(50, 0.1, 3))
fit3 <- lm(Income ~ Frost + Illiteracy + Murder +
           Population + Area + `Life Exp` + `HS Grad`,
           data = states, weights = runif(50, 0.1, 3))

# Plot all 3 regressions with custom predictor labels,
# standardized coefficients, and robust standard errors
plot_summs(fit1, fit2, fit3,
           coefs = c("Frost Days" = "Frost", "% Illiterate" = "Illiteracy",
                     "Murder Rate" = "Murder"),
           scale = TRUE, robust = TRUE)
```

---

predict_merMod *Alternative interface for* merMod *predictions*

Description

This function generates predictions for merMod models, but with the ability to get standard errors as well.

Usage

```
predict_merMod(
  object,
  newdata = NULL,
  se.fit = FALSE,
  use.re.var = FALSE,
  allow.new.levels = FALSE,
  type = c("link", "response", "terms"),
  na.action = na.pass,
  re.form = NULL,
  boot = FALSE,
  sims = 100,
  prog.arg = "none",
  ...
)
```

Arguments

object	a fitted model object
newdata	data frame for which to evaluate predictions.
se.fit	Include standard errors with the predictions? Note that these standard errors by default include only fixed effects variance. See details for more info. Default is FALSE.
use.re.var	If se.fit is TRUE, include random effects variance in standard errors? Default is FALSE.
allow.new.levels	logical if new levels (or NA values) in newdata are allowed. If FALSE (default), such new values in newdata will trigger an error; if TRUE, then the prediction will use the unconditional (population-level) values for data with previously unobserved levels (or NAs).
type	character string - either "link", the default, or "response" indicating the type of prediction object returned.
na.action	function determining what should be done with missing values for fixed effects in newdata. The default is to predict NA: see na.pass.
re.form	(formula, NULL, or NA) specify which random effects to condition on when predicting. If NULL, include all random effects; if NA or ~0, include no random effects.
boot	Use bootstrapping (via lme4::bootMer()) to estimate variance for se.fit? Default is FALSE
sims	If boot is TRUE, how many simulations should be run? Default is 100.
prog.arg	If boot and se.fit are TRUE, a character string - type of progress bar to display. Default is "none"; the function will look for a relevant *ProgressBar function, so "txt" will work in general; "tk" is available if the tcltk package is loaded; or "win" on Windows systems. Progress bars are disabled (with a message) for parallel operation.
...	When boot and se.fit are TRUE, any additional arguments are passed to lme4::bootMer().

Details

The developers of **lme4** omit an `se.fit` argument for a reason, which is that it's not perfectly clear how best to estimate the variance for these models. This solution is a logical one, but perhaps not perfect. Bayesian models are one way to do better.

The method used here is based on the one described here: http://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#predictions-andor-confidence-or-prediction-intervals-on-predictions

---

scale_mod *Scale variables in fitted regression models*

Description

`scale_mod` (previously known as `scale_lm`) takes fitted regression models and scales all predictors by dividing each by 1 or 2 standard deviations (as chosen by the user).

Usage

```
scale_mod(model, ...)

## Default S3 method:
scale_mod(
  model,
  binary.inputs = "0/1",
  n.sd = 1,
  center = TRUE,
  scale.response = FALSE,
  center.only = FALSE,
  scale.only = FALSE,
  data = NULL,
  vars = NULL,
  apply.weighted.contrasts = getOption("jtools-weighted.contrasts", FALSE),
  ...
)
```

Arguments

`model`	A regression model of type `lm`, `glm`, `svyglm`, or lme4::merMod. Other model types may work as well but are not tested.
`...`	Arguments passed on to `gscale()`.
`binary.inputs`	Options for binary variables. Default is `"0/1"`; `"0/1"` keeps original scale; `"-0.5,0.5"` rescales 0 as -0.5 and 1 as 0.5; `center` subtracts the mean; and `full` treats them like other continuous variables.
`n.sd`	How many standard deviations should you divide by for standardization? Default is 1, though some prefer 2.

`center` Default is `TRUE`. If `TRUE`, the predictors are also mean-centered. For binary predictors, the `binary.inputs` argument supersedes this one.

`scale.response` Should the response variable also be rescaled? Default is `FALSE`.

`center.only` Rather than actually scale predictors, just mean-center them.

`scale.only` A logical value indicating whether you would like to scale the values, but not mean-center them.

`data` If you provide the data used to fit the model here, that data frame is used to re-fit the model instead of the `stats::model.frame()` of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.

`vars` A character vector of variable names that you want to be scaled. If NULL, the default, it is all predictors.

`apply.weighted.contrasts`
Factor variables cannot be scaled, but you can set the contrasts such that the intercept in a regression model will reflect the true mean (assuming all other variables are centered). If set to TRUE, the argument will apply weighted effects coding to all factors. This is similar to the R default effects coding, but weights according to how many observations are at each level. An adapted version of `contr.wec()` from the `wec` package is used to do this. See that package's documentation and/or Grotenhuis et al. (2016) for more info.

Details

This function will scale all continuous variables in a regression model for ease of interpretation, especially for those models that have interaction terms. It can also mean-center all of them as well, if requested.

The scaling happens on the input data, not the terms themselves. That means interaction terms are still properly calculated because they are the product of standardized predictors, not a standardized product of predictors.

This function re-estimates the model, so for large models one should expect a runtime equal to the first run.

Value

The functions returns a re-fitted model object, inheriting from whichever class was supplied.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

Bauer, D. J., & Curran, P. J. (2005). Probing interactions in fixed and multilevel regression: Inferential and graphical techniques. *Multivariate Behavioral Research*, *40*(3), 373-400.

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analyses for the behavioral sciences* (3rd ed.). Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

See Also

sim_slopes performs a simple slopes analysis.

interact_plot creates attractive, user-configurable plots of interaction models.

Other standardization: center_mod(), center(), gscale(), standardize()

Examples

```
fit <- lm(formula = Murder ~ Income * Illiteracy,
          data = as.data.frame(state.x77))
fit_scale <- scale_mod(fit)
fit_scale <- scale_mod(fit, center = TRUE)

# With weights
fitw <- lm(formula = Murder ~ Income * Illiteracy,
           data = as.data.frame(state.x77),
           weights = Population)
fitw_scale <- scale_mod(fitw)
fitw_scale <- scale_mod(fitw, center = TRUE, binary.input = "0/1")

# With svyglm
if (requireNamespace("survey")) {
library(survey)
data(api)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
regmodel <- svyglm(api00~ell*meals,design=dstrat)
regmodel_scale <- scale_mod(regmodel)
regmodel_scale <- scale_mod(regmodel, binary.input = "0/1")
}
```

---

set_summ_defaults *Set defaults for* summ() *functions*

Description

This function is convenience wrapper for manually setting options using options(). This gives a handy way to, for instance, set the arguments to be used in every call to summ() in your script/session.

To make the settings persist across sessions, you can run this in your .Rprofile file.

Note that arguments that do not apply (e.g., robust for merMod models) are silently ignored when those types of models are used.

Usage

