Package ‘tabxplor’

September 18, 2026

Title User-Friendly Tables with Color Helpers for Data Exploration

Version 2.0.1

Description Make it easy to deal with multiple cross-tables in data exploration, by creating them, manipulating them, and adding color helpers to highlight deviations (differences from totals, comparisons between lines or columns, contributions to variance, odds ratios, etc.) and significance (confidence intervals, stars, etc.).
Create the same kind of tables for regression models, with a framework to compare model effects with their crude/observed counterpart systematically.
All functions render data frames which can be easily manipulated.
All tables can be exported with formats and colors to 'Excel', html and markdown.

URL https://github.com/BriceNocenti/tabxplor, https://bricenocenti.github.io/tabxplor/

BugReports https://github.com/BriceNocenti/tabxplor/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Suggests fansi (>= 0.5.0), htmltools (>= 0.5.0), jmvcore (>= 2.4.0), knitr, mirai (>= 2.5.0), openxlsx2 (>= 1.0.0), pkgload (>= 1.3.0), parallelly (>= 1.32.0), R6, RhpcBLASctl, ggplot2 (>= 3.5.0), rmarkdown, rstudioapi (>= 0.1), testthat (>= 3.0.0), brant (>= 0.3.0), svyVGAM (>= 1.2), VGAM (>= 1.1.0), marginaleffects (>= 0.20.0), withr (>= 3.0.0), yaml, gridExtra (>= 2.3.0), clipr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first tab-reg, plots, tab-xl, reg-estimand, survey-variance, tab-classes, golden, tab-render-html, reg-assumptions, tab-display, tab, tab-transpose-render, tab-export, tab-palettes, reg-influence, survey-design, jmvtab, tab-export-prep, reg-cross, tab-args

Config/Needs/website pkgdown, BriceNocenti/txtheme

Config/Needs/coverage covr

Imports dplyr (>= 1.0.3), forcats (>= 0.5.0), purrr (>= 0.3.0), rlang (>= 1.0.0), tibble (>= 3.1.0), tidyr (>= 1.1.0), vctrs (>= 0.3.0), cli (>= 2.0.0), tidyselect (>= 1.2.0), pillar (>= 1.6.0), stats (>= 4.0.0), utils, lifecycle (>= 1.0.0), data.table, grid, survey (>= 3.0.0), nnet (>= 7.3.0), MASS (>= 7.3.0)

Depends R (>= 4.1.0)

VignetteBuilder knitr, rmarkdown

Config/potools/style explicit

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Brice Nocenti [aut, cre]

Maintainer Brice Nocenti <brice.nocenti@protonmail.com>

Repository CRAN

Date/Publication 2026-09-18 07:00:02 UTC

Contents

car_arrests . . . 3
car_salaries . . . 4
conf_level_to_z . . . 5
facto_tea . . . 6
fct_recode_helper . . . 7
fmt . . . 8
fmt_attr . . . 16
fmt_attributes . . . 17
fmt_fields . . . 21
fmt_get_color_code . . . 23
forest_plot . . . 24
get_test . . . 26
gss_cat_data_formatting . . . 27
is_tab . . . 28
jmvtab . . . 28
jmvtabreg . . . 33
new_tab . . . 37
questionr_hdv . . . 39
reg_check_plots . . . 40
reg_formulas . . . 42
reg_measures . . . 43
score_from_lv1 . . . 44
set_bars . . . 45
set_caption . . . 46
set_color_palette . . . 47
set_footer_tabs . . . 51

set_legend_words . . . 52
set_subtext . . . 53
shape_numeric_var . . . 54
tab . . . 56
tabxplor-base-coercion . . . 62
tabxplor-display . . . 64
tabxplor-footer . . . 66
tabxplor-options . . . 68
tabxplor-type . . . 72
tab_columns . . . 73
tab_compact . . . 74
tab_counts . . . 75
tab_css . . . 77
tab_export . . . 80
tab_footer_text . . . 81
tab_html . . . 82
tab_many . . . 85
tab_md . . . 87
tab_note . . . 89
tab_num . . . 90
tab_parallel_stop . . . 92
tab_pipe . . . 92
tab_plain . . . 93
tab_reg . . . 94
tab_spread . . . 104
tab_structure . . . 105
tab_transpose . . . 106
tab_wrap_text . . . 107
tab_xl . . . 108

Index **112**

`car_arrests` *Marijuana-possession arrests in Toronto, 1997-2002*

Description

5 226 people arrested for possession of a small quantity of marijuana. The outcome is whether the person was **released with a summons** rather than held. This is the running example of the *All else equal* article: 86 % of white arrestees were released against 74 % of black arrestees, and the article asks what survives of that gap when people alike on everything else are compared.

Usage

```
car_arrests
```

Format

A tibble of 5 226 rows and 8 columns.

released Released with a summons? "Yes" first – it is what the article studies.

colour The arrestee's race as the police recorded it: "White" first, then "Black".

year 1997 to 2002.

age Age in years.

sex Female or Male.

employed Employed? "Yes" first.

citizen A Canadian citizen? "Yes" first.

checks On how many of six police databases the person's name already appeared, 0 to 6.

Source

The complete `Arrests` data of the **carData** package (John Fox, Sanford Weisberg and Brad Price), GPL (>= 2) – with thanks; gathered by Michael Friendly for a series in the *Toronto Star*. tabxplor's copy changes only the level order: the yes/no items read "Yes" first, and `colour` starts at "White".

Examples

```
tab(car_arrests, colour, released, pct = "row", color = "difference")
```

---

car_salaries *Salaries of US college professors, 2008-09*

Description

Nine months' salary for 397 professors at one US college, collected by the institution to monitor a pay gap. Used in the *Reading a regression* article to show a gap that **grows** under adjustment, and then a **mediator**: rank explains the gap away, but rank is itself part of what is unequal.

Usage

```
car_salaries
```

Format

A tibble of 397 rows and 7 columns.

rank AsstProf, AssocProf, then Prof.

discipline "A" (theoretical) or "B" (applied).

yrs.since.phd Years since the PhD.

yrs.service Years of service.

sex Female or Male.

salary Nine-month salary, in US dollars.

is_prof Full professor or not: the same information as `rank`, asked as the yes/no question a percentage can answer. Added by tabxplor.

Source

The complete `Salaries` data of the **carData** package (John Fox, Sanford Weisberg and Brad Price), GPL (>= 2) – with thanks. tabxplor's copy adds `is_prof` and orders `rank` from assistant to full professor.

Examples

```
tab_reg(car_salaries, "salary", c("sex", "discipline"))
```

---

conf_level_to_z *Convert confidence levels into z thresholds*

Description

Turn one or several confidence levels into the two-sided normal (z) thresholds they correspond to, rounded for readability. It is a convenience for writing the `zscore` color break scale (`set_color_breaks`) in the vocabulary you already use elsewhere — confidence levels — instead of remembering that 95 % is 1.96. The scale itself always stores plain z magnitudes, so `conf_level_to_z(0.95)` and `1.96` are strictly interchangeable.

Usage

```
conf_level_to_z(conf_level, digits = 2)
```

Arguments

`conf_level`	A numeric vector of confidence levels, each between 0 and 1 (e.g. `c(0.95, 0.99)`).
`digits`	Number of digits to round to (default 2). Rounding keeps color legends readable (`"+1.96"` rather than `"+1.959964"`); pass `Inf` for the exact values.

Value

A numeric vector of positive z thresholds, the same length as `conf_level`.

Examples

```
conf_level_to_z(c(0.95, 0.99))

# the default `zscore` break scale (color = "contrib", color_signif = "guaranteed_effect")
conf_level_to_z(c(0.95, 0.99, 0.9999, 1 - 2e-9))


set_color_breaks(zscore = conf_level_to_z(c(0.95, 0.999)))
set_color_breaks(zscore = c(2, 3, 4, 6))  # or plain z values, identically
```

`facto_tea` *Tea drinkers: when, where and why they drink it*

Description

A survey of 300 tea drinkers, used here for its three **batteries of yes/no items** – the shape a multiple-answer question ("which of these apply to you?") arrives in. Six items say *when* people drink tea, six say *where*, and fourteen say what they think it does for them. See `vignette("tabxplor")` for what a battery does in a table, and `score_from_lv1()` for turning one into a single summed score.

Usage

```
facto_tea
```

Format

A tibble of 300 rows and 36 columns. The ones the vignettes use:

breakfast, tea.time, evening, lunch, dinner, always *When* do you drink tea?

home, work, tearoom, friends, resto, pub *Where* do you drink tea?

Sport Do you play a sport?

SPC Socio-professional category, 7 levels.

sex F or M.

The other columns describe the tea itself (`Tea`, `How`, `sugar`, `how`, `where`, `price`, `frequency`), the drinker (`age`, `age_Q`), and what they associate tea with (`healthy`, `relaxing`, `exciting`, `slimming`, and ten more). Every two-level item reads "yes" first.

Source

The complete `tea` data of the **FactoMineR** package (Francois Husson, Julie Josse, Sebastien Le and Jeremy Mazet), GPL (>= 2) – with thanks. tabxplor's copy changes only the level order: in each yes/no item the "yes" answer comes first, and its label loses the separator dot the original spells it with (`"Not.tea time"` becomes `"Not tea time"`).

Examples

```
tab(facto_tea, SPC, c(breakfast, evening), pct = "row", levels = "first", na = "drop")
```

fct_recode_helper *Write the code to recode several factors*

Description

Recoding a factor with `forcats::fct_recode()` means typing every level name exactly, and a typo is silently ignored. `fct_recode_helper()` writes that code for you: it prints a ready-to-paste `mutate()` call with one `fct_recode()` per variable, each level already written as `"level" = "level"`. You then only edit the new names on the left, and delete the lines you keep.

With a few variables, each level carries its frequency and count as a comment, which is what tells you which small levels to merge. A column with a `label` attribute (data imported by **haven**) gets that label as a comment title.

Usage

```
fct_recode_helper(
  data,
  .cols = -where(is.numeric),
  name_in,
  name_out,
  freq = NULL,
  style = c("mutate", "base"),
  reminder = TRUE,
  cat = TRUE
)
```

Arguments

`data`	The data frame.
`.cols`	<tidy-select> The variables to recode. Default: every non-numeric column.
`name_in`	The input data frame's name (default: the expression given as `data`).
`name_out`	The output data frame's name, if different from `name_in` (used by `style = "base"`).
`freq`	Print each level's frequency and count as a comment; defaults to `TRUE` when 5 or fewer variables are given.
`style`	`"mutate"` (default) writes a `dplyr::mutate()` call; `"base"` writes `data$var <-`.
`reminder`	Print a `"new" = "old"` syntax reminder. Default `TRUE`.
`cat`	Print to console, or open a temporary file when there are more than 5 variables; `FALSE` returns a data frame of the recode text instead.

Value

With `cat = TRUE` (default), the text printed to console (or written to a temp R file for more than 5 variables), returned invisibly. With `cat = FALSE`, a `tibble` of the recode text.

Examples

```
fct_recode_helper(forcats::gss_cat, c(marital, race))
```

---

fmt *Create an* fmt *vector, the tabxplor cell*

Description

fmt vectors, of class tabxplor_fmt, powers **tabxplor** and tab tibbles. As a record, they stores all data necessary to calculate percentages, Chi2 metadata or confidence intervals, but also to format and color the table to help the user read it. You can access this data with vctrs::field, or change it with vctrs:field<-. Its per-cell **fields** are listed below. The other arguments are **attributes**, attached not to each value but to the whole vector, like scale, col_var, totcol or color. You can get them with attr and modify them with attr<-. Special functions listed below are made to facilitate programming with with **tabxplor** formatted numbers. taxplfmt vectors can use all standard operations, like +, -, sum(), or c(), using **vctrs**.

Usage

```
fmt(
  n = integer(),
  scale = "level_n",
  digits = rep(0L, length(n)),
  display = est_default_display(scale[1]),
  wn = rep(NA_real_, length(n)),
  pct = rep(NA_real_, length(n)),
  mean = rep(NA_real_, length(n)),
  diff = rep(NA_real_, length(n)),
  ratio = rep(NA_real_, length(n)),
  ctr = rep(NA_real_, length(n)),
  var = rep(NA_real_, length(n)),
  ci = rep(NA_real_, length(n)),
  ci_inf = rep(NA_real_, length(n)),
  ci_sup = rep(NA_real_, length(n)),
  pvalue = rep(NA_real_, length(n)),
  or = rep(NA_real_, length(n)),
  tot_n = rep(NA_real_, length(n)),
  n_eff = rep(NA_real_, length(n)),
  obs = rep(NA_real_, length(n)),
  gap_se = rep(NA_real_, length(n)),
  row_kind = rep("data", length(n)),
  in_tottab = rep(FALSE, length(n)),
  in_refrow = rep(FALSE, length(n)),
  in_totrow = NULL,
  comp_all = NA,
  ref = "",
```

```
  pct_type = "none",
  col_var = "",
  col_group = "",
  totcol = FALSE,
  refcol = FALSE,
  color = "",
  color_signif = "ignore",
  model_family = "",
  role = "",
  conf_level = NA_real_,
  degf = NA_real_,
  basis = "n",
  ci_method = "",
  ...
)

is_fmt(x)
```

Arguments

`n` The underlying count, as an integer vector of length `n()`. It is used to calculate confidence intervals.

`scale` What the column estimates, as a single string (an attribute, not a field): one key into the declared library of estimate scales. It says which field holds the estimate, what its null value is, whether the scale is additive or multiplicative, and which colour ladder it reads.

- `"level_n"`: counts
- `"level_pct"`: percentages (`pct_type` says of what)
- `"level_mean"`: means (from numeric variables)
- `"points"`: a difference between two percentages, in percentage points
- `"mean_diff"`: a difference between two means, in the outcome's own units
- `"raw_diff"`: a regression coefficient / marginal effect in the outcome's units
- `"pct_ratio"`, `"mean_ratio"`: the ratio of two percentages / two means
- `"odds_ratio"`: a multiplicative effect (odds ratio, risk ratio, ratio of means)
- `"log_coef"`: a link-scale coefficient (a log-odds, a log-rate)
- `"mixed"`: what binding columns of unlike scales collapses to

`digits` The number of digits, as an integer, or an integer vector the length of `n`.

`display` The display type : the name of the field you want to show when printing the vector, as a single string or a character vector the length of `n`. Every accepted value is listed in *Every display token* below; a named layout or a `{}` template combining several (e.g. `"\{pct\} (n=\{n\})"`) is also accepted — see tabxplor-display.

`wn` The underlying weighted counts, as a double vector the length of `n`. It is used in certain operations on `fmt`, like means.

`pct` The percentages, as a double vector the length of `n`. Calculate with `tab_pct`.

`mean` The means, as a double vector the length of `n`.

`diff` The differences (from totals or first cells), as a double vector the length of `n`. Used to set colors for means and row or col percentages. Built by `tab`.

`ratio` The ratio to the reference (relative risk for percentages, mean ratio for means), as a double vector the length of `n`.

`ctr` The contributions of cells to (sub)tables variances, as a double vector the length of `n`. Used to print colors when `color = "contrib"`. The mean contribution of each (sub)table is written on total rows (then, colors don't print well without total rows). Built by `tab`. The cell's adjusted standardized residual is not a field of its own: it is recovered from `pvalue` and this field's sign, and readable with `display = "resid"` (see `tab`).

`var` The cells variances, as a double vector the length of `n`. Used with `scale = "level_mean"` to calculate confidence intervals.

`ci` The confidence interval half-width (margin of error), as a double vector the length of `n`. Kept for backward compatibility: it is stored as the symmetric bounds `ci_inf`/`ci_sup` and read back by `get_ci()`.

`ci_inf`, `ci_sup` The lower and upper bounds of the confidence interval, as double vectors the length of `n`. Built by `tab`.

`pvalue` The per-cell significance p-value, as a double vector the length of `n`.

`or` The odds ratio (for a 3+ level variable, the OR of each level versus the reference), as a double vector the length of `n`.

`tot_n` The cell's own (unweighted) percentage base, as a double vector the length of `n`.

`n_eff` The effective sample size used for this cell's confidence interval, `p(1-p) / Var_design(p)` (a mean: `s^2 / Var_design(mean)`): from `survey::svyrecvar` under a `survey::svydesign`, from the closed-form flat-design variance when the weighted basis is asked for (`tab(design_effect = TRUE)`), else `NA` (the CI falls back to the raw unweighted base). It records *the base that was used*: a finite value where the design or weights corrected it, `NA` where nothing did, and the **raw count** where a correction was asked for but this cell could not carry one. Populated for descriptive cells (a crosstab/mean cell, a `tab_reg` `Obs_*` column whose interval came from a closed form); a coefficient column, and any column whose interval came from a fit instead, carry none. A double vector the length of `n`. Non-displayed.

`obs` The value this cell's estimate is COMPARED TO by the `tab_reg` colour measures `"adjustment"` / `"between_groups"`, on the cell's own scale: the observed (crude) effect beside a model effect, or – under `tab_vars` with `color = "between_groups"` – the reference group's estimate. `NA` on cross-tables and wherever there is no counterpart (leaving those cells uncoloured). A double vector the length of `n`; displayable as `display = "\{obs\}"`.

`gap_se` The standard error of the GAP between this cell's estimate and `obs`, on the estimate's own test scale. Written by `tab_reg` where the two estimates are independent (`tab_vars` groups), so `color = "between_groups"` can honour `color_signif`; `NA` elsewhere. A double vector the length of `n`. Non-displayed.

`row_kind` What kind of row the cell sits in — one of `"data"` (an ordinary body row), `"total"`, and the synthetic display rows `"n"`, `"pct"`, `"pvalue"`, `"gof"`, `"blank"`. A character vector the length of `n`. It supersedes the logical `in_totrow` field, kept as a soft-deprecated argument and read-only `$in_totrow`.

`in_tottab` TRUE when the cell is part of a total table

`in_refrow` TRUE when the cell is part of a reference row (cf. `ref`)

`in_totrow` **[Deprecated]** Use `row_kind = "total"`.

`comp_all` FALSE when the comparison level is the subtable/group, TRUE when it is the whole table

`ref` The type of difference of the vector. Cf. `tab`.

`pct_type` For a percentage column, what the percentage is a percentage OF, and hence which axis its reference lies on (as a single string): `"row"`, `"col"`, `"all"` (frequencies by subtable / group, i.e. by `tab_vars`), `"all_tabs"` (frequencies for the whole table), or `"none"` (counts, means, coefficients).

`col_var` The name of the `col_var` used to calculate the vector

`col_group` The sub-population this column's block belongs to: a level of a `spread_vars` variable (`tab_spread`), or a `tab_reg` `tab_vars` group. `""` (the default) when the table was never spread. Together with `col_var` it identifies a column BLOCK: two blocks may show the same variable for two sub-populations, and exports head them on two lines.

`totcol` TRUE when the vector is a total column

`refcol` TRUE when the vector is a reference column

`color` The colour measure, as a single string — how a cell's value is compared to colour it (significance is handled separately by `color_signif`):

- `"no"`: no colors are printed.
- `"diff"` (`"difference"`): the cell's difference from the reference (a total, or the first cell when `ref = "first"`) — percentage points for factors, a standardized difference for means.
- `"ratio"`: the ratio to the reference (relative risk for percentages, mean ratio).
- `"or"` (`"odds_ratio"`): the odds ratio, for row/col percentages.
- `"contrib"`: the cell's contribution to the table's variance. Under `color_signif = "guaranteed_effect"` it switches to the absolute adjusted standardized residual — see `tab`.
- `"adjustment"` / `"between_groups"`: the two `tab_reg` measures, which compare a cell to *another column* rather than to a reference row. A hand-built column may carry them, provided it fills the `obs` field they score.

The value is **validated and normalised**: every accepted spelling — the discipline's acronyms included (`"RD"`, `"RR"`, `"RoM"`, `"OR"` and their lowercase twins) — is stored as its canonical measure name, and an unknown one is an error. The tabxplor 1.x combined strings `"diff_ci"` / `"after_ci"` still work but are superseded by the `color` + `color_signif` pair; here they resolve to their *measure* half only, so pass the significance policy through `color_signif`.

`color_signif` How significance gates the color, as a single string (`"ignore"` / `"grey_non_signif"` / `"guaranteed_effect"`). See `tab`.

`model_family` For regression tables (`tab_reg`): the column's model family (`"binomial"`, `"gaussian"`, `"poisson"`, `"multinomial"`, `"ordinal"`), as a single string. Empty (`""`) on cross-tables. Lets a table mix several outcomes with different families, each column keeping its own effect wording.

role — For regression tables (tab_reg): the column's role, "model" for a model-estimate column or "emp" for an empirical (crude) companion column. Empty ("") on cross-tables. Read by the colour legend to name each column's effect without matching its label.

conf_level — The confidence level this column's interval and thresholds were computed at, as a single number in (0, 1). NA (default) means "unknown" — the colour engine then falls back to options("tabxplor.conf_level"). Stored per COLUMN, because colours are resolved per column at print time and cannot see the table's conf_level argument.

degf — The degrees of freedom this column's interval is referred to. On a cross-table that is the survey design's #PSU - #strata, which matters below ~30 primary sampling units; on a regression it is the fitted model's own residual df (for an svyglm, degf + 1 - p), so a model column and its observed companion legitimately differ. NA (default) means "refer to the normal quantile".

basis — How this column's interval and significance were computed — "n" (the raw sample size), "weights" (the design effect of the weights), "design" (a full survey design), or "design_partial" (a design was given but its variance could not be computed). Default "n". A per-COLUMN fact, so a table states honestly what its numbers carry even after a pipeline drops the table's metadata; binding columns keeps the WEAKEST basis.

ci_method — Which interval ENGINE built this column's bounds — "wilson", "wald", "beta" (a cell proportion), "newcombe", "ac" (a difference of proportions), "katz" (a ratio of proportions), "welch", "student", "ols" (a difference of means), "robust", "quasipoisson", "poisson" (a ratio of means), "woolf", "wald_log", "profile"; "" (default) when the column carries no interval. Read back by the colour legend, so it always names the method the bounds were built with.

... — In fmt(), it exists only for the arguments retired in tabxplor 2.0.0: type is translated into scale + pct_type (see tabxplor-type), ci_type gets an error naming its replacement. In the accessor methods below, to add arguments in the future.

x — The object to test, to get a field in, or to modify.

Value

A vector of class tabxplor_fmt.

A logical vector.

Functions

- is_fmt(): a test function for class fmt.

The fields of a cell

A fmt cell carries 21 fields. Many are NA when the quantity was not requested; read one with x$field or vctrs::field(), and see them all with vctrs::vec_data():

- n — the unweighted count.

- display — which field this cell shows (a bare name, or a {} template).
- digits — how many decimals this cell prints.
- wn — the weighted count.
- pct — the percentage.
- mean — the mean, on a numeric column variable.
- diff — the difference from the reference cell (percentage points, or the outcome's own units).
- ratio — the ratio to the reference cell (a relative risk, or a ratio of means).
- ctr — the cell's contribution to the table's Chi-2.
- var — the column's variance quantity – which one is given by its scale.
- ci_inf — the lower bound of the confidence interval.
- ci_sup — the upper bound of the confidence interval.
- pvalue — the cell's own significance p-value, which the stars read.
- or — the odds ratio against the ref2 level.
- tot_n — the cell's own base — the count its percentage is computed on.
- n_eff — the effective sample size its interval was computed on (weights or a survey design).
- obs — tab_reg() only: the observed (crude) effect the modelled one is compared to.
- gap_se — tab_reg() only: the standard error of the gap between the estimate and obs.
- row_kind — what kind of row the cell sits in — see get_row_kind().
- in_tottab — is the cell in a total table (logical).
- in_refrow — is the cell in a reference row (logical).

Every display token

Generated from the package's own display table, so it cannot drift from what get_num() reads. Each of pct, n, wn, mean, diff, ratio, or, ctr, var, obs, pvalue shows the field of the same name, described above. The rest are composed or derived by the pipeline itself, and the last few are not meant to be typed:

- est — the estimate, whatever this column estimates — an odds ratio, a risk difference, a coefficient, a percentage. The one token that means the same thing on every table.
- base — the level the estimate sits on: the percentage, the mean or the count. On a plain percentage table it is the same number as est; beside a regression effect it is the adjusted prediction.
- ci — the confidence interval of whatever the column compares, as [low;high].
- moe — the margin of error — the same interval as ci, written as the half-width +/-x around the estimate. Void where the column compares a RATIO: a ratio's interval is symmetric on the LOG scale, so it has no half-width.
- odds — the odds — pct / (1 - pct), the quantity an odds ratio is a ratio of. Printed on the odds ratio's own ladder, so a cell below 1 reads its inverse ("1/2.60") unless options(tabxplor.ratio_print = "raw") asks for the plain number. Void where the percentage is 1 and the odds infinite.
- sd — the standard deviation, in the variable's own unit.

- cv — the coefficient of variation — the standard deviation as a percentage of the mean.
- resid — the adjusted standardized residual – whether the cell departs from independence. Derived from the p-value and the sign of ctr, so it is read-only.
- coef — the estimate on the model's LINK scale — the coefficient a linear or log-link model fitted. The same number as est where the column is already additive, its logarithm where the column shows a ratio.
- gap — how far adjustment moved the effect: the gap between the modelled estimate and its observed counterpart, on the estimate's own scale. What color = "adjustment" grades — readable in print and Excel, not only in an html tooltip.
- gof — a model-fit statistic (N, R2, AIC, BIC, dispersion).
- gof_warn — a model-fit statistic past the threshold its check is read against.
- n_range — the unweighted base: one count, or a min-max range over the table.
- blank — nothing: a cell masked by n_min.
- rr — the legacy synonym of ratio, still accepted.
- OR — the acronym spelling of or, still accepted.

See Also

tabxplor-display for the {} grammar and the named layouts display accepts; fmt_fields and fmt_attributes for the accessors.

