Package 'broom.helpers'

July 6, 2022

Title Helpers for Model Coefficients Tibbles

Version 1.8.0

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/

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

Depends R (>= 3.4)

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

Suggests biglm, biglmm, brms (>= 2.13.0), broom.mixed, cmprsk, covr, datasets, emmeans, fixest (>= 0.10.0), forcats, gam, gee, geepack, ggplot2, glmmTMB, glue, gt, gtsummary, knitr, lavaan, lfe, lme4 (>= 1.1.28), MASS, mgcv, mice, nnet, ordinal, parameters, parsnip, plm, rmarkdown, rstanarm, spelling, survey, survival, testthat, tidycmprsk, VGAM

VignetteBuilder knitr

RdMacros lifecycle

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.2.0

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 2022-07-05 22:40:09 UTC

R topics documented:

.clean_backticks *Remove backticks around variable names*

Description

Remove backticks around variable names

Usage

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

Arguments

x	a character vector to be cleaned
variable_names	list 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	a character vector

See Also

Other other_helpers: `.clean_backticks()`

`.formula_list_to_named_list` *Convert formula selector to a named list*

Description

Functions takes a list of formulas, a named list, or a combination of named elements with formula elements and returns a named list. For example, `list(age = 1, starts_with("stage") ~ 2)`.

Usage

```
.formula_list_to_named_list(
  x,
  data = NULL,
  var_info = NULL,
  arg_name = NULL,
  select_single = FALSE,
  type_check = NULL,
  type_check_msg = NULL,
  null_allowed = TRUE
)
```

Arguments

`x`	list of selecting formulas
`data`	A data frame to select columns from. Default is NULL
`var_info`	A data frame of variable names and attributes. May also pass a character vector of variable names. Default is NULL
`arg_name`	Optional string indicating the source argument name. This helps in the error messaging. Default is NULL.
`select_single`	Logical indicating whether the result must be a single variable. Default is `FALSE`
`type_check`	A predicate function that checks the elements passed on the RHS of the formulas in `x=` (or the element in a named list) satisfy the function.
`type_check_msg`	When the `type_check=` fails, the string provided here will be printed as the error message. When `NULL`, a generic error message will be printed.
`null_allowed`	Are `NULL` values accepted for the right hand side of formulas?

Shortcuts

A shortcut for specifying an option be applied to all columns/variables is omitting the LHS of the formula. For example, `list(~ 1)` is equivalent to passing `list(everything() ~ 1)`.

Additionally, a single formula may be passed instead of placing a single formula in a list; e.g. `everything() ~ 1` is equivalent to passing `list(everything() ~ 1)`

.generic_selector *Generate a custom selector function*

Description

Generate a custom selector function

Usage

```
.generic_selector(variable_column, select_column, select_expr, fun_name)

.is_selector_scoped(variable_column, select_column)
```

Arguments

variable_column	string indicating column variable names are stored
select_column	character vector of columns used in the select_expr= argument
select_expr	unquoted predicate command to subset a data frame to select variables
fun_name	quoted name of function where .generic_selector() is being used. This helps with error messaging.

Details

.is_selector_scoped() checks if a selector has been properly registered in env_variable_type$df_var_info.

Value

custom selector functions

.select_to_varnames *Variable selector*

Description

Function takes select()-like inputs and converts the selector to a character vector of variable names. Functions accepts tidyselect syntax, and additional selector functions defined within the package

Usage

```
.select_to_varnames(
  select,
  data = NULL,
  var_info = NULL,
  arg_name = NULL,
  select_single = FALSE
)
```

Arguments

select	A single object selecting variables, e.g. c(age, stage), starts_with("age")
data	A data frame to select columns from. Default is NULL
var_info	A data frame of variable names and attributes. May also pass a character vector of variable names. Default is NULL
arg_name	Optional string indicating the source argument name. This helps in the error messaging. Default is NULL.
select_single	Logical indicating whether the result must be a single variable. Default is FALSE

Value

A character vector of variable names

---

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

Description

The function 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.

Usage

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

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

Arguments

pkg	Package required
fn	Calling function from the user perspective. Used to write informative error messages.
pkg_search	the package the function will search for a minimum required version from.
boolean	logical indicating 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.

