Package 'jtools'

September 23, 2018

Type Package

Title Analysis and Presentation of Social Scientific Data

Version 1.1.1

Description This is a collection of tools that the author (Jacob) has written for the purpose of more efficiently understanding and sharing the results of (primarily) regression analyses. There are a number of functions focused specifically on the interpretation and presentation of interactions. Just about everything supports models from the survey package.

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

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports ggplot2, crayon, cli, magrittr

Suggests boot, broom, brms, cowplot, ggstance, glue, huxtable (>= 3.0.0), kableExtra, lme4, lmerTest, methods, pbkrtest, quantreg, RColorBrewer, rstanarm, sandwich, stats4, survey, weights, knitr, rmarkdown, rmdformats, testthat

RoxygenNote 6.1.0.9000

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Jacob A. Long <long.1377@osu.edu>

Repository CRAN

Date/Publication 2018-09-23 13:30:02 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?

as_huxtable.sim_slopes *Create tabular output for simple slopes analysis*

Description

This function converts a `sim_slopes` object into a `huxtable` object, making it suitable for use in external documents.

Usage

```
as_huxtable.sim_slopes(x, format = "{estimate} ({std.error})",
  sig.levels = c(`***` = 0.001, `**` = 0.01, `*` = 0.05, `#` = 0.1),
  digits = getOption("jtools-digits", 2), conf.level = 0.95, ...)
```

Arguments

x	The sim_slopes() object.
format	The method for sharing the slope and associated uncertainty. Default is "{estimate} ({std.error})". See the instructions for the error_format argument of export_summs() for more on your options.
sig.levels	A named vector in which the values are potential p value thresholds and the names are significance markers (e.g., "*") for when p values are below the threshold. Default is c(`***` = .001, `**` = .01, `*` = .05, `#` = .1).
digits	How many digits should the outputted table round to? Default is 2.
conf.level	How wide the confidence interval should be, if it is used. .95 (95% interval) is the default.
...	Ignored.

Details

For more on what you can do with a huxtable, see **huxtable**.

---

cat_plot *Plot interaction effects between categorical predictors.*

Description

cat_plot is a complementary function to interact_plot() that is designed for plotting interactions when both predictor and moderator(s) are categorical (or, in R terms, factors).

Usage

```
cat_plot(model, pred, modx = NULL, mod2 = NULL, data = NULL,
  geom = c("point", "line", "bar", "boxplot"), pred.values = NULL,
  modx.values = NULL, mod2.values = NULL, interval = TRUE,
  plot.points = FALSE, point.shape = FALSE, vary.lty = FALSE,
  centered = "all", int.type = c("confidence", "prediction"),
  int.width = 0.95, line.thickness = 1, point.size = 1.5,
  pred.point.size = 3.5, jitter = 0.1, geom.alpha = NULL,
  dodge.width = NULL, errorbar.width = NULL,
  interval.geom = c("errorbar", "linerange"),
  outcome.scale = "response", robust = FALSE, cluster = NULL,
  vcov = NULL, pred.labels = NULL, modx.labels = NULL,
  mod2.labels = NULL, set.offset = 1, x.label = NULL,
  y.label = NULL, main.title = NULL, legend.main = NULL,
  color.class = "CUD Bright", ...)
```

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	A categorical predictor variable that will appear on the x-axis.
modx	A categorical moderator variable.
mod2	For three-way interactions, the second categorical moderator.
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.
geom	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. • "boxplot": This geom plots a dot and whisker plot. These can be useful for understanding the distribution of the observed data without having to plot all the observed points (especially helpful with larger data sets). **However**, it is important to note the boxplots are not based on the model whatsoever.
pred.values	Which values of the predictor should be included in the plot? By default, all levels are included.
modx.values	For which values of the moderator should lines be plotted? Default is NULL. If NULL, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If "plus-minus", plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose "terciles" to split the data into equally-sized groups and choose the point at the mean of each of those groups. If the moderator is a factor variable and modx.values is NULL, each level of the factor is included. You may specify any subset of the factor levels (e.g., c("Level 1", "Level 3")) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

`mod2.values` For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as `modx.values`.

`interval` Logical. If `TRUE`, plots confidence/prediction intervals.

`plot.points` Logical. If `TRUE`, plots the actual data points as a scatterplot on top of the interaction lines. Note that if `geom = "bar"`, this will cause the bars to become transparent so you can see the points.

`point.shape` For plotted points—either of observed data or predicted values with the "point" or "line" geoms—should the shape of the points vary by the values of the factor? This is especially useful if you aim to be black and white printing- or colorblind-friendly.

`vary.lty` Should the resulting plot have different shapes for each line in addition to colors? Defaults to `TRUE`.

`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`, `modx`, and `mod2` variables are never centered.

`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.

`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 2.

`pred.point.size`
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.

`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 0.1, but increase as needed if your points are overlapping too much or set to 0 for no jitter. 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.

`geom.alpha` What should the alpha aesthetic be for the plotted lines/bars? Default is NULL, which means it is set depending on the value of `geom` and `plot.points`.

`dodge.width` What should the `width` argument to `ggplot2::position_dodge()` be? Default is NULL, which means it is set depending on the value of `geom`.

`errorbar.width` How wide should the error bars be? Default is NULL, meaning it is set depending on the value `geom`. Ignored if `interval` is FALSE.

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.
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.
pred.labels	A character vector of equal length to the number of factor levels of the predictor (or number specified in predvals). If NULL, the default, the factor labels are used.
modx.labels	A character vector of labels for each level of the moderator values, provided in the same order as the modx.values argument. If NULL, the values themselves are used as labels unless modx,values is also NULL. In that case, "+1 SD" and "-1 SD" are used.
mod2.labels	A character vector of labels for each level of the 2nd moderator values, provided in the same order as the mod2.values argument. If NULL, the values themselves are used as labels unless mod2.values is also NULL. In that case, "+1 SD" and "-1 SD" are used.
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.
main.title	A character object that will be used as an overall title for the plot. If NULL, no main title is used.
legend.main	A character object that will be used as the title that appears above the legend. If NULL, the name of the moderating variable is used.
color.class	Any palette argument accepted by scale_colour_brewer. Default is "Set2". You may also simply supply a vector of colors accepted by ggplot2 and of equal length to the number of moderator levels.
...	extra arguments passed to make_predictions

Details

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

The function is designed for two and three-way interactions. For additional terms, the `effects` package may be better suited to the task.

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(variable)`, provide only the variable name to `pred`, `modx`, or `mod2` and supply the original data separately to the `data` argument.

Info about offsets:

Offsets are partially supported by this function with important limitations. First of all, only a single offset per model is supported. Second, it is best in general to specify offsets with the offset argument of the model fitting function rather than in the formula. You are much more likely to have success if you provide the data 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.

