Package ‘broom.helpers’

April 24, 2025

Title Helpers for Model Coefficients Tibbles

Version 1.21.0

Description Provides suite of functions to work with regression model 'broom::tidy()' tibbles. The suite includes functions to group regression model terms by variable, insert reference and header rows for categorical variables, add variable labels, and more.

License GPL (>= 3)

URL https://larmarange.github.io/broom.helpers/, https://github.com/larmarange/broom.helpers

BugReports https://github.com/larmarange/broom.helpers/issues

Depends R (>= 4.1)

Imports broom (>= 0.8), cards, cli, dplyr (>= 1.1.0), labelled, lifecycle, purrr, rlang (>= 1.0.1), stats, stringr, tibble, tidyr, tidyselect

Suggests betareg, biglm, brms (>= 2.13.0), broom.mixed, cmprsk, covr, datasets, effects, emmeans, fixest (>= 0.10.0), forcats, gam, gee, geepack, ggplot2, ggeffects (>= 1.3.2), ggstats (>= 0.2.1), glmmTMB, glmtoolbox, glue, gt, gtsummary (>= 2.0.0), knitr, lavaan, lfe, lme4 (>= 1.1.28), logitr (>= 0.8.0), marginaleffects (>= 0.21.0), margins, MASS, mgcv, mice, mmrm (>= 0.3.6), multgee, nnet, ordinal, parameters, parsnip, patchwork, plm, pscl, rmarkdown, rstanarm, scales, spelling, survey, survival, testthat (>= 3.0.0), tidycmprsk, VGAM, svyVGAM

VignetteBuilder knitr

RdMacros lifecycle

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

Config/testthat/edition 3

NeedsCompilation no

Author Joseph Larmarange [aut, cre] (<https://orcid.org/0000-0001-7097-700X>),
Daniel D. Sjoberg [aut] (<https://orcid.org/0000-0003-0862-2018>)

Maintainer Joseph Larmarange <joseph@larmarange.net>

Repository CRAN

Date/Publication 2025-04-24 08:50:02 UTC

Contents

`.clean_backticks` *Remove backticks around variable names*

Description

Remove backticks around variable names

Usage

```
.clean_backticks(x, variable_names = x)
```

Arguments

`x`	`(string)` A character vector to be cleaned.
`variable_names`	`(string)` Optional vector of variable names, could be obtained with model_list_variables(only_variable = TRUE), to properly take into account interaction only terms/variables.

See Also

Other other_helpers: `.escape_regex()`

`.escape_regex` *Escapes any characters that would have special meaning in a regular expression*

Description

This functions has been adapted from `Hmisc::escapeRegex()`

Usage

```
.escape_regex(string)
```

Arguments

`string`	(`string`) A character vector.

See Also

Other other_helpers: `.clean_backticks()`

`assert_package` *Check a package installation status or minimum required version*

Description

The function `.assert_package()` checks whether a package is installed and returns an error or `FALSE` if not available. If a package search is provided, the function will check whether a minimum version of a package is required. The function `.get_package_dependencies()` returns a tibble with all dependencies of a specific package. Finally, `.get_min_version_required()` will return, if any, the minimum version of pkg required by `pkg_search`, `NULL` if no minimum version required.

Usage

```
.assert_package(pkg, fn = NULL, pkg_search = "broom.helpers", boolean = FALSE)

.get_package_dependencies(pkg_search = "broom.helpers")

.get_all_packages_dependencies(
  pkg_search = NULL,
  remove_duplicates = FALSE,
  lib.loc = NULL
)

.get_min_version_required(pkg, pkg_search = "broom.helpers")
```

Arguments

`pkg`	`(string)` Name of the required package.
`fn`	`(string)` Name of the calling function from the user perspective. Used to write informative error messages.
`pkg_search`	`(string)` Name of the package the function will search for a minimum required version from.
`boolean`	`(logical)` Whether to return a `TRUE`/`FALSE`, rather than error when package/package version not available. Default is `FALSE`, which will return an error if `pkg` is not installed.
`remove_duplicates`	`(logical)` If several versions of a package are installed, should only the first one be returned?
`lib.loc`	`(string)` Location of `R` library trees to search through, see `utils::installed.packages()`.

Details

`get_all_packages_dependencies()` could be used to get the list of dependencies of all installed packages.

Value

logical or error for `.assert_package()`, `NULL` or character with the minimum version required for `.get_min_version_required()`, a tibble for `.get_package_dependencies()`.

Examples

```
.assert_package("broom", boolean = TRUE)
.get_package_dependencies()
.get_min_version_required("brms")
```

`model_compute_terms_contributions` *Compute a matrix of terms contributions*

Description

Used for `model_get_n()`. For each row and term, equal 1 if this row should be taken into account in the estimate of the number of observations, 0 otherwise.

Usage

```
model_compute_terms_contributions(model)

## Default S3 method:
model_compute_terms_contributions(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Details

This function does not cover `lavaan` models (`NULL` is returned).

See Also

Other model_helpers: `model_get_assign()`, `model_get_coefficients_type()`, `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
mod <- lm(Sepal.Length ~ Sepal.Width, iris)
mod |> model_compute_terms_contributions()

mod <- lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars)
mod |> model_compute_terms_contributions()

mod <- glm(
  response ~ stage * grade + trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(
    stage = contr.sum,
    grade = contr.treatment(3, 2),
    trt = "contr.SAS"
  )
)
mod |> model_compute_terms_contributions()

mod <- glm(
  response ~ stage * trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(stage = contr.poly)
)
mod |> model_compute_terms_contributions()
mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq, family = binomial
)
mod |> model_compute_terms_contributions()

d <- dplyr::as_tibble(Titanic) |>
  dplyr::group_by(Class, Sex, Age) |>
  dplyr::summarise(
    n_survived = sum(n * (Survived == "Yes")),
    n_dead = sum(n * (Survived == "No"))
  )
mod <- glm(cbind(n_survived, n_dead) ~ Class * Age + Sex, data = d, family = binomial)
mod |> model_compute_terms_contributions()
```

model_get_assign *Get the assign attribute of model matrix of a model*

Description

Return the assign attribute attached to the object returned by `stats::model.matrix()`.

Usage

```
model_get_assign(model)

## Default S3 method:
model_get_assign(model)

## S3 method for class 'vglm'
model_get_assign(model)

## S3 method for class 'svy_vglm'
model_get_assign(model)

## S3 method for class 'model_fit'
model_get_assign(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

`stats::model.matrix()`

Other model_helpers: `model_compute_terms_contributions()`, `model_get_coefficients_type()`, `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variat`
`model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_assign()
```

`model_get_coefficients_type`

Get coefficient type

Description

Indicate the type of coefficient among "generic", "logistic", "poisson", "relative_risk" or "prop_hazard".

Usage

```
model_get_coefficients_type(model)

## Default S3 method:
model_get_coefficients_type(model)

## S3 method for class 'glm'
model_get_coefficients_type(model)

## S3 method for class 'negbin'
model_get_coefficients_type(model)

## S3 method for class 'geeglm'
model_get_coefficients_type(model)

## S3 method for class 'fixest'
model_get_coefficients_type(model)

## S3 method for class 'biglm'
model_get_coefficients_type(model)

## S3 method for class 'glmerMod'
model_get_coefficients_type(model)
```

```
## S3 method for class 'clogit'
model_get_coefficients_type(model)

## S3 method for class 'polr'
model_get_coefficients_type(model)

## S3 method for class 'multinom'
model_get_coefficients_type(model)

## S3 method for class 'svyolr'
model_get_coefficients_type(model)

## S3 method for class 'clm'
model_get_coefficients_type(model)

## S3 method for class 'clmm'
model_get_coefficients_type(model)

## S3 method for class 'coxph'
model_get_coefficients_type(model)

## S3 method for class 'crr'
model_get_coefficients_type(model)

## S3 method for class 'tidycrr'
model_get_coefficients_type(model)

## S3 method for class 'cch'
model_get_coefficients_type(model)

## S3 method for class 'model_fit'
model_get_coefficients_type(model)

## S3 method for class 'LORgee'
model_get_coefficients_type(model)

## S3 method for class 'vglm'
model_get_coefficients_type(model)

## S3 method for class 'vgam'
model_get_coefficients_type(model)

## S3 method for class 'svy_vglm'
model_get_coefficients_type(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_coefficients_type()

df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
glm(Survived ~ Class + Age * Sex, data = df, weights = df$n, family = binomial) |>
  model_get_coefficients_type()
```

---

model_get_contrasts *Get contrasts used in the model*

Description

Get contrasts used in the model

Usage

```
model_get_contrasts(model)

## S3 method for class 'model_fit'
model_get_contrasts(model)

## S3 method for class 'zeroinfl'
model_get_contrasts(model)

## S3 method for class 'hurdle'
model_get_contrasts(model)

## S3 method for class 'betareg'
model_get_contrasts(model)

## S3 method for class 'svy_vglm'
model_get_contrasts(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`,
`model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`,
`model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`,
`model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`,
`model_list_terms_levels()`, `model_list_variables()`

Examples

```
glm(
  am ~ mpg + factor(cyl),
  data = mtcars,
  family = binomial,
  contrasts = list(`factor(cyl)` = contr.sum)
) |>
  model_get_contrasts()
```

`model_get_model` *Get the model from model objects*

Description

Most model objects are proper R model objects. There are, however, some model objects that store the proper object internally (e.g. mice models). This function extracts that model object in those cases.

Usage

```
model_get_model(model)

## Default S3 method:
model_get_model(model)

## S3 method for class 'mira'
model_get_model(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_model()
```

`model_get_model_frame` *Get the model frame of a model*

Description

The structure of the object returned by `stats::model.frame()` could slightly differ for certain types of models. `model_get_model_frame()` will always return an object with the same data structure or `NULL` if it is not possible to compute model frame from `model`.

Usage

```
model_get_model_frame(model)

## Default S3 method:
model_get_model_frame(model)

## S3 method for class 'coxph'
model_get_model_frame(model)

## S3 method for class 'svycoxph'
model_get_model_frame(model)

## S3 method for class 'survreg'
model_get_model_frame(model)

## S3 method for class 'biglm'
model_get_model_frame(model)

## S3 method for class 'model_fit'
model_get_model_frame(model)

## S3 method for class 'fixest'
model_get_model_frame(model)
```

```
## S3 method for class 'svy_vglm'
model_get_model_frame(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

`stats::model.frame()`

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_model_frame() |>
  head()
```

`model_get_model_matrix` *Get the model matrix of a model*

Description

The structure of the object returned by `stats::model.matrix()` could slightly differ for certain types of models. `model_get_model_matrix()` will always return an object with the same structure as `stats::model.matrix.default()`.

Usage

```
model_get_model_matrix(model, ...)

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

## S3 method for class 'multinom'
model_get_model_matrix(model, ...)

## S3 method for class 'clm'
model_get_model_matrix(model, ...)
```

```
## S3 method for class 'brmsfit'
model_get_model_matrix(model, ...)

## S3 method for class 'glmmTMB'
model_get_model_matrix(model, ...)

## S3 method for class 'plm'
model_get_model_matrix(model, ...)

## S3 method for class 'biglm'
model_get_model_matrix(model, ...)

## S3 method for class 'model_fit'
model_get_model_matrix(model, ...)

## S3 method for class 'fixest'
model_get_model_matrix(model, ...)

## S3 method for class 'LORgee'
model_get_model_matrix(model, ...)

## S3 method for class 'betareg'
model_get_model_matrix(model, ...)

## S3 method for class 'cch'
model_get_model_matrix(model, ...)

## S3 method for class 'vglm'
model_get_model_matrix(model, ...)

## S3 method for class 'vgam'
model_get_model_matrix(model, ...)

## S3 method for class 'svy_vglm'
model_get_model_matrix(model, ...)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.
`...`	Additional arguments passed to `stats::model.matrix()`.

Details

For models fitted with `glmmTMB::glmmTMB()`, it will return a model matrix taking into account all components ("cond", "zi" and "disp"). For a more restricted model matrix, please refer to `glmmTMB::model.matrix.glmmTMB()`.