Examples

```
library(dplyr)

f <- fmt(n = c(7, 19, 2), pct = c(0.25, 0.679, 0.07),
         scale = "level_pct", pct_type = "row")
f

# To get the currently displayed field :
get_num(f)

# To modify the currently displayed field :
set_num(f, c(1, 0, 0))


# See all the underlying fields of a fmt vector (a data frame with a number of rows
#  equal to the length of the vector) :
vctrs::vec_data(f)

# To get the numbers of digits :
vctrs::field(f, "digits")
f$digits

# To get the count :
vctrs::field(f, "n")
f$n
```

```
# To get the display :
vctrs::field(f, "display")
f$display

# To modify a field, you can use `dplyr::mutate` on the fmt vector,
# referring to the names of the columns of the underlying data.frame (`vctrs::vec_data`) :
vctrs::`field<-`(f, "pct", c(1, 0, 0))
mutate(f, pct = c(1, 0, 0))

# See all the attributes of a fmt vector :
attributes(f)

# To modify the "pct_type" attribute of a fmt vector (what the percentage is a percentage OF) :
set_pct_type(f, "col")

# To modify the "color" attribute of a fmt vector :
set_color(f, "contrib")




tabs <- tab(starwars, sex, hair_color, gender, na = "drop", pct = "row",
            other_if_less_than = 5)

# To identify the total columns, and work with them :
is_totcol(tabs)
tabs |> mutate(across(where(is_totcol), ~ "total column"))

# To identify the total rows, and work with them :
is_totrow(tabs)
tabs |>
  mutate(across(
    where(is_fmt),
    ~ if_else(is_totrow(.), true = "into_total_row", false = "normal_cell")
  ))

# To identify the total tables, and work with them :
tottabs <- is_tottab(tabs)
tabs |> tibble::add_column(tottabs) |>
  mutate(total = if_else(tottabs, "part of a total table", "normal cell"))

# To access the displayed numbers, as numeric vectors :
tabs |> mutate(across(where(is_fmt), get_num))

# To access the displayed numbers, as character vectors (without colors) :
tabs |> mutate(across(where(is_fmt), format))

# To access the (non-displayed) differences of the cells percentages from totals :
tabs |> mutate(across(where(is_fmt), ~ vctrs::field(., "diff")))


# To do more complex operations, like creating a new column with standard deviation and
# print it with 2 decimals, use `dplyr::mutate` on all the fmt columns of a table :
```

```
tab(forcats::gss_cat, race, c(age, tvhours), marital, digits = 1L, comp = "all",
    color = "auto") |>
  dplyr::mutate(dplyr::across( #Mutate over the whole table.
    c(age, tvhours),
    ~ dplyr::mutate(.,         #Mutate over each fmt vector's underlying data.frame.
                    var     = sqrt(var),
                    display = "var",
                    digits  = 2L) |>
      set_color("no"),
    .names = "{.col}_sd"
  ))
```

`fmt_attr` *Read or write one* `fmt` *column attribute, by name*

Description

The generic form of the `get_*()` / `set_*()` family: one function covering every per-column attribute a `tabxplor_fmt` vector carries, so a helper can loop over them instead of naming each. The named accessors (`get_scale()`, `get_col_var()`, `is_totcol()`, ...) remain the readable way to address one known attribute.

Usage

```
fmt_attr(x, name)

fmt_attr(x, name) <- value
```

Arguments

`x`	A `tabxplor_fmt` vector, or a data.frame (then every `fmt` column is read).
`name`	The attribute: one of `"scale"`, `"comp_all"`, `"ref"`, `"pct_type"`, `"col_var"`, `"col_group"`, `"totcol"`, `"refcol"`, `"color"`, `"color_signif"`, `"model_family"`, `"role"`, `"conf_level"`, `"degf"`, `"basis"`, `"ci_method"`. An unknown name is an error naming the set.
`value`	The new value. Written through the attribute's own setter, so it is validated exactly as `set_scale()` and friends validate it.

Value

The stored value (its declared default when the attribute is unset). On a data.frame, one entry per `fmt` column, named. Writing returns the modified vector.

See Also

`tab_columns()` for every column's attributes at once; `fmt()` for what each one means.

Examples

```
x <- fmt(n = c(10, 20), pct = c(0.3, 0.7), scale = "level_pct", pct_type = "row")
fmt_attr(x, "scale")
fmt_attr(x, "col_var") <- "region"
fmt_attr(x, "col_var")
```

`fmt_attributes` *Per-column attributes of a* `fmt` *vector*

Description

Read and write the facts that hold for a whole column: what it estimates, which percentage base it rests on, which reference it is compared to, how it is coloured, and how its confidence interval was built. They are stored on the vector, not on the table, so a column keeps them when it is extracted, renamed or piped through `dplyr`.

`fmt_attr()` reaches any of them by name — these are the readable way to address one you know. `tab_columns()` shows them all, for every column of a table at once.

Usage

```
get_scale(x, ...)

set_scale(x, scale)

get_pct_type(x, ...)

set_pct_type(x, pct_type)

is_totcol(x, ...)

as_totcol(x, totcol = TRUE)

get_comp_all(x, replace_na = TRUE)

set_comp_all(x, comp_all = FALSE)

get_ref_type(x, ...)

set_ref_type(x, ref)

set_diff_type(x, ref)

get_col_var(x, ...)

set_col_var(x, col_var)
```

```
get_col_group(x, ...)

get_model_family(x, ...)

set_model_family(x, model_family)

get_ci_method(x, ...)

is_refcol(x, ...)

as_refcol(x, refcol = TRUE)

get_color(x, ...)

get_color_bg(x, ...)

get_color_signif(x, ...)

set_color(x, color)

set_color_signif(x, color_signif)
```

Arguments

`x` The object to test, to get a field in, or to modify.

`...` In `fmt()`, it exists only for the arguments retired in tabxplor 2.0.0: `type` is translated into `scale` + `pct_type` (see `tabxplor-type`), `ci_type` gets an error naming its replacement. In the accessor methods below, to add arguments in the future.

`scale` What the column estimates, as a single string (an attribute, not a field): one key into the declared library of estimate scales. It says which field holds the estimate, what its null value is, whether the scale is additive or multiplicative, and which colour ladder it reads.

- `"level_n"`: counts
- `"level_pct"`: percentages (`pct_type` says of what)
- `"level_mean"`: means (from numeric variables)
- `"points"`: a difference between two percentages, in percentage points
- `"mean_diff"`: a difference between two means, in the outcome's own units
- `"raw_diff"`: a regression coefficient / marginal effect in the outcome's units
- `"pct_ratio"`, `"mean_ratio"`: the ratio of two percentages / two means
- `"odds_ratio"`: a multiplicative effect (odds ratio, risk ratio, ratio of means)
- `"log_coef"`: a link-scale coefficient (a log-odds, a log-rate)
- `"mixed"`: what binding columns of unlike scales collapses to

`pct_type` For a percentage column, what the percentage is a percentage OF, and hence which axis its reference lies on (as a single string): `"row"`, `"col"`, `"all"` (fre-

quencies by subtable / group, i.e. by tab_vars), "all_tabs" (frequencies for the whole table), or "none" (counts, means, coefficients).

totcol — TRUE when the vector is a total column

replace_na — By default, get_comp_all takes NA in comparison level to be a FALSE (=comparison at subtables/groups level). Set to FALSE to avoid this behavior.

comp_all — FALSE when the comparison level is the subtable/group, TRUE when it is the whole table

ref — The type of difference of the vector. Cf. tab.

col_var — The name of the col_var used to calculate the vector

model_family — For regression tables (tab_reg): the column's model family ("binomial", "gaussian", "poisson", "multinomial", "ordinal"), as a single string. Empty ("") on cross-tables. Lets a table mix several outcomes with different families, each column keeping its own effect wording.

refcol — TRUE when the vector is a reference column

color — The colour measure, as a single string — how a cell's value is compared to colour it (significance is handled separately by color_signif):

- "no": no colors are printed.
- "diff" ("difference"): the cell's difference from the reference (a total, or the first cell when ref = "first") — percentage points for factors, a standardized difference for means.
- "ratio": the ratio to the reference (relative risk for percentages, mean ratio).
- "or" ("odds_ratio"): the odds ratio, for row/col percentages.
- "contrib": the cell's contribution to the table's variance. Under color_signif = "guaranteed_effect" it switches to the absolute adjusted standardized residual — see tab.
- "adjustment" / "between_groups": the two tab_reg measures, which compare a cell to *another column* rather than to a reference row. A hand-built column may carry them, provided it fills the obs field they score.

The value is **validated and normalised**: every accepted spelling — the discipline's acronyms included ("RD", "RR", "RoM", "OR" and their lowercase twins) — is stored as its canonical measure name, and an unknown one is an error. The tabxplor 1.x combined strings "diff_ci" / "after_ci" still work but are superseded by the color + color_signif pair; here they resolve to their *measure* half only, so pass the significance policy through color_signif.

color_signif — How significance gates the color, as a single string ("ignore" / "grey_non_signif" / "guaranteed_effect"). See tab.

Value

A getter returns the stored value — its declared default where the attribute was never set ("" for a name, FALSE for a flag, NA where there is none) — and answers once per fmt column when given a data.frame. A setter returns the modified fmt vector.

Functions

- `get_scale()`: get the estimate scale of fmt columns (at `fmt` level or `tab` level)
- `set_scale()`: set the estimate scale attribute of a `fmt` vector
- `get_pct_type()`: get which kind of percentage fmt columns hold (at `fmt` level or `tab` level)
- `set_pct_type()`: set the percentage-type attribute of a `fmt` vector
- `is_totcol()`: test function for total columns (at `fmt` level or `tab` level)
- `as_totcol()`: set the "totcol" attribute of a `fmt` vector
- `get_comp_all()`: get comparison level of fmt columns
- `set_comp_all()`: set the comparison level attribute of a `fmt` vector
- `get_ref_type()`: get differences type of fmt columns (at `fmt` level or `tab` level)
- `set_ref_type()`: set the reference attribute of a `fmt` vector — which row or column a comparison is made against. It is the writer of the attribute `get_ref_type()` reads.
- `set_diff_type()`: **[Deprecated]** Use `set_ref_type()`, which shares its stem with the getter `get_ref_type()` and with the `ref` attribute both of them address.
- `get_col_var()`: get names of column variable of fmt columns (at `fmt` level or `tab` level)
- `set_col_var()`: set the "col_var" attribute of a `fmt` vector
- `get_col_group()`: get the sub-population of fmt columns (at `fmt` level or `tab` level)
- `get_model_family()`: get the regression model family of fmt columns (at `fmt` or `tab` level)
- `set_model_family()`: set the "model_family" attribute of a `fmt` vector (the per-column regression family, "" on crosstabs)
- `get_ci_method()`: get the interval method of fmt columns (at `fmt` level or `tab` level)
- `is_refcol()`: test function for reference columns (at `fmt` level or `tab` level)
- `as_refcol()`: set the "ref_col" attribute of a `fmt` vector
- `get_color()`: get color (at `fmt` level or `tab` level)
- `get_color_bg()`: get the background-channel color measure (`NA` when there is none)
- `get_color_signif()`: get the significance policy (`"ignore"` / `"grey_non_signif"` / `"guaranteed_effect"`)
- `set_color()`: set the "color" attribute of a `fmt` vector
- `set_color_signif()`: set the significance policy attribute of a `fmt` vector

See Also

`fmt_attr()` to address an attribute by name; `tab_columns()` for a whole table; `fmt()` for what every attribute means; fmt_fields for the per-cell values.

fmt_fields — *Per-cell fields of a* `fmt` *vector*

Description

Read and write the values that vary from cell to cell: the displayed number, its decimals, its p-value, and the kind of row each cell sits in. Every `fmt` vector carries all of them, an inapplicable one stored as `NA` — so these always answer, even on a column where the field means nothing.

Use them on a single `fmt` vector or, through `dplyr::across(where(is_fmt), ...)`, on a whole table. To reach a field these do not name, use `x$<field>` or `vctrs::field()`.

Usage

```
get_num(x)

set_num(x, value)

is_totrow(x, ...)

get_row_kind(x)

set_row_kind(x, row_kind)

as_totrow(x, in_totrow = TRUE)

is_tottab(x, ...)

as_tottab(x, in_tottab = TRUE)

set_display(x, value)

is_refrow(x, ...)

as_refrow(x, in_refrow = TRUE)

get_digits(x)

get_pvalue(x)

set_digits(x, value)

set_pvalue(x, value)
```

Arguments

`x`	The object to test, to get a field in, or to modify.

value	The value you want to inject in some fmt vector's vctrs::field or attribute using a given "set" function.
...	In fmt(), it exists only for the arguments retired in tabxplor 2.0.0: type is translated into scale + pct_type (see tabxplor-type), ci_type gets an error naming its replacement. In the accessor methods below, to add arguments in the future.
row_kind	The kind of row a cell sits in (see get_row_kind).
in_totrow	**[Deprecated]** Use row_kind = "total".
in_tottab	TRUE when the cell is part of a total table
in_refrow	TRUE when the cell is part of a reference row (cf. ref)

Value

A getter returns a vector the length of x; a setter the modified fmt vector. Given a data.frame, a getter answers once per fmt column.

Functions

- get_num(): get the currently displayed field
- set_num(): set the currently displayed field (not changing display type)
- is_totrow(): test function to detect cells in total rows (at fmt level or tab level)
- get_row_kind(): get the "row_kind" field: what kind of row each cell sits in (see the *Row kinds* section).
- set_row_kind(): set the "row_kind" field
- as_totrow(): set the "total" row kind (belong to total row)
- is_tottab(): test function to detect cells in total tables (at fmt level or tab level)
- as_tottab(): set the "in_tottab" field (belong to total table)
- set_display(): set the "display" vctrs::field of a fmt vector, or of all of them in the whole tibble.
- is_refrow(): test function to detect cells in reference rows (at fmt level or tab level)
- as_refrow(): set the "in_refrow" field (belong to reference row)
- get_digits(): get the "digits" field
- get_pvalue(): get the per-cell p-value (what the significance stars read)
- set_digits(): set the "digits" field
- set_pvalue(): set the per-cell p-value. set_pvalue(x, NA_real_) is how a duplicated, purely descriptive copy of a column loses its stars: the stored p-value is their only source.

Row kinds

The row_kind field says what kind of row a cell sits in.

"data" a real body row.

"total" a total row (the only kind is_totrow() asks about).

"n" the base-count row.

"pct" the add_pct percentage row.

"pvalue" an appended test row: a crosstab's chi2 or F, a regression's per-term test.

"gof" a regression goodness-of-fit / model-summary footer row.

"blank" a spacer row between footer blocks.

See Also

fmt() for what every field means and how to build a cell; fmt_attributes for the per-column facts; vignette("tabxplor-programming").

---

fmt_get_color_code *The html color code of a* fmt *vector*

Description

The html color code of a fmt vector

Usage

```
fmt_get_color_code(x, type = "text", theme = "light", ...)
```

Arguments

x	The fmt vector to get the html color codes from.
type	The style type, "text" to color the text, "bg" to color the background.
theme	Is your console or html table background "light" or "dark" ? Default to the current setting (RStudio theme when detectable, else "light").
...	Absorbs deprecated arguments (e.g. html_24_bit); ignored.

Value

A character vector with html color codes, of the length of the initial vector.

Examples

```
tabs <- tab(forcats::gss_cat, race, marital, pct = "row", color = "difference")
dplyr::mutate(tabs, across(where(is_fmt), fmt_get_color_code))
```

forest_plot *Forest plot of any tabxplor table*

Description

Draws every estimate of a table with its confidence interval, its significance and its colour – for a cross-table from tab as much as for a regression table from tab_reg. It **reads the table and never re-fits anything**: every number and every colour comes from the cell it was printed from, so the figure and the table cannot disagree. (Its sibling reg_check_plots is the opposite: model checks *always* re-fit, because they are about residuals, which no table carries.)

Usage

```
forest_plot(
  x,
  columns = NULL,
  what = c("auto", "effect", "level"),
  observed = c("auto", "band", "point", "ci", "none"),
  center = c("n", "estimate", "none"),
  display = NULL,
  layout = c("keep", "auto", "transpose"),
  facet = NULL,
  color = TRUE,
  guide = c("gridlines", "bands", "none"),
  intercept = FALSE,
  totals = FALSE,
  offset = 0.25,
  label_offset = 0.3,
  max_size = 6,
  footer = c("short", "full", "none"),
  footer_width = 130L,
  legend = "auto",
  theme = NULL,
  lang = NULL,
  caption = NULL,
  subtext = TRUE,
  return_data = FALSE,
  ...
)
```

Arguments

x — A table made with tab or tab_reg, or a list of tab. A list of tables sharing the same col_vars (and no tab_vars) is merged into one; any other list — several row_vars and/or tab_vars — is rendered one table after another, each keeping its own sub-tables.

columns	Value columns to draw, by name. NULL (the default) draws the model columns of a regression table and every value column of a cross-table.
what	"auto" (the quantity the table's own interval is centred on), "effect" (the contrast: difference, ratio or odds ratio) or "level" (the percentage or mean – for a regression table this needs effect = "marginal").
observed	For a regression table with empirical = TRUE: "auto", "band" (the observed value with the margin of error of the gap), "point", "ci" (the classic two-interval figure) or "none".
center	What marks the estimate: "n" (the default) a square whose area is the level's own base, with the value printed just above it; "estimate" the value alone; "none" a constant square and no value, for a plot with many panels.
display	What that value prints – a {} display template, as set_display takes ("\{est\} (\{base\})", "est_ci", ...). NULL (the default) prints the cell's own primary token.
layout	Which axis is read and which is faceted: "keep" (the default) reads the table's rows, "transpose" reads its columns, "auto" picks whichever has more levels (more legible whenever the table is much wider than tall). A regression table is never transposed.
facet	NULL for one panel per estimate column, FALSE for a single panel.
color	Set to FALSE for a plain plot with no colour measure.
guide	"gridlines" (the default), "bands" (shade the panel between the colour breaks – the teaching mode, which makes a cell's colour and its position one statement) or "none".
intercept	Draw the regression Constant row.
totals	Draw total rows and total columns.
offset	How far below the estimate the observed value sits, as a fraction of a row; raise it for a tall figure with few rows. Under an adjustment colour the arrow takes this row and the observed value drops one further.
label_offset	How far above the estimate its value is printed, as a fraction of a row.
max_size	Area of the largest marker, when center = "n" maps the base to it.
footer	"short" (the default) the console's own footer, "full" the exports' longer one, or "none". Both are wrapped and set flush left.
footer_width	Characters per footer line, since a ggplot caption does not wrap on its own. Use a larger number for a wide figure, smaller for a narrow one.
legend	Where the colour legend goes: "auto" (the bottom), "right", "left", "top" or "none". When several ladders apply it cannot be a guide and goes to the caption instead.
theme	"light", "dark" or one of the black-and-white publication palettes ("print_ready" and friends – a mark then reads its magnitude off a grey ramp). NULL follows getOption("tabxplor.export_theme").
lang	Colour-legend language: NULL (auto from the R/OS locale, English fallback), "en" or "fr".
caption	A caption. NULL keeps the table's own.

subtext	Include the table's subtext and footer lines in the caption.
return_data	Return the long estimate tibble instead of the plot.
...	Retired arguments, accepted and ignored with a deprecation message since 2.0.0. Anything else is an error naming the argument you meant, as it already was in tab().

Details

What is drawn. Always a **deviation**: the effect a regression estimates, or, for a cross-table, the comparison its color = grades. The **level** it sits on (the percentage, the mean, the adjusted probability) is printed above each whisker instead, so position and number say two different things; what = "level" swaps them.

The gridlines are the table's colour ladder (set_color_breaks), labelled with the same glyphs as the footer and continued as far as the data goes. The whisker takes the colour of its cell whole, so significance is read off it and there are no stars.

A table that mixes units (an odds ratio beside a mean difference) gets one axis per panel, each in its own transform, with panels measuring the same thing sharing one comparable range.

The observed comparison. With empirical = TRUE, a regression estimate carries its crude counterpart. observed = "band" (the default when testable) draws a bracket at plus-or-minus the margin of error of the difference: the modelled point falls outside it exactly when the gap test rejects. Two correlated intervals should not be compared by overlap, which is why the crude one is not drawn by default; observed = "ci" restores it.

Value

A ggplot (or a list of them), ready for + theme() and ggsave().

See Also

reg_check_plots for the model checks, tab_export to export the table itself.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  t <- tab(forcats::gss_cat, race, marital, pct = "row", ci = "ref", color = TRUE)
  forest_plot(t)
  forest_plot(t, guide = "bands")          # the teaching mode
}
```

---

get_test *Read a table's statistical tests*

Description

The tests a table carries and prints under itself: a crosstab's chi-squared or ANOVA, a regression's model-fit statistics and global tests. `get_test()` hands them back as a tidy tibble — one row per test, keyed by `var` (the row variable, the predictor, or `""` for the whole table) and `col` (the column variable it keys under) — so a test can be filtered, reshaped or reported like any other data. A new kind of test is new rows, never new columns.

The remaining columns name the statistic (`test`, `statistic`, `df1`, `df2`, `pvalue`), the base it was computed on (`n`, `min_e`, `deff`) and its effect size (`effect_size`, `es_type`). Per-CELL contributions to the chi-squared are not here: they are the `ctr` field of the cells themselves (`tabs$Total$ctr`).

Usage

```
get_test(x)
```

Arguments

`x`	A `tabxplor_tab`.

Value

A tibble of tests — empty, with the same columns, when the table ran none (build them with `tab(test = TRUE)`); `NULL` only when `x` has lost its attributes.

See Also

`tab()` for `test =`, `tab_structure()` and `tab_columns()` for the rest of a table's metadata.

Examples

```
tabs <- tab(forcats::gss_cat, race, marital, test = TRUE)
get_test(tabs)
```

---

gss_cat_data_formatting

A General Social Survey extract, formatted for cross-tables

Description

`forcats::gss_cat` with levels merged into readable groups, and each variable's first level chosen as the reference the colors and the models compare everything else to. The data set most examples and vignettes are built on.

Usage

```
gss_cat_data_formatting()
```

Value

A tibble of 21483 rows: the US General Social Survey, 2000-2014.

is_tab — *Is this a tabxplor table?*

Description

TRUE for a table built by tab, tab_reg or any of their variants — i.e. for a tabxplor_tab (a tabxplor_grouped_tab with tab_vars). tab_structure answers the fuller question: what structure is it, and what can be done with it.

Usage

```
is_tab(x)
```

Arguments

x	An object to test.

Value

A single logical.

See Also

tab_structure(), tab_get_vars().

jmvtab — *Crosstables*

Description

Crosstables

Usage

```
jmvtab(
  data,
  row_vars = NULL,
  col_vars = NULL,
  tab_vars = NULL,
  wt = NULL,
  pct = "no",
  color = "no",
  color_signif = "ignore",
  test = FALSE,
  anova = "welch",
  na = "keep",
```