See Also

Other interaction tools: `interact_plot`, `johnson_neyman`, `probe_interaction`, `sim_slopes`

Examples

```
library(ggplot2)
fit <- lm(price ~ cut * color, data = diamonds)
cat_plot(fit, pred = color, modx = cut, interval = TRUE)

# 3-way interaction

## Will first create a couple dichotomous factors to ensure full rank
mpg2 <- mpg
mpg2$auto <- "auto"
mpg2$auto[mpg2$trans %in% c("manual(m5)", "manual(m6)")] <- "manual"
mpg2$auto <- factor(mpg2$auto)
mpg2$fwd <- "2wd"
mpg2$fwd[mpg2$drv == "4"] <- "4wd"
mpg2$fwd <- factor(mpg2$fwd)
## Drop the two cars with 5 cylinders (rest are 4, 6, or 8)
mpg2 <- mpg2[mpg2$cyl != "5",]
mpg2$cyl <- factor(mpg2$cyl)
## Fit the model
```

```
fit3 <- lm(cty ~ cyl * fwd * auto, data = mpg2)

# The line geom looks good for an ordered factor predictor
cat_plot(fit3, pred = cyl, modx = fwd, mod2 = auto, geom = "line",
 interval = TRUE)
```

---

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 = TRUE, 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 TRUE.
`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, scaling, and centering tools: `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 `wec::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 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 <<long.1377@osu.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, scaling, and centering tools: `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, centered = "all", plot.points = FALSE,
  interval = FALSE, data = 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, color.class = NULL, line.thickness = 1.1,
  point.size = 2.5, jitter = 0, rug = FALSE, rug.sides = "b", ...)
```

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.
`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`, `modx`, and `mod2` 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.

`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 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. 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.

`color.class` See jtools_colors for details on the types of arguments accepted. Default is "CUD Bright" for factor moderators, "Blues" for +/- SD and user-specified `modx.values` values.

`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 2.

`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 "b", meaning bottom. "t" and/or "b" show the distribution of the predictor while "l" and/or "r" show the distribution of the response. "bl" is a good option to show both the predictor and response.

`...` 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 <<long.1377@osu.edu>>

See Also

`interact_plot` plots interaction effects, producing plots like this function but with separate lines for different levels of a moderator.

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.squar
- summ.svyglm = c(N = "nobs", R2 = "r.squared"),
- summ.merMod = c(N = "nobs", AIC = "AIC", BIC = "BIC", `R2 (fixed)` = "r.squar
- 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)
```

```
}
```

`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 = TRUE, 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 TRUE.
`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 `wec::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 <<long.1377@osu.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, scaling, and centering tools: `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` *Plot interaction effects in regression models*

Description

`interact_plot` plots regression lines at user-specified levels of a moderator variable to explore interactions. The plotting is done with `ggplot2` rather than base graphics, which some similar functions use.

Usage

```
interact_plot(model, pred, modx, modx.values = NULL, mod2 = NULL,
  mod2.values = NULL, centered = "all", data = NULL,
  plot.points = FALSE, interval = FALSE, int.type = c("confidence",
  "prediction"), int.width = 0.95, outcome.scale = "response",
  linearity.check = FALSE, facet.modx = FALSE, robust = FALSE,
  cluster = NULL, vcov = NULL, set.offset = 1, x.label = NULL,
  y.label = NULL, pred.labels = NULL, modx.labels = NULL,
  mod2.labels = NULL, main.title = NULL, legend.main = NULL,
  color.class = NULL, line.thickness = 1.1, vary.lty = TRUE,
  point.size = 2.5, point.shape = FALSE, jitter = 0, rug = FALSE,
  rug.sides = "b", ...)
```

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.
`modx`	The name of the moderator variable involved in the interaction. This can be a bare name or string.
`modx.values`	For which values of the moderator should lines be plotted? Default is `NULL`. If `NULL`, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If `"plus-minus"`, plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose `"terciles"` to split the data into equally-sized groups and choose the point at the mean of each of those groups. If the moderator is a factor variable and `modx.values` is `NULL`, each level of the factor is included. You may specify any subset of the factor levels (e.g., `c("Level 1", "Level 3")`) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

mod2 — Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.

mod2.values — For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as modx.values.

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, modx, and mod2 variables are never centered.

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.

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.

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".

linearity.check — For two-way interactions only. If TRUE, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set plot.points = TRUE and use modx.values = "terciles" with this option.

facet.modx — Create separate panels for each level of the moderator? Default is FALSE, except when linearity.check is TRUE.

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 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If `NULL`, the default, the factor labels are used.

`modx.labels` A character vector of labels for each level of the moderator values, provided in the same order as the `modx.values` argument. If `NULL`, the values themselves are used as labels unless `modx,values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`mod2.labels` A character vector of labels for each level of the 2nd moderator values, provided in the same order as the `mod2.values` argument. If `NULL`, the values themselves are used as labels unless `mod2.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" 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.

`legend.main` A character object that will be used as the title that appears above the legend. If `NULL`, the name of the moderating variable is used.

`color.class` See jtools_colors for details on the types of arguments accepted. Default is "CUD Bright" for factor moderators, "Blues" for +/- SD and user-specified `modx.values` values.

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

`vary.lty` Should the resulting plot have different shapes for each line in addition to colors? Defaults to `TRUE`.

`point.size` What size should be used for observed data when `plot.points` is TRUE? Default is 2.

`point.shape` For plotted points—either of observed data or predicted values with the "point" or "line" geoms—should the shape of the points vary by the values of the factor? This is especially useful if you aim to be black and white printing- or colorblind-friendly.

`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 "b", meaning bottom. "t" and/or "b" show the distribution of the predictor while "l" and/or "r" show the distribution of the response. "bl" is a good option to show both the predictor and response.

`...` extra arguments passed to `make_predictions`

Details

This function provides a means for plotting conditional effects for the purpose of exploring interactions in regression models.

The function is designed for two and three-way interactions. For additional terms, the **effects** package may be better suited to the task.

This function supports nonlinear and generalized linear models and by default will plot them on their original scale (`outcome.scale = "response"`). To plot them on the linear scale, use "link" for `outcome.scale`.

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(variable)`, put its name in quotes or backticks in the argument.

Details on how observed data are split in multi-pane plots:

If you set `plot.points = TRUE` and request a multi-pane (facetted) plot either with a second moderator, `linearity.check = TRUE`, or `facet.modx = TRUE`, the observed data are split into as many groups as there are panes and plotted separately. If the moderator is a factor, then the way this happens will be very intuitive since it's obvious which values go in which pane. The rest of this section will address the case of continuous moderators.

My recommendation is that you use `modx.values = "terciles"` or `mod2.values = "terciles"` when you want to plot observed data on multi-pane plots. When you do, the data are split into three approximately equal-sized groups with the lowest third, middle third, and highest third of the data split accordingly. You can replicate this procedure using `Hmisc::cut2()` with `g = 3` from the `Hmisc` package. Sometimes, the groups will not be equal in size because the number of observations is not divisible by 3 and/or there are multiple observations with the same value at one of the cut points.