```
set_summ_defaults(
  digits = NULL,
  model.info = NULL,
```

```
  model.fit = NULL,
  pvals = NULL,
  robust = NULL,
  confint = NULL,
  ci.width = NULL,
  vifs = NULL,
  conf.method = NULL,
  table.format = NULL
)
```

Arguments

digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with `options("jtools-digits" = digits)` where digits is the desired number.
model.info	Toggles printing of basic information on sample size, name of DV, and number of predictors.
model.fit	Toggles printing of model fit statistics.
pvals	Show p values? If FALSE, these are not printed. Default is TRUE.
robust	If not FALSE, reports heteroskedasticity-robust standard errors instead of conventional SEs. These are also known as Huber-White standard errors. There are several options provided by `sandwich::vcovHC()`: "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". Default is FALSE. This requires the sandwich package to compute the standard errors.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.
vifs	If TRUE, adds a column to output with variance inflation factors (VIF). Default is FALSE.
conf.method	Argument passed to `lme4::confint.merMod()`. Default is "Wald", but "profile" or "boot" are better when accuracy is a priority. Be aware that both of the alternate methods are sometimes very time-consuming.
table.format	A format understood by `md_table()`

---

standardize *Standardize vectors, data frames, and survey designs*

Description

This function is a wrapper around `gscale()` that is configured to do a conventional standardization of continuous variables, mean-centering and dividing by one standard deviation.

Usage

```
standardize(
  data = NULL,
  vars = NULL,
  binary.inputs = "center",
  binary.factors = FALSE,
  weights = NULL
)
```

Arguments

`data`	A data frame or survey design. Only needed if you would like to rescale multiple variables at once. If `x = NULL`, all columns will be rescaled. Otherwise, `x` should be a vector of variable names. If `x` is a numeric vector, this argument is ignored.
`vars`	If `data` is a data.frame or similar, you can scale only select columns by providing a vector column names to this argument.
`binary.inputs`	Options for binary variables. Default is `center`; `0/1` keeps original scale; `-0.5/0.5` rescales 0 as -0.5 and 1 as 0.5; `center` subtracts the mean; and `full` subtracts the mean and divides by 2 sd.
`binary.factors`	Coerce two-level factors to numeric and apply scaling functions to them? Default is FALSE.
`weights`	A vector of weights equal in length to `x`. If iterating over a data frame, the weights will need to be equal in length to all the columns to avoid errors. You may need to remove missing values before using the weights.

Details

Some more information can be found in the documentation for `gscale()`

Value

A transformed version of the `data` argument.

See Also

Other standardization: `center_mod()`, `center()`, `gscale()`, `scale_mod()`

Examples

```
# Standardize just the "qsec" variable in mtcars
standardize(mtcars, vars = "qsec")
```

summ *Regression summaries with options*

Description

To get specific documentation, choose the appropriate link to the type of model that you want to summarize from the details section.

Usage

```
summ(model, ...)

j_summ(model, ...)
```

Arguments

model	A lm, glm, svyglm, merMod, rq object.
...	Other arguments to be passed to the model-specific function.

Details

- summ.lm
- summ.glm
- summ.svyglm
- summ.merMod
- summ.rq

summ.glm *Generalized linear regression summaries with options*

Description

summ() prints output for a regression model in a fashion similar to summary(), but formatted differently with more options.

Usage

```
## S3 method for class 'glm'
summ(
  model,
  scale = FALSE,
  confint = getOption("summ-confint", FALSE),
  ci.width = getOption("summ-ci.width", 0.95),
  robust = getOption("summ-robust", FALSE),
```

```
  cluster = NULL,
  vifs = getOption("summ-vifs", FALSE),
  digits = getOption("jtools-digits", default = 2),
  exp = FALSE,
  pvals = getOption("summ-pvals", TRUE),
  n.sd = 1,
  center = FALSE,
  transform.response = FALSE,
  scale.only = FALSE,
  data = NULL,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE),
  which.cols = NULL,
  vcov = NULL,
  ...
)
```

Arguments

model	A glm object.
scale	If TRUE, reports standardized regression coefficients by scaling and mean-centering input data (the latter can be changed via the scale.only argument). Default is FALSE.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.
robust	If not FALSE, reports heteroskedasticity-robust standard errors instead of conventional SEs. These are also known as Huber-White standard errors. There are several options provided by sandwich::vcovHC(): "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". Default is FALSE. This requires the sandwich package to compute the standard errors.
cluster	For clustered standard errors, provide the column name of the cluster variable in the input data frame (as a string). Alternately, provide a vector of clusters. Note that you must set robust to either "HC1", "HC2", or "HC3" in order to have clustered standard errors ("HC4" and "HC5" are not supported.
vifs	If TRUE, adds a column to output with variance inflation factors (VIF). Default is FALSE.
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
exp	If TRUE, reports exponentiated coefficients with confidence intervals for exponential models like logit and Poisson models. This quantity is known as an odds ratio for binary outcomes and incidence rate ratio for count models.

pvals	Show p values? If FALSE, these are not printed. Default is TRUE.
n.sd	If scale = TRUE, how many standard deviations should predictors be divided by? Default is 1, though some suggest 2.
center	If you want coefficients for mean-centered variables but don't want to standardize, set this to TRUE. Note that setting this to false does not affect whether scale mean-centers variables. Use scale.only for that.
transform.response	Should scaling/centering apply to response variable? Default is FALSE.
scale.only	If you want to scale but not center, set this to TRUE. Note that for legacy reasons, setting scale = TRUE and center = FALSE will not achieve the same effect. Default is FALSE.
data	If you provide the data used to fit the model here, that data frame is used to re-fit the model (if scale is TRUE) instead of the stats::model.frame() of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.
model.info	Toggles printing of basic information on sample size, name of DV, and number of predictors.
model.fit	Toggles printing of model fit statistics.
which.cols	Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.
vcov	You may provide your own variance-covariance matrix for the regression coefficients if you want to calculate standard errors in some way not accommodated by the robust and cluster options.
...	Among other things, arguments are passed to scale_mod() or center_mod() when center or scale is TRUE.

Details

By default, this function will print the following items to the console:

- The sample size
- The name of the outcome variable
- The chi-squared test, (Pseudo-)R-squared value and AIC/BIC.
- A table with regression coefficients, standard errors, z values, and p values.

There are several options available for robust. The heavy lifting is done by sandwich::vcovHC(), where those are better described. Put simply, you may choose from "HC0" to "HC5". Based on the recommendation of the developers of **sandwich**, the default is set to "HC3". Stata's default is "HC1", so that choice may be better if the goal is to replicate Stata's output. Any option that is understood by vcovHC() will be accepted. Cluster-robust standard errors are computed if cluster is set to the name of the input data's cluster variable or is a vector of clusters.

The scale and center options are performed via refitting the model with scale_mod() and center_mod(), respectively. Each of those in turn uses gscale() for the mean-centering and scaling.

Value

If saved, users can access most of the items that are returned in the output (and without rounding).

`coeftable`	The outputted table of variables and coefficients
`model`	The model for which statistics are displayed. This would be most useful in cases in which `scale = TRUE`.

Much other information can be accessed as attributes.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

King, G., & Roberts, M. E. (2015). How robust standard errors expose methodological problems they do not fix, and what to do about it. *Political Analysis*, *23*(2), 159–179. doi: 10.1093/pan/mpu015

Lumley, T., Diehr, P., Emerson, S., & Chen, L. (2002). The Importance of the Normality Assumption in Large Public Health Data Sets. *Annual Review of Public Health*, *23*, 151–169. doi: 10.1146/annurev.publhealth.23.100901.140546

See Also

`scale_mod()` can simply perform the standardization if preferred.

`gscale()` does the heavy lifting for mean-centering and scaling behind the scenes.

Other summ: `summ.lm()`, `summ.merMod()`, `summ.rq()`, `summ.svyglm()`

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson)

# Summarize with standardized coefficients
summ(glm.D93, scale = TRUE)
```

summ.lm *Linear regression summaries with options*

Description

summ() prints output for a regression model in a fashion similar to summary(), but formatted differently with more options.