```
  lvs = "all",
  cleannames = TRUE,
  ref_levels = NULL,
  levels_order = NULL,
  levels_collapse = NULL,
  shape = NULL,
  ref = "auto",
  ref2 = "first",
  comp = "tab",
  ci = "auto",
  conf_level = 0.95,
  stars = FALSE,
  design_effect = FALSE,
  ci_method_cell = "wilson",
  ci_method_diff = "newcombe",
  ci_method_mean_diff = "welch",
  ci_method_mean_ratio = "robust",
  tab_theme = "light",
  totaltab = "line",
  wrap_rows = 35,
  wrap_cols = 15,
  display = "auto",
  n = "range",
  add_pct = FALSE,
  subtext = "",
  digits = "0",
  n_min = 0,
  export_format = "excel",
  exportExcel = FALSE,
  export_dir = "~/Documents",
  export_filename = "Table",
  resetPath = FALSE,
  xl_replace = FALSE
)
```

Arguments

data	A data.frame.
row_vars	The row variable, which will be printed with one level per line. If numeric, it will be converted to factor. If several row variables are provided, it's not possible to add any tab_vars.
col_vars	One column is printed for each level of each column variable. For numeric variables means are calculated, in a single column.
tab_vars	One subtable is made for each combination of levels of the tab variables. All tab variables are converted to factor. Leave empty to make a simple table. Not used when there are several row_vars.
wt	A weight variable, of class numeric. Leave empty for unweighted results.

`pct` The type of percentages to calculate :

- `"row"`: row percentages.
- `"col"`: column percentages.
- `"all"`: frequencies for each subtable/group, if there is `tab_vars`.
- `"all_tabs"`: frequencies for the whole (set of) table(s).

`color` Which measure to use for color helpers, as a single string. The values are the measure names `tab()` itself takes (the short spellings `"diff"` / `"OR"` stay valid aliases in R).

- `"no"`: by default, no colors are printed.
- `"auto"`: a smart per-column-type default (percentage-point difference on the text plus a relative-risk highlight on the background for factors, mean ratio for numeric columns).
- `"difference"`: color the difference of each cell from its total (or reference cell). For factors this is a percentage-point difference; for numeric columns the standardized (SD-scaled) mean difference.
- `"ratio"`: color the relative risk (factors) or mean ratio (numeric).
- `"odds_ratio"`: for `pct == "col"` or `pct == "row"`, color based on odds ratios. To PRINT them, set `display` (the colour and the printed quantity are two questions).
- `"contrib"`: color cells based on their contribution to variance (factor columns only).

How significance gates these colors is set separately by `color_signif`.

`color_signif` How statistical significance gates the colors, as a single string.

- `"ignore"`: by default, color every deviation by its observed size.
- `"grey_non_signif"`: color by observed size, but grey out cells whose deviation is not significant (at `conf_level`). A confidence interval on the difference is computed automatically.
- `"guaranteed_effect"`: color by the guaranteed (confidence-bound) effect – all cells whose interval clears the threshold show, with dimmer colors.

`test` Set to `TRUE` to add a test p-value row: a Chi-square test for categorical column variables and an ANOVA F-test for numeric ones (chosen automatically per column type). Also enables colouring cells by their contribution to variance.

`anova` Which F statistic to display for numeric column variables when the test is on: Welch's F (default, does not assume equal variances) or the classic pooled F.

`na` The policy to adopt with missing values. It must be a single string.

- `na = "keep"`: by default, prints `NA`'s as explicit `"NA"` level.
- `na = "drop"`: each column variable drops its own `NA`'s, so tables made with different column variables may have a different number of observations.
- `na = "drop_all"`: drops every observation missing on the row variable, any column variable or a tab variable (all columns then share one base).
- `na = "common_base"`: fixes a single population (non-missing on the row variable and the first column variable), while secondary column variables keep their own `NA`'s. Reproduces the historical `tab()` behaviour.

`lvs` The levels of `col_vars` to keep.

- `"all"`: by default, all levels are kept.
- `"first"`: only keep the first level of each `col_vars`
- `"auto"`: keep the first level when `col_var` is only two levels, keep all levels otherwise.

`cleannames` By default, clean levels names, by removing prefix numbers like "1-", and text in parenthesis. Set to `FALSE` to avoid this behaviour.

`ref_levels` .

`levels_order` .

`levels_collapse` .

`shape` .

`ref` The reference cell to calculate differences and ratios (used to print `colors`) :

- `"auto"`: by default, cell difference from the corresponding total (rows or cols depending on `pct = "row"` or `pct = "col"`) is used for `diff` ; cell ratio from the first line (or col) is use for `OR` (odds ratio/relative risks ratio).
- `"tot"`: totals are always used.
- `"first"`: calculate cell difference or ratio from the first cell of the row or column (useful to color temporal developments).
- `n`: when `ref` is an integer, the nth row (or column) is used for comparison.
- `"regex"`: when `ref` is a string, it it used as a regular expression, to match with the names of the rows (or columns). Be precise enough to match only one column or row, otherwise you get a warning message.
- `"no"`: not use ref and not calculate diffs to gain calculation time.

`ref2` With `OR = OR` (odds-ratios) and a 3+ levels factor, a second reference cell is needed to calculate relative risks ratios. First cell by default.

`comp` The comparison level : by subtables/groups, or for the whole table.

`ci` **What the confidence interval is anchored on** – one question, four answers. The GEOMETRY of the interval is not asked here: it follows the comparison the table makes (set by `color` / `display`), so a difference table gets a difference interval and a ratio table a ratio one.

- `"auto"`: build the comparison interval when something reads it (`stars`, or a `color_signif` policy), and none otherwise.
- `"no"`: no interval at all.
- `"cell"`: each cell's own interval (a percentage / a mean with its own bounds). It anchors nothing to compare, so `stars` and `color_signif` are informed and switched off.
- `"ref"`: the interval of the comparison with the reference cell – what `stars` and `color_signif` read.

`conf_level` The confidence level, a single numeric between 0 and 1 — 0.95 by default.

`stars` With `ci = "diff"`, print significance stars (`* ** ***`) for the difference of each cell from its reference. Read from the same confidence interval that is displayed, so stars and bracket always agree.

`design_effect` For a WEIGHTED table, make the confidence intervals, the significance stars, the colour thresholds AND the p-values account for the unequal weighting (the exact flat survey-design variance) instead of using the raw number of respondents. Sets options(tabxplor.design_effect). Off by default; it moves every interval in the table, not only the p-value.

`ci_method_cell` The proportion confidence-interval method for `ci = "cell"`: `"wilson"` (the score interval, default) or `"wald"` (the normal approximation).

`ci_method_diff` The proportion confidence-interval method for `ci = "diff"`. `"newcombe"` (default) is the dual of the two-proportion score test, so the interval and the significance stars always agree.

`ci_method_mean_diff`
The confidence-interval method for the difference of numeric means (means with `ci = "diff"`): Welch (default), Student (the two groups pooled) or OLS (pooled over every level of the variable, i.e. the interval a linear model gives that coefficient).

`ci_method_mean_ratio`
The confidence-interval method for a ratio of numeric means (means with `ci = "ratio"`).

`tab_theme` How the table is painted, in the results panel and in every export. `"light"` is the colour palette; `"print_ready"` says the same thing typographically — bold, italics, underlines and marks instead of blue and red — for a page that has no colour. See `tab_css`.

`totaltab` The total table, if there are subtables/groups (i.e. when `tab_vars` is provided). Vectorised over `row_vars`.

- `"line"`: by default, add a general total line (necessary for calculations with `comp = "all"`)
- `"table"`: add a complete total table (i.e. `row_var` by `col_vars` without `tab_vars`).
- `"no"`: not to draw any total table.

`wrap_rows` By default, rownames are wrapped when larger than 30 characters.

`wrap_cols` By default, colnames are wrapped when larger than 12 characters.

`display` What each cell shows. Every value here is a `tab(display =)` value: a bare field name, a named layout such as `"base_ci"` (each value with its interval), or a `{}` template combining fields. `"auto"` keeps whatever the table was built with. A template naming a field the table does not carry renders empty and says which argument would fill it.

`n` How many people the table is about: `"range"` prints the unweighted base beside the Total cell (as `min-max` when the column variables rest on different people), `"min"` the smallest base only, `"no"` no count at all.

`add_pct` Set to `TRUE` to add a column with the frequencies of the row variable (for `pct = "row"`) or a row with the frequencies of the column variable (for `pct = "col"`)

`subtext` A character vector to print rows of legend under the table.

`digits` The number of digits to print, as a single integer (0-6). In R, `tab()` also accepts an integer vector the same length as `col_vars`.

n_min	A pure display filter (0 = off). A row is dropped only when its largest base across the column variables is below n_min; surviving cells whose own base is below n_min are blanked. Under pct = "col" weak columns are dropped instead. Totals, the added-n row/column and the p-value line are always kept. Recomputes nothing.
export_format	The export file format: Excel (.xlsx), HTML (.html) or Markdown (.md).
exportExcel	Press to export the table to the chosen format (the button label follows the format).
export_dir	The folder to save the exported file in. Blank or ~/Documents auto-detects your real Documents folder (a redirected D:/Documents or network Documents included). Type any other folder to override; a leading ~ there expands to your home folder.
export_filename	The bare file name, with NO extension (the chosen format adds it). Illegal characters are removed automatically. Blank saves as "Table".
resetPath	Reset the folder and file name to their defaults (your Documents folder and "Table").
xl_replace	"Set to TRUE to overwrite an existing file."

Value

A results object containing:

results$html_table	a html
results$cache_state	an image

---

jmvtabreg *Regressions*

Description

Regressions

Usage

```
jmvtabreg(
  data,
  outcome = NULL,
  predictors = NULL,
  tab_vars = NULL,
  wt = NULL,
  family = NULL,
  link = NULL,
  outcome_level = NULL,
```

```
  trials = NULL,
  effect = "auto",
  measure = "auto",
  empirical = TRUE,
  models = NULL,
  na = "drop_by_outcome",
  run_compare = FALSE,
  levels_order = NULL,
  levels_collapse = NULL,
  crosses = NULL,
  ref_levels = NULL,
  shape = NULL,
  multiplier = NULL,
  conf_level = 0.95,
  ci_method = "wald",
  stars = TRUE,
  color = "measure",
  color_signif = "grey_non_signif",
  display = "auto",
  n = "range",
  digits = "0",
  cleannames = TRUE,
  subtext = "",
  tab_theme = "light",
  wrap_rows = 35,
  wrap_cols = 15,
  export_format = "excel",
  exportExcel = FALSE,
  export_dir = "~/Documents",
  export_filename = "Reg_model",
  resetPath = FALSE,
  xl_check = FALSE,
  xl_replace = FALSE
)
```

Arguments

data	A data.frame.
outcome	The outcome variable(s). One model is built per outcome. Set each outcome's family (and, for a binomial outcome, its modelled level or number of trials) in the Model table.
predictors	The explanatory variables of the model. Factors are shown one line per level (the reference level as the neutral value); numeric predictors as a single line.
tab_vars	A grouping variable. The same model is fitted within each of its levels and the tables are stacked (like tab_vars for crosstables).
wt	A survey weight variable. Switches to design-based estimation (scale-invariant sandwich standard errors). Leave empty for unweighted results.

`family` .

`link` WHICH MEASURE THE MODEL ESTIMATES – the only argument that changes the model. A link IS a measure, so it takes the same words as `measure`. Chosen per outcome, in the Model table beside `family`: `"auto"` is the family's own (a logistic regression for a binary outcome, a linear one for a quantity, a Poisson one for a count), and `"odds_ratio"` / `"ratio"` / `"difference"` name the model whose coefficient IS that measure – on a binary outcome, the logistic fit, the modified Poisson (Zou 2004) and the identity-link additive-risk one. The picker only ever offers the links the chosen family can be fitted on.

`outcome_level` .

`trials` .

`effect` WHERE THE NUMBER COMES FROM, once the model and the reported measure are fixed.

- `"auto"`: the coefficients when the reported measure IS the model's own, the model's predictions otherwise. Nobody needs to change this.
- `"conditional"`: read off the model's own coefficients.
- `"marginal"`: worked out from the model's predictions for every observed person, then averaged.
- `"at_reference"`: the same, at one profile (every other predictor at its reference level / mean).

`measure` WHICH MEASURE IS REPORTED. It never changes the model: where it is not the measure the model estimates (see `link`), it is worked out from the model's predictions.

- `"auto"`: the model's own measure – follow from the left. On a prediction route it steps back to the outcome's own (a percentage reads as "x times as likely"), because a marginal odds ratio is a specialist quantity, asked for by name.
- `"odds_ratio"` / `"ratio"` / `"difference"`: the named measure, when the outcome's level can carry it. One it cannot says so, and lists what it does offer.
- `"raw_coefficient"`: the model's own coefficient, un-transformed — the log of the reported measure wherever that measure is multiplicative, and the additive estimate itself on a model that is already additive.

`empirical` Show the crude, unadjusted, single-predictor effect beside each model effect — the bivariate association that IS the modelised quantity when there is a single predictor, so the gap between the two is what adjustment changed.

`models` .

`na` "drop_by_outcome" (default) fits every model OF ONE OUTCOME on the same complete cases, which is what makes the observed columns comparable to the model beside them and lets the likelihood-ratio comparison run; "drop_by_model" gives each model its own complete cases (a model on a different population then gets no observed effect); "drop_all" shares one population across every outcome as well.

`run_compare` .

`levels_order` .
`levels_collapse`
.

`crosses` .

`ref_levels` .

`shape` .

`multiplier` .

`conf_level` The confidence level for intervals and the significance stars.

`ci_method` Wald intervals (also the only option for weighted models) or profile-likelihood intervals (unweighted binomial / poisson only; needs MASS). A profile interval is an output of the likelihood at one confidence level, so it cannot be cached: every change refits the models.

`stars` Show per-cell significance stars (the colours read the confidence interval either way).

`color` WHAT the effect cells are coloured by. The colour LADDER always comes from what the column estimates (an odds ratio is read on the odds-ratio scale, a beta on the standardized-difference one), so what is left to choose is what the estimate is compared TO.

- `"measure"`: the effect's own size (compared to no effect).
- `"no"`: no colours.
- `"adjustment"`: how far the ADJUSTED effect moved from the crude one – needs `empirical`.
- `"between_groups"`: how far each group's effect is from the first group's – needs `tab_vars`.

`color_signif` How significance interacts with the colours: observed size + grey out non-significant cells, colour only the guaranteed (error-adjusted) effect, ignore significance.

`display` The estimate-cell LAYOUT (never the estimand: a display may fold in another quantity of the SAME fit, it can never change the fit). The same named layouts `tab()` offers, written in the same `{}` grammar: `est` is whatever the column estimates and `base` the level it sits on (an adjusted probability, an adjusted mean). `"auto"` keeps the built-in layout.

`n` The column giving the number of observations behind each predictor level: `"range"` prints `min-max` when several models were fitted on different people, `"min"` the smallest count only, `"no"` no column at all.

`digits` The minimum number of digits to print, as a single integer (0-6): each measure keeps its own precision where that is finer (an odds ratio reads at two decimals, a mean score at one). In R, `tab_reg()` also names one display field at a time, `digits = c(ratio = 3)`.

`cleannames` Strip numeric prefixes from factor level labels.

`subtext` A free note printed below the table.

`tab_theme` How the table is painted, in the results panel and in every export. `"light"` is the colour palette; `"print_ready"` says the same thing typographically — bold, italics, underlines and marks instead of blue and red — for a page that has no colour. See `tab_css`.

wrap_rows	.
wrap_cols	.
export_format	.
exportExcel	Press to export the table to the chosen format (the button label follows the format).
export_dir	The folder to save the exported file in. Blank or ~/Documents auto-detects your real Documents folder (a redirected D:/Documents or network Documents included). Type any other folder to override; a leading ~ there expands to your home folder.
export_filename	The bare file name, with NO extension (the chosen format adds it).
resetPath	Reset the folder and file name to their defaults (your Documents folder and "Regression").
xl_check	Excel export only: draw the model-check plots (reg_check_plots()) under each table in the workbook — the panels that apply to the fitted family (tab_xl(check = "auto")). Needs ggplot2 and gridExtra; without them the export says so and writes the table alone.
xl_replace	"Set to TRUE to overwrite an existing file."

Value

A results object containing:

results$html_table	a html
results$cache_state	an image
results$compare_state	an image

---

new_tab *Build a* tabxplor_tab *around* fmt *columns*

Description

Build a tabxplor_tab around fmt columns

Usage

```
new_tab(
  tabs = tibble::tibble(),
  subtext = "",
  test = new_test_tibble(),
  chi2 = NULL,
  meta = NULL,
  ...,
```

```
  class = character()
)

new_grouped_tab(
  tabs = tibble::tibble(),
  groups,
  subtext = "",
  test = new_test_tibble(),
  chi2 = NULL,
  meta = NULL,
  ...,
  class = character()
)
```

Arguments

`tabs` A table, stored into a `tibble` data.frame. It is generally made with `tab`, `tab_reg` or `tab_plain`.

`subtext` The footer's text, as a template: one element per line, every `<placeholder>` **tabxplor** generates and every line you write, in the order they print. See `set_subtext`.

`test` A tidy tibble storing whole-table test results (Chi2 for factor columns, ANOVA F for mean columns), filled by `tab_chi2`.

`chi2` **[Deprecated]** Soft-deprecated alias of `test`.

`meta` The table's metadata, as a single named list gathering (all optional, `NULL` when unset):

- `spec` — the table's identity, `list(kind =, vars =, call =)`: its `kind` (`"crosstab"` or `"regression"`); `vars`, what no column can carry (the weight, the caption, the outcomes, the variable labels — see `set_caption`), the rest of the variable model being derived from the declared index columns and from the columns' own `col_var`; and `call`, the producer's own recipe (a regression's model record — family, outcome, predictors, reference level, and the `fit_spec` `reg_check_plots` refits from).
- `render_extras` — display-only intent for the base count and the `add_pct` companion, `list(n =, add_pct =)`, materialised at print/export time rather than baked into the table.
- `empirical_tips` — multinomial crude-companion tooltip data (a `tibble` keyed by column, predictor and level), set by `tab_reg(empirical = TRUE)`.
- `assumptions` — one record PER OUTCOME, keyed by it, each holding the observed curve of every continuous predictor (weighted quantile bins of the outcome on the family's link scale, one block per `tab_vars` group), set by `tab_reg`: the data behind the shape table under the footer, and behind `reg_check_plots`'s linearity panel.
- `color_breaks` — a per-table override of the colour break scales, set by `tab(color_breaks =)` and merged over the global option (`set_color_breaks`) at render time.
- `legend_words` — what this table's colour legend CALLS each measure — naming only, never a number (see `set_legend_words`).

- `footer_tabs` — the tables and notes rendered UNDER this one by every medium, set by `set_footer_tabs`: a `tabxplor_tab` renders as a table, any other data.frame as a grey note (`tab_note`). In the console they print ABOVE the table, so the last thing printed is the object you can go on to pipe. A footer table's own are never rendered.
- `bars` — the columns drawn as data bars in html and in Excel, each named with the ceiling a full bar means (`NA` = the column's own largest data cell, so that one reference serves the whole column — `set_bars`).

`meta` sub-fields left `NULL` are dropped, so a table given nothing carries no attribute.

`...` Needed to implement subclasses.

`class` Needed to implement subclasses.

`groups` The grouping data.

Value

A `tibble` of class `tabxplor_tab`.

A `tibble` of class `tabxplor_grouped_tab`.

`questionr_hdv` *Histoire de vie: leisure, work and beliefs in France, 2003*

Description

A French national survey, used to show what **adjustment** does to a relationship: going to the cinema is strongly patterned by occupation, and also by age – and the two are entangled. See the *Reading a regression* article for the analysis. It also carries a real sampling weight (`poids`), so it is the data set to try `wt =` on; see `vignette("tabxplor-weights")`.

Usage

```
questionr_hdv
```

Format

A tibble of 2 000 rows and 20 columns. The ones the article uses:

cinema Went to the cinema in the last 12 months? "Oui" first.

qualif Occupational qualification, 7 levels, "Cadre" (senior professional) first. 347 values are missing, as in the original.

age Age in years, 18 to 97.

poids The survey's own sampling weight.

The rest describe the respondent (`sexe`, `nivetud`, `occup`, `freres.soeurs`), their views (`clso`, `relig`, `trav.imp`, `trav.satisf`) and six more leisure activities (`sport`, `cuisine`, `bricol`, `lecture.bd`, `peche.chasse`, `hard.rock`), plus `heures.tv` and `id`.

Source

The complete `hdv2003` data of the **questionr** package (Julien Barnier, Francois Briatte and Joseph Larmarange), GPL (>= 2) – with thanks. It comes from the *Histoire de vie* survey run in 2003 by INSEE, the French national statistics institute. tabxplor's copy changes only the level order: every "Oui"/"Non" item reads "Oui" first, and `qualif` starts at "Cadre".

Examples

```
tab(questionr_hdv, qualif, cinema, pct = "row", na = "drop", color = "difference")
```

---

reg_check_plots *Diagnostic plots of a regression model*

Description

A teaching companion, not a decision tool. Every verdict these panels illustrate is already a row in the table's own footer, for every model column, with no plotting package installed (see the `stats` argument of `tab_reg()`). This function exists to show what a violation looks like.

One call diagnoses every model in the table: one titled grid per model, drawing the panels its own family allows. Pass a `tab_reg()` table — the data it was built from is usually found on its own — or a fitted model directly.

Usage

```
reg_check_plots(
  x,
  data = NULL,
  check = "auto",
  predictors = NULL,
  ncol = NULL,
  facet_ncol = NULL,
  theme = NULL,
  lang = NULL,
  max_points = 2000L,
  nbins = 10L,
  conf = 0.95,
  seed = 20260810,
  ...
)
```

Arguments

`x`	A `tab_reg()` table, or a fitted model (`lm` / `glm` / `svyglm` / `polr` / `multinom` / `svyolr`).

data	The data frame or survey::svydesign the table was built from. **Usually unnecessary**: a table records the name it was called with, and when that name still holds data of the same size, it is used — otherwise the call stops rather than draw the wrong model. Give data explicitly when the table was built from an expression rather than a named object (tab_reg(gss |> dplyr::filter(...), ...)), or when the name has since changed. Ignored with a bare model.
check	Which panels to draw. "auto" (default) draws the panels that *decide* something the footer cannot say in one number — linearity, residuals, normality, influence, and proportionality for an ordinal outcome. "all" adds dispersion and collinearity, whose footer row is normally enough. Or name them: any of "linearity", "residuals", "normality", "dispersion", "influence", "collinearity", "proportionality" — the same words the footer rows and tab_reg()'s stats argument use.
predictors	Optional: restrict the linearity panel to these continuous predictors.
ncol	Number of panel columns in the assembled grid (default: as square as it can be, 3 at most).
facet_ncol	Number of facet columns *inside* a panel (default: 2 for linearity, 4 for proportionality).
theme	"light", "dark", or a black-and-white publication palette ("print_ready" and friends). Defaults to options("tabxplor.theme"), like the table exporters.
lang	Language of the titles and captions ("en", "fr", ...). Defaults to options("tabxplor.lang").
max_points	Thin the raw-point layers to about this many observations; statistics and verdicts are always computed on the full data.
nbins	Bins of the linearity panel's observed curve (default 10).
conf	Confidence level of the Q-Q band. Default 0.95.
seed	Seed of the randomised quantile residuals (NULL for a fresh draw each time).
...	Unused, for future extension.

Value

Invisibly, the assembled gtable — or, with several models, the named list of them, one per model, all drawn on the current graphics device.

See Also

tab_reg() and its stats argument (the same checks as footer rows), and forest_plot() for the RESULTS – its opposite contract: it reads the finished table and never re-fits, where a model check always must.

Examples

```
# \donttest: building a multi-panel ggplot grid costs a few seconds of CPU.

d <- forcats::gss_cat |>
  dplyr::mutate(married = factor(dplyr::if_else(marital == "Married",
                                                "Married", "Not married")))
```

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("gridExtra", quietly = TRUE)) {
  t <- tab_reg(d, "married", c("race", "age"), family = "binomial")
  reg_check_plots(t)
}
```

`reg_formulas` *The model formulas a regression table fitted*

Description

Shows the formula behind every column of a `tab_reg()` table — exactly what reached `stats::glm()`, `survey::svyglm()`, `nnet::multinom()` or `MASS::polr()`. Use it to check what a `shape =`, a `trials =` or a model formula really built.

Usage

```
reg_formulas(x)
```

Arguments

`x`	A table built by `tab_reg()`.

Details

One row per model: several `outcomes` give one each, a `predictors` list one per model. Two things the list does not repeat: under `tab_vars` the same formula is fitted **within each group**, and `color = "between_groups"` (or `stats = "group_interaction"`) fits one extra pooled model for the footer test only.

A summed score (`trials =`) is fitted on a success / failure pair, so its formula names the two internal columns tabxplor builds for it (`.gb_succ`, `.gb_fail`, and `.gb_trials` in the offset of the risk-ratio link).

The formula names the columns as the user wrote them, but a continuous predictor is fitted **anchored** at its `ref` (its mean by default), and a `shape =` may have recoded it — neither changes any effect, only what the Constant row means.

Value

A tibble with one row per model: `model` (its name in the table), `outcome`, `family` (the outcome family), `link` (the measure that model estimates — the word `link =` takes), `fit` (the R call it was fitted with) and `formula`. A `svyglm()` row also means robust (Huber-White) standard errors: survey's design-based variance IS the sandwich, which is why a ratio or a difference on a binary outcome is fitted through it.

See Also

`tab_reg()`, `reg_measures()` (what an outcome can be modelled as).

Examples

```
d <- forcats::gss_cat
d$married <- as.integer(d$marital == "Married")
reg_formulas(tab_reg(d, "married", c("race", "age"), family = "binomial"))
```

`reg_measures` *What can this outcome be modelled as?*

Description

Lists what `tab_reg()` can report for one outcome: which models it could fit, and which measure of deviation each of them yields. It reads the same runtime table the argument validator and the error messages read, so what it prints is what the function does. The section below is the same table for every kind of outcome, read without any data.

Usage

```
reg_measures(data, outcome, family = "auto", link = "auto")
```

Arguments

`data`	A data frame (or a `survey` design), as for `tab_reg()`.
`outcome`	The outcome column name.
`family`	The model family. `"auto"` (default) lists every family this kind of outcome can take, the detected one first — which is the choice to make before the others.
`link`	Which measure the model estimates. `"auto"` (default) reads the table at each family's own model; `"all"` adds every other link it fits; name one to read it at that model alone.

Details

The table has two blocks, because the grid factors:

- one row per model you could fit — its `link`, and the measure that model's own coefficients carry (`effect = "conditional"`);
- then the measures read off the model's **predictions**, which are the same whichever model you fit — `link` reads `"(any)"` there.

So `link` is the choice that matters, and it decides only which measure comes with a *coefficient*: everything else is available from any of them.

By default only each family's **own** model is listed — the one it fits unless told otherwise. `link = "all"` adds the others, which are specialist choices, and marks the family's own with `base_link`.

Only what can be built is listed. A measure this kind of outcome does not have simply has no row, and the message says why (an odds ratio needs a probability to take the odds of). One state exists only at run time: a link that does not converge on your data. `tab_reg()` says so and, for the risk difference, falls back to the linear probability model.

Value

A tibble of `family` (only when several are listed), `link`, `measure`, `effect`, `header` (the column name it would produce) and `reads_as` (what that header's acronym means). With `link = "all"`, a `base_link` column says which model is the family's own (`NA` on the prediction rows, which belong to no link in particular).

Which models each outcome offers, and which measures

Generated from the package's own resolution table, so it cannot drift from what `tab_reg()` builds. A measure that IS the model's own is read off its coefficients; any other is computed from its predictions (`effect = "marginal"` or `"at_reference"`).

- **gaussian** — a cell is a mean. Models: `link = "difference"`, the default (linear regression); `link = "ratio"` (log-link mean regression (Poisson pseudo-likelihood, robust standard errors)). Reported: `measure = "difference"`, `"ratio"`, `"raw_coefficient"`.
- **binomial** — a cell is a percentage. Models: `link = "odds_ratio"`, the default (logistic regression); `link = "ratio"` (modified Poisson regression); `link = "difference"` (additive-risk regression (identity link, robust standard errors)). Reported: `measure = "ratio"`, `"difference"`, `"odds_ratio"`, `"raw_coefficient"`.
- **poisson** — a cell is a count. Models: `link = "ratio"`, the default (Poisson regression). Reported: `measure = "difference"`, `"ratio"`, `"raw_coefficient"`.
- **multinomial** — a cell is a percentage. Models: `link = "odds_ratio"`, the default (multinomial logistic regression). Reported: `measure = "ratio"`, `"difference"`, `"odds_ratio"`, `"raw_coefficient"`.
- **ordinal** — a cell is a position on an ordered scale. Models: `link = "odds_ratio"`, the default (ordinal logistic regression). Reported: `measure = "difference"`, `"ratio"`, `"odds_ratio"`, `"raw_coefficient"`.

See Also

`tab_reg()` to build the table, `reg_formulas()` to see the formula each column was fitted with.