For `plm::plm()` models, constant columns are not removed.

For `fixest` models, will recreate a model matrix with both main variables and instrumental variables. For more options, see fixest::model.matrix.fixest.

See Also

`stats::model.matrix()`

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type` `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable(` `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_model_matrix() |>
  head()
```

`model_get_n` *Get the number of observations*

Description

For binomial and multinomial logistic models, will also return the number of events.

Usage

```
model_get_n(model)

## Default S3 method:
model_get_n(model)

## S3 method for class 'glm'
model_get_n(model)

## S3 method for class 'glmerMod'
model_get_n(model)

## S3 method for class 'multinom'
model_get_n(model)

## S3 method for class 'LORgee'
model_get_n(model)

## S3 method for class 'coxph'
model_get_n(model)
```

```
## S3 method for class 'survreg'
model_get_n(model)

## S3 method for class 'model_fit'
model_get_n(model)

## S3 method for class 'tidycrr'
model_get_n(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Details

For Poisson models, will return the number of events and exposure time (defined with `stats::offset()`).

For Cox models (`survival::coxph()`), will return the number of events, exposure time and the number of individuals.

For competing risk regression models (`tidycmprsk::crr()`), `n_event` takes into account only the event of interest defined by `failcode`.

See `tidy_add_n()` for more details.

The total number of observations (`N_obs`), of individuals (`N_ind`), of events (`N_event`) and of exposure time (`Exposure`) are stored as attributes of the returned tibble.

This function does not cover `lavaan` models (`NULL` is returned).

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars) |>
  model_get_n()

mod <- glm(
  response ~ stage * grade + trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(stage = contr.sum, grade = contr.treatment(3, 2), trt = "contr.SAS")
)
mod |> model_get_n()
```

```
## Not run:
mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq, family = binomial
)
mod |> model_get_n()

d <- dplyr::as_tibble(Titanic) |>
  dplyr::group_by(Class, Sex, Age) |>
  dplyr::summarise(
    n_survived = sum(n * (Survived == "Yes")),
    n_dead = sum(n * (Survived == "No"))
  )
mod <- glm(cbind(n_survived, n_dead) ~ Class * Age + Sex, data = d, family = binomial)
mod |> model_get_n()

mod <- glm(response ~ age + grade * trt, gtsummary::trial, family = poisson)
mod |> model_get_n()

mod <- glm(
  response ~ trt * grade + offset(ttdeath),
  gtsummary::trial,
  family = poisson
)
mod |> model_get_n()

dont
df <- survival::lung |> dplyr::mutate(sex = factor(sex))
mod <- survival::coxph(survival::Surv(time, status) ~ ph.ecog + age + sex, data = df)
mod |> model_get_n()

mod <- lme4::lmer(Reaction ~ Days + (Days | Subject), lme4::sleepstudy)
mod |> model_get_n()

mod <- lme4::glmer(response ~ trt * grade + (1 | stage),
  family = binomial, data = gtsummary::trial
)
mod |> model_get_n()

mod <- lme4::glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
  family = binomial, data = lme4::cbpp
)
mod |> model_get_n()

## End(Not run)
```

`model_get_nlevels` *Get the number of levels for each factor used in* `xlevels`

Description

Get the number of levels for each factor used in `xlevels`

Usage

```
model_get_nlevels(model)

## Default S3 method:
model_get_nlevels(model)

## S3 method for class 'svy_vglm'
model_get_nlevels(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Value

a tibble with two columns: `"variable"` and `"var_nlevels"`

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_nlevels()
```

`model_get_offset` *Get model offset*

Description

This function does not cover `lavaan` models (`NULL` is returned).

Usage

```
model_get_offset(model)

## Default S3 method:
model_get_offset(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`,
`model_get_n()`, `model_get_nlevels()`, `model_get_pairwise_contrasts()`, `model_get_response()`,
`model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`,
`model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`,
`model_list_terms_levels()`, `model_list_variables()`

Examples

```
mod <- glm(
  response ~ trt + offset(log(ttdeath)),
  gtsummary::trial,
  family = poisson
)
mod |> model_get_offset()
```

`model_get_pairwise_contrasts`
Get pairwise comparison of the levels of a categorical variable

---

Description

It is computed with `emmeans::emmeans()`.

Usage

```
model_get_pairwise_contrasts(
  model,
  variables,
  pairwise_reverse = TRUE,
  contrasts_adjust = NULL,
  conf.level = 0.95,
  emmeans_args = list()
)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.
`variables`	(`tidy-select`) Variables to add pairwise contrasts.

`pairwise_reverse`
: (logical)
Determines whether to use "`pairwise`" (if `TRUE`) or "`revpairwise`" (if `FALSE`), see `emmeans::contrast()`.

`contrasts_adjust`
: optional adjustment method when computing contrasts, see `emmeans::contrast()` (if `NULL`, use `emmeans` default)

`conf.level`
: (`numeric`)
Level of confidence for confidence intervals (default: 95%).

`emmeans_args`
: (`logical`)
List of additional parameter to pass to `emmeans::emmeans()` when computing pairwise contrasts.

Details

For `pscl::zeroinfl()` and `pscl::hurdle()` models, pairwise contrasts are computed separately for each component, using `mode = "count"` and `mode = "zero"` (see documentation of `emmeans`) and a component column is added to the results.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type` `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_response()`, `model_get_response_variabl` `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
mod <- lm(Sepal.Length ~ Species, data = iris)
mod |> model_get_pairwise_contrasts(variables = "Species")
mod |>
  model_get_pairwise_contrasts(
    variables = "Species",
    contrasts_adjust = "none"
  )
```

`model_get_response` *Get model response*

Description

This function does not cover `lavaan` models (`NULL` is returned).

Usage

```
model_get_response(model)

## Default S3 method:
model_get_response(model)

## S3 method for class 'glm'
model_get_response(model)

## S3 method for class 'glmerMod'
model_get_response(model)

## S3 method for class 'model_fit'
model_get_response(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars) |>
  model_get_response()

mod <- glm(
  response ~ stage * grade + trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(stage = contr.sum, grade = contr.treatment(3, 2), trt = "contr.SAS")
)
mod |> model_get_response()

mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq,
  family = binomial
)
mod |> model_get_response()
```

```
d <- dplyr::as_tibble(Titanic) |>
  dplyr::group_by(Class, Sex, Age) |>
  dplyr::summarise(
    n_survived = sum(n * (Survived == "Yes")),
    n_dead = sum(n * (Survived == "No"))
  )
mod <- glm(cbind(n_survived, n_dead) ~ Class * Age + Sex, data = d, family = binomial, y = FALSE)
mod |> model_get_response()
```

---

model_get_response_variable

Get the name of the response variable

Description

Get the name of the response variable

Usage

```
model_get_response_variable(model)

## Default S3 method:
model_get_response_variable(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars) |>
  model_get_response_variable()

mod <- glm(
  response ~ stage * grade + trt,
  gtsummary::trial,
  family = binomial
)
mod |> model_get_response_variable()
mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq,
  family = binomial
)
mod |> model_get_response_variable()
```

`model_get_terms` *Get the terms of a model*

Description

Return the result of `stats::terms()` applied to the model or `NULL` if it is not possible to get terms from `model`.

Usage

```
model_get_terms(model)

## Default S3 method:
model_get_terms(model)

## S3 method for class 'brmsfit'
model_get_terms(model)

## S3 method for class 'glmmTMB'
model_get_terms(model)

## S3 method for class 'model_fit'
model_get_terms(model)

## S3 method for class 'betareg'
model_get_terms(model)

## S3 method for class 'betareg'
model_get_terms(model)

## S3 method for class 'cch'
model_get_terms(model)

## S3 method for class 'fixest'
model_get_terms(model)

## S3 method for class 'svy_vglm'
model_get_terms(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Details

For models fitted with `glmmTMB::glmmTMB()`, it will return a terms object taking into account all components ("cond" and "zi"). For a more restricted terms object, please refer to `glmmTMB::terms.glmmTMB()`.

For `fixest` models, return a term object combining main variables and instrumental variables.

See Also

`stats::terms()`

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_terms()
```

`model_get_weights` *Get sampling weights used by a model*

Description

This function does not cover `lavaan` models (`NULL` is returned).

Usage

```
model_get_weights(model)

## Default S3 method:
model_get_weights(model)

## S3 method for class 'svyglm'
model_get_weights(model)

## S3 method for class 'svrepglm'
model_get_weights(model)

## S3 method for class 'model_fit'
model_get_weights(model)
```

```
## S3 method for class 'svy_vglm'
model_get_weights(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Note

For class `svrepglm` objects (GLM on a survey object with replicate weights), it will return the original sampling weights of the data, not the replicate weights.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
mod <- lm(Sepal.Length ~ Sepal.Width, iris)
mod |> model_get_weights()

mod <- lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars, weights = mtcars$gear)
mod |> model_get_weights()

mod <- glm(
  response ~ stage * grade + trt,
  gtsummary::trial,
  family = binomial
)
mod |> model_get_weights()

mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq,
  family = binomial
)
mod |> model_get_weights()

d <- dplyr::as_tibble(Titanic) |>
  dplyr::group_by(Class, Sex, Age) |>
  dplyr::summarise(
    n_survived = sum(n * (Survived == "Yes")),
    n_dead = sum(n * (Survived == "No"))
  )
```

```
mod <- glm(cbind(n_survived, n_dead) ~ Class * Age + Sex, data = d, family = binomial)
mod |> model_get_weights()
```

model_get_xlevels   *Get xlevels used in the model*

Description

Get xlevels used in the model

Usage

```
model_get_xlevels(model)

## Default S3 method:
model_get_xlevels(model)

## S3 method for class 'lmerMod'
model_get_xlevels(model)

## S3 method for class 'glmerMod'
model_get_xlevels(model)

## S3 method for class 'felm'
model_get_xlevels(model)

## S3 method for class 'brmsfit'
model_get_xlevels(model)

## S3 method for class 'glmmTMB'
model_get_xlevels(model)

## S3 method for class 'plm'
model_get_xlevels(model)

## S3 method for class 'model_fit'
model_get_xlevels(model)

## S3 method for class 'svy_vglm'
model_get_xlevels(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`,
`model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`,
`model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`,
`model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variables()`,
`model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl), mtcars) |>
  model_get_xlevels()
```

`model_identify_variables`

Identify for each coefficient of a model the corresponding variable

---

Description

It will also identify interaction terms and intercept(s).