Otherwise, a more ad hoc procedure is used to split the data. Quantiles are found for each `mod2.values` or `modx.values` value. These are not the quantiles used to split the data, however, since we want the plotted lines to represent the slope at a typical value in the group. The next step, then, is to take the mean of each pair of neighboring quantiles and use these as the cut points.

For example, if the `mod2.values` are at the 25th, 50th, and 75th percentiles of the distribution of the moderator, the data will be split at the 37.5th and and 62.5th percentiles. When the variable is normally distributed, this will correspond fairly closely to using terciles.

Info about offsets:

Offsets are partially supported by this function with important limitations. First of all, only a single offset per model is supported. Second, it is best in general to specify offsets with the offset argument of the model fitting function rather than in the formula. You are much more likely to have success if you provide the data 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 <<long.1377@osu.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. http://dx.doi.org/10.1207/s15327906mbr4003_5

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.

Hainmueller, J., Mummolo, J., & Xu, Y. (2016). How much should we trust estimates from multiplicative interaction models? Simple tools to improve empirical practice. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.2739221

See Also

`plotSlopes` from **rockchalk** performs a similar function, but with R's base graphics—this function is meant, in part, to emulate its features.

`sim_slopes` performs a simple slopes analysis with a similar argument syntax to this function.

Other interaction tools: `cat_plot`, `johnson_neyman`, `probe_interaction`, `sim_slopes`

Examples

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

# Using interval feature
fit <- lm(accel ~ mag * dist, data = attenu)
interact_plot(fit, pred = mag, modx = dist, interval = TRUE,
  int.type = "confidence", int.width = .8)

# Using second moderator
fit <- lm(Income ~ HSGrad * Murder * Illiteracy, data = states)
interact_plot(model = fit, pred = Murder, modx = Illiteracy, mod2 = HSGrad)
```

```
# 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)
interact_plot(regmodel, pred = ell, modx = meals)
}

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

## End(Not run)
```

---

johnson_neyman *Calculate Johnson-Neyman intervals for 2-way interactions*

Description

johnson_neyman finds so-called "Johnson-Neyman" intervals for understanding where simple slopes are significant in the context of interactions in multiple linear regression.

Usage

```
johnson_neyman(model, pred, modx, vmat = NULL, alpha = 0.05,
  plot = TRUE, control.fdr = FALSE, line.thickness = 0.5,
  df = "residual", digits = getOption("jtools-digits", 2),
  critical.t = NULL, sig.color = "#00BFC4", insig.color = "#F8766D",
  mod.range = NULL, title = "Johnson-Neyman plot")
```

Arguments

model	A regression model. It is tested with lm, glm, and svyglm objects, but others may work as well. It should contain the interaction of interest. Be aware that just because the computations work, this does not necessarily mean the procedure is appropriate for the type of model you have.
pred	The predictor variable involved in the interaction.
modx	The moderator variable involved in the interaction.

vmat	Optional. You may supply the variance-covariance matrix of the coefficients yourself. This is useful if you are using robust standard errors, as you could if using the **sandwich** package.
alpha	The alpha level. By default, the standard 0.05.
plot	Should a plot of the results be printed? Default is TRUE. The ggplot2 object is returned either way.
control.fdr	Logical. Use the procedure described in Esarey & Sumner (2017) to limit the false discovery rate? Default is FALSE. See details for more on this method.
line.thickness	How thick should the predicted line be? This is passed to geom_path as the size argument, but because of the way the line is created, you cannot use geom_path to modify the output plot yourself.
df	How should the degrees of freedom be calculated for the critical test statistic? Previous versions used the large sample approximation; if alpha was .05, the critical test statistic was 1.96 regardless of sample size and model complexity. The default is now "residual", meaning the same degrees of freedom used to calculate p values for regression coefficients. You may instead choose any number or "normal", which reverts to the previous behavior. The argument is not used if control.fdr = TRUE.
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.
critical.t	If you want to provide the critical test statistic instead relying on a normal or *t* approximation, or the control.fdr calculation, you can give that value here. This allows you to use other methods for calculating it.
sig.color	Sets the color for areas of the Johnson-Neyman plot where the slope of the moderator is significant at the specified level. "black" can be a good choice for greyscale publishing.
insig.color	Sets the color for areas of the Johnson-Neyman plot where the slope of the moderator is insignificant at the specified level. "grey" can be a good choice for greyscale publishing.
mod.range	The range of values of the moderator (the x-axis) to plot. By default, this goes from one standard deviation below the observed range to one standard deviation above the observed range and the observed range is highlighted on the plot. You could instead choose to provide the actual observed minimum and maximum, in which case the range of the observed data is not highlighted in the plot. Provide the range as a vector, e.g., c(0, 10).
title	The plot title. "Johnson-Neyman plot" by default.

Details

The interpretation of the values given by this function is important and not always immediately intuitive. For an interaction between a predictor variable and moderator variable, it is often the case that the slope of the predictor is statistically significant at only some values of the moderator. For example, perhaps the effect of your predictor is only significant when the moderator is set at some high value.

The Johnson-Neyman interval provides the two values of the moderator at which the slope of the predictor goes from non-significant to significant. Usually, the predictor's slope is only significant *outside* of the range given by the function. The output of this function will make it clear either way.

One weakness of this method of probing interactions is that it is analagous to making multiple comparisons without any adjustment to the alpha level. Esarey & Sumner (2017) proposed a method for addressing this, which is implemented in the `interactionTest` package. This function implements that procedure with modifications to the `interactionTest` code (that package is not required to use this function). If you set `control.fdr = TRUE`, an alternative *t* statistic will be calculated based on your specified alpha level and the data. This will always be a more conservative test than when `control.fdr = FALSE`. The printed output will report the calculated critical *t* statistic.

This technique is not easily ported to 3-way interaction contexts. You could, however, look at the J-N interval at two different levels of a second moderator. This does forgo a benefit of the J-N technique, which is not having to pick arbitrary points. If you want to do this, just use the `sim_slopes` function's ability to handle 3-way interactions and request Johnson-Neyman intervals for each.

Value

`bounds`	The two numbers that make up the interval.
`cbands`	A dataframe with predicted values of the predictor's slope and lower/upper bounds of confidence bands if you would like to make your own plots
`plot`	The `ggplot` object used for plotting. You can tweak the plot like you could any other from `ggplot`.

Author(s)

Jacob Long <<long.1377@osu.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. http://doi.org/10.1207/s15327906mbr4003_5

Esarey, J., & Sumner, J. L. (2017). Marginal effects in interaction models: Determining and controlling the false positive rate. Comparative Political Studies, 1–33. Advance online publication. https://doi.org/10.1177/0010414017730080

Johnson, P.O. & Fay, L.C. (1950). The Johnson-Neyman technique, its theory and application. *Psychometrika*, *15*, 349-367. http://doi.org/10.1007/BF02288864

See Also

Other interaction tools: `cat_plot`, `interact_plot`, `probe_interaction`, `sim_slopes`

Examples