Examples

```
reg_measures(car_arrests, "released")
reg_measures(car_salaries, "salary")
```

---

score_from_lv1 — *Score a set of factors by counting their first level*

Description

Builds an integer score column counting, for each row, how many of the listed factors sit at their **first level** (1 if so, 0 otherwise) – the score ranges 0 to `length(vars_list)`. The natural way to sum a battery of yes/no survey items into one score, feeding the grouped-binomial outcome of `tab_reg()` (its `trials` argument).

Usage

```
score_from_lv1(data, name, vars_list)
```

Arguments

data	A data.frame.
name	The name of the score variable to create (unquoted or a string); an existing column of that name is replaced.
vars_list	The factors to count, as a character vector. For each one only its **first level** counts (as 1); every other level, including missing values, counts as 0.

Details

The "first level" is `levels(as.factor(x))[1]`. Non-factor columns are coerced with `as.factor()`; missing values are folded into an explicit `"NA"` level first (via `forcats::fct_na_value_to_level()`), so `NA` never counts as the first level.

Value

`data` with the integer score column `name` added (or replaced).

See Also

`tab_reg()` and its `trials` argument for modelling a summed score as a grouped binomial; `vignette("tabxplor")`, section "Multiple-answer questions", for a worked example.

Examples

```
data <- tibble::tibble(group = factor(c("G1", "G1", "G2", "G2", "G3", "G3")),
                       a = factor(c("Oui", "Oui", "Oui", "Oui", "Non", "Oui")),
                       b = factor(c("Oui", "Non", "Non", "Oui", "Non", "Oui")),
                       c = factor(c("Oui", "Oui", "Non", "Non", "Oui", "Oui")))
data |>
  score_from_lv1("score", vars_list = c("a", "b", "c")) |>
  tab(group, score, digits = 1)
```

---

set_bars *Draw a column as data bars*

Description

Names the columns whose cells carry a horizontal bar behind their figure — a bar chart inside the table, at no cost in width, in html and in Excel. Total and footer rows take none.

One reference per column: every bar in a column is a share of the same ceiling, or two bars could not be compared. That ceiling is the column's own largest data cell by default, which is what spreads the bars over the width available; `max` states it instead, so that two tables can be read against each other.

It is a display intent, like a caption: the numbers are untouched, and a medium with nowhere to put a bar (the console, a pipe table) ignores it.

Usage

```
set_bars(x, cols, max = NULL)

get_bars(x)
```

Arguments

x	A tabxplor_tab.
cols	Column names, or NULL to remove.
max	The ceiling a full bar means, in the column's **stored** unit — a percentage is stored between 0 and 1, so max = 1 means 100 %. NULL (the default) or NA takes the column's largest data cell. Unnamed values are recycled over cols, a named one applies to that column.

Value

x, with its bar columns set (set_bars) ; a named vector of their ceilings, NA where the column's largest is used, or NULL (get_bars).

See Also

new_tab() for the whole meta record.

Examples

```
t <- tab(forcats::gss_cat, race, marital, pct = "row")
t <- set_bars(t, "Married")
get_bars(t)

# a full bar means 100 %, whatever the column holds
get_bars(set_bars(t, "Married", max = 1))
```

---

set_caption *Store a caption on a table*

Description

Records a caption/title on a tabxplor_tab that survives a dplyr pipeline (it is kept in the table's meta$vars$caption, carried through every verb) and is read by the exporters (tab_md, tab_kable, tab_xl) as the table title, ahead of a regression table's auto-title, when the exporter's own caption= argument is not supplied. get_caption() reads it back (NULL when none is stored).

Usage

```
set_caption(x, caption)

get_caption(x)
```

Arguments

x	A tabxplor_tab (or a tabxplor_tabs list of them).
caption	A single string, or NULL / NA to remove any stored caption.

Value

x, with its stored caption set (set_caption) ; the caption or NULL (get_caption).

---

set_color_palette *Colours: palettes, styles and breaks*

Description

Everything that decides what a coloured cell LOOKS like, and at which value it changes shade. set_color_palette() sets the hues (and the console's light/dark theme); set_color_breaks() sets the thresholds each measure is read on; get_color_style() and get_color_breaks() read them back. All of them act globally, through options(), so one call at the top of a script restyles every table it builds — see tabxplor-options. A single table can override the thresholds with tab(color_breaks =).

Color breaks are a named list of the ten measure scales pct_diff, pct_ratio, odds_ratio, mean_diff, mean_ratio, contrib, zscore, adj_ratio, adj_diff and adj_diff_std. Each is a vector of positive-only thresholds (the under-represented side is mirrored automatically), 1 to 4 values, one per color step.

Every default is the same ladder in another measure, read at ONE reference cell of 50 %: 5 / 10 / 20 / 30 percentage points is also 0.1 / 0.2 / 0.4 / 0.8 SD, x1.1 / x1.2 / x1.5 / x2 as a ratio and x1.2 / x1.5 / x2 / x4 as an odds ratio – so a shade means the same size of deviation whichever measure a table is read on. pct_diff colors percentage-point differences, pct_ratio the relative risk, odds_ratio the odds ratio (color = "odds_ratio"), mean_diff the standardized mean difference (Glass's delta) by default (supply data-unit values for absolute coloring), mean_ratio the mean ratio, contrib the chi2 contribution (in multiples of the mean cell contribution) and zscore an absolute z scale (the adjusted standardized residual) – the absolute scale color = "contrib" switches to under color_signif = "guaranteed_effect". Its default c(1.96, 2.58, 3.89, 6) is written as conf_level_to_z(c(0.95, 0.99, 0.9999,1 - 2e-9)), and its FIRST value is re-anchored to the significance threshold at print time, so the remaining ones are read as spacings from it. adj_ratio, adj_diff and adj_diff_std are the tab_reg-only scales of color = "adjustment" / "between_groups" – how far a modelled effect sits from the observed one (or from the reference group's). Which one a column reads follows the estimate's own scale: adj_ratio for a multiplicative effect (odds / risk ratio or ratio of means), adj_diff for a probability-scale marginal effect (in percentage points), and adj_diff_std for an additive effect in the outcome's own units (a gaussian beta, a count marginal effect), where the gap is divided by SD(Y) so the same threshold means the same thing whatever unit the outcome is recorded in. An empty/NULL scale drops that measure for its column type.

Two rules shape a default, and a custom one is free to break them. A ladder is MIRRORED unless the quantity it grades is bounded above: a percentage ratio is capped at 1 / base, so a cell can sit far below its reference and never far above it, and pct_ratio is stricter below (list(over = c(1.1,

1.2, 1.5, 2), under = c(1.1, 1.25, 2, 4))) – a ratio of means and a ratio of two estimates have no ceiling and stay symmetric. And a fill is read at a glance, so on the BACKGROUND channel the two ratio scales keep their two loudest rungs only: with the default color = TRUE the text grades every deviation and the background flags the ones whose RELATIVE size is out of proportion.

Usage

```
set_color_palette(
  text_colors = NULL,
  text_colors_neg = NULL,
  background_colors = NULL,
  background_colors_neg = NULL,
  dark_text_colors = NULL,
  dark_text_colors_neg = NULL,
  dark_background_colors = NULL,
  dark_background_colors_neg = NULL,
  bg_legend_colors = NULL,
  bg_legend_colors_neg = NULL,
  theme = NULL
)

set_color_style(
  type = c("text", "bg"),
  theme = NULL,
  html_24_bit = NULL,
  custom_palette = NULL
)

get_color_style(
  mode = c("crayon", "color_code", "face"),
  type = NULL,
  theme = NULL,
  ...
)

set_color_breaks(breaks = NULL, ...)

get_color_breaks(brk, type = c("positive", "all"))
```

Arguments

text_colors, text_colors_neg, background_colors, background_colors_neg
: Light-theme palettes (4 hex each): the text (font) and background (fill) colours for the over- (*_colors) and under-represented (*_colors_neg) sides.

dark_text_colors, dark_text_colors_neg, dark_background_colors, dark_background_colors_neg
: The dark-theme counterparts (4 hex each).

bg_legend_colors, bg_legend_colors_neg
: (4 hex each) The FONT stand-in for background_colors in the colour legend of media that cannot fill a run (Excel); the defaults are the background colours at -0.2 OKLCH lightness. Setting background_colors without these makes them follow it unchanged (readable only if your fills already are). There is no dark counterpart: an Excel legend cell is on a white page whatever the theme, and the dark fills read there as-is.

theme
: Which palette theme. In set_color_palette(): "light" or "dark" for the console / exports, or "auto" to detect the console's colour scheme now (the RStudio theme, the Positron theme, or COLORFGBG; "light" when it cannot be told). Detection is best-effort and resolved ONCE: call again after changing your editor's theme. (This is the console only — tab_css / tab_html take their own theme = "auto", which follows the reader's browser.) In get_color_style(): "light", "dark" or one of the black-and-white publication palettes ("print_minimalistic", "print_emphasis", "print_marks"); "auto" resolves to "light" there, a palette being always one definite thing. Both default to the current setting.

type
: Which palette, or which half of a break scale — the word means one thing per function, and both are given here because they share this page. In get_color_style() and the deprecated set_color_style(): "text" (font colour), "bg" (background fill), or "bg_legend" (mode = "color_code" only), the darker FONT stand-in for the background palette, for media that cannot fill a run (an Excel rich-text run) – see the colour legend. In get_color_breaks(): "positive" (the default) returns a readable form – a plain vector of magnitudes when the scale is symmetric, a list(over =, under =) otherwise – and "all" the signed / reciprocal thresholds the engine actually compares against (c(-x, x) for additive scales, c(1/x, x) for multiplicative ones).

html_24_bit
: **[Deprecated]** Inert since 2.0.0 (exports are always 24-bit).

custom_palette
: **[Deprecated]** A former 10/11-slot palette; its 4 over- and 4 under-represented colours are mapped onto set_color_palette().

mode
: By default, get_color_style returns a list of terminal (ANSI) coloring functions (the historical value "crayon", now built with **cli**). Set to "color_code" to return html color codes, or "face" to return the palette's TYPOGRAPHY – a list bold / italic / underline of 8 logicals each (plus a semantic flag), which is how a publication palette says "over-represented cells are bold, under-represented ones italic". The colour palettes report bold on every text slot and nothing on the background ones, i.e. exactly how they have always been drawn.

...
: Scales passed individually and named, e.g. set_color_breaks(pct_diff = c(0.05, 0.1, 0.2), mean_ratio = c(1.15, 1.5, 2, 4)). Each value is either a plain vector of signed / reciprocal literals (negatives, or ratios < 1, are the under-represented side; a one-sided vector auto-mirrors; NA skips an intensity slot) or a list(over =, under =) of magnitudes (no mirror; omit a side to switch it off, e.g. list(over = 2) for the "only x2" rule). The old pct_breaks / mean_breaks / contrib_breaks arguments are soft-deprecated but still work (mapped onto the new scales).

breaks
: A named list of scales to set, e.g. list(pct_diff = c(0.05, 0.1, 0.2, 0.3), pct_ratio = list(over = 2)). Unset scales keep their current value.

brk When missing, return the full named list of the ten break scales – the same shape set_color_breaks accepts, so it round-trips (an asymmetric scale comes back as list(over =, under =), a standardized one with std = TRUE). Specify one scale name to return only its breaks. The old aliases "pct" (-> pct_diff) and "mean" (-> mean_ratio) are still accepted.

Details

set_color_palette() customises the palette used to print tab. Each palette is 4 hex codes ordered faint -> strong. Provide only the ones you want to change; the OKLCH defaults are used otherwise. The ANSI styles are (re)built once, not per cell.

Value

Sets the internal color palettes (invisibly) and the option "tabxplor.color_style_theme".

A list of 8 terminal (ANSI) color-style functions, a vector of 8 color html codes, or (mode = "face") the palette's typography record.

Sets the global option "tabxplor.color_breaks" (a named list of scales) and returns it invisibly.

The color breaks as a double vector or a list(over =, under =), or a named list of these.

Functions

- set_color_style(): **[Deprecated]** Superseded by set_color_palette(). Kept as a back-compat shim: type/theme still take effect (as options); custom_palette maps its over/under colours onto the new 4+4 palette; html_24_bit is inert (exports are always 24-bit).
- get_color_style(): get the color palette as terminal (ANSI) style functions or html codes: an 8-element vector (4 over-represented intensities then 4 under-represented), indexed by the engine slot.
- set_color_breaks(): set the breaks used to print colors.
- get_color_breaks(): get the color breaks currently in use, in the canonical shape.

Examples

```
set_color_palette(text_colors = c("#02a5b3", "#0891c9", "#0267c7", "#300dfd"))
set_color_breaks(
  pct_diff   = c(0.05, 0.15, 0.3),
  pct_ratio  = list(over = 2),
  mean_ratio = c(1.15, 2, 4),
  contrib    = c(1, 2, 5)
)
set_color_breaks(get_color_breaks())   # a no-op: the shape round-trips
```

set_footer_tabs *Attach subordinate tables under a table*

Description

Records one or more tabxplor_tabs that BELONG to x and are rendered under it by every medium — the console, tab_md, tab_html and tab_xl — exactly as if they had been passed in one list, and which travel with x through a dplyr pipeline (they are kept in x's meta$footer_tabs).

The use is a fact that belongs to the table without being a row of it: the eigenvalues of the axes beside a factorial-analysis summary, a sample description beside the result it describes.

Usage

```
set_footer_tabs(x, tabs)

get_footer_tabs(x)
```

Arguments

x	A tabxplor_tab.
tabs	A tabxplor_tab, a list of them, or NULL to remove whatever is attached. A named element is captioned with its name (set_caption) unless it carries a caption already.

Value

x, with its subordinate tables set (set_footer_tabs) ; the list of them, or NULL when none (get_footer_tabs).

See Also

new_tab() for the whole meta record.

Examples

```
main <- tab(forcats::gss_cat, race, marital, pct = "row")
side <- tab(forcats::gss_cat, race)
main <- set_footer_tabs(main, list("Base" = side))
get_footer_tabs(main)
```

`set_legend_words` *Re-state what the colour legend calls a measure*

Description

The generated legend names a measure with the discipline's own words — `color = "contrib"` says *contribution to Chi2*. A table may legitimately grade the same ladder on another quantity: a factorial axis has no chi-squared, and its cells' contribution is to the **variance of the axis**. `set_legend_words()` re-states the words and changes nothing else, so the swatches, the ladder, both registers, the publication palettes, the plot guide and every medium keep working — the console included, which no exporter argument can reach.

Prefer it to writing a legend of your own: replacing the sentence (through `set_subtext`) costs you the terse/prose pair and the publication palettes' wording.

Usage

```
set_legend_words(x, ...)

get_legend_words(x)
```

Arguments

`x`	A `tabxplor_tab`.
`...`	One argument per measure (`difference`, `ratio`, `odds_ratio`, `contrib`, `adjustment`, `between_groups`), each either a single string (the measure's short word) or a named list of the fields below. `NULL` removes an override.

Details

The fields a table may re-state — naming only, never a number:

- `word` — the short word (the console, a plot guide); `word_long` — the same named in full, for the export footers; `word_std` / `word_long_std` their SD-scale twins.
- `word_guar` — the `color_signif = "guaranteed_effect"` head, a template taking the confidence level (`"%s%%-guaranteed contribution"`).
- `subject` — the noun for what is graded, when it is not the cell itself.
- `ref` — the baseline noun (*the mean contribution*), for a measure compared to a **concept** rather than to a row of the table: the terse form brackets it with its preposition and the prose one points at it bare, both from this one field. Give `ref_word` / `ref_phrase` instead only where those two nouns genuinely differ. Re-stating any of them on a measure whose reference is a category or a total is refused — there the legend names what the table itself shows.
- `unit_word` — the unit the thresholds are counted in.
- `lead_over` / `lead_under` — the sentence each side of the ladder opens with, as a template taking `%1$s` the subject, `%2$s` the reference and `%3$s` the null value. **Only** `%1$s` **is always there**: a line names its baseline in full on its first side and not again on its second, so `%2$s` comes back empty there — where the baseline is the measure's own (a mean contribution, an axis), write it into the sentence rather than interpolate it.

- caveat — one sentence of honesty appended to the line.

An engine fact (raw, scale, sig_source, bounds) and a ladder glyph (break_over, threshold_mult) are refused: a table attribute must never change a number, and a column pulled out of its table must still colour identically.

Written in the language of the call — unlike the words **tabxplor** generates, which follow lang = at render.

Value

x, with its legend words set (set_legend_words) ; the named list of them, or NULL (get_legend_words).

See Also

set_subtext() for the footer template, set_color_breaks() for the ladder itself.

Examples

```
t <- tab(forcats::gss_cat, race, marital, pct = "row", color = "contrib")
cat(tab_footer_text(t), sep = "\n")

t <- set_legend_words(t, contrib = "contribution to the axis variance")
cat(tab_footer_text(t), sep = "\n")
```

---

set_subtext *The lines printed under a table*

Description

A table's footer text, as a **template**: everything **tabxplor** generates is a <placeholder> and everything you write is a line, so the order of the lines is the order of the footer. Re-order them and it re-orders; drop <legend> and no colour legend is generated, in the console too.

A subtext naming **no** placeholder is simply appended to the template — which is what a note has always done, so tab(subtext = "Field: GSS 2000") is unchanged. Writing one placeholder on a line of its own takes the layout over: only what you name is printed. An unknown <...> is not a placeholder and passes through verbatim (\< escapes a literal <).

The template a producer writes names only what **this** table can say: no <weight> on an unweighted table, no <model> outside a regression. What you read back is therefore what prints, and the way to drop a line is to delete it.

A line opening on a short label and a colon ("Field: GSS 2000") has that label set in bold in every medium; options(tabxplor.subtext_bold_label = FALSE) prints it as written.

Usage

```
set_subtext(x, subtext)

get_subtext(x)
```

Arguments

x	A tabxplor_tab.
subtext	A character vector, one element per line, or NULL to restore the default template. (There is no per-table way to print nothing at all: the exporters' subtext = FALSE is the one-off.)

Value

x, with its footer template set (set_subtext) ; the template, as a character vector (get_subtext).

See Also

tab_footer_text() to see what the template prints, set_legend_words() to re-word the generated legend, set_footer_tabs() for a table or a note under the whole block.

Examples

```
t <- tab(forcats::gss_cat, race, marital, pct = "row", color = "diff")
get_subtext(t)

# your own sentence, with tabxplor's own pieces inside it
t <- set_subtext(t, c("<measure> (<ref>): <breaks>", "<stars>"))
cat(tab_footer_text(t), sep = "\n")
```

---

shape_numeric_var *Cut or transform a numeric variable*

Description

Turn a number into the form you want to read it in: cut it into quantile groups or into bands at the mean and one standard deviation either side (it becomes an ordinary factor), or transform it and keep it a number. This is the same operation tab(shape =) and tab_reg()(shape =) perform, exposed on its own vector, so a column you cut by hand is identical to one they cut.

Usage

```
shape_numeric_var(x, shape, w = NULL, name = NULL, ordered = TRUE)
```

Arguments

x	A numeric vector.
shape	A single string, or a number of groups — see *The shapes* below.
w	Optional weights, the same length as x. Quantile breaks and the mean/SD landmarks are then weighted — equal shares of the *population*, not of the sample.

`name` The variable's name, written onto the **first level only** (`"age: [18,35) low"`), so a table that names the variable nowhere else still says what the levels are levels *of*. `NULL` (the default here) writes nothing. It applies to the two CUTS, whose levels are intervals and band words; `"values_to_levels"` keeps the raw values, which name themselves.

`ordered` Whether the resulting factor is `ordered`. Bands and quantile groups do have a real order; a model fit does not want one (an ordered factor takes polynomial contrasts instead of contrasts against a reference), which is why `tab_reg()` cuts unordered.

Value

For a cut: a factor whose levels carry the real cut points and, in words, where each group sits. For `"log"` / `"sqrt"`: a numeric vector.

The mean of a transform is not the transform of the mean

`"log"` and `"sqrt"` replace the values, so a crosstab column then shows the mean *of the logarithm* — a different quantity from the logarithm of the mean, and smaller. `tab()` renames the column for that reason (`age` becomes `log_age`), so no header promises the wrong quantity; do the same if you shape a column by hand.

The shapes

- `"linear"`: the number as it is — one slope in a model, one mean in a crosstab. The default for a column variable, and what `shape` is spelled out as when nothing is done.
- `"log"`: replace the variable by its logarithm — diminishing returns. Needs strictly positive values.
- `"sqrt"`: replace the variable by its square root. Needs non-negative values.
- `"sd_bands"`: four bands cut at the mean and one standard deviation either side. Each level names its own cut (`[30,48) ; < mean`), so the label can be checked against the interval beside it. The cut points mean the same thing across sub-samples of one variable, where quantile breaks move with each one; but the bands are NOT balanced, and on a skewed variable a landmark falling outside the data is dropped (an exponential variable gets three bands, not four).
- `"median"`: two groups of equal size, cut at the median — the coarsest reading of a number.
- `"terciles"`: three groups of equal size.
- `"quartiles"`: four groups of equal size. The counts are balanced, so every group answers on a comparable base.
- `"quintiles"`: five groups of equal size.
- `"deciles"`: ten groups of equal size. Reads a gradient, but ten rows need a large sample to keep each base usable.
- `"values_to_levels"`: one level per distinct value, in numeric order. Right for a counted number or a 1-7 scale; unreadable for a continuous one, which is what `"auto"` decides.
- an integer `k` (2 to 20): `k` quantile groups of equal size.

See Also

tab() and tab_reg(), whose shape argument is this function applied once to the whole population, before any sub-table or sub-model is split off.

Examples

```
age <- forcats::gss_cat$age
table(shape_numeric_var(age, "sd_bands", name = "age"))
table(shape_numeric_var(age, "quartiles"))
```

---

tab *Cross-tables with color helpers*

Description

tab() builds a cross-table of one or several row variables by one or several column variables, and colors the cells so the table is easy to read at a glance — in the R console, or exported to Excel, HTML or Word. Cells can show counts, row or column percentages, or (for a numeric column variable) means, optionally with differences, confidence intervals and tests.

The result is a tibble (of class tabxplor_tab), so every **dplyr** verb keeps working on it.

New to the package? Four arguments are enough to begin — data, row_vars, col_vars and pct — then add color when you want reading helpers. The Introduction (vignette("tabxplor")) works through them. Package-wide defaults are options(), listed at tabxplor-options.

Usage

```
tab(
  data,
  row_vars,
  col_vars,
  tab_vars,
  wt,
  ...,
  pct = "no",
  color = "no",
  color_signif = "ignore",
  test = FALSE,
  na = "keep",
  levels = "all",
  cleannames = NULL,
  other_if_less_than = 0,
  ref = "auto",
  ref2 = "first",
  comp = "tab",
  ci = "auto",
```

```
  conf_level = NULL,
  stars = NULL,
  ci_method = NULL,
  anova = NULL,
  design_effect = NULL,
  totaltab = "line",
  common_totrow = FALSE,
  n = NULL,
  n_min = 0,
  add_pct = FALSE,
  subtext = "",
  caption = NULL,
  digits = 0,
  display = NULL,
  color_breaks = NULL,
  output_list = FALSE,
  spread_vars,
  filter
)
```

Arguments

`data` A data frame.

`row_vars` <tidy-select> The row variable(s), printed with one level per line, and the column variable(s), one level per column. A numeric variable gives a single column of means. Each accepts one variable or several, `c(var1, var2)`.

`col_vars` <tidy-select> The column variable(s) — see `row_vars`. **An interaction** is written `a*b`, as in `tab_reg()`, and only `col_vars` takes one: two factors give one column per observed cell of the pair, a number crossed with a factor one mean column per level. See `vignette("tabxplor")`.

`tab_vars` <tidy-select> Tab variables: one subtable per combination of their levels. Leave empty for a simple cross-table.

`wt` A weight variable, of class numeric. Leave empty for unweighted results.

`...` Arguments taken by name, and kept out of the signature. Past the variable roles every argument must be named, and an unknown name is refused with a suggestion. `tot` Which totals to show: `c("row", "col")` or `"both"` (default), `"row"`, `"col"`, or `"no"` (removed after the calculations that need them). `shape` How a **numeric** variable enters the table. Cut it into groups and it becomes an ordinary factor — one row (or column) per group. One value for every numeric variable, or one per variable: `shape = c(age = "quintiles")`. On the row and tab axes a number always gets one, `"auto"` by default; a numeric `col_vars` keeps its means. `shape_numeric_var` lists the whole vocabulary. `shape_name` Whether a shaped variable writes its own name onto its **first** level (`"age: [18,30) low"`), so a table whose leading text columns are stripped still says what the levels are levels of. `FALSE` by default. **Retired in 2.0.0**, still taken by name, each warning once and saying what to use instead: `row_var`, `col_var`, `sup_cols`, `totaltab_name`, `total_names`, `OR`, `chi2`, `method_cell`, `method_diff`, `add_n`,

names_prefix, names_sort, other_level. The dot-prefixed names (.cache, .defer_level_merge, .return_armed, .levels_order, .levels_collapse) are internal plumbing, not user arguments.

pct
: The percentages to calculate, as a single string or a vector the same length as col_vars: "row", "col", "all" (frequencies within each subtable), "all_tabs" (frequencies over every table) or "no" (default, counts). Everything else — the reference, the interval, the colour — follows from this choice.

color
: Which **measure of deviation** to color — a deviation being how far a cell sits from its reference, the measure which of the ways of expressing it you read. "no" (default, FALSE equivalently) prints no color; TRUE picks one per column type. Otherwise:

- "difference" ("RD", "diff"): the cell's difference from its reference (percentage points for factors, Glass's Δ for means).
- "ratio" ("RR", "RoM"): relative risk (factors) or mean ratio (numerics) vs the reference.
- "odds_ratio" ("OR"): the odds ratio, on percentage tables, coloured on its own symmetric scale.
- "contrib": signed contribution to the chi-squared (reference-free).