Usage

```
model_identify_variables(model)

## Default S3 method:
model_identify_variables(model)

## S3 method for class 'lavaan'
model_identify_variables(model)

## S3 method for class 'aov'
model_identify_variables(model)

## S3 method for class 'clm'
model_identify_variables(model)

## S3 method for class 'clmm'
model_identify_variables(model)

## S3 method for class 'gam'
model_identify_variables(model)

## S3 method for class 'model_fit'
model_identify_variables(model)

## S3 method for class 'logitr'
```

```
model_identify_variables(model)

## S3 method for class 'svy_vglm'
model_identify_variables(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Value

A tibble with four columns:

- `term`: coefficients of the model
- `variable`: the corresponding variable
- `var_class`: class of the variable (cf. `stats::.MFclass()`)
- `var_type`: `"continuous"`, `"dichotomous"` (categorical variable with 2 levels), `"categorical"` (categorical variable with 3 or more levels), `"intercept"` or `"interaction"`
- `var_nlevels`: number of original levels for categorical variables

See Also

`tidy_identify_variables()`

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type` `model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_list_contrasts()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
glm(
  Survived ~ Class + Age * Sex,
  data = df, weights = df$n,
  family = binomial
) |>
  model_identify_variables()

lm(
  Sepal.Length ~ poly(Sepal.Width, 2) + Species,
  data = iris,
  contrasts = list(Species = contr.sum)
) |>
  model_identify_variables()
```

`model_list_contrasts` *List contrasts used by a model*

Description

List contrasts used by a model

Usage

```
model_list_contrasts(model)

## Default S3 method:
model_list_contrasts(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

Details

For models with no intercept, no contrasts will be applied to one of the categorical variable. In such case, one dummy term will be returned for each level of the categorical variable.

Value

A tibble with three columns:

- `variable`: variable name
- `contrasts`: contrasts used
- `contrasts_type`: type of contrasts ("treatment", "sum", "poly", "helmert", "sdiff, "other" or "no.contrast")
- `reference`: for variables with treatment, SAS or sum contrasts, position of the reference level

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_higher_order_variables()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
glm(
  am ~ mpg + factor(cyl),
  data = mtcars,
  family = binomial,
  contrasts = list(`factor(cyl)` = contr.sum)
) |>
  model_list_contrasts()
```

model_list_higher_order_variables

List higher order variables of a model

Description

List higher order variables of a model

Usage

```
model_list_higher_order_variables(model)

## Default S3 method:
model_list_higher_order_variables(model)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_terms_levels()`, `model_list_variables()`

Examples

```
lm(hp ~ mpg + factor(cyl) + disp:hp, mtcars) |>
  model_list_higher_order_variables()

mod <- glm(
  response ~ stage * grade + trt:stage,
  gtsummary::trial,
  family = binomial
)
```

```
mod |> model_list_higher_order_variables()

mod <- glm(
  Survived ~ Class * Age + Sex,
  data = Titanic |> as.data.frame(),
  weights = Freq,
  family = binomial
)
mod |> model_list_higher_order_variables()
```

`model_list_terms_levels` *List levels of categorical terms*

Description

Only for categorical variables with treatment, SAS, sum or successive differences contrasts (cf. `MASS::contr.sdif()`), and categorical variables with no contrast.

Usage

```
model_list_terms_levels(
  model,
  label_pattern = "{level}",
  variable_labels = NULL,
  sdif_term_level = c("diff", "ratio")
)

## Default S3 method:
model_list_terms_levels(
  model,
  label_pattern = "{level}",
  variable_labels = NULL,
  sdif_term_level = c("diff", "ratio")
)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.
`label_pattern`	(`glue pattern`) A glue pattern for term labels (see examples).
`variable_labels`	(`list` or `string`) An optional named list or named vector of custom variable labels passed to `model_list_variables()`

`sdif_term_level`
: (string)
For successive differences contrasts, how should term levels be named? `"diff"` for `"B - A"` (default), `"ratio"` for `"B / A"`.

Value

A tibble with ten columns:

- `variable`: variable
- `contrasts_type`: type of contrasts ("sum" or "treatment")
- `term`: term name
- `level`: term level
- `level_rank`: rank of the level
- `reference`: logical indicating which term is the reference level
- `reference_level`: level of the reference term
- `var_label`: variable label obtained with `model_list_variables()`
- `var_nlevels`: number of levels in this variable
- `dichotomous`: logical indicating if the variable is dichotomous
- `label`: term label (by default equal to term level) The first nine columns can be used in `label_pattern`.

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`, `model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`, `model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variab`
`model_list_variables()`

Examples

```
glm(
  am ~ mpg + factor(cyl),
  data = mtcars,
  family = binomial,
  contrasts = list(`factor(cyl)` = contr.sum)
) |>
  model_list_terms_levels()

df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))

mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, weights = df$n, family = binomial,
  contrasts = list(Age = contr.sum, Class = "contr.helmert")
)
mod |> model_list_terms_levels()
mod |> model_list_terms_levels("{level} vs {reference_level}")
mod |> model_list_terms_levels("{variable} [{level} - {reference_level}]")
mod |> model_list_terms_levels(
  "{ifelse(reference, level, paste(level, '-', reference_level))}"
)
```

---

model_list_variables *List all the variables used in a model*

Description

Including variables used only in an interaction.

Usage

```
model_list_variables(
  model,
  labels = NULL,
  only_variable = FALSE,
  add_var_type = FALSE,
  instrumental_suffix = " (instrumental)"
)

## Default S3 method:
model_list_variables(
  model,
  labels = NULL,
  only_variable = FALSE,
  add_var_type = FALSE,
  instrumental_suffix = " (instrumental)"
)

## S3 method for class 'lavaan'
model_list_variables(
  model,
  labels = NULL,
  only_variable = FALSE,
  add_var_type = FALSE,
  instrumental_suffix = " (instrumental)"
)

## S3 method for class 'logitr'
model_list_variables(
  model,
  labels = NULL,
  only_variable = FALSE,
  add_var_type = FALSE,
  instrumental_suffix = " (instrumental)"
)
```

Arguments

`model`	(a model object, e.g. `glm`) A model object.
`labels`	(`list` or `string`) An optional named list or named vector of custom variable labels.
`only_variable`	(`logical`) If `TRUE`, will return only "variable" column.
`add_var_type`	(`logical`) If `TRUE`, add `var_nlevels` and `var_type` columns.
`instrumental_suffix`	(`string`) Suffix added to variable labels for instrumental variables (`fixest` models). `NULL` to add nothing.

Value

A tibble with three columns:

- `variable`: the corresponding variable
- `var_class`: class of the variable (cf. `stats::.MFclass()`)
- `label_attr`: variable label defined in the original data frame with the label attribute (cf. `labelled::var_label()`)
- `var_label`: a variable label (by priority, `labels` if defined, `label_attr` if available, otherwise `variable`)

If `add_var_type = TRUE`:

- `var_type`: `"continuous"`, `"dichotomous"` (categorical variable with 2 levels), `"categorical"` (categorical variable with 3 or more levels), `"intercept"` or `"interaction"`
- `var_nlevels`: number of original levels for categorical variables

See Also

Other model_helpers: `model_compute_terms_contributions()`, `model_get_assign()`, `model_get_coefficients_type`
`model_get_contrasts()`, `model_get_model()`, `model_get_model_frame()`, `model_get_model_matrix()`,
`model_get_n()`, `model_get_nlevels()`, `model_get_offset()`, `model_get_pairwise_contrasts()`,
`model_get_response()`, `model_get_response_variable()`, `model_get_terms()`, `model_get_weights()`,
`model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `model_list_higher_order_variab`
`model_list_terms_levels()`

Examples

```
  df <- Titanic |>
    dplyr::as_tibble() |>
    dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
  glm(
    Survived ~ Class + Age:Sex,
    data = df, weights = df$n,
    family = binomial
  ) |>
  model_list_variables()

lm(
   Sepal.Length ~ poly(Sepal.Width, 2) + Species,
   data = iris,
   contrasts = list(Species = contr.sum)
  ) |>
  model_list_variables()

glm(
  response ~ poly(age, 3) + stage + grade * trt,
  na.omit(gtsummary::trial),
  family = binomial,
) |>
  model_list_variables()
```

`scope_tidy` *Scoping a tidy tibble allowing to tidy select*

Description

This function uses the information from a model tidy tibble to generate a data frame exposing the different variables of the model, data frame that could be used for tidy selection. In addition, columns `"var_type"`, `"var_class"` and `"contrasts_type"` are scoped and their values are added as attributes to the data frame. For example, if `var_type='continuous'` for variable `"age"`, then the attribute `attr(.$age, 'gtsummary.var_type') <- 'continuous'` is set. That attribute is then used in a selector like `all_continuous()`. Note: attributes are prefixed with `"gtsummary."` to be compatible with selectors provided by `{gtsummary}`.

Usage

```
scope_tidy(x, data = NULL)
```

Arguments

`x`	(`data.frame`) A tidy tibble, with a `"variable"` column, as returned by `tidy_identify_variables()`.

`data`	(`data.frame`) An optional data frame the attributes will be added to.

Value

A data frame.

Examples

```
mod <- lm(Sepal.Length ~ Sepal.Width * Species, data = iris)
tt <- mod |> tidy_and_attach() |> tidy_add_contrasts()

scope_tidy(tt) |> str()
scope_tidy(tt, data = model_get_model_frame(mod)) |> str()

scope_tidy(tt) |> dplyr::select(dplyr::starts_with("Se")) |> names()
scope_tidy(tt) |> dplyr::select(where(is.factor)) |> names()
scope_tidy(tt) |> dplyr::select(all_continuous()) |> names()
scope_tidy(tt) |> dplyr::select(all_contrasts()) |> names()
scope_tidy(tt) |> dplyr::select(all_interaction()) |> names()
scope_tidy(tt) |> dplyr::select(all_intercepts()) |> names()
```

`select_helpers` *Select helper functions*

Description

Set of functions to supplement the *tidyselect* set of functions for selecting columns of data frames (and other items as well).

- `all_continuous()` selects continuous variables
- `all_categorical()` selects categorical (including `"dichotomous"`) variables
- `all_dichotomous()` selects only type `"dichotomous"`
- `all_interaction()` selects interaction terms from a regression model
- `all_intercepts()` selects intercept terms from a regression model
- `all_contrasts()` selects variables in regression model based on their type of contrast
- `all_ran_pars()` and `all_ran_vals()` for random-effect parameters and values from a mixed model (see `vignette("broom_mixed_intro", package = "broom.mixed")`)

Usage

```
all_continuous(continuous2 = TRUE)

all_categorical(dichotomous = TRUE)

all_dichotomous()
```

```
all_interaction()

all_ran_pars()

all_ran_vals()

all_intercepts()

all_contrasts(
  contrasts_type = c("treatment", "sum", "poly", "helmert", "sdif", "other")
)
```

Arguments

`continuous2`	(`logical`) Whether to include continuous2 variables, default is `TRUE`. For compatibility with `{gtsummary}`), see `gtsummary::all_continuous2()`.
`dichotomous`	(`logical`) Whether to include dichotomous variables, default is `TRUE`.
`contrasts_type`	(`string`) Type of contrast to select. When `NULL`, all variables with a contrast will be selected. Default is `NULL`. Select among contrast types `c("treatment", "sum", "poly", "helmert", "sdif", "other")`.

Value

A character vector of column names selected.

See Also

`scope_tidy()`

Examples

```
glm(response ~ age * trt + grade, gtsummary::trial, family = binomial) |>
  tidy_plus_plus(exponentiate = TRUE, include = all_categorical())



  glm(response ~ age + trt + grade + stage,
    gtsummary::trial,
    family = binomial,
    contrasts = list(trt = contr.SAS, grade = contr.sum, stage = contr.poly)
  ) |>
    tidy_plus_plus(
      exponentiate = TRUE,
      include = all_contrasts(c("treatment", "sum"))
    )
```

seq_range *Sequence generation between min and max*

Description

Sequence generation between min and max

Usage

```
seq_range(x, length.out = 25)
```

Arguments

`x`	`(numeric)` A numeric vector.
`length.out`	`(integer)` Desired length of the sequence (a positive integer).

Details

`seq_range(x, length.out)` is a shortcut for `seq(min(x, na.rm = TRUE), max(x, na.rm = TRUE), length.out = length.out)`

Value

a numeric vector

Examples

```
seq_range(iris$Petal.Length)
```

supported_models *Listing of Supported Models*

Description

Listing of Supported Models

Usage

```
supported_models
```

Format

A data frame with one row per supported model

model Model

notes Notes

Supported models

model	notes
`betareg::betareg()`	Use `tidy_parameters()` as `tidy_fun` with `component` argument to control with coefficients to re
`biglm::bigglm()`	
`brms::brm()`	`broom.mixed` package required
`cmprsk::crr()`	Limited support. It is recommended to use `tidycmprsk::crr()` instead.
`fixest::feglm()`	May fail with R <= 4.0.
`fixest::femlm()`	May fail with R <= 4.0.
`fixest::feNmlm()`	May fail with R <= 4.0.
`fixest::feols()`	May fail with R <= 4.0.
`gam::gam()`	
`geepack::geeglm()`	
`glmmTMB::glmmTMB()`	`broom.mixed` package required
`glmtoolbox::glmgee()`	
`lavaan::lavaan()`	Limited support for categorical variables
`lfe::felm()`	
`lme4::glmer.nb()`	`broom.mixed` package required
`lme4::glmer()`	`broom.mixed` package required
`lme4::lmer()`	`broom.mixed` package required
`logitr::logitr()`	Requires logitr >= 0.8.0
`MASS::glm.nb()`	
`MASS::polr()`	
`mgcv::gam()`	Use default tidier `broom::tidy()` for smooth terms only, or `gtsummary::tidy_gam()` to include
`mice::mira`	Limited support. If `mod` is a `mira` object, use `tidy_fun = function(x, ...) {mice::pool(x)`
`mmrm::mmrm()`	
`multgee::nomLORgee()`	Use `tidy_multgee()` as `tidy_fun`.
`multgee::ordLORgee()`	Use `tidy_multgee()` as `tidy_fun`.
`nnet::multinom()`	
`ordinal::clm()`	Limited support for models with nominal predictors.
`ordinal::clmm()`	Limited support for models with nominal predictors.
`parsnip::model_fit`	Supported as long as the type of model and the engine is supported.
`plm::plm()`	
`pscl::hurdle()`	Use `tidy_zeroinfl()` as `tidy_fun`.
`pscl::zeroinfl()`	Use `tidy_zeroinfl()` as `tidy_fun`.
`rstanarm::stan_glm()`	`broom.mixed` package required
`stats::aov()`	Reference rows are not relevant for such models.
`stats::glm()`	
`stats::lm()`	
`stats::nls()`	Limited support
`survey::svycoxph()`	
`survey::svyglm()`	
`survey::svyolr()`	
`survival::cch()`	Experimental support.
`survival::clogit()`	
`survival::coxph()`	
`survival::survreg()`	
`svyVGAM::svy_vglm()`	Experimental support. It is recommended to use `tidy_svy_vglm()` as `tidy_fun`.