```
# Using a fitted lm model
states <- as.data.frame(state.x77)
states$HSGrad <- states$`HS Grad`
```

```
fit <- lm(Income ~ HSGrad + Murder*Illiteracy,
  data = states)
johnson_neyman(model = fit, pred = Murder,
  modx = Illiteracy)
```

---

jtools_colors *Color palettes in* jtools *functions*

Description

jtools combines several options into the color.class argument in plotting functions.

Details

The argument to color.class in functions like interact_plot, cat_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.

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 (e.g., 3 colors for 3 values of the moderator in interact_plot). 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 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 http://jfly.iam.u-tokyo.ac.jp/color/ for more.

j_summ *Regression summaries with options*

Description

j_summ is an alias for summ. To get specific documentation, choose the appropriate link to the type of model that you want to summarize from the details section.

Usage

```
j_summ(model, ...)
```

Arguments

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

Details

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

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_predictions` *Generate predicted data for plotting results of regression models*

Description

This is an alternate interface to the underlying tools that make up `interact_plot()`, `effect_plot()`, and `cat_plot()`. `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,
  modx = NULL, modx.values = NULL, mod2 = NULL, mod2.values = NULL,
  centered = "all", data = NULL, plot.points = FALSE,
  interval = FALSE, int.width = 0.95, outcome.scale = "response",
  linearity.check = FALSE, robust = FALSE, cluster = NULL,
  vcov = NULL, set.offset = 1, pred.labels = NULL,
  modx.labels = NULL, mod2.labels = NULL, int.type = c("confidence",
  "prediction"), preds.per.level = 100, force.cat = FALSE,
  facet.modx = linearity.check, ...)
```

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.
...	Ignored.
pred	The name of the predictor variable involved in the interaction. This must be a string.
pred.values	Which values of the predictor should be included in the plot? By default, all levels are included.
modx	The name of the moderator variable involved in the interaction. This must be a string.

`modx.values` For which values of the moderator should lines be plotted? Default is `NULL`. If `NULL`, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If `"plus-minus"`, plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose `"terciles"` to split the data into equally-sized groups and choose the point at the mean of each of those groups.

If the moderator is a factor variable and `modx.values` is `NULL`, each level of the factor is included. You may specify any subset of the factor levels (e.g., `c("Level 1", "Level 3")`) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

`mod2` Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.

`mod2.values` For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as `modx.values`.

`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`, `modx`, and `mod2` variables are never centered.

`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.

`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`.

`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"`.

`linearity.check`

For two-way interactions only. If `TRUE`, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See

Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set `plot.points = TRUE` and use `modx.values = "terciles"` with this option.

`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.

`pred.labels` A character vector of 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If `NULL`, the default, the factor labels are used.

`modx.labels` A character vector of labels for each level of the moderator values, provided in the same order as the `modx.values` argument. If `NULL`, the values themselves are used as labels unless `modx,values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`mod2.labels` A character vector of labels for each level of the 2nd moderator values, provided in the same order as the `mod2.values` argument. If `NULL`, the values themselves are used as labels unless `mod2.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

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

`preds.per.level` For continuous predictors, a series of equally spaced points across the observed range of the predictor are used to create the lines for each level of the moderator. Use this to choose how many points are used for that. Default is 100, but for complicated models larger numbers may better capture the curvature.

`force.cat` If `TRUE`, treats numeric predictor as categorical. This can be helpful when you have 0/1 dummy variables that you don't want to plot as if intermediate values are possible.

`facet.modx` Create separate panels for each level of the moderator? Default is FALSE, except when `linearity.check` is TRUE.

See Also

Other plotting tools: `plot_predictions`

`make_predictions.brmsfit` *Make predictions for brmsfit models*

Description

This method adds support for `plot_predictions`, `interact_plot`, `cat_plot`, and `effect_plot` for models fit with `brms`.

Usage

```
## S3 method for class 'brmsfit'
make_predictions(model, pred, pred.values = NULL,
  modx = NULL, modx.values = NULL, mod2 = NULL, mod2.values = NULL,
  centered = "all", data = NULL, plot.points = FALSE,
  interval = TRUE, int.width = 0.95, estimate = "mean",
  linearity.check = FALSE, set.offset = 1, pred.labels = NULL,
  modx.labels = NULL, mod2.labels = NULL, preds.per.level = 100,
  force.cat = FALSE, facet.modx = linearity.check, ...)
```

Arguments

`model`	A `brmsfit` model.
`pred`	The name of the predictor variable involved in the interaction. This must be a string.
`pred.values`	Which values of the predictor should be included in the plot? By default, all levels are included.
`modx`	The name of the moderator variable involved in the interaction. This must be a string.
`modx.values`	For which values of the moderator should lines be plotted? Default is `NULL`. If `NULL`, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If `"plus-minus"`, plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose `"terciles"` to split the data into equally-sized groups and choose the point at the mean of each of those groups. If the moderator is a factor variable and `modx.values` is `NULL`, each level of the factor is included. You may specify any subset of the factor levels (e.g., `c("Level 1", "Level 3")`) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.
`mod2`	Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.
`mod2.values`	For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as `modx.values`.

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, modx, and mod2 variables are never centered.

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.

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.

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.

estimate — Should estimates be based on mean or median simulation? Default is "mean".

linearity.check — For two-way interactions only. If TRUE, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set plot.points = TRUE and use modx.values = "terciles" with this option.

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.

pred.labels — A character vector of 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If NULL, the default, the factor labels are used.

modx.labels — A character vector of labels for each level of the moderator values, provided in the same order as the modx.values argument. If NULL, the values themselves are used as labels unless modx,values is also NULL. In that case, "+1 SD" and "-1 SD" are used.

mod2.labels — A character vector of labels for each level of the 2nd moderator values, provided in the same order as the mod2.values argument. If NULL, the values themselves are used as labels unless mod2.values is also NULL. In that case, "+1 SD" and "-1 SD" are used.

preds.per.level — For continuous predictors, a series of equally spaced points across the observed range of the predictor are used to create the lines for each level of the moderator.

	Use this to choose how many points are used for that. Default is 100, but for complicated models larger numbers may better capture the curvature.
force.cat	If TRUE, treats numeric predictor as categorical. This can be helpful when you have 0/1 dummy variables that you don't want to plot as if intermediate values are possible.
facet.modx	Create separate panels for each level of the moderator? Default is FALSE, except when linearity.check is TRUE.
...	Ignored.

---

make_predictions.merMod

Generate predicted data for merMod models

Description

This function produces predicted data for merMod models, including confidence intervals calculated via one of two methods if requested.