Usage

```
## S3 method for class 'lm'
summ(
  model,
  scale = FALSE,
  confint = getOption("summ-confint", FALSE),
  ci.width = getOption("summ-ci.width", 0.95),
  robust = getOption("summ-robust", FALSE),
  cluster = NULL,
  vifs = getOption("summ-vifs", FALSE),
  digits = getOption("jtools-digits", 2),
  pvals = getOption("summ-pvals", TRUE),
  n.sd = 1,
  center = FALSE,
  transform.response = FALSE,
  scale.only = FALSE,
  data = NULL,
  part.corr = FALSE,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE),
  which.cols = NULL,
  vcov = NULL,
  ...
)
```

Arguments

model	A lm object.
scale	If TRUE, reports standardized regression coefficients by scaling and mean-centering input data (the latter can be changed via the scale.only argument). Default is FALSE.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.

robust — If not FALSE, reports heteroskedasticity-robust standard errors instead of conventional SEs. These are also known as Huber-White standard errors. There are several options provided by sandwich::vcovHC(): "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5".

Default is FALSE.

This requires the sandwich package to compute the standard errors.

cluster — For clustered standard errors, provide the column name of the cluster variable in the input data frame (as a string). Alternately, provide a vector of clusters. Note that you must set robust to either "HC1", "HC2", or "HC3" in order to have clustered standard errors ("HC4" and "HC5" are not supported.

vifs — If TRUE, adds a column to output with variance inflation factors (VIF). Default is FALSE.

digits — An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.

pvals — Show p values? If FALSE, these are not printed. Default is TRUE.

n.sd — If scale = TRUE, how many standard deviations should predictors be divided by? Default is 1, though some suggest 2.

center — If you want coefficients for mean-centered variables but don't want to standardize, set this to TRUE. Note that setting this to false does not affect whether scale mean-centers variables. Use scale.only for that.

transform.response — Should scaling/centering apply to response variable? Default is FALSE.

scale.only — If you want to scale but not center, set this to TRUE. Note that for legacy reasons, setting scale = TRUE and center = FALSE will not achieve the same effect. Default is FALSE.

data — If you provide the data used to fit the model here, that data frame is used to re-fit the model (if scale is TRUE) instead of the stats::model.frame() of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.

part.corr — Print partial (labeled "partial.r") and semipartial (labeled "part.r") correlations with the table? Default is FALSE. See details about these quantities when robust standard errors are used.

model.info — Toggles printing of basic information on sample size, name of DV, and number of predictors.

model.fit — Toggles printing of model fit statistics.

which.cols — Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.

vcov — You may provide your own variance-covariance matrix for the regression coefficients if you want to calculate standard errors in some way not accommodated by the robust and cluster options.

... — Among other things, arguments are passed to scale_mod() or center_mod() when center or scale is TRUE.

Details

By default, this function will print the following items to the console:

- The sample size
- The name of the outcome variable
- The R-squared value plus adjusted R-squared
- A table with regression coefficients, standard errors, t-values, and p values.

There are several options available for `robust`. The heavy lifting is done by `sandwich::vcovHC()`, where those are better described. Put simply, you may choose from `"HC0"` to `"HC5"`. Based on the recommendation of the developers of **sandwich**, the default is set to `"HC3"`. Stata's default is `"HC1"`, so that choice may be better if the goal is to replicate Stata's output. Any option that is understood by `vcovHC()` will be accepted. Cluster-robust standard errors are computed if `cluster` is set to the name of the input data's cluster variable or is a vector of clusters.

The `scale` and `center` options are performed via refitting the model with `scale_mod()` and `center_mod()`, respectively. Each of those in turn uses `gscale()` for the mean-centering and scaling.

If using `part.corr = TRUE`, then you will get these two common effect size metrics on the far right two columns of the output table. However, it should be noted that these do not go hand in hand with robust standard error estimators. The standard error of the coefficient doesn't change the point estimate, just the uncertainty. However, this function uses *t*-statistics in its calculation of the partial and semipartial correlation. This provides what amounts to a heteroskedasticity-adjusted set of estimates, but I am unaware of any statistical publication that validates this type of use. Please use these as a heuristic when used alongside robust standard errors; do not report the "robust" partial and semipartial correlations in publications.

Value

If saved, users can access most of the items that are returned in the output (and without rounding).

`coeftable`	The outputted table of variables and coefficients
`model`	The model for which statistics are displayed. This would be most useful in cases in which `scale = TRUE`.

Much other information can be accessed as attributes.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

King, G., & Roberts, M. E. (2015). How robust standard errors expose methodological problems they do not fix, and what to do about it. *Political Analysis*, *23*(2), 159–179. doi: 10.1093/pan/mpu015

Lumley, T., Diehr, P., Emerson, S., & Chen, L. (2002). The Importance of the Normality Assumption in Large Public Health Data Sets. *Annual Review of Public Health*, *23*, 151–169. doi: 10.1146/annurev.publhealth.23.100901.140546

See Also

scale_mod() can simply perform the standardization if preferred.

gscale() does the heavy lifting for mean-centering and scaling behind the scenes.

Other summ: summ.glm(), summ.merMod(), summ.rq(), summ.svyglm()

Examples

```
# Create lm object
fit <- lm(Income ~ Frost + Illiteracy + Murder,
          data = as.data.frame(state.x77))

# Print the output with standardized coefficients and 3 digits
summ(fit, scale = TRUE, digits = 3)
```

---

summ.merMod *Mixed effects regression summaries with options*

Description

summ() prints output for a regression model in a fashion similar to summary(), but formatted differently with more options.