`tidycmprsk::crr()`	
`VGAM::vgam()`	Experimental support. It is recommended to use `tidy_vgam()` as `tidy_fun`.
`VGAM::vglm()`	Experimental support. It is recommended to use `tidy_vgam()` as `tidy_fun`.

tidy_add_coefficients_type

Add coefficients type and label as attributes

Description

Add the type of coefficients ("generic", "logistic", "poisson", "relative_risk" or "prop_hazard") and the corresponding coefficient labels, as attributes to `x` (respectively named `coefficients_type` and `coefficients_label`).

Usage

```
tidy_add_coefficients_type(
  x,
  exponentiate = attr(x, "exponentiate"),
  model = tidy_get_model(x)
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`exponentiate`	(`logical`) Whether or not to exponentiate the coefficient estimates. It should be consistent with the original call to `broom::tidy()`.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.

See Also

Other tidy_helpers: `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_rows()`, `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
ex1 <- lm(hp ~ mpg + factor(cyl), mtcars) |>
  tidy_and_attach() |>
  tidy_add_coefficients_type()
attr(ex1, "coefficients_type")
attr(ex1, "coefficients_label")

df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
ex2 <- glm(
  Survived ~ Class + Age * Sex,
  data = df,
  weights = df$n,
  family = binomial
) |>
  tidy_and_attach(exponentiate = TRUE) |>
  tidy_add_coefficients_type()
attr(ex2, "coefficients_type")
attr(ex2, "coefficients_label")
```

`tidy_add_contrasts` *Add contrasts type for categorical variables*

Description

Add a `contrasts` column corresponding to contrasts used for a categorical variable and a `contrasts_type` column equal to "treatment", "sum", "poly", "helmert", "other" or "no.contrast".

Usage

```
tidy_add_contrasts(x, model = tidy_get_model(x), quiet = FALSE)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether broom.helpers should not return a message when `tidy_disambiguate_terms()` was already applied

Details

If the `variable` column is not yet available in `x`, `tidy_identify_variables()` will be automatically applied.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_estimate_to_reference_rows()`, `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))

glm(
  Survived ~ Class + Age + Sex,
  data = df, weights = df$n, family = binomial,
  contrasts = list(Age = contr.sum, Class = "contr.helmert")
) |>
  tidy_and_attach() |>
  tidy_add_contrasts()
```

`tidy_add_estimate_to_reference_rows`

Add an estimate value to references rows for categorical variables

---

Description

For categorical variables with a treatment contrast (`stats::contr.treatment()`) or a SAS contrast (`stats::contr.SAS()`), will add an estimate equal to `0` (or 1 if `exponentiate` = TRUE) to the reference row.

Usage

```
tidy_add_estimate_to_reference_rows(
  x,
  exponentiate = attr(x, "exponentiate"),
  conf.level = attr(x, "conf.level"),
  model = tidy_get_model(x),
  quiet = FALSE
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`exponentiate`	(`logical`) Whether or not to exponentiate the coefficient estimates. It should be consistent with the original call to `broom::tidy()`

`conf.level`	(`numeric`) Confidence level, by default use the value indicated previously in `tidy_and_attach()`, used only for sum contrasts.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.

Details

For categorical variables with a sum contrast (`stats::contr.sum()`), the estimate value of the reference row will be equal to the sum of all other coefficients multiplied by `-1` (eventually exponentiated if `exponentiate = TRUE`), and obtained with `emmeans::emmeans()`. The `emmeans` package should therefore be installed. For sum contrasts, the model coefficient corresponds to the difference of each level with the grand mean. For sum contrasts, confidence intervals and p-values will also be computed and added to the reference rows.

For other variables, no change will be made.

If the `reference_row` column is not yet available in `x`, `tidy_add_reference_rows()` will be automatically applied.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(dplyr::across(where(is.character), factor))

glm(
  Survived ~ Class + Age + Sex,
  data = df, weights = df$n, family = binomial,
  contrasts = list(Age = contr.sum, Class = "contr.SAS")
) |>
  tidy_and_attach(exponentiate = TRUE) |>
  tidy_add_reference_rows() |>
  tidy_add_estimate_to_reference_rows()

glm(
  response ~ stage + grade * trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(
    stage = contr.treatment(4, base = 3),
    grade = contr.treatment(3, base = 2),
    trt = contr.treatment(2, base = 2)
  )
) |>
  tidy_and_attach() |>
  tidy_add_reference_rows() |>
  tidy_add_estimate_to_reference_rows()
```

`tidy_add_header_rows` *Add header rows variables with several terms*

Description

For variables with several terms (usually categorical variables but could also be the case of continuous variables with polynomial terms or splines), `tidy_add_header_rows()` will add an additional row per variable, where `label` will be equal to `var_label`. These additional rows could be identified with `header_row` column.

Usage

```
tidy_add_header_rows(
  x,
  show_single_row = NULL,
  model = tidy_get_model(x),
  quiet = FALSE,
  strict = FALSE
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`show_single_row`	(`tidy-select`) Names of dichotomous variables that should be displayed on a single row. See also `all_dichotomous()`.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.
`strict`	(`logical`) Whether `broom.helpers` should return an error when requested output cannot be generated. Default is `FALSE`.

Details

The `show_single_row` argument allows to specify a list of dichotomous variables that should be displayed on a single row instead of two rows.

The added `header_row` column will be equal to:

- `TRUE` for an header row;
- `FALSE` for a normal row of a variable with an header row;
- `NA` for variables without an header row.

If the `label` column is not yet available in `x`, `tidy_add_term_labels()` will be automatically applied.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))

res <-
  glm(
    Survived ~ Class + Age + Sex,
    data = df, weights = df$n, family = binomial,
    contrasts = list(Age = contr.sum, Class = "contr.SAS")
  ) |>
  tidy_and_attach() |>
  tidy_add_variable_labels(labels = list(Class = "Custom label for Class")) |>
  tidy_add_reference_rows()
res |> tidy_add_header_rows()
res |> tidy_add_header_rows(show_single_row = all_dichotomous())

glm(
  response ~ stage + grade * trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(
    stage = contr.treatment(4, base = 3),
    grade = contr.treatment(3, base = 2),
    trt = contr.treatment(2, base = 2)
  )
) |>
  tidy_and_attach() |>
  tidy_add_reference_rows() |>
  tidy_add_header_rows()
```

`tidy_add_n` *Add the (weighted) number of observations*

Description

Add the number of observations in a new column `n_obs`, taking into account any weights if they have been defined.

Usage

```
tidy_add_n(x, model = tidy_get_model(x))
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.

Details

For continuous variables, it corresponds to all valid observations contributing to the model.

For categorical variables coded with treatment or sum contrasts, each model term could be associated to only one level of the original categorical variable. Therefore, `n_obs` will correspond to the number of observations associated with that level. `n_obs` will also be computed for reference rows. For polynomial contrasts (defined with `stats::contr.poly()`), all levels will contribute to the computation of each model term. Therefore, `n_obs` will be equal to the total number of observations. For Helmert and custom contrasts, only rows contributing positively (i.e. with a positive contrast) to the computation of a term will be considered for estimating `n_obs`. The result could therefore be difficult to interpret. For a better understanding of which observations are taken into account to compute `n_obs` values, you could look at `model_compute_terms_contributions()`.

For interaction terms, only rows contributing to all the terms of the interaction will be considered to compute `n_obs`.

For binomial logistic models, `tidy_add_n()` will also return the corresponding number of events (`n_event`) for each term, taking into account any defined weights. Observed proportions could be obtained as `n_obs / n_event`.

Similarly, a number of events will be computed for multinomial logistic models (`nnet::multinom()`) for each level of the outcome (`y.level`), corresponding to the number of observations equal to that outcome level.

For Poisson models, `n_event` will be equal to the number of counts per term. In addition, a third column `exposure` will be computed. If no offset is defined, exposure is assumed to be equal to 1 (eventually multiplied by weights) per observation. If an offset is defined, `exposure` will be equal to the (weighted) sum of the exponential of the offset (as a reminder, to model the effect of `x` on the ratio `y / z`, a Poisson model will be defined as `glm(y ~ x + offset(log(z)), family = poisson)`). Observed rates could be obtained with `n_event / exposure`.

For Cox models (`survival::coxph()`), an individual could be coded with several observations (several rows). `n_obs` will correspond to the weighted number of observations which could be different from the number of individuals `n_ind`. `tidy_add_n()` will also compute a (weighted) number of events (`n_event`) according to the definition of the `survival::Surv()` object. Exposure time is also returned in `exposure` column. It is equal to the (weighted) sum of the time variable if only one variable time is passed to `survival::Surv()`, and to the (weighted) sum of `time2 - time` if two time variables are defined in `survival::Surv()`.

For competing risk regression models (`tidycmprsk::crr()`), `n_event` takes into account only the event of interest defined by `failcode`.

The (weighted) total number of observations (`N_obs`), of individuals (`N_ind`), of events (`N_event`) and of exposure time (`Exposure`) are stored as attributes of the returned tibble.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
lm(Petal.Length ~ ., data = iris) |>
  tidy_and_attach() |>
  tidy_add_n()

lm(Petal.Length ~ ., data = iris, contrasts = list(Species = contr.sum)) |>
  tidy_and_attach() |>
  tidy_add_n()

lm(Petal.Length ~ ., data = iris, contrasts = list(Species = contr.poly)) |>
  tidy_and_attach() |>
  tidy_add_n()

lm(Petal.Length ~ poly(Sepal.Length, 2), data = iris) |>
  tidy_and_attach() |>
  tidy_add_n()

df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))

glm(
  Survived ~ Class + Age + Sex,
  data = df, weights = df$n, family = binomial,
  contrasts = list(Age = contr.sum, Class = "contr.helmert")
) |>
```

```
  tidy_and_attach() |>
  tidy_add_n()

glm(
  Survived ~ Class * (Age:Sex),
  data = df, weights = df$n, family = binomial,
  contrasts = list(Age = contr.sum, Class = "contr.helmert")
) |>
  tidy_and_attach() |>
  tidy_add_n()

glm(response ~ age + grade * trt, gtsummary::trial, family = poisson) |>
  tidy_and_attach() |>
  tidy_add_n()

glm(
  response ~ trt * grade + offset(log(ttdeath)),
  gtsummary::trial,
  family = poisson
) |>
  tidy_and_attach() |>
  tidy_add_n()
```

tidy_add_pairwise_contrasts

Add pairwise contrasts for categorical variables

Description

Computes pairwise contrasts with `emmeans::emmeans()` and add them to the results tibble. Works only with models supported by emmeans, see `vignette("models", package = "emmeans")`.