Value

logical or error

Examples

```
.assert_package("broom", boolean = TRUE)

.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

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_frame()`, `model_get_model_matrix()`, `model_get_model()`, `model_get_nlevels()`, `model_get_n()`, `model_get_offset()`, `model_get_response()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`, `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 'model_fit'
model_get_assign(model)
```

Arguments

model	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_frame(), model_get_model_matrix(), model_get_model(), model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), 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), 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 'model_fit'
model_get_coefficients_type(model)
```

Arguments

model	a model object

See Also

Other model_helpers: model_compute_terms_contributions(), model_get_assign(), model_get_contrasts(), model_get_model_frame(), model_get_model_matrix(), model_get_model(), model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), 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), mtcars) %>%
  model_get_coefficients_type()
```

```
Titanic %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes"))) %>%
  glm(Survived ~ Class + Age * Sex, data = ., weights = .$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)
```

Arguments

`model`	a model object

See Also

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

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_nlevels(),
model_get_n(), model_get_offset(), model_get_response(), 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), 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 '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)
```

Arguments

model	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_matrix(), model_get_model(), model_get_nlevels(),
model_get_n(), model_get_offset(), model_get_response(), 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), 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, ...)
```

Arguments

`model`	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.

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_frame(), model_get_model(), model_get_nlevels(),
model_get_n(), model_get_offset(), model_get_response(), 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), 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 '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

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 and exposure time.

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 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_frame(), model_get_model_matrix(), model_get_model(),
model_get_nlevels(), model_get_offset(), model_get_response(), 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_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)
```

Arguments

`model`	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_frame(), model_get_model_matrix(), model_get_model(),
model_get_n(), model_get_offset(), model_get_response(), 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), 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

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_model(),
model_get_nlevels(), model_get_n(), model_get_response(), model_get_terms(), model_get_weights(),
model_get_xlevels(), model_identify_variables(), model_list_contrasts(), 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_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

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_model()`,
`model_get_nlevels()`, `model_get_n()`, `model_get_offset()`, `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_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_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)
```

Arguments

model	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().

See Also

stats::terms()

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_model(),
model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), 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), 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 'model_fit'
model_get_weights(model)
```

Arguments

model	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_model(),
model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), model_get_terms(),
model_get_xlevels(), model_identify_variables(), model_list_contrasts(), 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)
```

Arguments

model	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_model(),
model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), model_get_terms(),
model_get_weights(), model_identify_variables(), model_list_contrasts(), 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)
```

Arguments

`model`	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_frame(), model_get_model_matrix(), model_get_model(),
model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), model_get_terms(),
model_get_weights(), model_get_xlevels(), model_list_contrasts(), model_list_terms_levels(),
model_list_variables()

Examples

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

iris %>%
  lm(
    Sepal.Length ~ poly(Sepal.Width, 2) + Species,
    data = .,
    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

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", "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_frame()`, `model_get_model_matrix()`, `model_get_model()`, `model_get_nlevels()`, `model_get_n()`, `model_get_offset()`, `model_get_response()`, `model_get_terms()`, `model_get_weights()`, `model_get_xlevels()`, `model_identify_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_terms_levels

List levels of categorical terms

Description

Only for categorical variables with treatment, SAS or sum contrasts, and categorical variables with no contrast.

Usage

```
model_list_terms_levels(
  model,
  label_pattern = "{level}",
  variable_labels = NULL
)
```

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

Arguments

model	a model object
label_pattern	a glue pattern for term labels (see examples)
variable_labels	an optional named list or named vector of custom variable labels passed to model_list_variables()

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_frame(), model_get_model_matrix(), model_get_model(),
model_get_nlevels(), model_get_n(), model_get_offset(), model_get_response(), model_get_terms(),
model_get_weights(), model_get_xlevels(), model_identify_variables(), model_list_contrasts(),
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 <- df %>%
  glm(
    Survived ~ Class + Age + Sex,
    data = ., weights = .$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)

## Default S3 method:
model_list_variables(model, labels = NULL, only_variable = FALSE)

## S3 method for class 'lavaan'
model_list_variables(model, labels = NULL, only_variable = FALSE)
```

Arguments

model	a model object
labels	an optional named list or named vector of custom variable labels
only_variable	if TRUE, will return only "variable" column

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`)

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_model()`,
`model_get_nlevels()`, `model_get_n()`, `model_get_offset()`, `model_get_response()`, `model_get_terms()`,
`model_get_weights()`, `model_get_xlevels()`, `model_identify_variables()`, `model_list_contrasts()`,
`model_list_terms_levels()`

Examples

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

iris %>%
  lm(
    Sepal.Length ~ poly(Sepal.Width, 2) + Species,
    data = .,
    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()
```

`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()

all_dichotomous()

all_categorical(dichotomous = TRUE)

all_interaction()

all_ran_pars()

all_ran_vals()

all_intercepts()

all_contrasts(contrasts_type = NULL)
```

Arguments

`dichotomous`	Logical indicating whether to include dichotomous variables. Default is `TRUE`
`contrasts_type`	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", "other")`