Usage

```
## S3 method for class 'merMod'
summ(
  model,
  scale = FALSE,
  confint = getOption("summ-confint", FALSE),
  ci.width = getOption("summ-ci.width", 0.95),
  conf.method = getOption("summ-conf.method", c("Wald", "profile", "boot")),
  digits = getOption("jtools-digits", default = 2),
  r.squared = TRUE,
  pvals = getOption("summ-pvals", NULL),
  n.sd = 1,
  center = FALSE,
  transform.response = FALSE,
  scale.only = FALSE,
  data = NULL,
  exp = FALSE,
  t.df = NULL,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE),
  re.variance = getOption("summ-re.variance", c("sd", "var")),
  which.cols = NULL,
```

```
  re.table = getOption("summ-re.table", TRUE),
  groups.table = getOption("summ-groups.table", TRUE),
  ...
)
```

Arguments

model	A merMod object.
scale	If TRUE, reports standardized regression coefficients by scaling and mean-centering input data (the latter can be changed via the scale.only argument). Default is FALSE.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.
conf.method	Argument passed to lme4::confint.merMod(). Default is "Wald", but "profile" or "boot" are better when accuracy is a priority. Be aware that both of the alternate methods are sometimes very time-consuming.
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
r.squared	Calculate an r-squared model fit statistic? Default is TRUE, but if it has errors or takes a long time to calculate you may want to consider setting to FALSE.
pvals	Show p values? If FALSE, these are not printed. Default is TRUE, except for merMod objects (see details).
n.sd	If scale = TRUE, how many standard deviations should predictors be divided by? Default is 1, though some suggest 2.
center	If you want coefficients for mean-centered variables but don't want to standardize, set this to TRUE. Note that setting this to false does not affect whether scale mean-centers variables. Use scale.only for that.
transform.response	Should scaling/centering apply to response variable? Default is FALSE.
scale.only	If you want to scale but not center, set this to TRUE. Note that for legacy reasons, setting scale = TRUE and center = FALSE will not achieve the same effect. Default is FALSE.
data	If you provide the data used to fit the model here, that data frame is used to re-fit the model (if scale is TRUE) instead of the stats::model.frame() of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.
exp	If TRUE, reports exponentiated coefficients with confidence intervals for exponential models like logit and Poisson models. This quantity is known as an odds ratio for binary outcomes and incidence rate ratio for count models.
t.df	For lmerMod models only. User may set the degrees of freedom used in conducting t-tests. See details for options.

model.info	Toggles printing of basic information on sample size, name of DV, and number of predictors.
model.fit	Toggles printing of model fit statistics.
re.variance	Should random effects variances be expressed in standard deviations or variances? Default, to be consistent with previous versions of jtools, is "sd". Use "var" to get the variance instead.
which.cols	Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.
re.table	Show table summarizing variance of random effects? Default is TRUE.
groups.table	Show table summarizing the grouping variables? Default is TRUE.
...	Among other things, arguments are passed to scale_mod() or center_mod() when center or scale is TRUE.

Details

By default, this function will print the following items to the console:

- The sample size
- The name of the outcome variable
- The (Pseudo-)R-squared value and AIC/BIC.
- A table with regression coefficients, standard errors, and t-values.

The scale and center options are performed via refitting the model with scale_mod() and center_mod(), respectively. Each of those in turn uses gscale() for the mean-centering and scaling.

merMod models are a bit different than the others. The lme4 package developers have, for instance, made a decision not to report or compute p values for lmer() models. There are good reasons for this, most notably that the t-values produced are not "accurate" in the sense of the Type I error rate. For certain large, balanced samples with many groups, this is no big deal. What's a "big" or "small" sample? How much balance is necessary? What type of random effects structure is okay? Good luck getting a statistician to give you any clear guidelines on this. Some simulation studies have been done on fewer than 100 observations, so for sure if your sample is around 100 or fewer you should not interpret the t-values. A large number of groups is also crucial for avoiding bias using t-values. If groups are nested or crossed in a linear model, it is best to just get the **pbkrtest** package.

By default, this function follows lme4's lead and does not report the p values for lmer() models. If the user has **pbkrtest** installed, however, p values are reported using the Kenward-Roger d.f. approximation unless pvals = FALSE or t.df is set to something other than NULL. In publications, you should cite the Kenward & Roger (1997) piece as well as either this package or **pbkrtest** package to explain how the p values were calculated.

See pvalues from the **lme4** for more details. If you're looking for a simple test with no extra packages installed, it is better to use the confidence intervals and check to see if they exclude zero than use the t-test. For users of glmer(), see some of the advice there as well. While lme4 and by association summ() does as well, they are still imperfect.

You have some options to customize the output in this regard with the t.df argument. If NULL, the default, the degrees of freedom used depends on whether the user has **lmerTest** or **pbkrtest** installed. If lmerTest is installed, the degrees of freedom for each coefficient are calculated using

the Satterthwaite method and the p values calculated accordingly. If only `pbkrtest` is installed or `t.df` is `"k-r"`, the Kenward-Roger approximation of the standard errors and degrees of freedom for each coefficient is used. Note that Kenward-Roger standard errors can take longer to calculate and may cause R to crash with models fit to large (roughly greater than 5000 rows) datasets.

If neither is installed and the user sets `pvals = TRUE`, then the residual degrees of freedom is used. If `t.df = "residual"`, then the residual d.f. is used without a message. If the user prefers to use some other method to determine the d.f., then any number provided as the argument will be used.

About pseudo-R^2

There is no one way to calculate R^2 for mixed models or nonlinear models. Many caution against interpreting or even using such approximations outside of OLS regression. With that said, this package reports one version for your benefit, though you should of course understand that it is not an unambiguous measure of model fit.

This package calculates R^2 for mixed models using an adapted version of `rsquared()` from the `piecewiseSEM` package. This is an implementation of the Nakagawa & Schielzeth (2013) procedure with refinements by Johnson (2014). If you choose to report the pseudo-R^2 in a publication, you should cite Nakagawa & Schielzeth to explain how the calculation was done.

Value

If saved, users can access most of the items that are returned in the output (and without rounding).

`coeftable`	The outputted table of variables and coefficients
`rcoeftable`	The secondary table with the grouping variables and random coefficients.
`gvars`	The tertiary table with the grouping variables, numbers of groups, and ICCs.
`model`	The model for which statistics are displayed. This would be most useful in cases in which `scale = TRUE`.

Much other information can be accessed as attributes.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

Johnson, P. C. D. (2014). Extension of Nakagawa & Schielzeth's R^2_GLMM to random slopes models. *Methods in Ecology and Evolution*, *5*, 944–946. doi: 10.1111/2041210X.12225

Kenward, M. G., & Roger, J. H. (1997). Small sample inference for fixed effects from restricted maximum likelihood. *Biometrics*, *53*, 983. doi: 10.2307/2533558

Kuznetsova, A., Brockhoff, P. B., & Christensen, R. H. B. (2017). lmerTest package: Tests in linear mixed effects models. *Journal of Statistical Software*, *82*. doi: 10.18637/jss.v082.i13

Luke, S. G. (2017). Evaluating significance in linear mixed-effects models in R. *Behavior Research Methods*, *49*, 1494–1502. doi: 10.3758/s134280160809y

Nakagawa, S., & Schielzeth, H. (2013). A general and simple method for obtaining R^2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, *4*, 133–142. doi: 10.1111/j.2041210x.2012.00261.x

See Also

scale_mod() can simply perform the standardization if preferred.

gscale() does the heavy lifting for mean-centering and scaling behind the scenes.

pbkrtest::get_ddf_Lb() gets the Kenward-Roger degrees of freedom if you have **pbkrtest** installed.