Usage

```
## S3 method for class 'merMod'
make_predictions(model, pred, pred.values = NULL,
  modx = NULL, modx.values = NULL, mod2 = NULL, mod2.values = NULL,
  centered = "all", data = NULL, plot.points = FALSE,
  interval = FALSE, int.width = 0.95, outcome.scale = "response",
  add.re.variance = FALSE, linearity.check = FALSE, set.offset = 1,
  pred.labels = NULL, modx.labels = NULL, mod2.labels = NULL,
  int.type = c("confidence", "prediction"), preds.per.level = 100,
  boot = FALSE, sims = 100, progress = "txt", force.cat = FALSE,
  facet.modx = linearity.check, ...)
```

Arguments

model	A merMod model.
pred	The name of the predictor variable involved in the interaction. This must be a string.
pred.values	Which values of the predictor should be included in the plot? By default, all levels are included.
modx	The name of the moderator variable involved in the interaction. This must be a string.
modx.values	For which values of the moderator should lines be plotted? Default is NULL. If NULL, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If "plus-minus", plots lines when the moderator is at +/- 1 standard deviation without the mean. You may

also choose "terciles" to split the data into equally-sized groups and choose the point at the mean of each of those groups.

If the moderator is a factor variable and modx.values is NULL, each level of the factor is included. You may specify any subset of the factor levels (e.g., c("Level 1", "Level 3")) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

mod2 — Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.

mod2.values — For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as modx.values.

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, modx, and mod2 variables are never centered.

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.

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.

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".

add.re.variance

Experimental. Adds variance specific to the *random* effects in the model. Often overwhelms the fixed effects variances and makes the plot uninterpretable.

linearity.check

For two-way interactions only. If TRUE, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set plot.points = TRUE and use modx.values = "terciles" with this option.

`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.

`pred.labels` A character vector of 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If `NULL`, the default, the factor labels are used.

`modx.labels` A character vector of labels for each level of the moderator values, provided in the same order as the `modx.values` argument. If `NULL`, the values themselves are used as labels unless `modx,values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`mod2.labels` A character vector of labels for each level of the 2nd moderator values, provided in the same order as the `mod2.values` argument. If `NULL`, the values themselves are used as labels unless `mod2.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

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

`preds.per.level` For continuous predictors, a series of equally spaced points across the observed range of the predictor are used to create the lines for each level of the moderator. Use this to choose how many points are used for that. Default is 100, but for complicated models larger numbers may better capture the curvature.

`boot` Use `lme4::bootMer()` to generate confidence intervals instead of estimating fixed effects variance with variance-covariance matrices? Default is FALSE but this is probably more defensible for publication-level output. See Details for a little bit more information.

`sims` How many bootstrap simulations should be used? Default is 100, but should usually be much higher. Just be aware that runtime may be considerable.

`progress` Should a progress bar be shown during bootstrapping and if so, how should it look? Default is "txt", which is probably what you want to use, but "none" will suppress the progress bar.

`force.cat` If `TRUE`, treats numeric predictor as categorical. This can be helpful when you have 0/1 dummy variables that you don't want to plot as if intermediate values are possible.

`facet.modx` Create separate panels for each level of the moderator? Default is FALSE, except when `linearity.check` is TRUE.

`...` Extra arguments passed to `lme4::bootMer()` if `boot` is TRUE.

Details

The ability to bootstrap the variances is not available through `interact_plot`, `effect_plot`, and `cat_plot` to keep those functions as simple as possible. Internally, `lme4::bootMer()` is called with with default arguments (`type = "parametric"`, `use.u = FALSE`). To get parallel processing, add the arguments `parallel = "multicore"` or `parallel = "snow"` and `ncpus =` the number of cores.

make_predictions.rq *Make predictions for quantile regression models*

Description

This method adds support for plot_predictions, interact_plot, cat_plot, and effect_plot for models fit with rq.

Usage

```
## S3 method for class 'rq'
make_predictions(model, pred, pred.values = NULL,
  modx = NULL, modx.values = NULL, mod2 = NULL, mod2.values = NULL,
  centered = "all", data = NULL, plot.points = FALSE,
  int.width = 0.95, outcome.scale = "response",
  linearity.check = FALSE, set.offset = 1, pred.labels = NULL,
  modx.labels = NULL, mod2.labels = NULL, int.type = c("confidence",
  "prediction"), preds.per.level = 100, force.cat = FALSE,
  se = c("nid", "iid", "ker"), facet.modx = linearity.check, ...)
```

Arguments

model	A rq model.
pred	The name of the predictor variable involved in the interaction. This must be a string.
pred.values	Which values of the predictor should be included in the plot? By default, all levels are included.
modx	The name of the moderator variable involved in the interaction. This must be a string.
modx.values	For which values of the moderator should lines be plotted? Default is NULL. If NULL, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If "plus-minus", plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose "terciles" to split the data into equally-sized groups and choose the point at the mean of each of those groups. If the moderator is a factor variable and modx.values is NULL, each level of the factor is included. You may specify any subset of the factor levels (e.g., c("Level 1", "Level 3")) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.
mod2	Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.
mod2.values	For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as modx.values.

`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`, `modx`, and `mod2` variables are never centered.

`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.

`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.

`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"`.

`linearity.check` For two-way interactions only. If `TRUE`, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set `plot.points = TRUE` and use `modx.values = "terciles"` with this option.

`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.

`pred.labels` A character vector of 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If `NULL`, the default, the factor labels are used.

`modx.labels` A character vector of labels for each level of the moderator values, provided in the same order as the `modx.values` argument. If `NULL`, the values themselves are used as labels unless `modx,values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`mod2.labels` A character vector of labels for each level of the 2nd moderator values, provided in the same order as the `mod2.values` argument. If `NULL`, the values themselves are used as labels unless `mod2.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

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

preds.per.level
: For continuous predictors, a series of equally spaced points across the observed range of the predictor are used to create the lines for each level of the moderator. Use this to choose how many points are used for that. Default is 100, but for complicated models larger numbers may better capture the curvature.

force.cat
: If TRUE, treats numeric predictor as categorical. This can be helpful when you have 0/1 dummy variables that you don't want to plot as if intermediate values are possible.

se
: One of "nid", "iid", "ker", standard error options defined by summary.rq.

facet.modx
: Create separate panels for each level of the moderator? Default is FALSE, except when linearity.check is TRUE.

...
: Ignored.

---

make_predictions.stanreg

Make predictions for stanreg models

Description

This method adds support for plot_predictions, interact_plot, cat_plot, and effect_plot for models fit with rstanarm.

Usage

```
## S3 method for class 'stanreg'
make_predictions(model, pred, pred.values = NULL,
  modx = NULL, modx.values = NULL, mod2 = NULL, mod2.values = NULL,
  centered = "all", data = NULL, plot.points = FALSE,
  interval = TRUE, int.width = 0.95, estimate = "mean",
  linearity.check = FALSE, set.offset = 1, pred.labels = NULL,
  modx.labels = NULL, mod2.labels = NULL, preds.per.level = 100,
  force.cat = FALSE, facet.modx = linearity.check, ...)
```

Arguments

model
: A stanreg model.

pred
: The name of the predictor variable involved in the interaction. This must be a string.

pred.values
: Which values of the predictor should be included in the plot? By default, all levels are included.

modx
: The name of the moderator variable involved in the interaction. This must be a string.