The acronyms in brackets are permanent aliases, the same words tab_reg's measure takes. An acronym here always names a **measure**, where display = names a *field* and ref2 = a *level*. **Position picks the channel** (1st value -> text, 2nd -> background) and **names pick the column type**: c("difference", "ratio"), c(pct = "difference", mean = "ratio"), or both with a list(). Only difference / ratio may go on the background; thresholds come from set_color_breaks.

color_signif
: How significance gates the color, as a single string:

- "ignore" (default): color every deviation by its observed size.
- "grey_non_signif": color by the observed size, but grey out cells whose deviation is not significant at conf_level. A coloured cell is then significantly different from its reference; a grey one may still be significant, only too small to colour.
- "guaranteed_effect": color by the guaranteed (confidence-bound) effect – only cells whose interval clears the threshold, with dimmer, conservative colors.

With color = "contrib", which has no interval to floor, the first two color the **relative** contribution and "guaranteed_effect" the **adjusted standardized residual**. See vignette("tabxplor").

test
: Set to TRUE to test each (sub)table for independence: **Chi-squared** for factor col_vars, **Welch's F** for numeric ones, with an effect size beside it. Needed by color = "contrib", and added automatically for it. The footer names the test you actually got — see vignette("tabxplor-weights").

na
: The policy to adopt for missing values, as a single string :

- "keep" (default): every NA becomes an explicit "NA" level.
- "drop": each column is computed on its own non-missing observations, so bases can differ between col_vars.

- "drop_all": drop every observation missing on any variable, so all columns share one base.
- "common_base": one population — non-missing on the row_vars and the **first** col_vars — while the other col_vars keep their own NA's as a level within it. Microdata only.

levels
: The levels of col_vars to keep, as a single string or a vector the same length as col_vars: "all" (default), "first" (only the first level of each — a compact summary of many items), or "auto" (the first level of a two-level variable, all of them otherwise). For finer selections use dplyr::select on the finished table.

cleannames
: Set to TRUE to clean level names, by removing prefix numbers like "1-" and text in parentheses. NULL (default) reads options(tabxplor.cleannames) — FALSE.

other_if_less_than
: When set to a positive integer, levels with less count than it will be merged into an "Others" level.

ref
: The reference cell that differences and ratios are computed against:

- "auto" (default): the corresponding total for a difference, the first row (or column) for an odds ratio. "tot": always the total.
- "first" / "last": the first or last **level** — useful to color a temporal development. A total is not a level and is never selected.
- an **integer**, the nth row (or column); a **string**, a regular expression matched against the row (or column) names. "no": no reference.

One reference per row_vars with a named vector, ref = c(race = "first"); an unnamed one goes by position.

ref2
: The second reference level for odds ratios, needed only for a factor with **3 levels or more** (the "OR of each level versus ref2"); the first level by default. Ignored for a **binary** factor, where each level's OR is taken against the other. Same values as ref.

comp
: What each cell is compared with: "tab" (default) compares it inside its own tab_vars subtable, "all" against the total table's own reference line.

ci
: **What the confidence interval is anchored on**. Its *geometry* is not asked here: it follows the comparison the table makes, so an odds-ratio table gets an odds-ratio interval.

- "auto" (default): on the comparison where the table makes one, on the cell for plain frequencies, none where nothing needs one.
- "ref": on the cell's deviation from its reference. "cell": on the cell's own percentage or mean. "no": none.

"cell" and "no" anchor nothing to compare, so stars and color_signif are disabled (with a message). The method is chosen with ci_method and named in the table's legend.

conf_level
: The confidence level, as a single numeric between 0 and 1. NULL (default) reads options(tabxplor.conf_level) — 0.95.

stars
: Logical. With ci = "ref", print significance stars for each cell's difference from its reference, read from the displayed interval itself. NULL (default) reads options(tabxplor.stars) — FALSE.

`ci_method` The interval method, one kind at a time, as ONE named vector – partial, like `ref` or `pct`, so an unnamed kind keeps its default. Example: `ci_method = c(cell = "beta", diff = "ac")`.

- `cell`, a proportion's own interval: `"wilson"` (default), `"wald"`, `"beta"`.
- `diff`, a proportion minus its reference: `"newcombe"` (default), `"ac"`, `"wald"`.
- `mean_diff`: `"welch"` (default), `"student"`, `"ols"`.
- `mean_ratio`: `"robust"` (default), `"quasipoisson"`, `"poisson"`.

A proportion *ratio* has only one method (Katz), so it is not a choice.

`anova` Which one-way ANOVA **F** the p-value line shows for *numeric* `col_vars`: `"welch"` (does not assume equal variances) or `"classic"` (the pooled F). `NULL` (default) reads `options(tabxplor.anova)` — `"welch"`. Both are always computed, so this only chooses which row is shown.

`design_effect` Whether the intervals, stars and colour thresholds of a **weighted** table account for the weighting's own design effect instead of using the raw sample size. `NULL` (default) reads `options(tabxplor.design_effect)` — `FALSE`. Ignored without `wt`. See `vignette("tabxplor-weights")`.

`totaltab` The total table, when `tab_vars` makes subtables: `"line"` (default, a general total line), `"table"` (a complete total table — `row_vars` by `col_vars`, without `tab_vars`) or `"no"`.

`common_totrow` With several `row_vars`, `FALSE` (default) shows one Total row per row variable; `TRUE` collapses the identical ones into a single shared Total. Genuinely different totals (which only `na = "drop"` can produce) are never merged.

`n` How many people this table is about. `NULL` (default) reads `options(tabxplor.n)` — `"range"`, which prints the unweighted base beside the `Total` cell, `100% (9 838)`, or the whole range where the columns do not rest on the same people, `100% (6 712-9 838)`, so an unequal base cannot pass unnoticed. `"min"` prints the smallest base only; `"no"` none.

`n_min` A single positive integer (default `0`, off). A pure display filter applied last: it hides cells resting on too few people, without recomputing anything. Totals and the p-value line are always kept.

`add_pct` Set to `TRUE` to add a column with the frequencies of the row variable (for `pct = "row"`) or a row with the frequencies of the column variable (for `pct = "col"`).

`subtext` A character vector to print rows of legend under the table.

`caption` A title for the table. It is **stored on the table**, so it survives a **dplyr** pipeline and travels into every export — html, Markdown, Excel, `forest_plot()` — where an exporter's own `caption` still wins. `set_caption()` attaches one after the fact, `get_caption()` reads it back.

`digits` The number of digits to print, as a single integer, or an integer vector the same length as `col_vars`.

`display` What each value cell shows (text output only – the console, `tab_kable` and `tab_md`; Excel falls back to the primary field). `NULL` (default) keeps each cell's plain value. Three ways to ask: a **named layout** (`"est_ci"`, `"base_ratio"`), a **single field** (`"ci"`, `"diff"`), or a **{} template** of your own (`"\{pct\} (n=\{n\})"`). The whole vocabulary is in tabxplor-display; `set_display` changes it on a table

already built. A layout that names a confidence interval or a contribution to the chi-squared COMPUTES it, so `display = "base_ci"` needs no `ci =` beside it; an explicit `ci = "no"` still stands.

`color_breaks` A per-table override of the colour thresholds, in the form `set_color_breaks` accepts; unset scales keep the global ones.

`output_list` Logical (default `FALSE`). With several `row_var`, `FALSE` merges the mirror tables into a single `tabxplor_tab`; `TRUE` returns a list with one table per `row_var`. With `tab_vars`, tables stay a list regardless.

`spread_vars` <tidy-select> The `tab_vars` to show ACROSS the page instead of down it: each of their levels becomes a block of columns, and the table becomes as compact as it can be. A variable named here alone is added to `tab_vars` for you. Pair it with `comp = "all"` to compare every block against the overall total, and with `levels = "first"` to keep one column per block. Because the columns are multiplied, a cell layout you did not ask for narrows to its bare estimate — a numeric column shows its mean alone, without the coefficient of variation; name a layout with `display =` to keep one.

`filter` **[Superseded]** A `dplyr::filter` to apply to the data first, as a single string. Prefer filtering upstream of `tab()`.

Details

Ordered factors stay ordered through the whole pipeline, which is what lets `ref2 = "cumulative"` pick its column variables by class. One consequence is worth knowing: the synthetic `"Total"` and `"NA"` levels are appended *after* the real ones, so on an ordered column they compare as the greatest. They are labels, not points on the scale.

Weights and survey designs. A weight (`wt`) weights the estimates; the intervals still use the raw number of cases unless `design_effect = TRUE`. Pass a `survey::svydesign` as `data` and strata, clusters, `fpc` and calibration reach every interval, star and colour threshold. The footer always names what you got. See Weighted and survey data (`vignette("tabxplor-weights")`).

Value

A `tibble` of class `tabxplor_tab`. Every numeric column is an `fmt` vector holding all the data behind the number it shows; the `row_vars` and `tab_vars` columns are factors. Any factor column you add later is treated as a `tab_vars` and used for grouping, so keep added text columns as `character`.

Significance stars

With `stars = TRUE` and an interval anchored on the comparison (see `ci`), each cell says how sure we can be that its deviation from the reference is real and not sampling noise: `*` at the 10% level, `**` at 5%, `***` at 1%. The exact p-value is stored per cell, readable with `$pvalue` or `get_pvalue()`.

No separate test runs behind the scenes: a cell is significant exactly when the interval it prints no longer contains zero, so the stars and the `[inf; sup]` bracket can never contradict each other. Which classical test that amounts to follows `ci_method`, and the table's legend names it. An absolute cell interval compares nothing, so it carries no stars.

See Also

tab_reg() (regression tables) and tab_counts() (pre-aggregated counts); the superseded tab_num() (numeric variables) and tab_plain() (one bare cross-table). tabxplor-display says what a cell can show; set_color_breaks() / set_color_palette() customise the colours; tab_structure() reports what a finished table is and what accepts it. Export it with tab_xl() (Excel), tab_kable() (HTML) or tab_md() (Markdown), and chart it with forest_plot(). Package-wide defaults live in tabxplor-options.

color = "contrib" reads as a heatmap of the association pattern. For the specialist contingency-table models built on top of it — quasi-independence, Goodman's RC association models, UNIDIFF — see the **logmult** package (https://cran.r-project.org/package=logmult).

Examples

```
# A simple cross-table of counts:
tab(car_arrests, colour, released)


# Row percentages, with the difference to the total coloured:
tab(questionr_hdv, qualif, cinema, pct = "row", na = "drop", color = "difference")

# One subtable per level of a third variable, and colour only what is significant:
tab(questionr_hdv, qualif, cinema, sexe, pct = "row", na = "drop",
    color = "difference", color_signif = "grey_non_signif")

# Several col_vars at once, mixing factors and numeric variables (means):
tab(car_salaries, rank, c(discipline, salary, yrs.service), pct = "row")

# `levels = "first"` keeps one column per variable: a compact summary of many items.
tab(facto_tea, SPC, c(breakfast, evening, home), pct = "row", levels = "first")

# Each cell's contribution to the table's variance, as in a correspondence analysis:
tab(questionr_hdv, qualif, cinema, na = "drop", color = "contrib")

# The result is a tibble, so every dplyr verb works on it. Keep the total rows last:
tab(questionr_hdv, qualif, cinema, pct = "row", na = "drop") |>
  dplyr::arrange(is_totrow(dplyr::pick(dplyr::everything())), dplyr::desc(Oui))
```

---

tabxplor-base-coercion

Hand a table to base R

Description

as.matrix() gives the table's numbers as a plain numeric matrix; as.table() gives the same matrix as a base table, its dimnames named after the row and column variables. That is the shape base R and the packages built on it expect — a correspondence analysis, chisq.test(), mosaicplot():

```
FactoMineR::CA(as.matrix(tab(forcats::gss_cat, race, marital)), graph = FALSE)
```

Only the DATA cells come across. The total row, the total columns and the display-time rows (the base count, add_pct, the p-value and model-fit lines) are dropped, because a test or an analysis run on a table's own margins is wrong; totals = TRUE keeps them. Each cell contributes the number it *shows*, so a plain tab gives counts, a pct = "row" table proportions, and a numeric column means.

Usage

```
## S3 method for class 'tabxplor_tab'
as.matrix(x, totals = FALSE, ...)

## S3 method for class 'tabxplor_tab'
as.table(x, totals = FALSE, ...)
```

Arguments

x	A table made with tab, tab_counts or tab_reg.
totals	Set to TRUE to keep the total row, the total columns and the display-time rows.
...	Not used.

Value

A numeric matrix, or a base table.

Functions

- as.matrix(tabxplor_tab): the table's numbers as a numeric matrix
- as.table(tabxplor_tab): the same, as a base table with named dimnames

See Also

get_num, tab_export.

Examples

```
tabs <- tab(forcats::gss_cat, race, marital)
as.matrix(tabs)
as.table(tabs)

# a row-percentage table gives proportions, not counts:
as.matrix(tab(forcats::gss_cat, race, marital, pct = "row"))
```

tabxplor-display *What a table cell shows: the display grammar*

Description

Every function that builds a table takes a `display` argument, and `set_display()` changes it afterwards. This page is its vocabulary: the fields a cell may show, and the named layouts that arrange them.

Asking for a layout **in the call** computes what it prints: one naming a confidence interval or a contribution to the chi-squared turns that on, so `display = "base_ci"` needs no `ci =` beside it. Every other field is already stored in the cell (see fmt), so `set_display()` on a finished table changes no number — it only says which of them is shown, and names the argument that would have filled a field the table does not carry.

Details

Three ways to ask, from the shortest:

- a **named layout**: `display = "est_ci"`, `"base_ratio"`, `"mean_sd"`.
- a **single field**: `display = "ci"`, `"diff"`, `"n"`.
- a `{}` **template** of your own: `"{est} ({base})"`, `"{pct} [{n}]"`.

In a template, the **primary** field is the first one written *outside* brackets — so an aside may come first, `"({base}) {est}"`, without ceasing to be an aside. The primary carries the significance stars, it is what Excel writes and what `get_num()` returns, and it is the part the colours paint.

A field may carry **its own precision**, `"{est:3} ({base:1})"`, which beats every default — the only way to set an aside's decimals independently of the estimate's.

`est` and `base` are **scale-relative**: `est` is whatever the column estimates (a percentage, a mean difference, an odds ratio) and `base` the level it sits on. That is what lets one layout name mean the same thing on a `tab()` crosstab and on a `tab_reg()` regression table.

A field with nothing to show renders blank but keeps its width, so the column stays aligned; a field empty in the whole column is dropped, and a note says which argument would have filled it.

Display fields

The fields a `{}` template may name, and `display` may name on their own.

- `pct` — the percentage. Needs a factor col_var and percentages: pct = "row" / "col" / "all".
- `n` — the count.
- `wn` — the weighted count. Needs weights (`wt =`).
- `mean` — the mean. Needs a numeric col_var.
- `est` — the estimate, whatever this column estimates — an odds ratio, a risk difference, a coefficient, a percentage. The one token that means the same thing on every table.

- `base` — the level the estimate sits on: the percentage, the mean or the count. On a plain percentage table it is the same number as `est`; beside a regression effect it is the adjusted prediction. Needs a column that has a level beside its estimate.
- `diff` — the difference from the reference. Needs a `ref` to compare to and something to compare: pct = "row" / "col", or a numeric col_var.
- `ratio` — the ratio to the reference (relative risk, or a ratio of means). Needs a `ref` to compare to and something to compare: pct = "row" / "col", or a numeric col_var.
- `ci` — the confidence interval of whatever the column compares, as `[low;high]`.
- `moe` — the margin of error — the same interval as `ci`, written as the half-width `+/-x` around the estimate. Void where the column compares a RATIO: a ratio's interval is symmetric on the LOG scale, so it has no half-width.
- `or` — the odds ratio. Needs a factor col_var and a percentage base: pct = "row" / "col".
- `odds` — the odds — `pct / (1 - pct)`, the quantity an odds ratio is a ratio of. Printed on the odds ratio's own ladder, so a cell below 1 reads its inverse ("1/2.60") unless `options(tabxplor.ratio_print = "raw")` asks for the plain number. Void where the percentage is 1 and the odds infinite. Needs a factor col_var and a percentage base: pct = "row" / "col".
- `ctr` — the cell's contribution to the chi-squared. Needs a factor col_var.
- `var` — the variance. Needs a numeric col_var.
- `sd` — the standard deviation, in the variable's own unit. Needs a numeric col_var.
- `cv` — the coefficient of variation — the standard deviation as a percentage of the mean. Needs a numeric col_var whose mean is positive.
- `resid` — the adjusted standardized residual – whether the cell departs from independence. Derived from the p-value and the sign of `ctr`, so it is read-only. Needs a factor col_var.
- `obs` — the OBSERVED (crude) effect a modelled one is compared to. `tab_reg()` tables only. Needs tab_reg(empirical = TRUE) (an observed effect to compare the model to).
- `coef` — the estimate on the model's LINK scale — the coefficient a linear or log-link model fitted. The same number as `est` where the column is already additive, its logarithm where the column shows a ratio. Needs a `tab_reg()` column (a crosstab estimates no coefficient).
- `gap` — how far adjustment moved the effect: the gap between the modelled estimate and its observed counterpart, on the estimate's own scale. What `color = "adjustment"` grades — readable in print and Excel, not only in an html tooltip. Needs tab_reg(empirical = TRUE) (an observed effect to compare the model to).

Display layouts

The named layouts `display` accepts. They are spelt with the scale-relative `{est}` / `{base}` fields, so one name means the same thing on a crosstab and on a `tab_reg` table:

- `"est"` (`{est}`) — the estimate alone.
- `"est_ci"` (`{est} {ci}`) — the estimate with its confidence interval.
- `"est_base"` (`{est} ({base})`) — the estimate and, in parentheses, the level it sits on.
- `"est_base_once"` (`{est}`) — the estimate alone — the level is stated once, by the observed column beside it.

- "est_coef" ({est} ({coef})) — the estimate and, in parentheses, the model's own coefficient.
- "base_est_mdiff" ({est} ({diff})) — the estimate and, in parentheses, the same comparison as a difference.
- "base_est_mratio" ({est} ({ratio})) — the estimate and, in parentheses, the same comparison as a ratio.
- "est_obs" (({obs}) {est}) — the estimate and, before it in parentheses, the observed (crude) effect it is compared to.
- "base_est" (({base}) {est}) — the level, then the estimate — the mirror of est_base, which sets a crude and a modelled effect side by side.
- "base" ({base}) — the level alone: the percentage, the mean or the count.
- "base_ci" ({base} {ci}) — the level with its confidence interval.
- "base_moe" ({base} {moe}) — the level with its margin of error.
- "base_diff" ({base} ({diff})) — the level and, in parentheses, its difference to the reference.
- "base_ratio" ({base} ({ratio})) — the level and, in parentheses, its ratio to the reference.
- "base_or" ({base} ({or})) — the level and, in parentheses, its odds ratio.
- "or_base" ({or} ({base})) — the odds ratio and, in parentheses, the percentage it rests on.
- "mean_sd" ({mean} (SD{sd})) — the mean and, in parentheses, its standard deviation.
- "mean_cv" ({mean} (cv {cv})) — the mean and, in parentheses, its coefficient of variation — the spread as a percentage of the mean, comparable between columns measured in different units (the default where every mean is positive).

See Also

set_display() and get_display() change or read it on a built table; tab() and tab_reg() set it in the call; fmt describes every field a cell stores, and tabxplor-options the session-wide defaults.

---

tabxplor-footer *The lines under a table*

Description

Everything **tabxplor** prints beneath a table — the weight line, the Model: line, the colour legend, the significance-stars key, and your own notes — is **one template**, and the template is the table's subtext. get_subtext() shows it, set_subtext() replaces it:

```
t <- tab(forcats::gss_cat, race, marital, pct = "row", color = "diff")
get_subtext(t)
#> "<legend>" "<stars>"
```

Everything generated is a `<placeholder>`; everything you write is a line; **the order of the lines is the order of the footer**. Re-order them and it re-orders. Delete `<legend>` and no colour legend is generated — in the console too, which no exporter argument can reach.

A table names only what it can say. The template above has no `<weight>` because the table is unweighted and no `<model>` because it is not a regression — both are settled when the table is built and nothing can add them afterwards. `<legend>` and `<stars>` are always named: they are built from the columns, which `set_color()` and the **dplyr** verbs can still change.

The one rule

A `subtext` naming **no** placeholder is simply appended to the default footer, which is what a note has always done. Writing one placeholder on a line of its own takes the layout over: only what you name is printed. An unknown `<...>` is not a placeholder — raw html, `"n < 30"` and `"<30 ans>"` pass through verbatim and claim nothing (`\<` escapes a literal `<`).

What is built when

Every placeholder is resolved **at render**, so a footer follows the `lang =`, the `theme =` and the medium of the call that prints it, and a table edited after it was built (a `select()`, a `set_display()`, a `set_color_breaks()`) says the truth about what it now shows. Your own lines are frozen in the language you wrote them — the default template holds no prose, so nothing mixes unless you write it.

A table that has lost its attributes keeps what its **columns** can still say — the colour legend and the stars key — and drops the rest.

The placeholders

A line that IS one of these is built by **tabxplor**; a line that merely contains one of the last three keeps its own words and has that piece substituted into it. Anything else is your text, printed as written.

`<weight>` how the table was weighted, and what its intervals and tests rest on.

`<model>` a regression's family, outcome, predictors and estimand.

`<interaction>` the aggregated effect-modification test.

`<legend>` the colour legend: what each shade means. `<legend:terse>` / `<legend:prose>` pin the register. Change it with `set_legend_words()`.

`<stars>` what each significance star means.

`<breaks>` just the coloured ladder of a measure, inside a line of your own. `<breaks:over>` / `<breaks:under>` take one side; `<breaks:contrib>` names a measure where several compete.

`<measure>` what the colours grade, in the words `set_legend_words` gives it. Change it with `set_legend_words()`.

`<ref>` the baseline it is graded against, as the compact form brackets it (preposition included); `<ref:noun>` gives the bare noun a sentence points at. Change it with `set_legend_words()`.

`<method>` how the intervals were computed (*Wilson score interval, 95% confidence*).

`<cols>` the names of the columns a measure describes.

`<conf>` the confidence level, localised.

See Also

set_subtext() to edit the template, tab_footer_text() to read what it prints, set_legend_words() to re-state what the legend calls a measure, set_footer_tabs() and tab_note() for a table or a note under the whole block, tabxplor-options for the session-wide defaults.

---

tabxplor-options *The tabxplor options, and their defaults*

Description

tabxplor reads its display, colour, statistics and export defaults from options(), all prefixed tabxplor.. Set any of them for a session with options(), e.g. options(tabxplor.stars = TRUE), or once at the top of a script or .Rmd. The defaults are established when the package loads (.onLoad()); most also have a per-call argument on the relevant function, which always wins over the option.

Display and printing

tabxplor.print "console" (default): the medium a table auto-prints in. "html" renders the tab_html() table (in the Viewer pane in RStudio/Positron, and as a real html table in rmarkdown/Quarto documents) — recommended when you work in an IDE with a Viewer; "kable" is an accepted synonym of it (the pre-2.0.0 name). "md" renders the tab_md() table, so a bare tab() comes out as markdown — on standard output in a script, and raw (not as a verbatim block) in a knitted document, which is what an exploration notebook knitted straight to .md wants. Both media call their exporter bare, so tabxplor.theme and tabxplor.tab_kable_css govern them exactly as they govern an explicit call: options(tabxplor.tab_kable_css = FALSE) is what stops each table carrying its own stylesheet. There is no "xl" and no "forest": one writes a file, the other is a chart — ask for those by name.

tabxplor.stars FALSE (default): whether cells show significance stars, and at which cut-offs. FALSE (no stars), TRUE (the default ladder c("*" = 0.05, "**" = 0.01, "***" = 0.001)), or a named numeric giving your own – names are the glyphs, values the p-value cut-offs, e.g. options(tabxplor.stars = c("*" = 0.05, "**" = 0.01)). Off for tab(), on for tab_reg(). The LADDER is a render-time reading of each cell's stored p-value, so it is this option alone — change it and every table already built shows the new glyphs. Per-call stars =.

tabxplor.ratio_print "inverse" (default): prints a multiplicative value below its reference as the inverse — an odds ratio of 0.37 as 1/2.67, a mean ratio of 0.42 as /2.4 — so "2.7 times less" reads as strongly as "2.7 times more", and the same in a bracket. "raw" prints the plain number (0.37), the convention of most journals.

tabxplor.n "range" (default): how many people a table says are behind its numbers. "range" puts the unweighted base beside the Total cell of a crosstab (100% (9 838)) and in the n column of a regression table, printed as min-max when the blocks rest on different populations — several column variables losing different NAs, or several models. "min" prints the smallest base only, "no" shows no count at all. It replaces the add_n argument, deprecated in 2.0.0. Per-call n =.

`tabxplor.color_whole_cell` `FALSE` (default): EXPERT. A cell that prints SEVERAL fields reads as one number with an aside — `1/1.63*** (31%)` — so the cell's rendering grades the number and the aside is set slightly back from the table's own text, following the theme. That covers the colour and, under `theme = "print"`, the typography (bold, italic, underline) alike. Set to `TRUE` to extend the primary's own rendering over the whole cell instead (the pre-2.0.0 look). There is nothing to choose beyond that: which grey an aside takes belongs to the theme's palette, not to a per-cell option — see `set_color_palette()`. Console, html and Markdown; Excel renders a cell as a whole either way.

`tabxplor.var_names` `"both"` (default): which variable names the exporters annotate: `"both"`, `"rows"`, `"cols"`, `"none"`. Per-call `var_names =`.

`tabxplor.var_labels` `FALSE` (default): in *exports* (markdown / html / Excel / plot), show a variable's *label* (the `haven/labelled` `label` attribute, if it has one) instead of its name. Display only – the table structure keeps canonical names, so name-based `select()` and references still work; the console always shows names.

`tabxplor.cleannames` `FALSE` (default): clean up variable/level names in output. Also strips a `"1-"`-style prefix from `labelled` value labels turned into factor levels. Per-call `cleannames =`.