A tweaked version of piecewiseSEM::rsquared() is used to generate the pseudo-R-squared estimates for linear models.

Other summ: summ.glm(), summ.lm(), summ.rq(), summ.svyglm()

Examples

```
if (requireNamespace("lme4")) {
  library(lme4, quietly = TRUE)
  data(sleepstudy)
  mv <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)

  summ(mv) # Note lack of p values if you don't have lmerTest/pbkrtest

  # Without lmerTest/pbkrtest, you'll get message about Type 1 errors
  summ(mv, pvals = TRUE)

  # To suppress message, manually specify t.df argument
  summ(mv, t.df = "residual")

  # Confidence intervals may be better alternative to p values
  summ(mv, confint = TRUE)
  # Use conf.method to get profile intervals (may be slow to run)
  # summ(mv, confint = TRUE, conf.method = "profile")

}
```

---

summ.rq *Quantile regression summaries with options*

Description

summ() prints output for a regression model in a fashion similar to summary(), but formatted differently with more options.

Usage

```
## S3 method for class 'rq'
summ(
  model,
  scale = FALSE,
  confint = getOption("summ-confint", FALSE),
```

```
  ci.width = getOption("summ-ci.width", 0.95),
  se = c("nid", "rank", "iid", "ker", "boot"),
  boot.sims = 1000,
  boot.method = "xy",
  vifs = getOption("summ-vifs", FALSE),
  digits = getOption("jtools-digits", 2),
  pvals = getOption("summ-pvals", TRUE),
  n.sd = 1,
  center = FALSE,
  transform.response = FALSE,
  data = NULL,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE),
  which.cols = NULL,
  ...
)
```

Arguments

model	A rq model. At this time, rqs models (multiple tau parameters) are not supported.
scale	If TRUE, reports standardized regression coefficients by scaling and mean-centering input data (the latter can be changed via the scale.only argument). Default is FALSE.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.
se	One of "nid", "rank", "iid", "ker", or "boot". "nid" is default. See quantreg::summary.rq() documentation for more about these options.
boot.sims	If se = "boot", the number of bootstrap replications to perform. This is passed as the R argument to boot.rq
boot.method	If se = "boot", the type of bootstrapping method to use. Default is "xy", but see quantreg::boot.rq() for more options.
vifs	If TRUE, adds a column to output with variance inflation factors (VIF). Default is FALSE.
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
pvals	Show p values? If FALSE, these are not printed. Default is TRUE.
n.sd	If scale = TRUE, how many standard deviations should predictors be divided by? Default is 1, though some suggest 2.
center	If you want coefficients for mean-centered variables but don't want to standardize, set this to TRUE. Note that setting this to false does not affect whether scale mean-centers variables. Use scale.only for that.

`transform.response`
: Should scaling/centering apply to response variable? Default is `FALSE`.

`data`
: If you provide the data used to fit the model here, that data frame is used to re-fit the model (if `scale` is `TRUE`) instead of the `stats::model.frame()` of the model. This is particularly useful if you have variable transformations or polynomial terms specified in the formula.

`model.info`
: Toggles printing of basic information on sample size, name of DV, and number of predictors.

`model.fit`
: Toggles printing of model fit statistics.

`which.cols`
: Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.

`...`
: Among other things, arguments are passed to `scale_mod()` or `center_mod()` when `center` or `scale` is `TRUE`.

Details

This method implements most of the things I think most users would asking `summary.rq` for. `hs`, `U`, and `gamma` are ignored.

Note that when using `se = "rank"`, there are no standard errors, test statistics, or p values calculated.

About the R1 fit statistic: Described in Koenker & Machado (1999), this offers an interpretation similar to R-squared in OLS regression. While you could calculate R-squared for these models, it goes against the underlying theoretical rationale for them. Koenker himself is not a big fan of R1 either, but it's something. See Koenker & Machado (1999) for more info.

References

Koenker, R., & Machado, J. A. F. (1999). Goodness of fit and related inference processes for quantile regression. *Journal of the American Statistical Association*, *94*, 1296–1310. https://doi.org/10.1080/01621459.1999.1047

See Also

Other summ: `summ.glm()`, `summ.lm()`, `summ.merMod()`, `summ.svyglm()`

Examples

```
if (requireNamespace("quantreg")) {
 library(quantreg)
 data(engel)
 fitrq <- rq(income ~ foodexp, data = engel, tau = 0.5)
 summ(fitrq)
}
```

summ.svyglm *Complex survey regression summaries with options*

Description

summ() prints output for a regression model in a fashion similar to summary(), but formatted differently with more options.

Usage

```
## S3 method for class 'svyglm'
summ(
  model,
  scale = FALSE,
  confint = getOption("summ-confint", FALSE),
  ci.width = getOption("summ-ci.width", 0.95),
  digits = getOption("jtools-digits", default = 2),
  pvals = getOption("summ-pvals", TRUE),
  n.sd = 1,
  center = FALSE,
  transform.response = FALSE,
  scale.only = FALSE,
  exp = FALSE,
  vifs = getOption("summ-vifs", FALSE),
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE),
  which.cols = NULL,
  ...
)
```

Arguments

model	A svyglm object.
scale	If TRUE, reports standardized regression coefficients by scaling and mean-centering input data (the latter can be changed via the scale.only argument). Default is FALSE.
confint	Show confidence intervals instead of standard errors? Default is FALSE.
ci.width	A number between 0 and 1 that signifies the width of the desired confidence interval. Default is .95, which corresponds to a 95% confidence interval. Ignored if confint = FALSE.
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
pvals	Show p values? If FALSE, these are not printed. Default is TRUE.

n.sd	If scale = TRUE, how many standard deviations should predictors be divided by? Default is 1, though some suggest 2.
center	If you want coefficients for mean-centered variables but don't want to standardize, set this to TRUE. Note that setting this to false does not affect whether scale mean-centers variables. Use scale.only for that.
transform.response	Should scaling/centering apply to response variable? Default is FALSE.
scale.only	If you want to scale but not center, set this to TRUE. Note that for legacy reasons, setting scale = TRUE and center = FALSE will not achieve the same effect. Default is FALSE.
exp	If TRUE, reports exponentiated coefficients with confidence intervals for exponential models like logit and Poisson models. This quantity is known as an odds ratio for binary outcomes and incidence rate ratio for count models.
vifs	If TRUE, adds a column to output with variance inflation factors (VIF). Default is FALSE.
model.info	Toggles printing of basic information on sample size, name of DV, and number of predictors.
model.fit	Toggles printing of model fit statistics.
which.cols	Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.
...	Among other things, arguments are passed to scale_mod() or center_mod() when center or scale is TRUE.

Details

By default, this function will print the following items to the console:

- The sample size
- The name of the outcome variable
- The (Pseudo-)R-squared value and AIC.
- A table with regression coefficients, standard errors, t values, and p values.

The scale and center options are performed via refitting the model with scale_mod() and center_mod(), respectively. Each of those in turn uses gscale() for the mean-centering and scaling. These functions can handle svyglm objects correctly by calling svymean() and svyvar() to compute means and standard deviations. Weights are not altered. The fact that the model is refit means the runtime will be similar to the original time it took to fit the model.