`modx.values` For which values of the moderator should lines be plotted? Default is `NULL`. If `NULL`, then the customary +/- 1 standard deviation from the mean as well as the mean itself are used for continuous moderators. If `"plus-minus"`, plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose `"terciles"` to split the data into equally-sized groups and choose the point at the mean of each of those groups.

If the moderator is a factor variable and `modx.values` is `NULL`, each level of the factor is included. You may specify any subset of the factor levels (e.g., `c("Level 1", "Level 3")`) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

`mod2` Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.

`mod2.values` For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as `modx.values`.

`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`, `modx`, and `mod2` variables are never centered.

`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.

`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`.

`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.

`estimate` Should estimates be based on mean or median simulation? Default is "mean".

`linearity.check`

For two-way interactions only. If `TRUE`, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set `plot.points = TRUE` and use `modx.values = "terciles"` with this option.

`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.

`pred.labels` A character vector of 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If `NULL`, the default, the factor labels are used.

`modx.labels` A character vector of labels for each level of the moderator values, provided in the same order as the `modx.values` argument. If `NULL`, the values themselves are used as labels unless `modx.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`mod2.labels` A character vector of labels for each level of the 2nd moderator values, provided in the same order as the `mod2.values` argument. If `NULL`, the values themselves are used as labels unless `mod2.values` is also `NULL`. In that case, "+1 SD" and "-1 SD" are used.

`preds.per.level`
For continuous predictors, a series of equally spaced points across the observed range of the predictor are used to create the lines for each level of the moderator. Use this to choose how many points are used for that. Default is 100, but for complicated models larger numbers may better capture the curvature.

`force.cat` If `TRUE`, treats numeric predictor as categorical. This can be helpful when you have 0/1 dummy variables that you don't want to plot as if intermediate values are possible.

`facet.modx` Create separate panels for each level of the moderator? Default is FALSE, except when `linearity.check` is TRUE.

`...` Ignored.

`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.sim_slopes *Plot coefficients from simple slopes analysis*

Description

This creates a coefficient plot to visually summarize the results of simple slopes analysis.

Usage

```
## S3 method for class 'sim_slopes'
plot(x, ...)
```

Arguments

x	A jtools::sim_slopes() object.
...	arguments passed to jtools::plot_coefs()

plot_predictions *Plot predicted effects from make_predictions*

Description

The companion function to make_predictions(). This takes data from make_predictions() (or elsewhere) and plots them like effect_plot(), interact_plot(), and cat_plot(). Note that some arguments will be ignored if the inputted predictions

Usage

```
plot_predictions(predictions, pred = NULL, modx = NULL, mod2 = NULL,
  resp = NULL, data = NULL, geom = c("point", "line", "bar",
  "boxplot"), plot.points = FALSE, interval = FALSE,
  pred.values = NULL, modx.values = NULL, mod2.values = NULL,
  linearity.check = FALSE, facet.modx = FALSE, x.label = NULL,
  y.label = NULL, pred.labels = NULL, modx.labels = NULL,
  mod2.labels = NULL, main.title = NULL, legend.main = NULL,
  color.class = NULL, line.thickness = 1.1, vary.lty = NULL,
  jitter = 0, weights = NULL, rug = FALSE, rug.sides = "b",
  force.cat = FALSE, point.shape = FALSE, geom.alpha = NULL,
  dodge.width = NULL, errorbar.width = NULL,
  interval.geom = c("errorbar", "linerange"), pred.point.size = 3.5,
  point.size = 1, ...)
```

Arguments

predictions	Either the output from make_predictions() (an object of class "predictions") or a data frame of predicted values.
pred	The name of the predictor variable involved in the interaction. This can be a bare name or string.
modx	The name of the moderator variable involved in the interaction. This can be a bare name or string.
mod2	Optional. The name of the second moderator variable involved in the interaction. This can be a bare name or string.
resp	What is the name of the response variable? Use a string.
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.
geom	**For factor predictors only**: 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. • "boxplot": This geom plots a dot and whisker plot. These can be useful for understanding the distribution of the observed data without having to plot all the observed points (especially helpful with larger data sets). **However**, it is important to note the boxplots are not based on the model whatsoever.
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.
pred.values	Which values of the predictor should be included in the plot? By default, all levels are included.
modx.values	For which values of the moderator should lines be plotted? Default is NULL. If NULL, then the customary +/- 1 standard deviation from the mean as well as the

mean itself are used for continuous moderators. If "plus-minus", plots lines when the moderator is at +/- 1 standard deviation without the mean. You may also choose "terciles" to split the data into equally-sized groups and choose the point at the mean of each of those groups.

If the moderator is a factor variable and modx.values is NULL, each level of the factor is included. You may specify any subset of the factor levels (e.g., c("Level 1", "Level 3")) as long as there is more than 1. The levels will be plotted in the order you provide them, so this can be used to reorder levels as well.

mod2.values For which values of the second moderator should the plot be facetted by? That is, there will be a separate plot for each level of this moderator. Defaults are the same as modx.values.

linearity.check

For two-way interactions only. If TRUE, plots a pane for each level of the moderator and superimposes a loess smoothed line (in gray) over the plot. This enables you to see if the effect is linear through the span of the moderator. See Hainmueller et al. (2016) in the references for more details on the intuition behind this. It is recommended that you also set plot.points = TRUE and use modx.values = "terciles" with this option.

facet.modx Create separate panels for each level of the moderator? Default is FALSE, except when linearity.check is TRUE.

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 2 labels for the predictor if it is a 2-level factor or a continuous variable with only 2 values. If NULL, the default, the factor labels are used.

modx.labels A character vector of labels for each level of the moderator values, provided in the same order as the modx.values argument. If NULL, the values themselves are used as labels unless modx,values is also NULL. In that case, "+1 SD" and "-1 SD" are used.

mod2.labels A character vector of labels for each level of the 2nd moderator values, provided in the same order as the mod2.values argument. If NULL, the values themselves are used as labels unless mod2.values is also NULL. In that case, "+1 SD" and "-1 SD" 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.

legend.main A character object that will be used as the title that appears above the legend. If NULL, the name of the moderating variable is used.

color.class See jtools_colors for details on the types of arguments accepted. Default is "CUD Bright" for factor moderators, "Blues" for +/- SD and user-specified modx.values values.

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

`vary.lty` Should the resulting plot have different shapes for each line in addition to colors? Default is NULL, which will switch to FALSE if the `pred` is a factor and TRUE if `pred` is continuous.

`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.

`weights` If the data are weighted, provide a vector of weights here. This is only used if `plot.points = TRUE` and `data` is not NULL.

`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 "b", meaning bottom. "t" and/or "b" show the distribution of the predictor while "l" and/or "r" show the distribution of the response. "bl" is a good option to show both the predictor and response.

`force.cat` Force the predictor to be treated as if it is a factor, even if it isn't? Default is FALSE. Set to TRUE if you'd like to generate a type of plot normally reserved for categorical variables. This can be helpful for numeric variables that have a small number of unique values, for instance.