Value

A character vector of column names selected

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")))
```

`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
`biglm::bigglm()`	
`biglmm::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
`lavaan::lavaan()`	Limited support for categorical variables
`lfe::felm()`	

`lme4::glmer()`	`broom.mixed` package required
`lme4::glmer.nb()`	`broom.mixed` package required
`lme4::lmer()`	`broom.mixed` package required
`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_plus_plus(mod, tidy_fun = function(x, ..`
`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()`	
`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::clogit()`	
`survival::coxph()`	
`survival::survreg()`	
`tidycmprsk::crr()`	
`VGAM::vglm()`	Limited support. It is recommended to use `tidy_parameters()` 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	a tidy tibble
exponentiate	logical indicating whether or not to exponentiate the coefficient estimates. It should be consistent with the original call to broom::tidy()
model	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_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

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

ex2 <- Titanic %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes"))) %>%
  glm(Survived ~ Class + Age * Sex, data = ., weights = .$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	a tidy tibble
model	the corresponding model, if not attached to x
quiet	logical argument 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_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

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

df %>%
  glm(
    Survived ~ Class + Age + Sex,
    data = ., weights = .$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"),
  model = tidy_get_model(x),
  quiet = FALSE
)
```

Arguments

x	a tidy tibble
exponentiate	logical indicating whether or not to exponentiate the coefficient estimates. It should be consistent with the original call to broom::tidy()
model	the corresponding model, if not attached to x
quiet	logical argument 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 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_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

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

df %>%
  glm(
    Survived ~ Class + Age + Sex,
    data = ., weights = .$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`	a tidy tibble
`show_single_row`	a vector indicating the names of binary variables that should be displayed on a single row. Accepts tidyselect syntax. Default is `NULL`. See also `all_dichotomous()`
`model`	the corresponding model, if not attached to `x`
`quiet`	logical argument whether broom.helpers should not return a message when requested output cannot be generated. Default is FALSE
`strict`	logical argument 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_row
tidy_add_n(), 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

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

res <- df %>%
  glm(
    Survived ~ Class + Age + Sex,
    data = ., weights = .$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`	a tidy tibble
`model`	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. `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 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_rows()`, `tidy_add_header_rows()`, `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

```
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")))

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

df %>%
  glm(
```

```
    Survived ~ Class * (Age : Sex),
    data = ., weights = .$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_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()`) or a sum contrast (`stats::contr.sum()`), add a reference row.

Usage

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

Arguments

`x`	a tidy tibble
`no_reference_row`	a vector indicating the name of variables for those no reference row should be added. Accepts tidyselect syntax. Default is `NULL`. See also `all_categorical()` and `all_dichotomous()`
`model`	the corresponding model, if not attached to `x`
`quiet`	logical argument 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_row`, `tidy_add_header_rows()`, `tidy_add_n()`, `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

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

res <- df %>%
  glm(
    Survived ~ Class + Age + Sex,
    data = ., weights = .$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}",
  model = tidy_get_model(x),
  quiet = FALSE,
  strict = FALSE
)
```

Arguments

`x`	a tidy tibble
`labels`	an optional named list or named vector of custom term labels
`interaction_sep`	separator for interaction terms
`categorical_terms_pattern`	a glue pattern for labels of categorical terms with treatment or sum contrasts (see examples and `model_list_terms_levels()`)
`model`	the corresponding model, if not attached to `x`
`quiet`	logical argument whether broom.helpers should not return a message when requested output cannot be generated. Default is FALSE
`strict`	logical argument 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_rows(), tidy_add_header_rows(), tidy_add_n(), tidy_add_reference_rows(), tidy_add_variable_labels(), tidy_attach_model(), tidy_disambiguate_terms(), 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 <- df %>%
  glm(Survived ~ Class * Age * Sex, data = ., weights = .$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 = " * ",
```

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

Arguments

x	a tidy tibble
labels	an optional named list or named vector of custom variable labels
interaction_sep	separator for interaction terms
model	the corresponding model, if not attached to x
quiet	logical argument whether broom.helpers should not return a message when requested output cannot be generated. Default is FALSE
strict	logical argument whether broom.helpers should return an error when requested output cannot be generated. Default is FALSE

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_rows(), tidy_add_header_rows(), tidy_add_n(), tidy_add_reference_rows(), tidy_add_term_labels(), tidy_attach_model(), tidy_disambiguate_terms(), 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"
  )

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

`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,
  exponentiate = FALSE,
  ...
)

tidy_get_model(x)

tidy_detach_model(x)
```