Usage

```
tidy_add_pairwise_contrasts(
  x,
  variables = all_categorical(),
  keep_model_terms = FALSE,
  pairwise_reverse = TRUE,
  contrasts_adjust = NULL,
  conf.level = attr(x, "conf.level"),
  emmeans_args = list(),
  model = tidy_get_model(x),
  quiet = FALSE
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`variables`	include (`tidy-select`) Variables for those pairwise contrasts should be added. Default is `all_categorical()`.
`keep_model_terms`	(`logical`) Keep terms from the model?
`pairwise_reverse`	(`logical`) Determines whether to use `"pairwise"` (if `TRUE`) or `"revpairwise"` (if `FALSE`), see `emmeans::contrast()`.
`contrasts_adjust`	(`string`) Optional adjustment method when computing contrasts, see `emmeans::contrast()` (if `NULL`, use `emmeans` default).
`conf.level`	(`numeric`) Confidence level, by default use the value indicated previously in `tidy_and_attach()`.
`emmeans_args`	(`list`) List of additional parameter to pass to `emmeans::emmeans()` when computing pairwise contrasts.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.

Note

If the `contrasts` column is not yet available in `x`, `tidy_add_contrasts()` will be automatically applied.

For multi-components models, such as zero-inflated Poisson or beta regression, support of pairwise contrasts is still experimental.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
mod1 <- lm(Sepal.Length ~ Species, data = iris)
mod1 |>
```

```
    tidy_and_attach() |>
    tidy_add_pairwise_contrasts()

  mod1 |>
    tidy_and_attach() |>
    tidy_add_pairwise_contrasts(pairwise_reverse = FALSE)

  mod1 |>
    tidy_and_attach() |>
    tidy_add_pairwise_contrasts(keep_model_terms = TRUE)

  mod1 |>
    tidy_and_attach() |>
    tidy_add_pairwise_contrasts(contrasts_adjust = "none")

  if (.assert_package("gtsummary", boolean = TRUE)) {
    mod2 <- glm(
      response ~ age + trt + grade,
      data = gtsummary::trial,
      family = binomial
    )
    mod2 |>
      tidy_and_attach(exponentiate = TRUE) |>
      tidy_add_pairwise_contrasts()
  }
```

tidy_add_reference_rows

Add references rows for categorical variables

Description

For categorical variables with a treatment contrast (`stats::contr.treatment()`), a SAS contrast (`stats::contr.SAS()`) a sum contrast (`stats::contr.sum()`), or successive differences contrast (`MASS::contr.sdif()`) add a reference row.

Usage

```
tidy_add_reference_rows(
  x,
  no_reference_row = NULL,
  model = tidy_get_model(x),
  quiet = FALSE
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`no_reference_row`	(`tidy-select`) Variables for those no reference row should be added. See also `all_categorical()` and `all_dichotomous()`.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.

Details

The added `reference_row` column will be equal to:

- `TRUE` for a reference row;
- `FALSE` for a normal row of a variable with a reference row;
- `NA` for variables without a reference row.

If the `contrasts` column is not yet available in `x`, `tidy_add_contrasts()` will be automatically applied.

`tidy_add_reference_rows()` will not populate the label of the reference term. It is therefore better to apply `tidy_add_term_labels()` after `tidy_add_reference_rows()` rather than before. Similarly, it is better to apply `tidy_add_reference_rows()` before `tidy_add_n()`.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))

res <-
  glm(
    Survived ~ Class + Age + Sex,
    data = df, weights = df$n, family = binomial,
    contrasts = list(Age = contr.sum, Class = "contr.SAS")
  ) |>
  tidy_and_attach()
res |> tidy_add_reference_rows()
res |> tidy_add_reference_rows(no_reference_row = all_dichotomous())
res |> tidy_add_reference_rows(no_reference_row = "Class")

glm(
  response ~ stage + grade * trt,
  gtsummary::trial,
  family = binomial,
  contrasts = list(
    stage = contr.treatment(4, base = 3),
    grade = contr.treatment(3, base = 2),
    trt = contr.treatment(2, base = 2)
  )
) |>
  tidy_and_attach() |>
  tidy_add_reference_rows()
```

`tidy_add_term_labels` *Add term labels*

Description

Will add term labels in a `label` column, based on:

1. labels provided in `labels` argument if provided;
2. factor levels for categorical variables coded with treatment, SAS or sum contrasts (the label could be customized with `categorical_terms_pattern` argument);
3. variable labels when there is only one term per variable;
4. term name otherwise.

Usage

```
tidy_add_term_labels(
  x,
  labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  relabel_poly = FALSE,
  model = tidy_get_model(x),
  quiet = FALSE,
  strict = FALSE
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`labels`	(`list` or `string`) An optional named list or named vector of custom term labels.
`interaction_sep`	(`string`) Separator for interaction terms.
`categorical_terms_pattern`	(`glue pattern`) A glue pattern for labels of categorical terms with treatment or sum contrasts (see examples and `model_list_terms_levels()`).
`relabel_poly`	Should terms generated with `stats::poly()` be relabeled?
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.
`strict`	(`logical`) Whether `broom.helpers` should return an error when requested output cannot be generated. Default is `FALSE`.

Details

If the `variable_label` column is not yet available in `x`, `tidy_add_variable_labels()` will be automatically applied. If the `contrasts` column is not yet available in `x`, `tidy_add_contrasts()` will be automatically applied.

It is possible to pass a custom label for any term in `labels`, including interaction terms.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes"))) |>
  labelled::set_variable_labels(
    Class = "Passenger's class",
    Sex = "Sex"
  )

mod <-
```

```
    glm(Survived ~ Class * Age * Sex, data = df, weights = df$n, family = binomial)
  mod |>
    tidy_and_attach() |>
    tidy_add_term_labels()
  mod |>
    tidy_and_attach() |>
    tidy_add_term_labels(
      interaction_sep = " x ",
      categorical_terms_pattern = "{level} / {reference_level}"
    )
```

`tidy_add_variable_labels` *Add variable labels*

Description

Will add variable labels in a `var_label` column, based on:

1. labels provided in `labels` argument if provided;
2. variable labels defined in the original data frame with the `label` attribute (cf. `labelled::var_label()`);
3. variable name otherwise.

Usage

```
tidy_add_variable_labels(
  x,
  labels = NULL,
  interaction_sep = " * ",
  instrumental_suffix = " (instrumental)",
  model = tidy_get_model(x)
)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`labels`	(`formula-list-selector`) An optional named list or a named vector of custom variable labels.
`interaction_sep`	(`string`) Separator for interaction terms.
`instrumental_suffix`	(`string`) Suffix added to variable labels for instrumental variables (`fixest` models). `NULL` to add nothing.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.

Details

If the `variable` column is not yet available in `x`, `tidy_identify_variables()` will be automatically applied.

It is possible to pass a custom label for an interaction term in `labels` (see examples).

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes"))) |>
  labelled::set_variable_labels(
    Class = "Passenger's class",
    Sex = "Sex"
  )

glm(Survived ~ Class * Age * Sex, data = df, weights = df$n, family = binomial) |>
  tidy_and_attach() |>
  tidy_add_variable_labels(
    labels = list(
      "(Intercept)" ~ "Custom intercept",
      Sex ~ "Gender",
      "Class:Age" ~ "Custom label"
    )
  )
```

`tidy_all_effects` *Marginal Predictions at the mean with* `effects::allEffects()`

Description

Use `effects::allEffects()` to estimate marginal predictions and return a tibble tidied in a way that it could be used by `broom.helpers` functions. See `vignette("functions-supported-by-effects", package = "effects")` for a list of supported models.

Usage

```
tidy_all_effects(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `effects::allEffects()`.

Details

By default, `effects::allEffects()` estimate marginal predictions at the mean at the observed means for continuous variables and weighting modalities of categorical variables according to their observed distribution in the original dataset. Marginal predictions are therefore computed at a sort of averaged situation / typical values for the other variables fixed in the model.

For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

Note

If the model contains interactions, `effects::allEffects()` will return marginal predictions for the different levels of the interactions.

See Also

`effects::allEffects()`

Other marginal_tieders: `tidy_avg_comparisons()`, `tidy_avg_slopes()`, `tidy_ggpredict()`, `tidy_marginal_contrasts()`, `tidy_marginal_predictions()`, `tidy_margins()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_all_effects(mod)
tidy_plus_plus(mod, tidy_fun = tidy_all_effects)
```

tidy_attach_model *Attach a full model to the tibble of model terms*

Description

To facilitate the use of broom helpers with pipe, it is recommended to attach the original model as an attribute to the tibble of model terms generated by `broom::tidy()`.

Usage

```
tidy_attach_model(x, model, .attributes = NULL)

tidy_and_attach(
  model,
  tidy_fun = tidy_with_broom_or_parameters,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  model_matrix_attr = TRUE,
  ...
)

tidy_get_model(x)

tidy_detach_model(x)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`model`	(a model object, e.g. `glm`) A model to be attached/tidied.
`.attributes`	(`list`) Named list of additional attributes to be attached to `x`.
`tidy_fun`	(`function`) Option to specify a custom tidier function.
`conf.int`	(`logical`) Should confidence intervals be computed? (see `broom::tidy()`)
`conf.level`	(`numeric`) Level of confidence for confidence intervals (default: 95%).
`exponentiate`	(`logical`) Whether or not to exponentiate the coefficient estimates. This is typical for logistic, Poisson and Cox models, but a bad idea if there is no log or logit link; defaults to `FALSE`.

`model_matrix_attr`
: (`logical`)
  Whether model frame and model matrix should be added as attributes of `model` (respectively named `"model_frame"` and `"model_matrix"`) and passed through

`...`
: Other arguments passed to `tidy_fun()`.

Details

`tidy_attach_model()` attach the model to a tibble already generated while `tidy_and_attach()` will apply `broom::tidy()` and attach the model.

Use `tidy_get_model()` to get the model attached to the tibble and `tidy_detach_model()` to remove the attribute containing the model.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro` `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
mod <- lm(Sepal.Length ~ Sepal.Width + Species, data = iris)
tt <- mod |>
  tidy_and_attach(conf.int = TRUE)
tt
tidy_get_model(tt)
```

`tidy_avg_comparisons` *Marginal Contrasts with* `marginaleffects::avg_comparisons()`

Description

Use `marginaleffects::avg_comparisons()` to estimate marginal contrasts and return a tibble tidied in a way that it could be used by `broom.helpers` functions. See `marginaleffects::avg_comparisons()` for a list of supported models.

Usage

```
tidy_avg_comparisons(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`
: (a model object, e.g. `glm`)
  A model to be tidied.

`conf.int`
: (`logical`)
  Whether or not to include a confidence interval in the tidied output.

`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `marginaleffects::avg_comparisons()`.

Details

By default, `marginaleffects::avg_comparisons()` estimate average marginal contrasts: a contrast is computed for each observed value in the original dataset (counterfactual approach) before being averaged. Marginal Contrasts at the Mean could be computed by specifying `newdata = "mean"`. The `variables` argument can be used to select the contrasts to be computed. Please refer to the documentation page of `marginaleffects::avg_comparisons()`.

See also `tidy_marginal_contrasts()` for taking into account interactions. For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

See Also

`marginaleffects::avg_comparisons()`

Other marginal_tieders: `tidy_all_effects()`, `tidy_avg_slopes()`, `tidy_ggpredict()`, `tidy_marginal_contrasts()`, `tidy_marginal_predictions()`, `tidy_margins()`

Examples

```
# Average Marginal Contrasts

df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_avg_comparisons(mod)
tidy_plus_plus(mod, tidy_fun = tidy_avg_comparisons)

mod2 <- lm(Petal.Length ~ poly(Petal.Width, 2) + Species, data = iris)
tidy_avg_comparisons(mod2)

# Custumizing the type of contrasts
tidy_avg_comparisons(
  mod2,
  variables = list(Petal.Width = 2, Species = "pairwise")
)

# Marginal Contrasts at the Mean
tidy_avg_comparisons(mod, newdata = "mean")
tidy_plus_plus(mod, tidy_fun = tidy_avg_comparisons, newdata = "mean")
```

tidy_avg_slopes *Marginal Slopes / Effects with* marginaleffects::avg_slopes()

Description

Use marginaleffects::avg_slopes() to estimate marginal slopes / effects and return a tibble tidied in a way that it could be used by broom.helpers functions. See marginaleffects::avg_slopes() for a list of supported models.

Usage

```
tidy_avg_slopes(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

x	(a model object, e.g. glm) A model to be tidied.
conf.int	(logical) Whether or not to include a confidence interval in the tidied output.
conf.level	(numeric) The confidence level to use for the confidence interval (between 0 ans 1).
...	Additional parameters passed to marginaleffects::avg_slopes().