Value

If saved, users can access most of the items that are returned in the output (and without rounding).

coeftable	The outputted table of variables and coefficients
model	The model for which statistics are displayed. This would be most useful in cases in which scale = TRUE.

Much other information can be accessed as attributes.

Author(s)

Jacob Long <jacob.long@sc.edu>

See Also

scale_mod() can simply perform the standardization if preferred.

gscale() does the heavy lifting for mean-centering and scaling behind the scenes.

Other summ: summ.glm(), summ.lm(), summ.merMod(), summ.rq()

Examples

```
if (requireNamespace("survey")) {
  library(survey)
  data(api)
  dstrat <- svydesign(id = ~1, strata =~ stype, weights =~ pw,
                      data = apistrat, fpc =~ fpc)
  regmodel <- svyglm(api00 ~ ell * meals, design = dstrat)

  summ(regmodel)
}
```

---

svycor *Calculate Pearson correlations with complex survey data*

Description

svycor extends the survey package by calculating correlations with syntax similar to the original package, which for reasons unknown lacks such a function.

Usage

```
svycor(
  formula,
  design,
  na.rm = FALSE,
  digits = getOption("jtools-digits", default = 2),
  sig.stats = FALSE,
  bootn = 1000,
  mean1 = TRUE,
  ...
)
```

Arguments

formula	A formula (e.g., ~var1+var2) specifying the terms to correlate.
design	The survey.design or svyrep.design object.
na.rm	Logical. Should cases with missing values be dropped?
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 2. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
sig.stats	Logical. Perform non-parametric bootstrapping (using wtd.cor) to generate standard errors and associated t- and p-values. See details for some considerations when doing null hypothesis testing with complex survey correlations.
bootn	If sig.stats is TRUE, this defines the number of bootstraps to be run to generate the standard errors and p-values. For large values and large datasets, this can contribute considerably to processing time.
mean1	If sig.stats is TRUE, it is important to know whether the sampling weights should have a mean of 1. That is, should the standard errors be calculated as if the number of rows in your dataset is the total number of observations (TRUE) or as if the sum of the weights in your dataset is the total number of observations (FALSE)?
...	Additional arguments passed to svyvar().

Details

This function extends the survey package by calculating the correlations for user-specified variables in survey design and returning a correlation matrix.

Using the wtd.cor function, this function also returns standard errors and p-values for the correlation terms using a sample-weighted bootstrapping procedure. While correlations do not require distributional assumptions, hypothesis testing (i.e., $r > 0$) does. The appropriate way to calculate standard errors and use them to define a probability is not straightforward in this scenario since the weighting causes heteroskedasticity, thereby violating an assumption inherent in the commonly used methods for converting Pearson's correlations into t-values. The method provided here is defensible, but if reporting in scientific publications the method should be spelled out.

Value

If significance tests are not requested, there is one returned value:

cors	The correlation matrix (without rounding)

If significance tests are requested, the following are also returned:

p.values	A matrix of p values
t.values	A matrix of t values
std.err	A matrix of standard errors

Note

This function was designed in part on the procedure recommended by Thomas Lumley, the author of the survey package, on Stack Overflow. However, he has not reviewed or endorsed this implementation. All defects are attributed to the author.

Author(s)

Jacob Long <jacob.long@sc.edu>

See Also

wtd.cor, svymean()

Other survey package extensions: svysd()

Other survey tools: pf_sv_test(), svysd(), weights_tests(), wgttest()

Examples

```
if (requireNamespace("survey")) {
 library(survey)
 data(api)
 # Create survey design object
 dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw,
                     data = apistrat, fpc = ~fpc)

 # Print correlation matrix
 svycor(~api00 + api99 + dnum, design = dstrat)

 # Save the results, extract correlation matrix
 out <- svycor(~api00 + api99 + dnum, design = dstrat)
 out$cors

}
```

---

svysd *Calculate standard deviations with complex survey data*

Description

svysd extends the survey package by calculating standard deviations with syntax similar to the original package, which provides only a svyvar() function.

Usage

```
svysd(
  formula,
  design,
  na.rm = FALSE,
  digits = getOption("jtools-digits", default = 3),
  ...
)
```

Arguments

formula	A formula (e.g., ~var1+var2) specifying the term(s) of interest.
design	The survey.design or svyrep.design object.
na.rm	Logical. Should cases with missing values be dropped?
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 3. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.
...	Additional arguments passed to svyvar().

Details

An alternative is to simply do sqrt(svyvar(~term, design = design)). However, if printing and sharing the output, this may be misleading since the output will say "variance."

Note

This function was designed independent of the **survey** package and is neither endorsed nor known to its authors.

See Also

svyvar()

Other survey package extensions: svycor()

Other survey tools: pf_sv_test(), svycor(), weights_tests(), wgttest()

Examples

```
if (requireNamespace("survey")) {
 library(survey)
 data(api)
 # Create survey design object
 dstrat <- svydesign(id = ~1,strata = ~stype, weights = ~pw, data = apistrat,
                     fpc=~fpc)

 # Print the standard deviation of some variables
 svysd(~api00+ell+meals, design = dstrat)
}
```

theme_apa *Format ggplot2 figures in APA style*

Description

`theme_apa()` is designed to work like any other complete theme from `ggplot`. To the extent possible, it aligns with the (vague) APA figure guidelines.

Usage

```
theme_apa(
  legend.pos = "right",
  legend.use.title = FALSE,
  legend.font.size = 12,
  x.font.size = 12,
  y.font.size = 12,
  facet.title.size = 12,
  remove.y.gridlines = TRUE,
  remove.x.gridlines = TRUE
)
```

Arguments

`legend.pos`	One of `"right"`, `"left"`, `"top"`, `"bottom"`, `"topleft"`, `"topright"`, `"topmiddle"`, `"bottomleft"`, `"bottomright"`, or `"bottommiddle"`. Positions the legend, which will layer on top of any geoms, on the plane.
`legend.use.title`	Logical. Specify whether to include a legend title. Defaults to `FALSE`.
`legend.font.size`	Integer indicating the font size of the labels in the legend. Default and APA-recommended is 12, but if there are many labels it may be necessary to choose a smaller size.
`x.font.size`	Font size of x-axis label.
`y.font.size`	Font size of x-axis label.
`facet.title.size`	Font size of facet labels.
`remove.y.gridlines`	Should the coordinate grid on the y-axis (horizontal lines) be removed? Default is TRUE.
`remove.x.gridlines`	Should the coordinate grid on the x-axis (vertical lines) be removed? Default is TRUE.

Details

This function applies a theme to ggplot2 figures with a style that is roughly in line with APA guidelines. Users may need to perform further operations for their specific use cases.

There are some things to keep in mind about APA style figures:

- Main titles should be written in the word processor or typesetter rather than on the plot image itself.
- In some cases, users can forgo a legend in favor of describing the figure in a caption (also written in the word processor/typesetter).
- Legends are typically embedded on the coordinate plane of the figure rather than next to it, as is default in ggplot2.
- Use of color is generally discouraged since most of the applications for which APA figures are needed involve eventual publication in non-color print media.
- There are no hard and fast rules on font size, though APA recommends choosing between 8 and 14-point. Fonts in figures should be sans serif.