Arguments

`x`	a tibble of model terms
`model`	a model to be attached/tidied
`.attributes`	named list of additional attributes to be attached to `x`
`tidy_fun`	option to specify a custom tidier function
`exponentiate`	logical indicating 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`
`...`	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_rows()`, `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_disambiguate_terms()`, `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_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`	a tidy tibble
`sep`	character, separator added between group name and term
`model`	the corresponding model, if not attached to `x`
`quiet`	logical argument 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_rows()`, `tidy_add_header_rows()`, `tidy_add_n()`, `tidy_add_reference_rows()`, `tidy_add_term_labels()`, `tidy_add_variable_labels()`, `tidy_attach_model()`, `tidy_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
mod <- lme4::lmer(marker ~ stage + (1|grade) + (death|response), gtsummary::trial)
mod %>% tidy_and_attach() %>% tidy_disambiguate_terms()
```

`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`	a tidy tibble
`model`	the corresponding model, if not attached to `x`
`quiet`	logical argument 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.

See Also

`model_identify_variables()`

Other tidy_helpers: `tidy_add_coefficients_type()`, `tidy_add_contrasts()`, `tidy_add_estimate_to_reference_rows()`, `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_plus_plus()`, `tidy_remove_intercept()`, `tidy_select_variables()`

Examples

```
Titanic %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(Survived = factor(Survived, c("No", "Yes"))) %>%
  glm(Survived ~ Class + Age * Sex, data = ., weights = .$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_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
`conf.int`	logical indicating whether or not to include a confidence interval in the tidied output
`conf.level`	the confidence level to use for the confidence interval
`...`	additional parameters passed to `parameters::model_parameters()`

See Also

Other custom_tieders: `tidy_with_broom_or_parameters()`

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_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_add_coefficients_type()`
- `tidy_detach_model()`

Usage

```
tidy_plus_plus(
  model,
  tidy_fun = tidy_with_broom_or_parameters,
  conf.int = TRUE,
  exponentiate = FALSE,
  variable_labels = NULL,
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  disambiguate_terms = TRUE,
  disambiguate_sep = ".",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  add_estimate_to_reference_rows = TRUE,
  add_header_rows = FALSE,
  show_single_row = NULL,
  add_n = TRUE,
  intercept = FALSE,
  include = everything(),
```

```
  keep_model = FALSE,
  quiet = FALSE,
  strict = FALSE,
  ...
)
```

Arguments

model	a model to be attached/tidied
tidy_fun	option to specify a custom tidier function
conf.int	should confidence intervals be computed? (see broom::tidy())
exponentiate	logical indicating 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.
variable_labels	a named list or a named vector of custom variable labels
term_labels	a named list or a named vector of custom term labels
interaction_sep	separator for interaction terms
categorical_terms_pattern	a glue pattern for labels of categorical terms with treatment or sum contrasts (see model_list_terms_levels())
disambiguate_terms	should terms be disambiguated with tidy_disambiguate_terms()? (default TRUE)
disambiguate_sep	separator for tidy_disambiguate_terms()
add_reference_rows	should reference rows be added?
no_reference_row	variables (accepts tidyselect notation) for those no reference row should be added, when add_reference_rows = TRUE
add_estimate_to_reference_rows	should an estimate value be added to reference rows?
add_header_rows	should header rows be added?
show_single_row	variables that should be displayed on a single row (accepts tidyselect notation), when add_header_rows is TRUE
add_n	should the number of observations be added?
intercept	should the intercept(s) be included?
include	variables to include. Accepts tidyselect syntax. Use - to remove a variable. Default is everything(). See also all_continuous(), all_categorical(), all_dichotomous() and all_interaction()
keep_model	should the model be kept as an attribute of the final result?

quiet	logical argument whether broom.helpers should not return a message when requested output cannot be generated. Default is FALSE
strict	logical argument whether broom.helpers should return an error when requested output cannot be generated. Default is FALSE
...	other arguments passed to tidy_fun()

See Also

Other tidy_helpers: tidy_add_coefficients_type(), tidy_add_contrasts(), tidy_add_estimate_to_reference_row
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_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"
  )
if (interactive()) {
  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`	a tidy tibble
`model`	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_rows()`, `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_identify_variables()`, `tidy_plus_plus()`, `tidy_select_variables()`

Examples

```
Titanic %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(Survived = factor(Survived)) %>%
  glm(Survived ~ Class + Age + Sex, data = ., weights = .$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`	a tidy tibble
`include`	variables to include. Accepts tidyselect syntax. Use `-` to remove a variable. Default is `everything()`. See also `all_continuous()`, `all_categorical()`, `all_dichotomous()` and `all_interaction()`
`model`	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_rows()`, `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_identify_variables()`, `tidy_plus_plus()`, `tidy_remove_intercept()`

Examples

```
res <- Titanic %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(Survived = factor(Survived)) %>%
  glm(Survived ~ Class + Age * Sex, data = ., weights = .$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_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
`conf.int`	logical indicating whether or not to include a confidence interval in the tidied output
`conf.level`	the confidence level to use for the confidence interval
`...`	additional parameters passed to `broom::tidy()` or `parameters::model_parameters()`

See Also

Other custom_tieders: `tidy_parameters()`

Index