Details

By default, marginaleffects::avg_slopes() estimate average marginal effects (AME): an effect is computed for each observed value in the original dataset before being averaged. Marginal Effects at the Mean (MEM) could be computed by specifying newdata = "mean". Other types of marginal effects could be computed. Please refer to the documentation page of marginaleffects::avg_slopes().

For more information, see vignette("marginal_tidiers", "broom.helpers").

See Also

marginaleffects::avg_slopes()

Other marginal_tieders: tidy_all_effects(), tidy_avg_comparisons(), tidy_ggpredict(), tidy_marginal_contrasts(), tidy_marginal_predictions(), tidy_margins()

Examples

```
# Average Marginal Effects (AME)

df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_avg_slopes(mod)
tidy_plus_plus(mod, tidy_fun = tidy_avg_slopes)

mod2 <- lm(Petal.Length ~ poly(Petal.Width, 2) + Species, data = iris)
tidy_avg_slopes(mod2)

# Marginal Effects at the Mean (MEM)
tidy_avg_slopes(mod, newdata = "mean")
tidy_plus_plus(mod, tidy_fun = tidy_avg_slopes, newdata = "mean")
```

`tidy_broom` *Tidy with* `broom::tidy()` *and checks that all arguments are used*

Description

Tidy with `broom::tidy()` and checks that all arguments are used

Usage

```
tidy_broom(x, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`...`	Additional parameters passed to `broom::tidy()`.

See Also

Other custom_tieders: `tidy_multgee()`, `tidy_parameters()`, `tidy_svy_vglm()`, `tidy_vgam()`, `tidy_with_broom_or_parameters()`, `tidy_zeroinfl()`

`tidy_disambiguate_terms` *Disambiguate terms*

Description

For mixed models, the `term` column returned by `broom.mixed` may have duplicated values for random-effect parameters and random-effect values. In such case, the terms could be disambiguated be prefixing them with the value of the `group` column. `tidy_disambiguate_terms()` will not change any term if there is no `group` column in `x`. The original term value is kept in a new column `original_term`.

Usage

```
tidy_disambiguate_terms(x, sep = ".", model = tidy_get_model(x), quiet = FALSE)
```

Arguments

x	(data.frame) A tidy tibble as produced by `tidy_*()` functions.
sep	(string) Separator added between group name and term.
model	(a model object, e.g. glm) The corresponding model, if not attached to x.
quiet	(logical) Whether broom.helpers should not return a message when requested output cannot be generated. Default is FALSE.

See Also

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_ro
tidy_add_header_rows(), tidy_add_n(), tidy_add_pairwise_contrasts(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_add_variable_labels(), tidy_attach_model(), tidy_group_by(), tidy_identify_variables(), tidy_plus_plus(), tidy_remove_intercept(), tidy_select_variables()

Examples

```
if (
  .assert_package("lme4", boolean = TRUE) &&
    .assert_package("broom.mixed", boolean = TRUE) &&
    .assert_package("gtsummary", boolean = TRUE)
) {
  mod <- lme4::lmer(marker ~ stage + (1 | grade) + (death | response), gtsummary::trial)
  mod |>
    tidy_and_attach() |>
    tidy_disambiguate_terms()
}
```

---

tidy_ggpredict — *Marginal Predictions with* `ggeffects::ggpredict()`

Description

Use `ggeffects::ggpredict()` to estimate marginal predictions and return a tibble tidied in a way that it could be used by broom.helpers functions. See https://strengejacke.github.io/ggeffects/ for a list of supported models.

Usage

```
tidy_ggpredict(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `ggeffects::ggpredict()`.

Details

By default, `ggeffects::ggpredict()` estimate marginal predictions at the observed mean of continuous variables and at the first modality of categorical variables (regardless of the type of contrasts used in the model).

For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

Note

By default, `ggeffects::ggpredict()` estimates marginal predictions for each individual variable, regardless of eventual interactions.

See Also

`ggeffects::ggpredict()`

Other marginal_tieders: `tidy_all_effects()`, `tidy_avg_comparisons()`, `tidy_avg_slopes()`, `tidy_marginal_contrasts()`, `tidy_marginal_predictions()`, `tidy_margins()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_ggpredict(mod)
tidy_plus_plus(mod, tidy_fun = tidy_ggpredict)
```

tidy_group_by *Group results by selected columns*

Description

Indicates that results should be grouped. By default (group_by = auto_group_by()), results will be grouped according to the y.level column (for multinomial models) or the component column (multi-components models) if any.

Usage

```
tidy_group_by(
  x,
  group_by = auto_group_by(),
  group_labels = NULL,
  model = tidy_get_model(x)
)

auto_group_by()
```

Arguments

x	(data.frame) A tidy tibble as produced by tidy_*() functions.
group_by	(tidy-select) One or several variables to group by. Default is auto_group_by(). Use NULL to force ungrouping.
group_labels	(string) An optional named vector of custom term labels.
model	(a model object, e.g. glm) The corresponding model, if not attached to x.

Value

The x tibble with, if relevant, an additional group_by column.

See Also

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_ro
tidy_add_header_rows(), tidy_add_n(), tidy_add_pairwise_contrasts(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_add_variable_labels(), tidy_attach_model(), tidy_disambiguate_terms(), tidy_identify_variables(), tidy_plus_plus(), tidy_remove_intercept(), tidy_select_variables()

Examples

```
mod <- multinom(Species ~ Petal.Width + Petal.Length, data = iris)
mod |> tidy_and_attach() |> tidy_group_by()

mod |>
  tidy_and_attach() |>
  tidy_group_by(group_labels = c(versicolor = "harlequin blueflag"))

mod |> tidy_and_attach() |> tidy_group_by(group_by = NULL)

mod |>
  tidy_and_attach() |>
  tidy_identify_variables() |>
  tidy_group_by(group_by = variable)
```

---

tidy_identify_variables

Identify the variable corresponding to each model coefficient

---

Description

`tidy_identify_variables()` will add to the tidy tibble three additional columns: `variable`, `var_class`, `var_type` and `var_nlevels`.

Usage

```
tidy_identify_variables(x, model = tidy_get_model(x), quiet = FALSE)
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.

Details

It will also identify interaction terms and intercept(s).

`var_type` could be:

- `"continuous"`,
- `"dichotomous"` (categorical variable with 2 levels),
- `"categorical"` (categorical variable with 3 levels or more),

- "intercept"
- "interaction"
- "ran_pars (random-effect parameters for mixed models)
- "ran_vals" (random-effect values for mixed models)
- "unknown" in the rare cases where tidy_identify_variables() will fail to identify the list of variables

For dichotomous and categorical variables, var_nlevels corresponds to the number of original levels in the corresponding variables.

For fixest models, a new column instrumental is added to indicate instrumental variables.

See Also

model_identify_variables()

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_ro tidy_add_header_rows(), tidy_add_n(), tidy_add_pairwise_contrasts(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_add_variable_labels(), tidy_attach_model(), tidy_disambiguate_terms(), tidy_group_by(), tidy_plus_plus(), tidy_remove_intercept(), tidy_select_variables()

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
glm(
  Survived ~ Class + Age * Sex,
  data = df,
  weights = df$n,
  family = binomial
) |>
  tidy_and_attach() |>
  tidy_identify_variables()

lm(
  Sepal.Length ~ poly(Sepal.Width, 2) + Species,
  data = iris,
  contrasts = list(Species = contr.sum)
) |>
  tidy_and_attach(conf.int = TRUE) |>
  tidy_identify_variables()
```

---

tidy_marginal_contrasts *Marginal Contrasts with* marginaleffects::avg_comparisons()

Description

Use `marginaleffects::avg_comparisons()` to estimate marginal contrasts for each variable of a model and return a tibble tidied in a way that it could be used by `broom.helpers` functions. See `marginaleffects::avg_comparisons()` for a list of supported models.

Usage

```
tidy_marginal_contrasts(
  x,
  variables_list = "auto",
  conf.int = TRUE,
  conf.level = 0.95,
  ...
)

variables_to_contrast(
  model,
  interactions = TRUE,
  cross = FALSE,
  var_categorical = "reference",
  var_continuous = 1,
  by_categorical = unique,
  by_continuous = stats::fivenum
)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`variables_list`	(`list` or `string`) A list whose elements will be sequentially passed to `variables` in `marginaleffects::avg_comparison` (see details below); alternatively, it could also be the string `"auto"` (default), `"cross"` or `"no_interaction"`
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `marginaleffects::avg_comparisons()`.
`model`	(a model object, e.g. `glm`) A model.
`interactions`	(`logical`) Should combinations of variables corresponding to interactions be returned?
`cross`	(`logical`) If `interaction` is `TRUE`, should "cross-contrasts" be computed? (if `FALSE`, only the last term of an interaction is passed to `variable` and the other terms are passed to `by`)

`var_categorical`	(predictor values) Default `variable` value for categorical variables.
`var_continuous`	(predictor values) Default `variable` value for continuous variables.
`by_categorical`	(predictor values) Default by value for categorical variables.
`by_continuous`	(predictor values) Default by value for continuous variables.

Details

Marginal contrasts are obtained by calling, for each variable or combination of variables, `marginaleffects::avg_comparis`

`tidy_marginal_contrasts()` will compute marginal contrasts for each variable or combination of variables, before stacking the results in a unique tibble. This is why `tidy_marginal_contrasts()` has a `variables_list` argument consisting of a list of specifications that will be passed sequentially to the `variables` and the by argument of `marginaleffects::avg_comparisons()`.

Considering a single categorical variable named `cat`, `tidy_marginal_contrasts()` will call `avg_comparisons(model, variables = list(cat = "reference"))` to obtain average marginal contrasts for this variable.

Considering a single continuous variable named `cont`, `tidy_marginalcontrasts()` will call `avg_comparisons(model, variables = list(cont = 1))` to obtain average marginal contrasts for an increase of one unit.

For a combination of variables, there are several possibilities. You could compute "cross-contrasts" by providing simultaneously several variables to `variables` and specifying `cross = TRUE` to `marginaleffects::avg_compa` Alternatively, you could compute the contrasts of a first variable specified to `variables` for the different values of a second variable specified to by.

The helper function `variables_to_contrast()` could be used to automatically generate a suitable list to be used with `variables_list`. Each combination of variables should be a list with two named elements: `"variables"` a list of named elements passed to `variables` and `"by"` a list of named elements used for creating a relevant `datagrid` and whose names are passed to by.

`variables_list`'s default value, `"auto"`, calls `variables_to_contrast(interactions = TRUE, cross = FALSE)` while `"no_interaction"` is a shortcut for `variables_to_contrast(interactions = FALSE)`. `"cross"` calls `variables_to_contrast(interactions = TRUE, cross = TRUE)`

You can also provide custom specifications (see examples).

By default, *average marginal contrasts* are computed: contrasts are computed using a counterfactual grid for each value of the variable of interest, before averaging the results. *Marginal contrasts at the mean* could be obtained by indicating `newdata = "mean"`. Other assumptions are possible, see the help file of `marginaleffects::avg_comparisons()`.