`tabxplor.total_names` `c(row = "Total", col = "Total", tab = "Ensemble", other = "Others")` (default): the four synthetic labels a table carries: `row` and `col` name the total row and the total column, `tab` the total *table* (the one made when there are `tab_vars`), and `other` the level `other_if_less_than` lumps small levels into. A partial vector is allowed – `options(tabxplor.total_names = c(tab = "Ensemble", other = "Autres"))` leaves the first two alone. It replaces the `total_names` / `totaltab_name` / `other_level` arguments, deprecated in 2.0.0.

`tabxplor.shape_auto_max` `12L` (default): where `shape = "auto"` draws the line for a numeric row or tab variable: a column with at most this many distinct **whole** values is a counted number or a short scale, and keeps one level per value; anything else is continuous and is cut into `"sd_bands"`. Raise it for a long scale, lower it to band more eagerly — or name the variable in `shape` and decide yourself.

Colours and theme

`tabxplor.color_breaks` the colour-break scales (a named list of `pct_diff`, `pct_ratio`, `odds_ratio`, `mean_diff`, `mean_ratio`, `contrib`, `zscore`, `adj_ratio`, `adj_diff`, `adj_diff_std`). Set with `set_color_breaks()`. Per-call `color_breaks =`.

`tabxplor.color_style_theme` **(alias** `tabxplor.console_theme`**)** the *console* palette theme, `"light"` or `"dark"`; set by `set_color_palette()` (which auto-detects the editor theme on load). NOT the export theme (`tabxplor.theme` / `tabxplor.export_theme`).

`tabxplor.console_bold` whether to embolden the reference / total (and coloured) cells in the *console*, `TRUE` or `FALSE`. Auto-detected at load: `TRUE` in Positron and VS Code (which render ANSI bold at a fixed glyph width), `FALSE` in RStudio and unknown consoles (there bold is drawn wider and would break column alignment). Override it for your own front-end / font.

`tabxplor.theme` **(alias** `tabxplor.export_theme`**)** `"light"` (default): the *export* theme: `"light"`, `"dark"`, `"auto"` (follow the reader), or a black-and-white **publication** palette – `"print_ready"` picks one per table (marks for a cross-table, the emphasis ladder for a regression), or name it yourself: `"print_marks"`, `"print_emphasis"`, `"print_minimalistic"` (`"bw"` is a synonym of the last). See `tab_css()` for what each says. `"auto"` needs a stylesheet, so only `tab_html()`, `tab_md()` and `tab_css()` honour it; static backends resolve it to `"light"`. A publication palette reaches every backend, Excel included. Per-call `theme =`.

`tabxplor.print_rules` `TRUE` (default): every stylesheet `tab_css()` emits also carries a black-and-white publication palette inside an `@media print` block, so a table rendered in colour **prints** (or saves to PDF) publication-ready with no further action. Set `FALSE` if your printer is a colour one and the colours are the point, or name a palette (`"print_emphasis"`) to print in that one instead of the default `"print_minimalistic"`. `"print_marks"` and `"print_ready"` cannot be used here: their marks are cell text, and a print rule can restyle a page but not add characters to it. Per-call `print_rules =`.

`tabxplor.background` `"page"` (default): what a rendered table paints behind itself. `"page"` leaves it **transparent**, so the table sits on the page's own ground, whatever that is; `"theme"` paints the theme's own background, a card of its own; or name any CSS colour. Change it only where the page is not yours to follow — a dark table dropped into a light document, an html email. The interactive Viewer page paints itself either way, and a **publication** palette is always a sheet of white paper.

Statistics and confidence intervals

`tabxplor.anova` `"welch"` (default): which one-way ANOVA F is shown for mean columns: `"welch"` (robust) or `"classic"` (pooled variance). Both are always stored in the `test` attribute. Per-call `anova =`.

`tabxplor.design_effect` `FALSE` (default): a weighted `tab()` estimates the population but bases every interval and test on the raw number of respondents, so they carry no design effect — and the table's footer says so. Set `TRUE` and the same intervals **account for the unequal weighting, exactly**: a weight column IS a survey design (the flat one, `ids = ~1`), whose variance has a closed form in the per-cell `sum(w^2)` the aggregate already computes, so the base becomes `n_eff = p(1-p) / Var_design(p)` in **every weighted descriptive confidence interval** — factor proportions *and* means (cell, difference, ratio and the `color = "odds_ratio"` significance) — and the whole-table tests (`test = TRUE`) become `survey::svychisq` / a `svyglm` Wald F on that flat design. It reproduces `survey` to the last digit, Kish's `(sum w)^2 / sum(w^2)` being that same formula with each cell's own `sum(w^2)` discarded. Being exact rather than a bound, it can make an interval *narrower* as well as wider. It is blind to **clustering** and to **calibration**, which the weights do not record — and those are not symmetric: missing the calibration and the strata costs a few percent, in the safe direction, while missing the clusters of a face-to-face household survey can leave an interval several times too short (see the Weights section of `vignette("tabxplor")`). It needs the microdata weights, so `tab_counts()` on pre-aggregated counts cannot apply it (such a table states the raw basis in its footer rather than claiming a correction it does not have). **Scope:** `tab()` **and its leaves only.** `tab_reg()` never reads it — its crude `empirical =` companions are always on the weighted basis, beside a model column (`survey::svyglm`) that always was. For the full design effect — strata, clusters, `fpc`, calibration — pass a `survey::svydesign` as `data`; the option is then not consulted at all. Per-call `design_effect =`.

`tabxplor.conf_level` `0.95` (default): confidence level for the intervals and significance tests. Since 2.0.0 each column records the level it was built at, so the colour thresholds follow the argument and this option is the fallback for a column that never recorded one (a hand-built `fmt()`, or a table from an older session). Per-call `conf_level =`.

`tabxplor.legend_style` `"prose"` (default): the colour-legend style in exports (`tab_md()`, `tab_html()`, `tab_xl()`): `"prose"` (full sentences) or `"terse"` (the compact one-line form the console uses). The console itself is always terse.

`tabxplor.subtext_bold_label` `TRUE` (default): whether a note line opening on a short label and a colon (`"Champ : adults"`, `"Source: GSS"`) has that label set in bold, in every medium. `FALSE` prints the note as written. The label is at most four words, with no digit, so a free sentence is left alone.

`tabxplor.test_lines` `"summary"` (default): how many crosstab test rows the exporters (`tab_md()`, `tab_html()`, `tab_xl()`) append: `"summary"` (p-value + effect size), `"all"` (+ the raw statistic), `"stat"` (p-value + statistic), or `"pvalue"` (the single p-value row). The p-value row name states the test used ("pvalue (Chi2, Welch F; survey-design)") and the effect-size row its measure ("Cramer's V, eta2"). N is never added – it is already shown by the `n` column. The console block always shows N + p-value + effect size.

`tabxplor.shape_table` **(alias** `tabxplor.spark`**)** `"all"` (default): in a `tab_reg()` table, each continuous predictor's OBSERVED SHAPE — the outcome binned against the predictor, on the model's scale, with no model in it — drawn as a small curve in a **shape table** below the footer, beside the range it is a picture of (`13-57% (OR 8.7)`). It is the eye-half of the `Linearity` footer row, and the free one: no fit is involved. With `tab_vars`, one curve per group; with several outcomes, one per outcome. `"all"` draws it in every medium; `"console"` only where you are working, so exported tables stay unchanged; `"no"` never. `TRUE` / `FALSE` are accepted for the first and the last. The curve is drawn TO SCALE on the predictor as the model sees it, so a `shape` transform visibly straightens it when it is the right cure, and every predictor's curve is the same width. Its vertical window is floored by the data's own sampling noise, so a curve smaller than that is greyed and marked `ns` — read it as a flat line whatever its shape. In HTML the glyphs become an inline SVG; a plot never draws them (no graphics-device font has them). An ordinal or multinomial outcome has one curve per cut or per category and this draws only the first: `reg_check_plots()` shows them all.

HTML / `tab_html()` export

`tabxplor.tab_kable_css` **(alias** `tabxplor.kable_css`**)** `TRUE` (default): inline the stylesheet with each `tab_html()` / `tab_md()` table (self-contained). Set `FALSE` in a many-table document that emits `tab_css()` once at the top. Per-call `css =`.

`tabxplor.tab_kable_tooltips` `TRUE` (default): show the per-cell hover tooltips (counts, confidence intervals, differences...) in html tables. Set `FALSE` once per document when every table auto-prints and tooltips are unwanted. Per-call `tooltips =`.

`tabxplor.kable_popover` `FALSE` (default): use click popovers instead of hover tooltips. Per-call `popover =`.

`tabxplor.tab_kable_num_font` the HTML/markdown number-font CSS stack. Monospace by default so figures stay column-aligned (set a proportional stack to revert).

`tabxplor.output_kable` `FALSE` (default): **[Superseded]** makes `tab()` return an html table instead of a table: the value can no longer be piped, filtered or re-coloured. Use `options(tabxplor.print = "html")`, which renders the same html at print time and leaves the object a table. Still honoured, with one message per session.

Excel / `tab_xl()` export

`tabxplor.xl_font_text` `"DejaVu Sans Condensed"` (default): text (labels/headers) font. Per-call `font_text =`.

`tabxplor.xl_font_num` `"DejaVu Sans"` (default): number font without stars. xlsx records ONE name (no fallback list), so set a font installed where the workbook is opened. Per-call `font_num =`.

`tabxplor.xl_font_num_stars` `"Cascadia Mono"` (default): number font with stars (monospace, so stars align). Per-call `font_num_stars =`.

`tabxplor.xl_ratio_cells` `"fold"` (default): what a ratio / odds-ratio cell HOLDS in the workbook: `"fold"` (the default) the signed fold, so Excel prints what the console prints and the cell stays a number; `"raw"` the untransformed ratio; `"text"` the exact display string. Per-call `ratio_cells =`.

Plot, paths and language

`tabxplor.export_dir` `NULL` (default): default directory for exported files (`NULL` = the working / typed path).

`tabxplor.lang` `"auto"` (default): the colour-legend language: `"auto"` (follows the R/OS locale), `"en"` or `"fr"`. Per-call `lang =`.

Parallel build

`tabxplor.parallel` `FALSE` (default): build the independent units of one call on parallel CPU cores (needs the `mirai` package): the per-`row_var` tables of a `tab()`, the models / `tab_vars` groups / outcomes of a `tab_reg()`. The result is byte-identical to the serial one. `FALSE` (default) never dispatches. `"auto"` (or `TRUE`) takes **half the cores this session may actually use, at least 2 and at most 4** — so 2 on a dual-core laptop, 2 on a 4-core machine, 4 on 8 cores or more. An integer takes that many verbatim. The count respects `options(mc.cores)`, a container's CPU quota and an HPC allocation (it reads them through the `parallelly` package when installed), and never exceeds 2 under `R CMD check`. It stays OPT-IN because starting the pool BLOCKS for about a second, so the first parallel table of a session is always slower than the serial one; from the third it is ahead. It pays off for MANY evenly sized units against a small or medium data frame — 24 tables run about 2.8x faster on 4 workers — and is a loss for few units or multi-million-row data, where shipping the population to each worker eats the gain. Set it once at the top of a script: `options(tabxplor.parallel = "auto")`. A model comparison (`stats = "compare_*"`) is always serial and says so when asked: it is a test BETWEEN the fits, so they are built together. For one call only, wrap it in `withr::with_options(list(tabxplor.parallel = "auto"), ...)`. The pool persists for the session; release it with `tab_parallel_stop()`.

`tabxplor.parallel_min` `2L` (default): the smallest UNIT count worth dispatching – `row_vars` for `tab()`, models for `tab_reg()` (fewer runs serially, since the setup would outweigh the gain).

---

tabxplor-type *Column types, the tabxplor 1.x spelling*

Description

[Superseded]

In tabxplor 1.x a `fmt` column carried one `type` attribute, whose seven values conflated two facts. Since 2.0.0 they are two attributes: `get_scale()` / `set_scale()` say **what the column estimates**

(a key into the declared scale table), and get_pct_type() / set_pct_type() say **which kind of percentage** it holds. These two functions translate, so 1.x code keeps running; they are defunct in tabxplor 2.1.0.

get_type() re-fuses what 2.0.0 split, so it is a reading aid rather than an accessor: level_mean reads back "mean", level_n reads "n", a level percentage reads its own pct_type, and every effect scale (a difference, a ratio, an odds ratio, a coefficient) reads "coef" — distinctions 1.x could not make are lost on the way back.

Usage

```
set_type(x, type)

get_type(x, ...)
```

Arguments

x	A fmt vector, or a data frame of them.
type	One of "row", "col", "all", "all_tabs", "mean", "n", "coef".
...	Used in methods to add arguments in the future.

Value

get_type() a character vector; set_type() a modified fmt vector.

Functions

- set_type(): set the retired type attribute of a fmt vector
- get_type(): get the retired type of fmt columns

See Also

get_scale(), set_scale(), get_pct_type(), set_pct_type(), fmt_attr().

Examples

```
x <- fmt(n = c(10, 20), pct = c(0.3, 0.7), scale = "level_pct", pct_type = "row")
get_type(x)
```

---

tab_columns *Every* fmt *column of a table, and what it carries*

Description

[Experimental]

One row per numeric (tabxplor_fmt) column, with the per-column attributes that decide what it shows, what it estimates, how it is coloured and how its confidence interval was computed. The column-axis companion of tab_structure(), which describes the table as a whole.

Usage

```
tab_columns(x)
```

Arguments

x	A tabxplor_tab / tabxplor_grouped_tab, or any data.frame holding fmt columns.

Value

A tibble, one row per fmt column:

column the column name.

col_var, col_group the column variable, and the sub-population its block belongs to ("" when the table was never spread).

scale, pct_type what the column estimates, and on which percentage base.

display the display template(s) its cells carry.

ref, comp_all, totcol, refcol the comparison model: which baseline, whether it compares across sub-tables, and whether this column is a total or the reference.

color, color_bg, color_signif the colour measure of each channel and the significance policy.

conf_level, degf, basis, ci_method how this column's interval was computed — the level, the degrees of freedom it is referred to (NA = the normal quantile), whether it rests on the raw count, the weights or the survey design, and by which method.

model_family, role for a tab_reg() table: the column's model family, and whether it holds the model estimate ("model") or its observed counterpart ("emp").

See Also

tab_structure() for the table's own structure; fmt_attr() to read or write one attribute; fmt() for what each attribute means.

Examples

```
t <- tab(forcats::gss_cat, marital, race, pct = "row", ci = "ref")
tab_columns(t)
```

---

tab_compact *Bind a list of tables into one*

Description

Bind a list of tables into one

Usage

```
tab_compact(tabs)
```

Arguments

tabs	A list of tabxplor_tab (or a tabxplor_tab)

Value

A tabxplor_tab

Examples

```
forcats::gss_cat |>
  tab(c(race, rincome), marital, pct = "row", color = "difference", output_list = TRUE) |>
  tab_compact()
```

---

tab_counts *Cross-tables from already-aggregated counts*

Description

tab_counts() builds the same color-coded cross-table as tab(), but from data that is **already cross-tabulated** — a dplyr::count() result, a contingency table, or a published table of counts or percentages — instead of microdata (one row per individual). Percentages, differences, confidence intervals, chi-squared, colors and totals are all computed from the counts, and the result is identical to the table tab() would build from the microdata behind them.

It accepts four input shapes:

- **Long tidy counts** (the default): one row per row_var × col_var (× tab_vars) combination, with the count in counts (and the weighted count in wt_counts).
- **A wide** data.frame: a label (row_var) column plus one column per col_var level — select those level columns with cols and name the column variable with col_name.
- **A** table **/** xtabs **/** matrix **object**: melted automatically, the row and column variables read from the dimnames (or set with row_var / col_var).
- **Frequencies + base N**: the wide shape, plus input = "pct" and base (the column of row sample sizes); the integer counts are rebuilt from the percentages and the base.

With weighted data, give the real (unweighted) count in counts **and** the weighted count in wt_counts: estimates use the weighted counts, while confidence intervals and tests use the real unweighted sample size. Counts that are not whole numbers (weighted-only or frequency-only input) disable confidence intervals and chi-squared, with a message.

Usage

```
tab_counts(
  data,
  row_var,
  col_var,
  tab_vars,
  counts,
  wt_counts,
  cols,
  col_name = "variable",
  base,
  input = c("counts", "pct"),
  ...
)
```

Arguments

data	A data frame.
row_var, col_var	**[Deprecated]** Singular aliases of row_vars/col_vars (which now accept several variables). Kept working.
tab_vars	<tidy-select> Tab variables: one subtable per combination of their levels. Leave empty for a simple cross-table.
counts	The column holding the **unweighted** count for each cell (long tidy shape).
wt_counts	Optional column holding the **weighted** count for each cell. Leave empty for an unweighted table.
cols	<tidy-select> For a wide data.frame: the columns holding the col_var levels.
col_name	Name of the (synthesised) column variable when cols is used.
base	For input = "pct": the column holding each row's sample size N.
input	"counts" (default) or "pct" (with cols and base: the level columns hold frequencies, and counts are rebuilt from them and base).
...	Every other argument of tab() – pct, color, ci, tot, ... – passed by name. See tab(); a typo gets a suggestion.

Value

A tabxplor_tab (or tabxplor_grouped_tab when tab_vars are provided).

Examples

```
# Long tidy counts (as from dplyr::count()) reproduce the microdata table :
counts <- dplyr::count(forcats::gss_cat, marital, race)
tab_counts(counts, marital, race, counts = n, pct = "row")
# tab(forcats::gss_cat, marital, race, pct = "row")   # identical

# A contingency table object :
```

```
tab_counts(table(forcats::gss_cat$marital, forcats::gss_cat$race), pct = "row")

# A wide data.frame of counts :
wide <- tidyr::pivot_wider(counts, names_from = race, values_from = n)
tab_counts(wide, row_var = marital, cols = c(Other, Black, White),
           col_name = "race", pct = "row")
```

---

tab_css *The stylesheet an html table needs*

Description

The CSS that colours tabxplor tables. It is a **constant** – a pure function of the colour palette, the channel type and the theme – so it does not take a table: one stylesheet styles every table in a document, whatever their `color_breaks`.

Usage

```
tab_css(
  theme = NULL,
  format = c("html", "md"),
  style_tag = TRUE,
  file = NULL,
  print_rules = NULL,
  ...
)
```

Arguments

`theme`	`"light"`, `"dark"`, a black-and-white publication palette (`"print_ready"`, `"print_marks"`, `"print_emphasis"`, `"print_minimalistic"`; `"bw"` is a synonym of the last – see the section below), or – opt-in – `"auto"` to follow the reader's colour scheme (their operating system, and any dark-mode toggle of the host page: Quarto, Bootstrap 5.3, Tailwind). Defaults to `getOption("tabxplor.theme")`, i.e. `"light"`: a dark table is always a deliberate choice. `"auto"` emits every rule four times (a light base, the OS media query, then both toggle directions), which is also what lets `tab_html()`'s own Viewer page force the editor's theme.
`format`	Which output the stylesheet is for, in `tab_export()`'s own vocabulary. `"html"` (the default) is the full stylesheet `tab_html()` needs: the colour classes **and** the table's own look (font, background, border colours, the greys). `"md"` emits the colour classes only, which is what `tab_md()` wants — bare selectors you can map in your own editor's or publisher's CSS.
`style_tag`	Wrap the CSS in a `<style>` tag (default `TRUE`).
`file`	Optional path to write to instead of returning.

`print_rules` Also emit a black-and-white publication palette inside an `@media print` block, so a coloured page prints (or saves to PDF) publication-ready with no further action. Defaults to `getOption("tabxplor.print_rules")`. Set to `FALSE` if your printer is a colour one and the colours are the point, or name a palette (`"print_emphasis"`) to print in that one. `"print_marks"` cannot be used here: its marks are cell text, and a print rule can restyle a page but not add characters to it. It adds roughly 1.5 KB to a `light/dark` stylesheet and 6 KB to an `"auto"` one. `FALSE` drops the whole block, including the two declarations that are not about colour: the one that makes a background fill reach the paper, and the one that stops the scrollbox clipping a wide table there.

`...` Retired arguments, accepted and ignored with a deprecation message since 2.0.0 (`color_type`): the text channel always uses the text palette, and the colour CHANNEL is chosen by `color = c(text, background)` (see `tab()`). Anything else is an error naming the argument you meant, as it already was in `tab()`.

Details

Cells carry classes named after the palette **slot** (`.p1-.p4` over-represented text, `.m1-.m4` under-represented text, `.o1-.o4` / `.u1-.u4` for the background channel), so `tab_html()` and `tab_md()` share one vocabulary.

Value

The CSS, invisibly when `file` is given. Printed as-is by `knitr` with `results = "asis"`.

Two workflows

Self-contained (the default). `tab_html(css = TRUE)` and `tab_md(css = TRUE)` inline the stylesheet with the table, so a single file works anywhere (the RStudio/Positron Viewer, jamovi, a standalone `.html`). Nothing to do.

Once per document. In an `.Rmd`/`.qmd` with many tables, emit it once and let every table reuse it:

```
```{r, results = "asis"}
options(tabxplor.tab_kable_css = FALSE)
tab_css(theme = "auto")
```
```

Every later `tab_html()` then emits classes only. Two things to know: with `css = FALSE` and **no** `tab_css()` call the tables render uncoloured; and one stylesheet means one colour `theme` for the whole document. A black-and-white table (`theme = "print_ready"` or a publication palette on the table's own call) still renders as such under it: every stylesheet carries the publication palettes, scoped to the tables that wear them.

Restyling a table

Nothing is written inline on a cell, so **any** of the look can be overridden by adding your own rules after the stylesheet – no `!important` needed. The cell colour classes are also emitted scoped (`.tabxplor-tab .p1`) so they survive host pages that style table cells themselves, such as Bootstrap-based sites including pkgdown. Column widths in particular are left to the browser, which sizes each column to its content; to pin one, style its role:

```
.tabxplor-tab .tx-rv  { min-width: 10em; }   /* the row-variable levels column */
.tabxplor-tab .tx-tot { min-width: 5.5em; }  /* total columns                  */
.tabxplor-tab .tx-num { min-width: 4em; }    /* every number column            */
```

The roles a cell can carry: `.tx-l`/`.tx-r` (alignment), `.tx-num` (numbers), `.tx-rv` (the row-variable levels column), `.tx-tot` (total columns), `.tx-bl`/`.tx-br` (side borders), `.tx-b` (bold), `.tx-lbl`/`.tx-vname` (a variable name spanning its block), `.tx-pill` (a background-coloured value), `.tx-span` (the variable-name header row), `.tx-foot` (the footnote). Rows carry `.tx-bt`/`.tx-bb`/`.tx-bb2` (top / bottom / thick-bottom rules).

Each html table is wrapped in a `.tx-scrollbox`, which scrolls it sideways rather than let it widen the page. Its title stays outside, so it does not scroll away, and the `@media print` block lifts the clip (a printer has no scrollbar). To let a table widen the page on screen too: `.tx-scrollbox { overflow-x: visible; max-widt`

The black-and-white publication palettes

A greyscale print loses colour entirely — both direction ramps become the same grey — so these palettes say the same thing with something else. `theme = "print_ready"` **is the one to reach for**: it picks per table, the marks for a cross-table and the emphasis ladder for a regression, whose cells already carry their own direction symbol. Name one yourself to override that. They share ONE grey fill ramp (a background colour measure keeps carrying its magnitude) and differ in the text channel:

`"print_minimalistic"` direction by underline (over) and italic (under); magnitude by an ink ladder.

`"print_emphasis"` magnitude by an emphasis ladder (bold, then underline, then double underline) in pure black; direction by the cell's own measure symbol, plus italic under the null.

`"print_marks"` magnitude and direction by a repeated superscript mark after the value (no significance stars: the marks take their place, do not use with `tab_reg()`).

In all of them a non-significant cell is greyed out, and the significance stars stay — except under `"print_marks"`, where the marks take their place (one run of symbols after a value, not two). `"bw"` is a synonym of `"print_minimalistic"`. *One caveat*, and only for a document that emits `tab_css()` once and renders its tables with `css = FALSE`: a stylesheet is table-independent, so it carries ONE of them. A cross-table is fine whatever it carries (its marks are cell text), but a regression's ladder is css and nothing else — name it there, `tab_css(theme = "print_emphasis")`.

See Also

`tab_html()`, `tab_md()`, `set_color_palette()`, `set_color_breaks()`

Examples

```
cat(tab_css(theme = "auto"))
cat(tab_css(format = "md", style_tag = FALSE))  # the markdown flavour
```

tab_export *Export a table to html, Excel or Markdown (wrapper)*

Description

One entry point over the format-specific exporters tab_html (HTML), tab_md (Markdown), tab_xl (Excel) and forest_plot (a chart of the estimates). They share one set of display-option names and defaults; tab_export() forwards them and passes any format-specific argument through ....

Usage

```
tab_export(
  x,
  format = c("html", "md", "xl", "forest"),
  path = NULL,
  theme = NULL,
  color = TRUE,
  color_legend = TRUE,
  lang = NULL,
  transpose = FALSE,
  caption = NULL,
  var_names = NULL,
  ...
)
```

Arguments

x	A table made with tab or tab_reg, or a list of tab. A list of tables sharing the same col_vars (and no tab_vars) is merged into one; any other list — several row_vars and/or tab_vars — is rendered one table after another, each keeping its own sub-tables.
format	One of "html" (the default), "md" (Markdown), "xl" (Excel) or "forest" (a forest plot of the estimates, see forest_plot).
path	Optional output file. For "xl" it is the workbook path; for "md" and "html" the rendered text is written to it; ignored for "forest".
theme	By default ("light") a white table with black text; "dark" for the inverse (the colours follow the theme). "auto" follows the reader's colour scheme (their operating system, and any dark-mode toggle of the host page); it needs a stylesheet, so it works for format = "html" and "md" and resolves to "light" for the static "xl" backend. The black-and-white **publication** palettes render a table for a page that has no colour: "print_ready" picks the right one per table, or name it yourself – "print_marks", "print_emphasis", "print_minimalistic" ("bw"). Defaults to getOption("tabxplor.theme"). See tab_css for what each says.
color	Set to FALSE to render the table without colours (monochrome).
color_legend	Print the colour legend below the table (with the subtext). TRUE by default, and a no-op on a table that carries no colours.

lang	Colour-legend language: NULL (auto from the R/OS locale, English fallback), "en" or "fr".
transpose	Set to TRUE to transpose each table before export (rows become columns) – the col-percentages-with-several-row-variables use case.
caption	A single caption / title for the table.
var_names	Which variable names to write beside the table: "both" (the default), "rows", "cols" or "none". The row-variable name is the leading column a table with several row_vars uses to name each block (written once per block); the column-variable names are the spanning row above their level columns. Level headers always keep their name.
...	Format-specific arguments passed to the underlying exporter. Retired arguments (color_type, html_24_bit, engine, html_font, full_width, position, n_min, hide_near_zero) are caught here, reported once, and not forwarded; the exporter this hands to refuses anything else it cannot use.