Because APA style calls for positioning legends on the plane itself, this function includes options for choosing a position–top left, top right, bottom left, bottom right–to place the legend. ggplot2 provides no obvious way to automatically choose a position that overlaps least with the geoms (the plotted data), so users will need to choose one.

Facetting is supported, but APA guidelines are considerably less clear for such situations.

This theme was created with inspiration from Rudolf Cardinal's code, which required updating for newer versions of ggplot2 and adaptations for APA style.

Author(s)

Jacob Long <jacob.long@sc.edu>

References

American Psychological Association. (2010). *Publication manual of the American Psychological Association, Sixth Edition*. Washington, DC: American Psychological Association.

Nicol, A.A.M. & Pexman, P.M. (2010). *Displaying your findings: A practical guide for creating figures, posters, and presentations, Sixth Edition*. Washington, D.C.: American Psychological Association.

See Also

ggplot, theme

Examples

```
# Create plot with ggplot2
library(ggplot2)
plot <- ggplot(mpg, aes(cty, hwy)) +
  geom_jitter()
```

```
# Add APA theme with defaults
plot + theme_apa()
```

---

theme_nice *A nice, flexible* ggplot2 *theme*

Description

theme_nice is designed to work like any other complete theme from ggplot. It has a nice appearance.

Usage

```
theme_nice(
  legend.pos = "right",
  style = c("white", "light", "dark_blue", "dark_gray"),
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)
```

Arguments

legend.pos	One of "right", "left", "top", "bottom" (outside the plotting area), "topleft", "topright", "topmiddle", "bottomleft", "bottomright", or "bottommiddle" (inside the plotting area).
style	One of "white", "light", "dark_blue", or "dark_gray". "white" sets the background to white, "light" to light gray, "dark_gray" to dark gray, "dark_blue" to dark blue.
base_size	base font size, given in pts.
base_family	base font family
base_line_size	base size for line elements
base_rect_size	base size for rect elements

Author(s)

Jacob Long <jacob.long@sc.edu>

Examples

```
# Create plot with ggplot2
library(ggplot2)
plot <- ggplot(mpg, aes(cty, hwy)) +
  geom_jitter() + theme_nice()
```

`tidy.summ` *Broom extensions for summ objects*

Description

These are functions used for compatibility with broom's tidying functions to facilitate use with huxreg, thereby making `export_summs` works.

Usage

```
## S3 method for class 'summ'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'summ.merMod'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'summ.lm'
glance(x, ...)

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

## S3 method for class 'summ.svyglm'
glance(x, ...)

## S3 method for class 'summ.merMod'
glance(x, ...)

## S3 method for class 'summ.rq'
glance(x, ...)
```

Arguments

`x`	The `summ` object.
`conf.int`	Include confidence intervals? Default is FALSE.
`conf.level`	How wide confidence intervals should be, if requested. Default is .95.
`...`	Other arguments (usually ignored)

Value

A data.frame with columns matching those appropriate for the model type per `glance` documentation.

See Also

`glance`

`weights_tests` *Test whether sampling weights are needed*

Description

Use the tests proposed in Pfeffermann and Sverchkov (1999) and DuMouchel and Duncan (1983) to check whether a regression model is specified correctly without weights.

Usage

```
weights_tests(
  model,
  weights,
  data,
  model_output = TRUE,
  test = NULL,
  sims = 1000,
  digits = getOption("jtools-digits", default = 2)
)
```

Arguments

`model`	The fitted model, without weights
`weights`	The name of the weights column in `model`'s data frame or a vector of weights equal in length to the number of observations included in `model`.
`data`	The data frame with the data fed to the fitted model and the weights
`model_output`	Should a summary of the model with weights as predictor be printed? Default is TRUE, but you may not want it if you are trying to declutter a document.
`test`	Which type of test should be used in the ANOVA? The default, `NULL`, chooses based on the model type ("F" for linear models). This argument is passed to `anova`.
`sims`	The number of bootstrap simulations to use in estimating the variance of the residual correlation. Default is 1000, but for publications or when computing power/time is sufficient, a higher number is better.
`digits`	An integer specifying the number of digits past the decimal to report in the output. Default is 3. You can change the default number of digits for all jtools functions with `options("jtools-digits" = digits)` where digits is the desired number.

Details

This function is a wrapper for the two tests implemented in this package that test whether your regression model is correctly specified. The first is `wgttest`, an R adaptation of the Stata macro of the same name. This test can otherwise be referred to as the DuMouchel-Duncan test. The other test is the Pfeffermann-Sverchkov test, which can be accessed directly with `pf_sv_test`.

For more details on each, visit the documentation on the respective functions. This function just runs each of them for you.

References

DuMouchel, W. H. & Duncan, D.J. (1983). Using sample survey weights in multiple regression analyses of stratified samples. *Journal of the American Statistical Association*, *78*. 535-543.

Nordberg, L. (1989). Generalized linear modeling of sample survey data. *Journal of Official Statistics; Stockholm*, *5*, 223-239.

Pfeffermann, D., & Sverchkov, M. (1999). Parametric and semi-parametric estimation of regression models fitted to survey data. *Sankhya: The Indian Journal of Statistics*, *61*. 166-186.

See Also

Other survey tools: `pf_sv_test()`, `svycor()`, `svysd()`, `wgttest()`

Examples

```
# Note: This is a contrived example to show how the function works,
# not a case with actual sammpling weights from a survey vendor
if (requireNamespace("boot")) {
  states <- as.data.frame(state.x77)
  set.seed(100)
  states$wts <- runif(50, 0, 3)
  fit <- lm(Murder ~ Illiteracy + Frost, data = states)
  weights_tests(model = fit, data = states, weights = wts, sims = 100)
}
```

`wgttest` *Test whether sampling weights are needed*

Description

Use the DuMouchel-Duncan (1983) test to assess the need for sampling weights in your linear regression analysis.

Usage

```
wgttest(
  model,
  weights,
  data = NULL,
  model_output = FALSE,
  test = NULL,
  digits = getOption("jtools-digits", default = 3)
)
```

Arguments

model	The unweighted linear model (must be lm, glm, see details for other types) you want to check.
weights	The name of the weights column in model's data frame or a vector of weights equal in length to the number of observations included in model.
data	The data frame with the data fed to the fitted model and the weights
model_output	Should a summary of the model with weights as predictor be printed? Default is FALSE since the output can be very long for complex models.
test	Which type of test should be used in the ANOVA? The default, NULL, chooses based on the model type ("F" for linear models). This argument is passed to anova.
digits	An integer specifying the number of digits past the decimal to report in the output. Default is 3. You can change the default number of digits for all jtools functions with options("jtools-digits" = digits) where digits is the desired number.