For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

See Also

`marginaleffects::avg_comparisons()`, `tidy_avg_comparisons()`

Other marginal_tieders: `tidy_all_effects()`, `tidy_avg_comparisons()`, `tidy_avg_slopes()`, `tidy_ggpredict()`, `tidy_marginal_predictions()`, `tidy_margins()`

Examples

```
# Average Marginal Contrasts
df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_marginal_contrasts(mod)
tidy_plus_plus(mod, tidy_fun = tidy_marginal_contrasts)

mod2 <- lm(Petal.Length ~ poly(Petal.Width, 2) + Species, data = iris)
tidy_marginal_contrasts(mod2)
tidy_marginal_contrasts(
  mod2,
  variables_list = variables_to_predict(
    mod2,
    continuous = 3,
    categorical = "pairwise"
  )
)

# Model with interactions
mod3 <- glm(
  Survived ~ Sex * Age + Class,
  data = df, family = binomial
)
tidy_marginal_contrasts(mod3)
tidy_marginal_contrasts(mod3, "no_interaction")
tidy_marginal_contrasts(mod3, "cross")
tidy_marginal_contrasts(
  mod3,
  variables_list = list(
    list(variables = list(Class = "pairwise"), by = list(Sex = unique)),
    list(variables = list(Age = "all")),
    list(variables = list(Class = "sequential", Sex = "reference"))
  )
)

mod4 <- lm(Sepal.Length ~ Petal.Length * Petal.Width + Species, data = iris)
tidy_marginal_contrasts(mod4)
tidy_marginal_contrasts(
  mod4,
  variables_list = list(
    list(
      variables = list(Species = "sequential"),
      by = list(Petal.Length = c(2, 5))
    ),
    list(
      variables = list(Petal.Length = 2),
      by = list(Species = unique, Petal.Width = 2:4)
    )
  )
)

# Marginal Contrasts at the Mean
tidy_marginal_contrasts(mod, newdata = "mean")
tidy_marginal_contrasts(mod3, newdata = "mean")
```

`tidy_marginal_means` *Marginal Means with deprecated* `marginaleffects::marginal_means()`

Description

[Deprecated] This function is deprecated. `marginal_means()` is not anymore exported by `marginaleffects`. Use instead `tidy_marginal_predictions()` with the option `newdata = "balanced"`.

Usage

```
tidy_marginal_means(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters.

`tidy_marginal_predictions` *Marginal Predictions with* `marginaleffects::avg_predictions()`

Description

Use `marginaleffects::avg_predictions()` to estimate marginal predictions for each variable of a model and return a tibble tidied in a way that it could be used by `broom.helpers` functions. See `marginaleffects::avg_predictions()` for a list of supported models.

Usage

```
tidy_marginal_predictions(
  x,
  variables_list = "auto",
  conf.int = TRUE,
  conf.level = 0.95,
  ...
)

variables_to_predict(
  model,
  interactions = TRUE,
  categorical = unique,
  continuous = stats::fivenum
)

plot_marginal_predictions(x, variables_list = "auto", conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`variables_list`	(`list` or `string`) A list whose elements will be sequentially passed to `variables` in `marginaleffects::avg_prediction` (see details below); alternatively, it could also be the string `"auto"` (default) or `"no_interaction"`.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `marginaleffects::avg_predictions()`.
`model`	(a model object, e.g. `glm`) A model.
`interactions`	(`logical`) Should combinations of variables corresponding to interactions be returned?
`categorical`	(`predictor values`) Default values for categorical variables.
`continuous`	(`predictor values`) Default values for continuous variables.

Details

Marginal predictions are obtained by calling, for each variable, `marginaleffects::avg_predictions()` with the same variable being used for the `variables` and the `by` argument.

Considering a categorical variable named `cat`, `tidy_marginal_predictions()` will call `avg_predictions(model, variables = list(cat = unique), by = "cat")` to obtain average marginal predictions for this variable.

Considering a continuous variable named `cont`, `tidy_marginal_predictions()` will call `avg_predictions(model, variables = list(cont = "fivenum"), by = "cont")` to obtain average marginal predictions for this variable at the minimum, the first quartile, the median, the third quartile and the maximum of the observed values of `cont`.

By default, *average marginal predictions* are computed: predictions are made using a counterfactual grid for each value of the variable of interest, before averaging the results. *Marginal predictions at the mean* could be obtained by indicating `newdata = "mean"`. Other assumptions are possible, see the help file of `marginaleffects::avg_predictions()`.

`tidy_marginal_predictions()` will compute marginal predictions for each variable or combination of variables, before stacking the results in a unique tibble. This is why `tidy_marginal_predictions()` has a `variables_list` argument consisting of a list of specifications that will be passed sequentially to the `variables` argument of `marginaleffects::avg_predictions()`.

The helper function `variables_to_predict()` could be used to automatically generate a suitable list to be used with `variables_list`. By default, all unique values are retained for categorical variables and `fivenum` (i.e. Tukey's five numbers, minimum, quartiles and maximum) for continuous variables. When `interactions = FALSE`, `variables_to_predict()` will return a list of all individual variables used in the model. If `interactions = FALSE`, it will search for higher order combinations of variables (see `model_list_higher_order_variables()`).

`variables_list`'s default value, `"auto"`, calls `variables_to_predict(interactions = TRUE)` while `"no_interaction"` is a shortcut for `variables_to_predict(interactions = FALSE)`.

You can also provide custom specifications (see examples).

`plot_marginal_predictions()` works in a similar way and returns a list of plots that could be combined with `patchwork::wrap_plots()` (see examples).

For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

See Also

`marginaleffects::avg_predictions()`

Other marginal_tieders: `tidy_all_effects()`, `tidy_avg_comparisons()`, `tidy_avg_slopes()`, `tidy_ggpredict()`, `tidy_marginal_contrasts()`, `tidy_margins()`

Examples

```
# example code


# Average Marginal Predictions
df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_marginal_predictions(mod)
tidy_plus_plus(mod, tidy_fun = tidy_marginal_predictions)
if (require("patchwork")) {
  plot_marginal_predictions(mod) |> patchwork::wrap_plots()
  plot_marginal_predictions(mod) |>
    patchwork::wrap_plots() &
    ggplot2::scale_y_continuous(limits = c(0, 1), label = scales::percent)
}

mod2 <- lm(Petal.Length ~ poly(Petal.Width, 2) + Species, data = iris)
tidy_marginal_predictions(mod2)
if (require("patchwork")) {
  plot_marginal_predictions(mod2) |> patchwork::wrap_plots()
}
tidy_marginal_predictions(
  mod2,
  variables_list = variables_to_predict(mod2, continuous = "threenum")
)
tidy_marginal_predictions(
  mod2,
  variables_list = list(
    list(Petal.Width = c(0, 1, 2, 3)),
    list(Species = unique)
  )
)
tidy_marginal_predictions(
  mod2,
  variables_list = list(list(Species = unique, Petal.Width = 1:3))
)

# Model with interactions
mod3 <- glm(
  Survived ~ Sex * Age + Class,
  data = df, family = binomial
)
tidy_marginal_predictions(mod3)
tidy_marginal_predictions(mod3, "no_interaction")
if (require("patchwork")) {
  plot_marginal_predictions(mod3) |>
    patchwork::wrap_plots()
  plot_marginal_predictions(mod3, "no_interaction") |>
    patchwork::wrap_plots()
}
tidy_marginal_predictions(
  mod3,
  variables_list = list(
    list(Class = unique, Sex = "Female"),
    list(Age = unique)
  )
)

# Marginal Predictions at the Mean
```

```
tidy_marginal_predictions(mod, newdata = "mean")
if (require("patchwork")) {
  plot_marginal_predictions(mod, newdata = "mean") |>
    patchwork::wrap_plots()
}
```

`tidy_margins` *Average Marginal Effects with* `margins::margins()`

Description

[Superseded]

Usage

```
tidy_margins(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `margins::margins()`.

Details

The `margins` package is no longer under active development and may be removed from CRAN sooner or later. It is advised to use the `marginaleffects` package instead, offering more functionalities. You could have a look at the article dedicated to marginal estimates with `broom.helpers`. `tidy_avg_slopes()` could be used as an alternative.

Use `margins::margins()` to estimate average marginal effects (AME) and return a tibble tidied in a way that it could be used by `broom.helpers` functions. See `margins::margins()` for a list of supported models.

By default, `margins::margins()` estimate average marginal effects (AME): an effect is computed for each observed value in the original dataset before being averaged.

For more information, see `vignette("marginal_tidiers", "broom.helpers")`.

Note

When applying `margins::margins()`, custom contrasts are ignored. Treatment contrasts (`stats::contr.treatment()`) are applied to all categorical variables. Interactions are also ignored.

See Also

`margins::margins()`

Other marginal_tieders: `tidy_all_effects()`, `tidy_avg_comparisons()`, `tidy_avg_slopes()`, `tidy_ggpredict()`, `tidy_marginal_contrasts()`, `tidy_marginal_predictions()`

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  tidyr::uncount(n) |>
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes")))
mod <- glm(
  Survived ~ Class + Age + Sex,
  data = df, family = binomial
)
tidy_margins(mod)
tidy_plus_plus(mod, tidy_fun = tidy_margins)
```

`tidy_multgee` *Tidy a* `multgee` *model*

Description

A tidier for models generated with `multgee::nomLORgee()` or `multgee::ordLORgee()`. Term names will be updated to be consistent with generic models. The original term names are preserved in an `"original_term"` column.

Usage

```
tidy_multgee(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(`LORgee`) A `multgee::nomLORgee()` or a `multgee::ordLORgee()` model.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `parameters::model_parameters()`.

Details

To be noted, for `multgee::nomLORgee()`, the baseline `y` category is the latest modality of `y`.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_parameters()`, `tidy_svy_vglm()`, `tidy_vgam()`, `tidy_with_broom_or_parameters()`, `tidy_zeroinfl()`

Examples

```
library(multgee)

h <- housing
h$status <- factor(
  h$y,
  labels = c("street", "community", "independant")
)

mod <- multgee::nomLORgee(
  status ~ factor(time) * sec,
  data = h,
  id = id,
  repeated = time,
)
mod |> tidy_multgee()

mod2 <- ordLORgee(
  formula = y ~ factor(time) + factor(trt) + factor(baseline),
  data = multgee::arthritis,
  id = id,
  repeated = time,
  LORstr = "uniform"
)
mod2 |> tidy_multgee()
```

`tidy_parameters` *Tidy a model with parameters package*

Description

Use `parameters::model_parameters()` to tidy a model and apply `parameters::standardize_names(style = "broom")` to the output

Usage

```
tidy_parameters(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `parameters::model_parameters()`.

Note

For `betareg::betareg()`, the component column in the results is standardized with `broom::tidy()`, using `"mean"` and `"precision"` values.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_multgee()`, `tidy_svy_vglm()`, `tidy_vgam()`, `tidy_with_broom_or_parame`
`tidy_zeroinfl()`

Examples

```
lm(Sepal.Length ~ Sepal.Width + Species, data = iris) |>
  tidy_parameters()
```

`tidy_plus_plus` *Tidy a model and compute additional informations*

Description

This function will apply sequentially:

- `tidy_and_attach()`
- `tidy_disambiguate_terms()`
- `tidy_identify_variables()`
- `tidy_add_contrasts()`
- `tidy_add_reference_rows()`
- `tidy_add_pairwise_contrasts()`
- `tidy_add_estimate_to_reference_rows()`
- `tidy_add_variable_labels()`
- `tidy_add_term_labels()`
- `tidy_add_header_rows()`

- tidy_add_n()
- tidy_remove_intercept()
- tidy_select_variables()
- tidy_group_by()
- tidy_add_coefficients_type()
- tidy_detach_model()

Usage

```
tidy_plus_plus(
  model,
  tidy_fun = tidy_with_broom_or_parameters,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  model_matrix_attr = TRUE,
  variable_labels = NULL,
  instrumental_suffix = " (instrumental)",
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  relabel_poly = FALSE,
  disambiguate_terms = TRUE,
  disambiguate_sep = ".",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  add_pairwise_contrasts = FALSE,
  pairwise_variables = all_categorical(),
  keep_model_terms = FALSE,
  pairwise_reverse = TRUE,
  contrasts_adjust = NULL,
  emmeans_args = list(),
  add_estimate_to_reference_rows = TRUE,
  add_header_rows = FALSE,
  show_single_row = NULL,
  add_n = TRUE,
  intercept = FALSE,
  include = everything(),
  group_by = auto_group_by(),
  group_labels = NULL,
  keep_model = FALSE,
  tidy_post_fun = NULL,
  quiet = FALSE,
  strict = FALSE,
  ...
)
```

Arguments

`model`	(a model object, e.g. `glm`) A model to be attached/tidied.
`tidy_fun`	(`function`) Option to specify a custom tidier function.
`conf.int`	(`logical`) Should confidence intervals be computed? (see `broom::tidy()`)
`conf.level`	(`numeric`) Level of confidence for confidence intervals (default: 95%).
`exponentiate`	(`logical`) Whether or not to exponentiate the coefficient estimates. This is typical for logistic, Poisson and Cox models, but a bad idea if there is no log or logit link; defaults to `FALSE`.
`model_matrix_attr`	(`logical`) Whether model frame and model matrix should be added as attributes of `model` (respectively named `"model_frame"` and `"model_matrix"`) and passed through.
`variable_labels`	(`formula-list-selector`) A named list or a named vector of custom variable labels.
`instrumental_suffix`	(`string`) Suffix added to variable labels for instrumental variables (`fixest` models). `NULL` to add nothing.
`term_labels`	(`list` or `vector`) A named list or a named vector of custom term labels.
`interaction_sep`	(`string`) Separator for interaction terms.
`categorical_terms_pattern`	(`glue pattern`) A glue pattern for labels of categorical terms with treatment or sum contrasts (see `model_list_terms_levels()`).
`relabel_poly`	Should terms generated with `stats::poly()` be relabeled?
`disambiguate_terms`	(`logical`) Should terms be disambiguated with `tidy_disambiguate_terms()`? (default `TRUE`)
`disambiguate_sep`	(`string`) Separator for `tidy_disambiguate_terms()`.
`add_reference_rows`	(`logical`) Should reference rows be added?