Details

Each exporter is also callable on its own, which reads better in a pipe (x |> tab_xl()); use tab_export() when the format comes from a variable.

Value

The value of the underlying exporter: an HTML/knitr object ("html"), a markdown string ("md"), x invisibly with the Excel file written ("xl"), or a ggplot ("forest").

Examples

```
tabs <- tab(forcats::gss_cat, race, marital, pct = "row", color = "difference")
tab_export(tabs, "md")
```

---

tab_footer_text *Read the footer a table would print*

Description

The lines **tabxplor** prints under a table, resolved: the template's placeholders built, the user's own lines as written. It is what set_subtext() edits, after the fact.

Usage

```
tab_footer_text(
  x,
  medium = c("plain", "console", "html", "md"),
  style = NULL,
  lang = NULL,
  theme = NULL
)
```

Arguments

x	A tabxplor_tab.
medium	One of "plain" (default), "console", "html", "md".
style	"terse" (the console's compact one-liner) or "prose" (full sentences, the exports' default); NULL follows the medium.
lang	NULL (from getOption("tabxplor.lang")), "en" or "fr".
theme	Palette theme; NULL follows the medium's option.

Value

A character vector, one element per footer line.

See Also

set_subtext() for the template, set_legend_words() for the words.

Examples

```
t <- tab(forcats::gss_cat, race, marital, pct = "row", color = "diff")
cat(tab_footer_text(t), sep = "\n")
```

---

tab_html *Render a table as html*

Description

The HTML exporter behind tab_export: tab_export(x, format = "html") calls this, and tab_kable() is a permanent alias of tab_html(). Use it directly for HTML-specific arguments.

Usage

```
tab_html(
  tabs,
  theme = NULL,
  color = TRUE,
  tooltips = NULL,
  popover = NULL,
  color_legend = TRUE,
  lang = NULL,
  caption = NULL,
  transpose = FALSE,
  var_names = NULL,
  get_data = FALSE,
  cells = NULL,
  wrap_rows = 35,
  wrap_cols = 15,
```

```
  whitespace_only = TRUE,
  css = NULL,
  ...
)

tab_kable(
  tabs,
  theme = NULL,
  color = TRUE,
  tooltips = NULL,
  popover = NULL,
  color_legend = TRUE,
  lang = NULL,
  caption = NULL,
  transpose = FALSE,
  var_names = NULL,
  get_data = FALSE,
  cells = NULL,
  wrap_rows = 35,
  wrap_cols = 15,
  whitespace_only = TRUE,
  css = NULL,
  ...
)
```

Arguments

`tabs` A table made with `tab` or `tab_reg`, or a `list` of tab. A list of tables sharing the same `col_vars` (and no `tab_vars`) is merged into one; any other list — several `row_vars` and/or `tab_vars` — is rendered one table after another, each keeping its own sub-tables.

`theme` By default (`"light"`) a white table with black text; `"dark"` for a black table with white text; `"auto"` (opt-in) to follow whoever is **reading** the table:

- in a file or a knitted document, the reader's browser decides – their operating system, plus any dark-mode toggle of the host page (Quarto, Bootstrap 5.3, Tailwind);
- printed to the **Viewer**, your editor decides. Its webview reports the operating system rather than the editor's colour theme, so the theme is resolved in R instead (RStudio's, or Positron's, best-effort).

Defaults to `getOption("tabxplor.theme")`, i.e. `"light"` – a dark table is always a deliberate choice.

`"print_ready"`, `"print_marks"`, `"print_emphasis"` and `"print_minimalistic"` (`"bw"`) are the black-and-white **publication** palettes: a greyscale print loses the colour palette's direction entirely (both ramps convert to the same greys), so each says it with something else — see `tab_css` for what. Their typography is written as real <b>/<i>/<u> markup as well as CSS, so it survives a stylesheet-less destination (a paste into Word, GitHub's markdown). You rarely

need to ask for one: any coloured table already **prints** in the first, see tab_css's print_rules.

color — Set to FALSE to render the table without colours (monochrome).

tooltips — By default, takes getOption("tabxplor.tab_kable_tooltips") (TRUE unless set): html tooltips display additional informations at mouse hover. Set to FALSE to discard (or set the option to FALSE once per document, e.g. in a vignette or report where every table auto-prints).

popover — By default, takes getOption("tabxplor.kable_popover"). When FALSE, html tooltips are of the base kind: they can't be used with a floating table of contents in **rmarkdown** documents. Set to TRUE for click popovers instead, which are compatible with a floating toc. Both are bound automatically, in the Viewer and in a knitted document alike, provided **rmarkdown** and **htmltools** are installed.

color_legend — Print the colour legend below the table (with the subtext). TRUE by default, and a no-op on a table that carries no colours.

lang — Colour-legend language: NULL (auto from the R/OS locale, English fallback), "en" or "fr".

caption — The table caption. For formatting, you need to use a css with caption{}in rmarkdown.

transpose — Set to TRUE to transpose each table before export (rows become columns) – the col-percentages-with-several-row-variables use case.

var_names — Which variable names to write beside the table: "both" (the default), "rows", "cols" or "none". The row-variable name is the leading column a table with several row_vars uses to name each block (written once per block); the column-variable names are the spanning row above their level columns. Level headers always keep their name.

get_data — Get the transformed data instead of the html table.

cells — The write side of get_data: the same data.frame, with the cells you want to replace edited. A value still equal to the one the table renders means "keep", so handing the frame straight back changes nothing; anything else is written verbatim into that cell, markup and all. The cell keeps its classes and its tooltip, and loses the decorations that belonged to the text it replaced (the bold split, the background pill, the sparkline). Pass a list of one data.frame per table when several are rendered at once.

wrap_rows — Row labels are wrapped past this width (35 characters by default).

wrap_cols — Column headers are wrapped past this width (15 characters by default), at the seams a compound name is built from (_, ., *, a camelCase boundary). A col_var's spanning name is measured against the width its own columns leave it first: past that it wraps, and past what wrapping can do it is shown from the prefix it shares with the block before it (MUS_CONCERT_CLASSIQUE, then _ROCK), and held to this width in the last resort — never while there is room for the whole name.

whitespace_only — Set to FALSE to wrap also on non whitespace characters.

`css` Inline the stylesheet with the table, so the output is self-contained (default, from `getOption("tabxplor.tab_kable_css")`). Set `FALSE` in a many-table document that emits `tab_css` once at the top – the stylesheet is table-independent, so one copy styles every table. With `FALSE` and no `tab_css` call, tables render uncoloured.

`...` Retired arguments, accepted and ignored with a deprecation message since 2.0.0: `color_type`, `html_24_bit`, `engine`, `html_font`, `full_width`, `position`. The table is rendered by one dependency-free `<table>` engine whose every look is a CSS class you can restyle – font, width, colour and placement are all `tab_css`'s business now. Anything else is an error naming the argument you meant, as it already was in `tab`.

Details

`tab_kable()` is a permanent alias of `tab_html()` – the two are identical. `tab_html()` names the output (an HTML table); `tab_kable()` is the name it had when **kableExtra** rendered it.

Value

A html table. Printing it opens it in the Viewer, on a page painted to match the table – so a `theme = "dark"` table no longer sits in a white pane. Everything the cell has no room for – the confidence interval, the exact p-value, the other ways of reading the same comparison (difference, ratio, odds ratio), the chi-squared contribution and the base count – is one hover away, each line named after the field it shows.

Examples

```
tabs <- tab(forcats::gss_cat, race, marital, year, pct = "row", color = "difference")
tab_html(tabs, theme = "light")
```

`tab_many` *Many cross-tables as one, with color helpers*

Description

[Superseded]

Superseded (2.0.0) by `tab()`, the unified entry point: it accepts several `row_vars` / `col_vars` and merges them into one table by default (`output_list = TRUE` gives the list shape `tab_many()` returns).

`tab_many()` forwards everything to `tab()`, translating the five renamed arguments:

`tab_many()`	`tab()`
`chi2 = TRUE`	`test = TRUE`
`totrow = FALSE`	`tot = "col"`
`totcol = "no"`	`tot = "row"`

```
compact = TRUE          output_list = FALSE
na_drop_all = c(a, b)   filter = !is.na(a) & !is.na(b)
```

Everything else keeps its name and meaning — see `tab()`.

Usage

```
tab_many(
  data,
  row_vars,
  col_vars,
  tab_vars,
  wt,
  ...,
  chi2,
  totrow,
  totcol,
  compact,
  na_drop_all,
  filter
)
```

Arguments

`data` A data frame, or a `survey::svydesign`.

`row_vars`, `col_vars`, `tab_vars`, `wt`
: The variable roles — see `tab()`. With `data`, the only arguments that may be passed by position: everything else must be named, because `tab()`'s argument order differs.

`...` Passed on to `tab()`.

`chi2` **[Deprecated]** Use `tab()`'s `test`.

`totrow`, `totcol` **[Deprecated]** Use `tab()`'s `tot`. A total row is always computed and exactly one total column is shown, so both are cosmetic; `totcol = "each"` and `"all_col_vars"` now give that same single total column instead of erroring.

`compact` **[Deprecated]** Use `tab()`'s `output_list` (inverted).

`na_drop_all` **[Deprecated]** <tidy-select> Use `tab()`'s `filter`: `na_drop_all = c(a, b)` is `filter = !is.na(a) & !is.na(b)`.

`filter` **[Superseded]** A `dplyr::filter` to apply to the data frame first — see `tab()`. Prefer filtering upstream.

Value

What `tab()` returns: a `tabxplor_tab` (a `tabxplor_grouped_tab` with `tab_vars`), or a `tabxplor_tabs` list under `output_list = TRUE` / `compact = FALSE`.

Examples

```
# Make a summary table with many col_vars, showing only one specific level :

library(dplyr)
first_lvs <- c("Married", "$25000 or more", "Strong republican", "Protestant")
data <- forcats::gss_cat |> mutate(across(
  where(is.factor),
  ~ forcats::fct_relevel(., first_lvs[first_lvs %in% levels(.)])
))
tab_many(data, race, c(marital, rincome, partyid, relig, age, tvhours),
         levels = "first", pct = "row", test = TRUE, color = "auto")


# Can be used with map and tribble to program several tables with different parameters
#  all at once, in a readable way:

library(purrr)
library(tibble)
pmap(
  tribble(
    ~row_vars, ~col_vars      , ~pct , ~filter              , ~subtext               ,
    "race"  , "marital"       , "row", NULL                 , "Source: GSS 2000-2014",
    "relig" , c("race", "age"), "row", "year %in% 2000:2010", "Source: GSS 2000-2010",
    NA_character_, "race"     , "no" , NULL                 , "Source: GSS 2000-2014",
  ),
  .f = tab_many,
  data = forcats::gss_cat, color = "auto", test = TRUE)
```

---

tab_md *Render a table as Markdown*

Description

The Markdown exporter behind tab_export: `tab_export(x, format = "md")` calls this.

Usage

```
tab_md(
  tabs,
  bold_references = TRUE,
  special_formatting = TRUE,
  wrap_rows = NULL,
  subtext = TRUE,
  color = TRUE,
  color_legend = TRUE,
  lang = NULL,
  theme = NULL,
  caption = NULL,
```

```
  transpose = FALSE,
  var_names = NULL,
  css = NULL,
  clipboard = FALSE,
  file = NULL,
  print = NULL,
  title = lifecycle::deprecated(),
  col_var_names = lifecycle::deprecated(),
  ...
)
```

Arguments

- `tabs` — A table made with `tab` or `tab_reg`, or a `list` of tab. A list of tables sharing the same `col_vars` (and no `tab_vars`) is merged into one; any other list — several `row_vars` and/or `tab_vars` — is rendered one table after another, each keeping its own sub-tables.
- `bold_references` — Bold reference/total rows with markdown `**...**`.
- `special_formatting` — Passed to `format()`. When `TRUE`, shows "ref:" prefix on diff reference cells, "mean:" on ctr totals, sigma on means.
- `wrap_rows` — Max width for row labels before truncation. `NULL` (default) never truncates (lossless – the column grows); set a number to cap the label width. A markdown pipe cell cannot hold a raw newline, so md "wrapping" means "do not truncate".
- `subtext` — Print chi2/footnotes below the table.
- `color` — When `TRUE` (default) and the table carries colours (e.g. built with `tab(..., color = "difference")`), each fmt cell is wrapped in a short pandoc bracketed span `[value]{.class}` so the markdown renders coloured in Quarto / RMarkdown / pandoc (and `tab_css(format = "md")` styles the classes). `FALSE` produces plain monochrome markdown. Uncoloured tables never get spans.
- `color_legend` — Print the colour legend below the table (with the subtext). `TRUE` by default, and a no-op on a table that carries no colours.
- `lang` — Colour-legend language: `NULL` (auto from the R/OS locale, English fallback), `"en"` or `"fr"`.
- `theme` — Colour palette selector (as in `tab_html`); it only affects the CSS emitted by `css = TRUE` / `tab_css`, since the span *class names* are palette- and theme-independent. Accepts `"auto"` (follow the reader's colour scheme).
- `caption` — Optional table caption, rendered as a pandoc caption line `: caption` (captions only the first table of a list).
- `transpose` — Set to `TRUE` to transpose each table before export (rows become columns) – the col-percentages-with-several-row-variables use case.
- `var_names` — Which variable names to write beside the table: `"both"` (the default), `"rows"`, `"cols"` or `"none"`. The row-variable name is the leading column a table with several `row_vars` uses to name each block (written once per block); the column-variable names are the spanning row above their level columns. Level headers always keep their name.

`css`	Prepend an inline `<style>` block so the exported markdown is self-contained and renders coloured and compact on its own (default, from `getOption("tabxplor.tab_kable_css")`). Set `FALSE` inside an `.Rmd`/`.qmd` document once the host page brings the stylesheet (or call `tab_css` once at the top for the whole document) – otherwise the `<style>` block is duplicated per table. A plain uncoloured table renders byte-identical either way.
`clipboard`	Copy output to clipboard via `clipr::write_clip()` (requires **clipr**).
`file`	Path to write the markdown to a file. `NULL` (default) skips.
`print`	By default (`NULL`), `cat()` the markdown and return it invisibly – except while a document is being knitted, where the object is returned instead, so the chunk emits raw markdown rather than a verbatim block. `TRUE` or `FALSE` forces either.
`title`	**[Deprecated]** Renamed to `caption`.
`col_var_names`	**[Deprecated]** Replaced by `var_names`: `col_var_names = FALSE` is `var_names = "rows"` (or `"none"`).
`...`	Retired arguments, accepted and ignored with a deprecation message since 2.0.0 (`color_type`, `html_24_bit`): colour is a CSS class, and exports are always 24-bit. Anything else is an error naming the argument you meant, as it already was in `tab()`.

Value

A `tabxplor_md`: the markdown as a character string, which prints as the text it is and reaches a knitted document raw.

Examples

```
tab(forcats::gss_cat, race, marital, pct = "row") |> tab_md()
tab(forcats::gss_cat, race, marital, pct = "row", color = "difference") |> tab_md()
tab(forcats::gss_cat, race, marital, pct = "row", color = "difference") |>
  dplyr::mutate(dplyr::across(dplyr::where(is_fmt), ~set_display(., "diff"))) |>
  tab_md()
```

`tab_note` *A note under a table*

Description

A small grid of **already-rendered** character columns, printed under a table in the aside ink: a glossary, a range, a set of diagnostics — something that belongs to the table without being a row of it. Attach it with `set_footer_tabs`, beside (or instead of) a real table.

A plain `data.frame` passed to `set_footer_tabs()` already renders as a note, its own names as headers, everything left-aligned. `tab_note()` is for when that is not enough.

Usage

```
tab_note(
  df,
  headers = NULL,
  align = NULL,
  grey = NULL,
  note = character(0),
  kind = NULL
)
```

Arguments

df	A data.frame of character columns, already formatted.
headers	Column titles; defaults to `names(df)`.
align	One of `"left"` / `"right"` per column; defaults to all left.
grey	One logical per row: a row to render in the dimmer aside ink (a result the note itself marks as not to be read). `NULL` for none.
note	One or more lines printed under the grid, smaller still — what a cell cannot say for itself.
kind	Per column, `"text"` (default) or `"spark"`, a run of block glyphs the html back-end upgrades to an inline `<svg>`.

Value

A `tabxplor_note`.

See Also

`set_footer_tabs()` to attach one, `set_subtext()` for the footer's text.

Examples

```
n <- tab_note(data.frame(axis = c("1", "2"), variance = c("9.9%", "7.2%")),
              headers = c("Axis", "% variance"), align = c("left", "right"))
set_footer_tabs(tab(forcats::gss_cat, race, marital, pct = "row"), list("Axes" = n))
```

---

tab_num *Means table*

Description

[Superseded]

Cross categorical variables with numeric ones, and get a table of means. Superseded by `tab()`, which builds the same table whenever `col_vars` holds numeric variables – and everything around it (both kinds of variable at once, colours, totals, tests). It stays the smallest entry point into the numeric aggregate core, and takes the same arguments resolved by the same rules, so its numbers agree with `tab()`'s cell for cell.

Usage

```
tab_num(
  data,
  row_var,
  col_vars,
  tab_vars,
  wt,
  ...,
  num = FALSE,
  df = FALSE,
  .fine = NULL,
  .by_table = FALSE
)
```

Arguments

data	A data frame.
row_var	**[Deprecated]** Singular aliases of row_vars/col_vars (which now accept several variables). Kept working.
col_vars	<tidy-select> The column variable(s) — see row_vars. **An interaction** is written a*b, as in tab_reg(), and only col_vars takes one: two factors give one column per observed cell of the pair, a number crossed with a factor one mean column per level. See vignette("tabxplor").
tab_vars	<tidy-select> Tab variables: one subtable per combination of their levels. Leave empty for a simple cross-table.
wt	A weight variable, of class numeric. Leave empty for unweighted results.
...	Every other argument of tab() – color, ci, tot, digits, ... – passed by name. See tab(); a typo gets a suggestion.
num	Set to TRUE to obtain a table with normal numeric vectors (not fmt).
df	Set to TRUE to obtain a plain data.frame (not a tibble), with normal numeric vectors (not fmt). Useful, for example, to pass the table to correspondence analysis with **FactoMineR**.
.fine, .by_table	Internal. .fine is a pre-computed count-aggregate to roll up from instead of scanning the raw data (used by tab_counts and the scan-fusion path); .by_table forces the table-by-table path.

Value

A tibble of class tabxplor_tab. If ... (tab_vars) are provided, a tab of class tabxplor_grouped_tab. All non-text columns are fmt vectors of class tabxplor_fmt, storing all the data necessary to print formats and colors. Columns with row_var and tab_vars are of class factor : every added factor will be considered as a tab_vars and used for grouping. To add text columns without using them in calculations, be sure they are of class character.

Examples

```
data <- dplyr::storms |> dplyr::filter(!is.na(wind))
tab_num(data, category, wind, tot = "row",
        color = "difference", color_signif = "guaranteed_effect")
```

---

tab_parallel_stop *Stop the tabxplor parallel worker pool*

Description

Shuts down the persistent **mirai** daemons tabxplor starts under `options(tabxplor.parallel = )`. The pool is otherwise reused for the whole session and cleaned up when the package is unloaded; call this to release the workers earlier.

Usage

```
tab_parallel_stop()
```

Value

`invisible(NULL)`, called for its side effect.

See Also

tabxplor-options for `tabxplor.parallel`, the switch that starts the pool.

Examples

```
# after options(tabxplor.parallel = TRUE)
tab_parallel_stop()
```

---

tab_pipe *Render a table as a plain pipe table*

Description

The markdown grid without the markup: one GFM pipe table, its unit line kept (`<col%>`, `<var>`), no colour spans, no footer and no stylesheet. It is what the console prints under a table for each of its subordinate tables (`set_footer_tabs`), the same shape a regression's *shape table* takes there — a grid one can read as text and paste anywhere.

It is `tab_md` with three arguments fixed, not a second renderer: a pipe table that drifted from the markdown export would be a second answer to one question.

Usage

```
tab_pipe(tabs, ...)
```

Arguments

tabs	A tabxplor_tab, or a list of them.
...	Passed to tab_md — color = TRUE brings the colour spans back, subtext = TRUE the footer.

Value

A character vector, one element per line.

See Also

tab_md(), set_footer_tabs().

Examples

```
cat(tab_pipe(tab(forcats::gss_cat, race, marital, pct = "row")), sep = "\n")
```

---

tab_plain *Plain single cross-table*

Description

[Superseded]

One bare cross-table of counts or percentages, from ONE row variable and ONE column variable. Superseded by tab(), which does the same and everything around it (several variables, colours, totals, tests) – but it stays the smallest entry point into the aggregate core, and takes the same ci / ci_method / conf_level / stars / display arguments, resolved by the same rules, so its numbers agree with tab()'s cell for cell.

Usage

```
tab_plain(
  data,
  row_var,
  col_var,
  tab_vars,
  wt,
  ...,
  num = FALSE,
  df = FALSE,
  .fine = NULL,
  .by_table = FALSE
)
```

Arguments

data	A data frame.
row_var, col_var	**[Deprecated]** Singular aliases of row_vars/col_vars (which now accept several variables). Kept working.
tab_vars	<tidy-select> Tab variables: one subtable per combination of their levels. Leave empty for a simple cross-table.
wt	A weight variable, of class numeric. Leave empty for unweighted results.
...	Every other argument of tab() – pct, color, ci, tot, ... – passed by name. See tab(); a typo gets a suggestion.
num	Set to TRUE to obtain a table with normal numeric vectors (not fmt).
df	Set to TRUE to obtain a plain data.frame (not a tibble), with normal numeric vectors (not fmt). Useful, for example, to pass the table to correspondence analysis with **FactoMineR**.
.fine, .by_table	Internal. .fine is a pre-computed count-aggregate to roll up from instead of scanning the raw data (used by tab_counts and the scan-fusion path); .by_table forces the table-by-table path.

Value

A tibble of class tabxplor_tab. If ... (tab_vars) are provided, a tab of class tabxplor_grouped_tab. All non-text columns are fmt vectors of class tabxplor_fmt, storing all the data necessary to print formats and colors. Columns with row_var and tab_vars are of class factor : every added factor will be considered as a tab_vars and used for grouping. To add text columns without using them in calculations, be sure they are of class character.

Examples

```
# the leaf builds the cells AND their intervals: `ci` is resolved here exactly as in tab(),
# so tab_plain(ci = "ref") and tab(ci = "ref") agree cell for cell.
dplyr::starwars |>
  tab_plain(sex, hair_color, tot = c("row", "col"), pct = "row",
            ci = "ref", color = "difference", color_signif = "grey_non_signif")
```

---

tab_reg — *All-in-one tables for regressions, with each modelled effect beside its observed one*

Description

Fits one regression model per column and returns a tabxplor table of the per-family effect measure — a linear **mean difference** (gaussian), **odds ratios** (binomial), **ratios of means** (poisson counts), one **odds-ratio column per outcome category** (nominal 3+ level), a **cumulative odds ratio** (ordinal) — one row per predictor level, grouped by predictor, with the **observed (crude)** effect beside each adjusted one. Each cell stores its estimate, interval and p-value, so the table prints with stars, greys what is not significant, and exports like any tabxplor crosstab.

Usage

```
tab_reg(
  data,
  outcome,
  predictors = NULL,
  tab_vars = NULL,
  wt = NULL,
  family = "auto",
  link = "auto",
  measure = "auto",
  effect = "auto",
  outcome_level = NULL,
  trials = NULL,
  empirical = TRUE,
  n = NULL,
  color = "measure",
  color_signif = NULL,
  stars = TRUE,
  ref = NULL,
  multiplier = "2sd",
  shape = NULL,
  stats = "auto",
  conf_level = NULL,
  na = c("drop_by_outcome", "drop_by_model", "drop_all", "keep_for_predictors"),
  display = NULL,
  digits = 0,
  cleannames = NULL,
  subtext = "",
  caption = NULL,
  ...
)
```

Arguments

`data` A data frame, **or a prebuilt survey design** (`survey::svydesign()`). A design's weights, clustering, stratification and calibration drive the estimation, and `wt` is ignored.

`outcome` <`tidy-select`> The outcome variable(s) — bare names, quoted names, or any selection helper, exactly as in `tab()` — **or a model formula** (the escape hatch; leave `predictors` unset). Several names give one effect column per outcome; with a `predictors` list, a single name is required. `reg_formulas()` shows what was fitted.