`point.shape` For plotted points—either of observed data or predicted values with the "point" or "line" geoms—should the shape of the points vary by the values of the factor? This is especially useful if you aim to be black and white printing- or colorblind-friendly.

`geom.alpha` What should the alpha aesthetic be for the plotted lines/bars? Default is NULL, which means it is set depending on the value of `geom` and `plot.points`.

`dodge.width` What should the `width` argument to `ggplot2::position_dodge()` be? Default is NULL, which means it is set depending on the value of `geom`.

`errorbar.width` How wide should the error bars be? Default is NULL, meaning it is set depending on the value `geom`. Ignored if `interval` is FALSE.

`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.

`pred.point.size`
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.

`point.size` What size should be used for observed data when `plot.points` is TRUE? Default is 2.

`...` Ignored.

Details

This is designed to offer more flexibility than the canned functions (`effect_plot()`, `interact_plot()`, and `cat_plot()`), by letting you generate your own predicted data and iteratively experiment with the plotting options.

Note: `predictions` objects from `make_predictions()` store information about the arguments used to create the object. Unless you specify those arguments manually to this function, as a convenience `plot_predictions` will use the arguments stored in the `predictions` object. Those arguments are:

- `pred`, `modx`, and `mod2`
- `resp`
- `pred.values`, `modx.values`, and `mod2.values`
- `pred.labels`, `modx.labels`, and `mod2.labels`
- `data`
- `interval`
- `linearity.check`
- `weights`

See Also

Other plotting tools: `make_predictions`

`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,
  color.class = "CUD Bright", plot.distributions = FALSE,
  rescale.distributions = FALSE, exp = FALSE, point.shape = TRUE,
  legend.title = "Model", groups = NULL, facet.rows = NULL,
  facet.cols = NULL, facet.label.pos = "top")

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

Arguments

`...`	regression model(s).
`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.
`color.class`	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.
`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 specifiy 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".

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)
```

---

probe_interaction *Probe interaction effects via simple slopes and plotting*

Description

probe_interaction is a convenience function that allows users to call both sim_slopes and interact_plot with a single call.

Usage

```
probe_interaction(model, pred, modx, mod2 = NULL, ...)
```

Arguments

model	A regression model of type lm or svyglm. It should contain the interaction of interest.
pred	The predictor variable involved in the interaction.
modx	The moderator variable involved in the interaction.
mod2	Optional. The name of the second moderator variable involved in the interaction.
...	Other arguments accepted by sim_slopes and interact_plot

Details

This function simply merges the nearly-equivalent arguments needed to call both sim_slopes and interact_plot without the need for re-typing their common arguments. Note that each function is called separately and they re-fit a separate model for each level of each moderator; therefore, the runtime may be considerably longer than the original model fit. For larger models, this is worth keeping in mind.

Sometimes, you may want different parameters when doing simple slopes analysis compared to when plotting interaction effects. For instance, it is often easier to interpret the regression output when variables are standardized; but plots are often easier to understand when the variables are in their original units of measure.

probe_interaction does not support providing different arguments to each function. If that is needed, use sim_slopes and interact_plot directly.

Value

simslopes	The sim_slopes object created.
interactplot	The ggplot object created by interact_plot.

Author(s)

Jacob Long <<long.1377@osu.edu>>

See Also

Other interaction tools: cat_plot, interact_plot, johnson_neyman, sim_slopes

Examples

```
# Using a fitted model as formula input
fiti <- lm(Income ~ Frost + Murder * Illiteracy,
  data=as.data.frame(state.x77))
probe_interaction(model = fiti, pred = Murder, modx = Illiteracy,
                  modx.values = "plus-minus")
# 3-way interaction
```

```
fiti3 <- lm(Income ~ Frost * Murder * Illiteracy,
  data=as.data.frame(state.x77))
probe_interaction(model = fiti3, pred = Murder, modx = Illiteracy,
                  mod2 = Frost, mod2.values = "plus-minus")

# 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 + sch.wide, design = dstrat)
probe_interaction(model = regmodel, pred = ell, modx = meals,
                  modx.values = "plus-minus", cond.int = TRUE)

# 3-way with survey and factor input
regmodel3 <- svyglm(api00 ~ ell * meals * sch.wide, design = dstrat)
probe_interaction(model = regmodel3, pred = ell, modx = meals,
                  mod2 = sch.wide)
# Can try different configurations of 1st vs 2nd mod
probe_interaction(model = regmodel3, pred = ell, modx = sch.wide,
                  mod2 = meals)
}
```

---

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,
  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.
...	Ignored.

`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.

`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 `wec::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 <<long.1377@osu.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, scaling, and centering tools: `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` *function*

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)
```

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 and significance stars? 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.

---

sim_slopes *Perform a simple slopes analysis.*

Description

`sim_slopes` conducts a simple slopes analysis for the purposes of understanding two- and three-way interaction effects in linear regression.

Usage

```
sim_slopes(model, pred, modx, mod2 = NULL, modx.values = NULL,
  mod2.values = NULL, centered = "all", data = NULL,
  cond.int = FALSE, johnson_neyman = TRUE, jnplot = FALSE,
  jnalpha = 0.05, robust = FALSE, digits = getOption("jtools-digits",
  default = 2), pvals = TRUE, confint = FALSE, ci.width = 0.95,
  cluster = NULL, modx.labels = NULL, mod2.labels = NULL, ...)
```

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 predictor variable involved in the interaction.
modx	The moderator variable involved in the interaction.
mod2	Optional. The name of the second moderator variable involved in the interaction.
modx.values	For which values of the moderator should simple slopes analysis be performed? Default is NULL. If NULL, then the values will be the customary +/- 1 standard deviation from the mean as well as the mean itself. There is no specific limit on the number of variables provided. If "plus-minus", uses just +/- 1 standard deviation without the mean. You may also choose "terciles" to split the data into equally-sized groups and choose the point at the mean of each of those groups. Factor variables are not particularly suited to simple slopes analysis, but you could have a numeric moderator with values of 0 and 1 and give c(0,1) to compare the slopes at the different conditions. Two-level factor variables are coerced to numeric 0/1 variables, but are not standardized/centered like they could be if your input data had a numeric 0/1 variable.
mod2.values	Same as modx.values, but for the second moderator (mod2).
centered	A vector of quoted variable names that are to be mean-centered. If "all", all non-focal predictors as well as the pred variable 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, modx, and mod2 variables are never centered.
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.
cond.int	Should conditional intercepts be printed in addition to the slopes? Default is FALSE.
johnson_neyman	Should the Johnson-Neyman interval be calculated? Default is TRUE. This can be performed separately with johnson_neyman.

jnplot	Should the Johnson-Neyman interval be plotted as well? Default is FALSE.
jnalpha	What should the alpha level be for the Johnson-Neyman interval? Default is .05, which corresponds to a 95% confidence interval.
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.
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 and significance stars? If FALSE, these are not printed. Default is TRUE.
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.
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.
modx.labels	A character vector of labels for each level of the moderator values, provided in the same order as the modx.values argument. If NULL, the values themselves are used as labels unless modx,values is also NULL. In that case, "+1 SD" and "-1 SD" are used.
mod2.labels	A character vector of labels for each level of the 2nd moderator values, provided in the same order as the mod2.values argument. If NULL, the values themselves are used as labels unless mod2.values is also NULL. In that case, "+1 SD" and "-1 SD" are used.
...	Arguments passed to johnson_neyman and summ.