`no_reference_row`	(`tidy-select`) Variables for those no reference row should be added, when `add_reference_rows = TRUE`.
`add_pairwise_contrasts`	(`logical`) Apply `tidy_add_pairwise_contrasts()`?
`pairwise_variables`	(`tidy-select`) Variables to add pairwise contrasts.
`keep_model_terms`	(`logical`) Keep original model terms for variables where pairwise contrasts are added? (default is `FALSE`)
`pairwise_reverse`	(`logical`) Determines whether to use `"pairwise"` (if `TRUE`) or `"revpairwise"` (if `FALSE`), see `emmeans::contrast()`.
`contrasts_adjust`	(`string`) Optional adjustment method when computing contrasts, see `emmeans::contrast()` (if `NULL`, use `emmeans` default).
`emmeans_args`	(`list`) List of additional parameter to pass to `emmeans::emmeans()` when computing pairwise contrasts.
`add_estimate_to_reference_rows`	(`logical`) Should an estimate value be added to reference rows?
`add_header_rows`	(`logical`) Should header rows be added?
`show_single_row`	(`tidy-select`) Variables that should be displayed on a single row, when `add_header_rows` is `TRUE`.
`add_n`	(`logical`) Should the number of observations be added?
`intercept`	(`logical`) Should the intercept(s) be included?
`include`	(`tidy-select`) Variables to include. Default is `everything()`. See also `all_continuous()`, `all_categorical()`, `all_dichotomous()` and `all_interaction()`.
`group_by`	(`tidy-select`) One or several variables to group by. Default is `auto_group_by()`. Use `NULL` to force ungrouping.

`group_labels`	(`string`) An optional named vector of custom term labels.
`keep_model`	(`logical`) Should the model be kept as an attribute of the final result?
`tidy_post_fun`	(`function`) Custom function applied to the results at the end of `tidy_plus_plus()` (see note)
`quiet`	(`logical`) Whether `broom.helpers` should not return a message when requested output cannot be generated. Default is `FALSE`.
`strict`	(`logical`) Whether `broom.helpers` should return an error when requested output cannot be generated. Default is `FALSE`.
`...`	other arguments passed to `tidy_fun()`

Note

`tidy_post_fun` is applied to the result at the end of `tidy_plus_plus()` and receive only one argument (the result of `tidy_plus_plus()`). However, if needed, the model is still attached to the tibble as an attribute, even if `keep_model = FALSE`. Therefore, it is possible to use `tidy_get_model()` within `tidy_fun` if, for any reason, you need to access the source model.

See Also

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_ro`
`tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_pairwise_contrasts()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_disambiguate_terms()`, `tidy_group_by()`, `tidy_identify_variables()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
ex1 <- lm(Sepal.Length ~ Sepal.Width + Species, data = iris) |>
  tidy_plus_plus()
ex1

df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(
    Survived = factor(Survived, c("No", "Yes"))
  ) |>
  labelled::set_variable_labels(
    Class = "Passenger's class",
    Sex = "Gender"
  )
ex2 <- glm(
  Survived ~ Class + Age * Sex,
  data = df, weights = df$n,
  family = binomial
) |>
```

```
    tidy_plus_plus(
      exponentiate = TRUE,
      add_reference_rows = FALSE,
      categorical_terms_pattern = "{level} / {reference_level}",
      add_n = TRUE
    )
ex2



ex3 <-
  glm(
    response ~ poly(age, 3) + stage + grade * trt,
    na.omit(gtsummary::trial),
    family = binomial,
    contrasts = list(
      stage = contr.treatment(4, base = 3),
      grade = contr.sum
    )
  ) |>
  tidy_plus_plus(
    exponentiate = TRUE,
    variable_labels = c(age = "Age (in years)"),
    add_header_rows = TRUE,
    show_single_row = all_dichotomous(),
    term_labels = c("poly(age, 3)3" = "Cubic age"),
    keep_model = TRUE
  )
ex3
```

`tidy_remove_intercept` *Remove intercept(s)*

Description

Will remove terms where `var_type == "intercept"`.

Usage

```
tidy_remove_intercept(x, model = tidy_get_model(x))
```

Arguments

`x`	(`data.frame`) A tidy tibble as produced by `tidy_*()` functions.
`model`	(a model object, e.g. `glm`) The corresponding model, if not attached to `x`.

Details

If the variable column is not yet available in x, tidy_identify_variables() will be automatically applied.

See Also

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_ro
tidy_add_header_rows(), tidy_add_n(), tidy_add_pairwise_contrasts(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_add_variable_labels(), tidy_attach_model(), tidy_disambiguate_terms(), tidy_group_by(), tidy_identify_variables(), tidy_plus_plus(), tidy_select_variables()

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived))
glm(Survived ~ Class + Age + Sex, data = df, weights = df$n, family = binomial) |>
  tidy_and_attach() |>
  tidy_remove_intercept()
```

---

tidy_select_variables *Select variables to keep/drop*

Description

Will remove unselected variables from the results. To remove the intercept, use tidy_remove_intercept().

Usage

```
tidy_select_variables(x, include = everything(), model = tidy_get_model(x))
```

Arguments

x	(data.frame) A tidy tibble as produced by tidy_*() functions.
include	(tidy-select) Variables to include. Default is everything(). See also all_continuous(), all_categorical(), all_dichotomous() and all_interaction().
model	(a model object, e.g. glm) The corresponding model, if not attached to x.

Details

If the variable column is not yet available in x, tidy_identify_variables() will be automatically applied.

Value

The x tibble limited to the included variables (and eventually the intercept), sorted according to the include parameter.

See Also

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_ro tidy_add_header_rows(), tidy_add_n(), tidy_add_pairwise_contrasts(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_add_variable_labels(), tidy_attach_model(), tidy_disambiguate_terms(), tidy_group_by(), tidy_identify_variables(), tidy_plus_plus(), tidy_remove_intercept()

Examples

```
df <- Titanic |>
  dplyr::as_tibble() |>
  dplyr::mutate(Survived = factor(Survived))
res <-
  glm(Survived ~ Class + Age * Sex, data = df, weights = df$n, family = binomial) |>
  tidy_and_attach() |>
  tidy_identify_variables()

res
res |> tidy_select_variables()
res |> tidy_select_variables(include = "Class")
res |> tidy_select_variables(include = -c("Age", "Sex"))
res |> tidy_select_variables(include = starts_with("A"))
res |> tidy_select_variables(include = all_categorical())
res |> tidy_select_variables(include = all_dichotomous())
res |> tidy_select_variables(include = all_interaction())
res |> tidy_select_variables(
  include = c("Age", all_categorical(dichotomous = FALSE), all_interaction())
)
```

---

tidy_svy_vglm *Tidy a* svy_vglm *model*

Description

[Experimental] A tidier for models generated with svyVGAM::svy_vglm(). Term names will be updated to be consistent with generic models. The original term names are preserved in an "original_term" column. Depending on the model, additional column "group", "component" and/or "y.level" may be added to the results.

Usage

```
tidy_svy_vglm(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(`svy_vglm`) A `svyVGAM::svy_vglm()` model.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `parameters::model_parameters()`.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_multgee()`, `tidy_parameters()`, `tidy_vgam()`, `tidy_with_broom_or_para`
`tidy_zeroinfl()`

Examples

```
library(svyVGAM)

mod <- svy_vglm(
  Species ~ Sepal.Length + Sepal.Width,
  family = multinomial(),
  design = survey::svydesign(~1, data = iris)
)
mod |> tidy_svy_vglm(exponentiate = TRUE)
```

`tidy_vgam` *Tidy a* `vglm` *or a* `vgam` *model*

Description

[Experimental] A tidier for models generated with `VGAM::vglm()` or `VGAM::vgam()`. Term names will be updated to be consistent with generic models. The original term names are preserved in an `"original_term"` column. Depending on the model, additional column `"group"`, `"component"` and/or `"y.level"` may be added to the results.

Usage

```
tidy_vgam(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(`vglm` or `vgam`) A `VGAM::vglm()` or a `VGAM::vgam()` model.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `parameters::model_parameters()`.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_multgee()`, `tidy_parameters()`, `tidy_svy_vglm()`, `tidy_with_broom_or_parameters()`, `tidy_zeroinfl()`

Examples

```
library(VGAM)
mod <- vglm(
  Species ~ Sepal.Length + Sepal.Width,
  family = multinomial(),
  data = iris
)
mod |> tidy_vgam(exponentiate = TRUE)
mod <- vglm(
  Species ~ Sepal.Length + Sepal.Width,
  family = multinomial(parallel = TRUE),
  data = iris
)
mod |> tidy_vgam(exponentiate = TRUE)
```

`tidy_with_broom_or_parameters`

Tidy a model with broom or parameters

Description

Try to tidy a model with `broom::tidy()`. If it fails, will try to tidy the model using `parameters::model_parameters()` through `tidy_parameters()`.

Usage

```
tidy_with_broom_or_parameters(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	(a model object, e.g. `glm`) A model to be tidied.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`...`	Additional parameters passed to `broom::tidy()` or `parameters::model_parameters()`.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_multgee()`, `tidy_parameters()`, `tidy_svy_vglm()`, `tidy_vgam()`, `tidy_zeroinfl()`

`tidy_zeroinfl` *Tidy a* `zeroinfl` *or a* `hurdle` *model*

Description

A tidier for models generated with `pscl::zeroinfl()` or `pscl::hurdle()`. Term names will be updated to be consistent with generic models. The original term names are preserved in an `"original_term"` column.

Usage

```
tidy_zeroinfl(x, conf.int = TRUE, conf.level = 0.95, component = NULL, ...)
```

Arguments

`x`	(`zeroinfl` or `hurdle`) A `pscl::zeroinfl()` or a `pscl::hurdle()` model.
`conf.int`	(`logical`) Whether or not to include a confidence interval in the tidied output.
`conf.level`	(`numeric`) The confidence level to use for the confidence interval (between `0` ans `1`).
`component`	(`string`) `NULL` or one of `"all"`, `"conditional"`, `"zi"`, or `"zero_inflated"`.
`...`	Additional parameters passed to `parameters::model_parameters()`.

See Also

Other custom_tieders: `tidy_broom()`, `tidy_multgee()`, `tidy_parameters()`, `tidy_svy_vglm()`, `tidy_vgam()`, `tidy_with_broom_or_parameters()`

Examples

```
library(pscl)
mod <- zeroinfl(
  art ~ fem + mar + phd,
  data = pscl::bioChemists
)

mod |> tidy_zeroinfl(exponentiate = TRUE)
```

Index