Details

This is designed to be similar to the wgttest macro for Stata (http://fmwww.bc.edu/repec/bocode/w/wgttest.html). This method, advocated for by DuMouchel and Duncan (1983), is fairly straightforward. To decide whether weights are needed, the weights are added to the linear model as a predictor and interaction with each other predictor. Then, an omnibus test of significance is performed to compare the weights-added model to the original; if insignificant, weights are not significantly related to the result and you can use the more efficient estimation from unweighted OLS.

It can be helpful to look at the created model using model_output = TRUE to see which variables might be the ones affected by inclusion of weights.

This test can support most GLMs in addition to LMs, a use validated by Nordberg (1989). This, to my knowledge, is different from the Stata macro. It does not work for mixed models (e.g., lmer or lme) though it could plausibly be implemented. However, there is no scholarly consensus how to properly incorporate weights into mixed models. There are other types of models that may work, but have not been tested. The function is designed to be compatible with as many model types as possible, but the user should be careful to make sure s/he understands whether this type of test is appropriate for the model being considered. DuMouchel and Duncan (1983) were only thinking about linear regression when the test was conceived. Nordberg (1989) validated its use with generalized linear models, but to this author's knowledge it has not been tested with other model types.

References

DuMouchel, W. H. & Duncan, D.J. (1983). Using sample survey weights in multiple regression analyses of stratified samples. *Journal of the American Statistical Association*, *78*. 535-543.

Nordberg, L. (1989). Generalized linear modeling of sample survey data. *Journal of Official Statistics; Stockholm*, *5*, 223–239.

Winship, C. & Radbill, L. (1994). Sampling weights and regression analysis. *Sociological Methods and Research*, *23*, 230-257.

See Also

Other survey tools: pf_sv_test(), svycor(), svysd(), weights_tests()

Examples

```
# First, let's create some fake sampling weights
wts <- runif(50, 0, 5)
# Create model
fit <- lm(Income ~ Frost + Illiteracy + Murder,
          data = as.data.frame(state.x77))
# See if the weights change the model
wgttest(fit, weights = wts)

# With a GLM
wts <- runif(100, 0, 2)
x <- rnorm(100)
y <- rbinom(100, 1, .5)
fit <- glm(y ~ x, family = binomial)
wgttest(fit, wts)
## Can specify test manually
wgttest(fit, weights = wts, test = "Rao")

# Quasi family is treated differently than likelihood-based
## Dobson (1990) Page 93: Randomized Controlled Trial (plus some extra values):
counts <- c(18,17,15,20,10,20,25,13,12,18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,18)
treatment <- gl(3,6)
glm.D93 <- glm(counts ~ outcome + treatment, family = quasipoisson)
wts <- runif(18, 0, 3)
wgttest(glm.D93, weights = wts)
```

---

wrap_str *cat, message, warning, and stop wrapped to fit the console's width.*

Description

These are convenience functions that format printed output to fit the width of the user's console.

Usage

```
wrap_str(..., sep = "")

cat_wrap(..., brk = "")

warn_wrap(..., brk = "\n", class = NULL, call. = FALSE)

stop_wrap(
  ...,
```

```
  brk = "\n",
  trace = rlang::trace_back(bottom = rlang::caller_env()),
  class = NULL,
  call = rlang::caller_env()
)

msg_wrap(..., class = NULL, brk = "\n")
```

Arguments

...	Objects to print. For `stop_wrap()`, `warn_wrap()`, and `msg_wrap()`, any named objects are instead diverted to the ... argument of `rlang::abort()`, `rlang::warn()`, and `rlang::inform()`, respectively.
sep	Separator between ..., Default: "
brk	What should the last character of the message/warning/error be? Default is `"\n"`, meaning the console output ends with a new line.
class	Subclass of the condition.
call.	Here for legacy reasons. It is ignored.
trace	A trace object created by `trace_back()`.
call	The actual calling environment to report in the error message. By default, `rlang::caller_env()`.

Details

The point of these functions is to allow you to print output/messages/warnings/errors to the console without having to figure out where to place newline characters. These functions get the width of the console from the `"width"` option, which in most editors adjusts dynamically as you resize.

So instead of writing a warning like this:

```
warning("I have to give you this very important message that may be too\n",
        "wide for your screen")
```

You can do it like this:

```
warn_wrap("I have to give you this very important message that may be
          too wide for your screen")
```

And the function will automatically insert line breaks to fit the console. As a note, it will also ignore any newlines you insert. This means you can make your own fit your editor's screen and indent in the middle of a string without that formatting being carried over into the output.

wtd.sd *Weighted standard deviation calculation*

Description

This function calculates standard deviations with weights and is a counterpart to the built-in `weighted.mean` function.

Usage

```
wtd.sd(x, weights)
```

Arguments

`x`	A vector of values for which you want the standard deviation
`weights`	A vector of weights equal in length to `x`

%nin% *Not* %in%

Description

This function does the very opposite of `%in%`

Usage

```
x %nin% table
```

Arguments

`x`	An object
`table`	The object you want see if `x` is not in

Value

A logical vector

See Also

Other subsetters: `%not%()`

`%not%` *Subsetting operators*

Description

`%just%` and `%not%` are subsetting convenience functions for situations when you would do `x[x %in% y]` or `x[x %nin% y]`. See details for behavior when `x` is a data frame or matrix.

Usage

```
x %not% y

x %not% y <- value

x %just% y

x %just% y <- value

## Default S3 method:
x %not% y

## Default S3 method:
x %not% y <- value

## S3 method for class 'data.frame'
x %not% y

## S3 method for class 'data.frame'
x %not% y <- value

## S3 method for class 'matrix'
x %not% y

## S3 method for class 'matrix'
x %not% y <- value

## S3 method for class 'list'
x %not% y

## S3 method for class 'list'
x %not% y <- value

## Default S3 method:
x %just% y

## Default S3 method:
x %just% y <- value
```

```
## S3 method for class 'data.frame'
x %just% y

## S3 method for class 'data.frame'
x %just% y <- value

## S3 method for class 'matrix'
x %just% y

## S3 method for class 'matrix'
x %just% y <- value

## S3 method for class 'list'
x %just% y

## S3 method for class 'list'
x %just% y <- value
```

Arguments

x	Object to subset
y	List of items to include if they are/aren't in x
value	The object(s) to assign to the subsetted x

Details

The behavior of %not% and %just% are different when you're subsetting data frames or matrices. The subset y in this case is interpreted as **column** names or indices.

You can also make assignments to the subset in the same way you could if subsetting with brackets.

Value

All of x that are in y (%just%) or all of x that are not in y (%not%).

See Also

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Other subsetters: %nin%()

Examples

```
x <- 1:5
y <- 3:8

x %just% y # 3 4 5
x %not% y # 1 2

# Assignment works too
x %just% y <- NA # 1 2 NA NA NA
x %not% y <- NA # NA NA 3 4 5

mtcars %just% c("mpg", "qsec", "cyl") # keeps only columns with those names
mtcars %not% 1:5 # drops columns 1 through 5

# Assignment works for data frames as well
mtcars %just% c("mpg", "qsec") <- gscale(mtcars, c("mpg", "qsec"))
mtcars %not% c("mpg", "qsec") <- gscale(mtcars %not% c("mpg", "qsec"))
```

Index