Details

This allows the user to perform a simple slopes analysis for the purpose of probing interaction effects in a linear regression. Two- and three-way interactions are supported, though one should be warned that three-way interactions are not easy to interpret in this way.

For more about Johnson-Neyman intervals, see johnson_neyman.

The function is tested with lm, glm, svyglm, and merMod inputs. Others may work as well, but are not tested. In all but the linear model case, be aware that not all the assumptions applied to simple slopes analysis apply.

Value

A list object with the following components:

slopes	A table of coefficients for the focal predictor at each value of the moderator
ints	A table of coefficients for the intercept at each value of the moderator

modx.values	The values of the moderator used in the analysis
mods	A list containing each regression model created to estimate the conditional coefficients.
jn	If johnson_neyman = TRUE, a list of johnson_neyman objects from johnson_neyman. These contain the values of the interval and the plots. If a 2-way interaction, the list will be of length 1. Otherwise, there will be 1 johnson_neyman object for each value of the 2nd moderator for 3-way interactions.

Author(s)

Jacob Long <<long.1377@osu.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. http://dx.doi.org/10.1207/s15327906mbr4003_5

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

interact_plot accepts similar syntax and will plot the results with ggplot.

testSlopes performs a hypothesis test of differences and provides Johnson-Neyman intervals.

simpleSlope performs a similar analysis.

Other interaction tools: cat_plot, interact_plot, johnson_neyman, probe_interaction

Examples

```
# Using a fitted model as formula input
fiti <- lm(Income ~ Frost + Murder * Illiteracy,
  data = as.data.frame(state.x77))
sim_slopes(model = fiti, pred = Murder, modx = Illiteracy)

# 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)
sim_slopes(regmodel, pred = ell, modx = meals)

# 3-way with survey and factor input
regmodel <- svyglm(api00 ~ ell * meals * sch.wide, design = dstrat)
sim_slopes(regmodel, pred = ell, modx = meals, mod2 = sch.wide)
}
```

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 = TRUE, 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 TRUE.
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, scaling, and centering tools: 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, ...)
```

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, data = NULL,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE), which.cols = NULL,
  ...)
```

Arguments

model	A glm object.
scale	If TRUE, reports standardized regression coefficients. 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.
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 and significance stars? 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.
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.
...	This just captures extra arguments that may only work for other types of models.

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 `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 <<long.1377@osu.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. https://doi.org/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. https://doi.org/10.1146/annurev.publhealth.23.100901.140546

See Also

`scale_lm` can simply perform the standardization if preferred.

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

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, data = NULL, part.corr = FALSE,
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE), model.check = FALSE,
  which.cols = NULL, ...)
```

Arguments

model	A lm object.
scale	If TRUE, reports standardized regression coefficients. 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.

`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 and significance stars? 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.

`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.

`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.

`model.check` Toggles whether to perform Breusch-Pagan test for heteroskedasticity and print number of high-leverage observations. See details for more info.

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

`...` This just captures extra arguments that may only work for other types of models.

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 `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.

There are two pieces of information given for `model.check`, provided that the model is an `lm` object. First, a Breusch-Pagan test is performed with `ncvTest`. This is a hypothesis test for which the alternative hypothesis is heteroskedastic errors. The test becomes much more likely to be statistically significant as the sample size increases; however, the homoskedasticity assumption becomes less important to inference as sample size increases (Lumley, Diehr, Emerson, & Lu, 2002). Take the result of the test as a cue to check graphical checks rather than a definitive decision. Note that the use of robust standard errors can account for heteroskedasticity, though some oppose this approach (see King & Roberts, 2015).

The second piece of information provided by setting `model.check` to `TRUE` is the number of high leverage observations. There are no hard and fast rules for determining high leverage either, but in this case it is based on Cook's Distance. All Cook's Distance values greater than (4/N) are included in the count. Again, this is not a recommendation to locate and remove such observations, but rather to look more closely with graphical and other methods.

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 <<long.1377@osu.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. https://doi.org/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. https://doi.org/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.

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, 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. 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 and significance stars? 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.
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.
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.
...	This just captures extra arguments that may only work for other types of models.

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_lm` and `center_lm`, 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 `sem.model.fits` from the **piecewiseSEM** package. This is an implementation of the Nakagawa & Schielzeth (2013) pro-

cedure 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 <<long.1377@osu.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. https://doi.org/10.1111/2041-210X.12225

Kenward, M. G., & Roger, J. H. (1997). Small sample inference for fixed effects from restricted maximum likelihood. *Biometrics*, *53*, 983. https://doi.org/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*. https://doi.org/10.18637/jss.v082.i13

Luke, S. G. (2017). Evaluating significance in linear mixed-effects models in R. *Behavior Research Methods*, *49*, 1494–1502. https://doi.org/10.3758/s13428-016-0809-y

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. https://doi.org/10.1111/j.2041-210x.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::sem.model.fits()` is used to generate the pseudo-R-squared estimates for linear models.

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. 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 and significance stars? 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.
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.
...	This just captures extra arguments that may only work for other types of models.

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

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, exp = FALSE,
  vifs = getOption("summ-vifs", FALSE),
  model.info = getOption("summ-model.info", TRUE),
  model.fit = getOption("summ-model.fit", TRUE), model.check = FALSE,
  which.cols = NULL, ...)
```

Arguments

model	A svyglm object.
scale	If TRUE, reports standardized regression coefficients. 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 and significance stars? 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.
transform.response	Should scaling/centering apply to response variable? 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.
model.check	Toggles whether to perform Breusch-Pagan test for heteroskedasticity and print number of high-leverage observations. See details for more info.
which.cols	Developmental feature. By providing columns by name, you can add/remove/reorder requested columns in the output. Not fully supported, for now.
...	This just captures extra arguments that may only work for other types of models.

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_lm and center_lm, 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 <<long.1377@osu.edu>>

See Also

scale_lm can simply perform the standardization if preferred.

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

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 <<long.1377@osu.edu>>

See Also

wtd.cor, svyvar

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 <<long.1377@osu.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()
```

`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

```
tidy.summ(x, conf.int = FALSE, conf.level = 0.95, ...)

glance.summ.lm(x, ...)

glance.summ.glm(x, ...)

glance.summ.svyglm(x, ...)

glance.summ.merMod(x, ...)

tidy.sim_slopes(x, conf.level = 0.95, ...)

glance.sim_slopes(x, ...)

glance.summ.rq(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)
```

Index