`predictors` <`tidy-select`> The predictors of one model — or a **named list**, one model per element, its name labelling the column, each element selected on its own (`list(m1 = c(race, age), m2 = starts_with("inc"))`), which is how models are compared. Leave `NULL` when `outcome` is a formula. A bare name is a column of `data` first, then an object, so a variable holding names works without `all_of()`.

a*b **is an interaction**, R's own spelling, bare or quoted — *"a's effect, allowed to vary with b"*. Two categorical variables give one row per **cell** of the pair; a continuous a gives its **slope within each level** of b. An interaction supplies both its variables, so do not list them beside it, which is what makes "with and without" an ordinary model comparison. a:b, which drops the main effects, is refused. See the regression vignette.

tab_vars

<tidy-select> Optional. One grouping variable — the same argument as tab()'s: one sub-table per group, the same model(s) fitted **within each level**. Two readings of "does this effect hold in every subgroup?" come with it: color = "between_groups" colours and tests each effect against the first group's, row by row, and stats = c(..., "group_interaction") adds the aggregated test, once per predictor. For an interaction between two PREDICTORS of one model, write it in predictors as a*b instead.

wt

<tidy-select> Optional. One weight column, switching to design-based survey estimation (survey::svyglm()). For clustering, stratification, a finite-population correction or calibration, build the design with survey::svydesign() and pass it as data. See vignette("tabxplor-weights").

family

The model family, **resolved per outcome** so several outcomes with different families can share one table. "auto" (default) detects each one and says so: a binary outcome gives "binomial", an ordered 3+ level "ordinal", a nominal 3+ level "multinomial", any other numeric "gaussian" — a genuine count is yours to name. Or set it: "gaussian" (linear), "binomial" (logistic), "poisson" / "quasipoisson" (counts), "multinomial", "ordinal". A **scalar** applies to every outcome; a **vector** aligned to outcome, or a **named** vector keyed by outcome (e.g. c(income = "poisson", satisfied = "binomial")), sets one family per outcome. Mixed families need a character predictors.

It answers **one** question — what kind of number the outcome is — and never picks a link behind your back: on a binary outcome family = "poisson" is refused, naming the two things it could have meant, link = "ratio" and measure = "ratio".

link

Which measure the model estimates — the only argument that changes the model. A link *is* a measure (the one a model gives you directly), so it takes measure's own words:

- "auto" (default) — the family's own: logistic for a binary outcome, linear for a quantity, Poisson for a count.
- "odds_ratio" — the logit fit (binomial, multinomial, ordinal).
- "ratio" — the log link: the **modified Poisson** on a binary outcome (a conditional risk ratio), Poisson on a count, Poisson pseudo-likelihood on a continuous one.
- "difference" — the identity link; on a binary outcome the **risk difference**. It can fail to converge, and the linear probability model then takes over, with a message.

Reach for it when you want the model's *coefficient* to be that measure; to report a measure without changing the model, set measure instead. "log" is the one word the two arguments do not share: here the **log link**, on measure a spelling of "raw_coefficient".

`measure` **Which measure of deviation is reported** — a deviation being how far a group sits from the reference, the measure which of the ways of expressing it you read. The one argument most readers ever set, and the one that never changes the model. `"auto"` (default) is the model's own. The full word is canonical, the discipline's acronym a synonym:

- `"odds_ratio"` (`"OR"`) — the odds of the outcome, times what.
- `"ratio"` (`"RR"`, `"RoM"`) — how many times as likely, as frequent, as large. Reach for it when the outcome is **common**, where an odds ratio is far from the risk ratio people hear in it, and because a risk ratio stays comparable across nested models.
- `"difference"` (`"RD"`, `"diff"`) — how much more, in the outcome's own units.
- `"raw_coefficient"` (`"coef"`, `"log"`, ...) — the model's own coefficient, un-transformed.

On an **ordered** outcome the first three read the whole predicted distribution rather than one category, so they stay in one column: Somers' `D` and the win ratio. Where the measure IS the model's own it is read off the coefficients, otherwise from its predictions — so it is available whichever model you fit. `"auto"` never lands on a **predicted odds ratio**, a specialist quantity to be asked for by name. Call `reg_measures()` to see what an outcome offers.

`effect` **Where the reported number comes from**, once the model and the measure are fixed. `"auto"` (default) needs no thought: the model's own coefficients when the reported measure is the model's, its predictions otherwise. The other values name a reading:

- `"conditional"` — read off the coefficients ("holding the other predictors constant"). Only where `measure` is the model's own; otherwise the abort names the two cures.
- `"marginal"` — the **average marginal effect**: the model's prediction for every observed person, averaged. Comparable across models (Mood 2010), and always available.
- `"at_reference"` — the same at **one profile**, every other predictor at its reference.

The contrast is a **marker on the measure** in the header, so the acronym stays the one thing to look up: `Model_OR`, `Model_mRR`, `Model_refRD`. The observed companion carries the measure alone (`Obs_RR`), a univariable effect having no adjustment to be marginal over.

`outcome_level` Which level of the **outcome** to single out, as a named vector keyed by outcome name — `outcome_level = c(married = "Married")`. It is the twin of `ref`: `ref` **names the level you compare AGAINST,** `outcome_level` **the one you MODEL.**

- **binomial**: the level whose probability is estimated; it becomes the column header. Defaults to the outcome's **first** level. A 0/1 numeric outcome may be named either way.
- **multinomial**: the baseline category the other categories' columns are compared to.

- **ordinal and numeric outcomes**: refused, with the reason.

`trials` Grouped-binomial (summed-score) outcomes only. The number of items behind the score, fitting `cbind(score, trials - score)` as a binomial. `NULL` (default) fits an ordinary binary logit; an integer (or a vector named by outcome) sets the item count; `TRUE` uses each outcome's observed maximum. Requires `family = "binomial"`.

`empirical` Show the **observed, unadjusted (crude)** effect beside each modelled one — the same quantity fitted with a single predictor, on exactly the same people. The distance between the two is what adjustment changed, read left to right; it is the feature the package exists for. `TRUE` (**the default**) or `FALSE`; three spellings say *where* it goes, and in every one but `"no"` it is stored in the `obs` field and read by `$obs`, `color = "adjustment"`, `forest_plot()` and the hover tooltip:

- `TRUE` — a crude **column** beside the model one, except where that would double a table already wide (`tab_vars` groups, a 3+ level outcome), which take `"tooltip"`.
- `"column"` — always the column, per outcome category if that is what it takes.
- `"tooltip"` — computed, printed nowhere. The narrowest table.
- `"cell"` — **inside** the model cell, `(1/1.69) 1/1.63***`. `display` overrides it.

The two columns are the same column twice: same estimand, same colour ladder, one legend block. Each cell prints the effect with the level it sits on — the observed percentage or mean on the crude side, the **adjusted** prediction on the model side. A **continuous** predictor has no levels, so its crude cell is the univariable slope, which assumes linearity: check that with `shape` first.

`n` How many people the table is about. `NULL` (default) reads `options(tabxplor.n)` — `"range"`, which adds an `n` column holding the **unadjusted count** behind each predictor level, on the model's own complete cases. Where several models rest on different people it prints the whole range (`5 139-9 862`), so an unequal base cannot pass unnoticed; `"min"` shows the smallest count only, `"no"` none. Continuous predictors are left blank: their count is the model N, the first footer row.

`color`, `color_signif`

Colouring of the effect cells. `color = "measure"` (default, `TRUE` equivalently) grades each cell on **its own measure**, so the ladder follows what the column estimates; `color = FALSE` turns colouring off. `color_signif` is the significance policy — `NULL` (default) is `"grey_non_signif"` here, where `tab()` defaults to `"ignore"`.

What is left to choose is what each effect is compared **to**. `color` is positional, `c(text, background)`, so `color = c("measure", "adjustment")` answers "how strong is this effect?" and "how much did the model change it?" in one glance:

- `"adjustment"` — how far each **modelled** effect sits from its **observed** (crude) counterpart: what adjusting for the other predictors did to it. It turns `empirical = TRUE` on. Set its thresholds with `set_color_breaks()` (`adj_ratio`, `adj_diff`, `adj_diff_std`).

- "between_groups" — with tab_vars, how far each group's effect sits from the **first** group's: a per-predictor reading of effect modification.

The two are mutually exclusive, and each always tests its own gap: a gap whose interval covers zero is greyed whatever color_signif says — so a cell can be filled while neither estimate carries a star, which is correct rather than odd. A conditional **odds ratio** is not collapsible, so there the colours stay descriptive and are not tested. Read a coloured cell as "adjustment moved this effect", not as "this variable is a confounder". See the regression vignette.

stars

Logical (default TRUE for regression tables, where significance stars are standard). When FALSE, the per-cell p-value is dropped and no stars are shown (colours still read the interval).

ref

The reference every effect is measured **from** — one argument, one meaning per kind of predictor. For a **factor** it is the level the others are compared against (a level name, or "first" (default) / "last"); for a **continuous** predictor the value it is **anchored** at (a number, or "mean" (default), "median", "min", "max").

Anchoring a continuous predictor **does not change its own effect** — a slope is the same wherever you start reading it from — but it does move the **Constant** row; its own row says where the anchor sits, per SD/13.5 (at mean/42.4). The default is the mean because zero is usually outside the data: nobody is 0 years old.

ref, multiplier and shape share one grammar: a value **on its own** is the default for every predictor it can apply to, a **named** one overrides that variable — ref = c("median", "last", race = "Black").

For the level of the **outcome**, see outcome_level: ref names the level you compare AGAINST, outcome_level the one you MODEL.

multiplier

How a **continuous** predictor's effect is scaled — the unit its row reports. One unit is rarely a readable amount (a one-year change in age barely moves the odds), so the default is **two standard deviations**: roughly the span a binary predictor's own contrast covers, which is what makes a continuous row and a factor row comparable at a glance (Gelman 2008). Values: "2sd" (default), "sd", or a number of units (10 = per decade). Same grammar as ref: multiplier = c("sd", age = 10).

The estimate, its interval and the observed companion all scale together; the p-value does not move. **Because the default is not 1, a continuous predictor's** Model_* **cell does not equal** exp(coef(glm(...))) **unless you pass** multiplier = 1**.**

shape

How a **continuous** predictor enters the model, when one straight line is not enough. The Linearity footer row and the little curve drawn in the predictor's n cell tell you *whether* a line is enough; this argument is how you fix it without leaving the framework. Same grammar as multiplier and ref — shape = "quintiles" cuts every continuous predictor, shape = c(age = "quadratic") only that one, and anything unnamed stays linear:

"linear" one slope (the default).

"quintiles" **/** "quartiles" **/ an integer** k cut into k quantile groups, so the predictor becomes an ordinary **factor**: one estimate per group, its own observed companion, counts and colours per group — the non-linearity be-

comes visible in the printed numbers. Start here; it is the most readable answer.

"sd_bands" cut at the **mean and one standard deviation either side** — the classic low / average / high reading, whose cut points mean the same thing across sub-samples, where quantiles move with each one. The bands are not balanced: prefer quantiles when the group sizes matter.

"quadratic" adds a curvature term, so the predictor takes **two rows** — the slope at the mean, and whether it flattens or accelerates away from it.

"log" / "sqrt" fit log(x) / sqrt(x) instead of x — diminishing returns.

Example: shape = c(age = "quadratic", income = "log"). The observed companion is fitted with the same shape, so the comparison stays like with like. It is the vocabulary tab() takes (shape_numeric_var()) plus "quadratic", which is a model term.

stats

The statistics shown in the model-summary **footer** (one block per model). "auto" (default) uses the per-family set — R square, adjusted R square, the overall F-test and the residual SD for a linear model, the likelihood-ratio test against the null model, McFadden's pseudo-R square, AIC and BIC otherwise — plus the default **model checks** (see below). A weighted model shows the survey-appropriate set. Pass a character vector to pick them: "n", "lr_null", "mcfadden_r2", "aic", "bic", "phi", "r2", "r2_adj", "f_model", "sigma", "global", "interaction", "group_interaction", "linearity", "proportionality", "dispersion", "influence", "collinearity"; "all" for everything, or NULL / FALSE / "no" to hide the footer.

Model comparison happens by default wherever it means anything — when predictors is a list of several models. Where each nests in the next, every model is tested against the previous one; otherwise each against the first. "compare_sequential" and "compare_baseline" (optionally naming the model) override that, and naming any footer statistic drops it.

Three tests are asked for by name: "global", one **overall test per predictor** — "is this variable associated with the outcome at all?", which a block of stars against a reference category cannot answer; "interaction", whether each **crossed pair** in predictors is real or the additive model is enough; and "group_interaction", one aggregated **effect-modification** test per predictor across tab_vars groups.

conf_level

Confidence level for the intervals. NULL (default) reads options(tabxplor.conf_level) — 0.95. It drives every interval, the significance stars, the greying under color_signif and the model-versus-observed gap interval, and each column records the level it was built at.

na

Which rows each model is fitted on — the grain at which missing values are dropped. "drop_by_outcome" (default) gives every model **of one outcome** the same complete-case population, which is what makes the comparisons honest: the observed columns are computed on exactly the model's rows, and nested models get equal N. "drop_by_model" lets each model use its own complete cases — more rows, at the price of comparability. "drop_all" shares one population across the whole call.

"keep_for_predictors" drops nothing but a missing **outcome**: every predictor keeps its missing values as an ordinary NA level, with its own row, count and

effect — often the fastest way to find out whether non-response is itself patterned. A number has no level to put them in, so a numeric predictor that has any is cut into bands.

`display` What each effect cell shows — `tab()`'s display grammar, same names, same meaning, on every family and on the crude column as well as the model one. `NULL` (default) shows the plain estimate, or, with `empirical`, the estimate with the level it sits on beside it. The whole vocabulary — the named layouts, the `{}` templates and the per-token precision `"{est:3} ({base:1})"` — is in tabxplor-display; the ones this table uses most are `"est_ci"` (`1/2.22*** [1/2.47;1/1.99]`), `"est_base"` (`1/2.22 (32.8%)`), `"est_coef"` and `"base_est_mdiff"` / `"base_est_mratio"`, which read the same comparison the other way. The **Constant** row holds the quantity the column's effects operate on, so it is read in one step: a baseline *odds* on an odds-ratio column, the level itself on an additive one.

`display` is **post-hoc**: every quantity it can name is already stored, so choosing a layout never triggers a computation and never changes a number — `set_display()` on a built table gives the same result. It never changes the estimand, which is `measure`'s job alone.

`digits` The number of decimals. A single integer sets every cell (`0`, the default, means "each measure's own"), and a measure finer than the level it sits on keeps its own precision. Name a display field to set just that one, an aside included — `digits = c(ratio = 3)`, `digits = c(1, or = 3)`; a template may carry its own, `display = "{est:3} ({base:1})"`. Left alone, a column reading in the outcome's own units (a mean, a mean difference) follows that outcome's magnitude, so a six-figure salary prints no decimals and a rate prints two; `digits =` always raises it back.

`cleannames` Logical. If `TRUE`, strips numeric prefixes from factor levels for display. Uses `getOption("tabxplor.cleannames")` when `NULL`.

`subtext` Optional character. A note shown below the table.

`caption` A title for the table, stored on it and carried into every export. Without one a regression table titles itself from the model it shows.

`...` One rarely-typed argument, plus internal plumbing.

`ci_method` — how the interval and p-value are computed: the same argument, and the same named-vector grammar, as in `tab()`, whose fifth slot is this producer's. `"wald"` (default) matches standard software output and is the only option for weighted models; `"profile"` uses the profile-likelihood interval and the likelihood-ratio test — more accurate near separation, unweighted binomial / poisson only.

Every argument removed or renamed while `tab_reg()` was in development is still accepted here, and gives an error naming its replacement rather than R's bare "unused argument". The dot-prefixed names are the jamovi live-cache plumbing, not user arguments.

Details

To **learn** what such a table says, read Reading a regression (`vignette("tabxplor-reading-a-regression")`); to **look something up**, the regression vignette (`vignette("tabxplor-reg")`).

New to regressions with tabxplor? A first model needs three arguments: `data`, `outcome` and `predictors`. The model follows the outcome's type — a two-level factor gives logistic **odds ratios**, a numeric a linear **mean difference**, a count Poisson **ratios of means**, a 3+ level factor multinomial or ordinal odds ratios — so you rarely set `family` by hand.

The estimand is a cascade: `family` -> `link` -> `measure` -> `effect`, where `"auto"` means *follow from the left*. `family` says what kind of number the outcome is; `link` **which measure the model estimates** (a link *is* a measure, so it takes the same words); `measure` **which one is reported**; `effect` where that number comes from. Set any one and everything to its right re-derives, so most tables set none of them — and the one most people ever set is `measure`.

A **coefficient** exists only where the reported measure IS the model's own. Ask for another and it is worked out from the model's predictions instead, averaged over the sample. So `measure = "ratio"` on a binary outcome gives a *marginal* risk ratio from the logistic fit, while `link = "ratio"` fits the modified Poisson and gives its *conditional* one — two different quantities, and now two different arguments.

Value

A `tabxplor_grouped_tab` (grouped by predictor), one effect column per model / outcome.

Model checks

Beside the fit statistics the footer carries five **model checks**, each naming an assumption and the instrument that measured it: **Linearity** and **Proportionality** (p-values) say whether the estimate means what it claims, **Dispersion** whether the intervals are wide enough, **Influence** whether one respondent carries the result, **Collinearity** why the intervals are wide. Four are shown by default; `stats = "linearity"` adds the fifth, and `shape` is the cure for what it flags. `reg_check_plots()` draws them all. What each one asks, and how to read it: the regression vignette.

Out of scope

`tab_reg()` covers linear, logistic, Poisson, multinomial and ordinal models, with survey designs. Three families of models are deliberately **not** supported, and are unlikely to be: **survival / Cox** models, **mixed / multilevel** models, and pooling over **multiply-imputed** datasets. Fit those with their own packages.

The header acronyms

A column header names the **measure**; the **contrast** is a marker on it — no marker for a conditional effect, `m` for a marginal one, `ref` at the reference profile — and `measure = "log"` wraps it (`Model_mRR`, `Model_refRD`, `Model_log(OR)`). The observed companion carries the measure alone (`Obs_RR`).

- `OR` — odds ratio (binomial, multinomial)
- `cumOR` — cumulative odds ratio (ordinal)
- `RR` — risk ratio (binomial, multinomial)
- `RD` — risk difference (binomial, multinomial)
- `RoM` — ratio of means (gaussian, poisson)
- `diff` — mean difference (gaussian, poisson)

- D — Somers' D (ordinal)
- WR — win ratio (ordinal)

References

Clogg, C. C., Petkova, E. & Haritou, A. (1995). Statistical Methods for Comparing Regression Coefficients between Models. *American Journal of Sociology*, 100(5), 1261-1293 — the comparison `color = "adjustment"` implements.

Zou, G. (2004). A Modified Poisson Regression Approach to Prospective Studies with Binary Data. *American Journal of Epidemiology*, 159(7), 702-706 — `link = "ratio"`.

Altman, D. G. & Bland, J. M. (2003). Interaction revisited: the difference between two estimates. *BMJ*, 326, 219 — the `color = "between_groups"` test.

See Also

`reg_formulas()` shows the formula each column was fitted with, and `reg_measures()` what an outcome can be modelled as. `forest_plot()` draws the finished table — every effect with its interval, its stars and its colour, and (with `empirical = TRUE`) the observed effect beside it with the margin of error of the gap. `reg_check_plots()` draws the model checks. tabxplor-display says what a cell can show, `tab()` builds cross-tables.

Examples

```
# The shape table a continuous predictor draws under the footer is noise in an example:
.opt <- options(tabxplor.shape_table = "no")

# Logistic: the odds of being released, adjusted, beside the observed (crude) odds ratio.
tab_reg(car_arrests, "released", c("colour", "checks"))


# Linear: a mean difference in dollars.
tab_reg(car_salaries, "salary", c("sex", "discipline", "rank"))

# A count outcome: ratios of mean counts.
tab_reg(car_arrests, "checks", c("colour", "employed"), family = "poisson")

# `measure` reports another measure WITHOUT changing the model: a MARGINAL risk ratio,
# averaged over the sample, still from the logistic fit.
tab_reg(car_arrests, "released", c("colour", "checks"), measure = "ratio")

# `link` changes the model: the CONDITIONAL risk ratio of a modified-Poisson fit.
tab_reg(car_arrests, "released", c("colour", "checks"), link = "ratio")

# A named list of predictor sets: one column per model, compared in the footer.
tab_reg(car_salaries, "salary",
        list("sex alone" = "sex",
             "+ field"   = c("sex", "discipline"),
             "+ rank"    = c("sex", "discipline", "rank")),
        empirical = FALSE)

# A continuous predictor cut into groups, on French survey data:
```

```
tab_reg(questionr_hdv, "cinema", c("qualif", "age"), shape = c(age = "quartiles"))

options(.opt)
```

---

tab_spread *Turn a sub-table variable into columns*

Description

Turns each level of a `tab_vars` variable into a **block of columns**: fewer rows, more columns, and every column stores which sub-population it belongs to (`col_group`) beside the variable it shows (`col_var`). Every total row merges into ONE, named `totname` — the remaining `tab_vars` are still index columns of their own, so the label does not repeat them. A total *table*'s own line joins that row when no `tab_vars` is left to hold it, and is dropped when one is. `tab()`'s `spread_vars` calls it for you, and takes care of the totals beforehand.

Usage

```
tab_spread(tabs, spread_vars, names_prefix, names_sort = FALSE, totname = NULL)
```

Arguments

`tabs`	A `tibble` of class `tab`, made with `tab`, `tab_reg` or `tab_plain`.
`spread_vars`	<tidy-select> The tab variables to pass to column, with a syntax of type `c(var1, var2, ...)`.
`names_prefix`	String added to the start of every variable name.
`names_sort`	If no `names_prefix` is given, new names takes the form `spread_var_col_var_level`. Should then the column names be sorted ? If `FALSE`, the default, column names are ordered by first appearance.
`totname`	The name the merged total row takes, as a single string. `NULL` (default) uses the one `options(tabxplor.total_names)` declares.

Value

A `tibble` of class `tab`, with less rows and more columns.

Examples

```
 data <- forcats::gss_cat |> dplyr::filter(year %in% c(2000, 2014))

tabs <-
  tab(data, relig, marital, c(year, race), pct = "row", totaltab = "no",
      color = "difference", tot = "row", other_if_less_than = 30)

tabs |>
  dplyr::select(year, race, relig, Married) |>
  tab_spread(race)
```

tab_structure *The structure of a table*

Description

[Experimental]

What kind of object a `tabxplor` result is, read from its own declared model — the row-index columns (their stored roles) and the table's stated kind — rather than guessed from column names.

Usage

```
tab_structure(x)
```

Arguments

`x`	A `tabxplor_tab` / `tabxplor_grouped_tab`, or a list of them (`output_list = TRUE`).

Value

A named list:

`container` `"table"` or `"list"`.

`kind` `"crosstab"` or `"regression"` (`NA` when the table carries no metadata).

`merged` `TRUE` when several row variables are stacked in one table (a `var` column names each row's variable).

`grouped` `TRUE` when the table has `tab_vars` (sub-tables).

`row_vars`, `tab_vars`, `col_vars` the variables on each axis.

`same_col_vars`, `same_tab_vars` for a list only: whether its tables agree.

See Also

`tab_columns()` for the column-axis view.

Examples

```
t <- tab(forcats::gss_cat, c(marital, relig), race, pct = "row")
tab_structure(t)$merged
```

tab_transpose *Swap the rows and columns of a cross-table*

Description

Turns a table's rows into its columns and its columns into its rows, and returns a real `tabxplor_tab` — one you can keep piping through **dplyr**, colour, print and export. Row percentages become column percentages, and the old total column and total row swap places.

Its job is the **profile table**: many variables down the page, a few groups across it. It is also the only way to put a *mean* on a row, since a number given to `row_vars` is always cut into levels — build the means as columns (`tab(data, groups, numeric_vars)`) and transpose.

Use the exporters' `transpose = TRUE` argument instead whenever you only need the OUTPUT: it flips the finished render model after every colour and cell string is computed, so it handles what a data-level flip cannot (several row variables, `tab_vars` sub-tables, columns of unlike kinds).

```
tab(data, row_vars, col_vars, pct = "row") |> tab_kable(transpose = TRUE)  # or tab_md() / tab_xl()
```

Usage

```
tab_transpose(tabs, name = NULL)
```

Arguments

`tabs`	A single table made with `tab` (one row variable, one column variable; not a subtabled table with `tab_vars`, and at most one total row and one total column).
`name`	The name to give the new first (label) column, holding the old column-variable levels. `NULL` (default) uses the old column-variable name.

Value

A transposed `tabxplor_tab`.

Columns of unlike kinds

A transposed column stacks whatever the original rows held, so transposing a table that mixes percentage and mean columns gives a `mixed` column. Its numbers and its cell layouts are exact; only the colour ladder is shared, so an *additive* measure (`color = "difference"`) grades the percentage cells and leaves the others uncoloured, while a *multiplicative* one (`color = "ratio"`) grades them all — the percentage and mean ratio ladders being the same rungs.

See Also

`tab_spread`, `tab_compact`, `tab_export`.

Examples

```
# race x marital, read as marital x race:
tab(forcats::gss_cat, marital, race, pct = "row") |> tab_transpose()


# the profile table: mean rows come from mean columns
tab(forcats::gss_cat, marital, c(age, tvhours)) |> tab_transpose()
```

`tab_wrap_text` *Wrap column names and long labels*

Description

Wrap column names and long labels

Usage

```
tab_wrap_text(
  tabs,
  wrap_rows = 35L,
  wrap_cols = 15L,
  exdent = 1,
  whitespace_only = TRUE,
  unbreakable_spaces = TRUE,
  brk = "\n"
)
```

Arguments

`tabs`	A `tabxplor_tab` or a `tibble` .
`wrap_rows`	Row labels are wrapped past this width (35 by default), as prose — on whitespace.
`wrap_cols`	Column NAMES are wrapped past this width (15 by default). A name is a compound word, not prose, so it breaks at the seams a name is built from (_, . , *, and a camelCase boundary) as well as at spaces.
`exdent`	On the second lines or more, the number or characters to use for indentation.
`whitespace_only`	Set to `FALSE` to wrap row labels also on non whitespace characters.
`unbreakable_spaces`	Set to `FALSE` to keep normal spaces in text (auto-break).
`brk`	The string to use for linebreak : `\n` in text, but ` ` in html.

Value

The same `tabxplor_tab` or `tibble`.

Examples

```
tab(forcats::gss_cat, race, marital, pct = "row", color = "difference") |>
  tab_wrap_text(wrap_rows = 5L, wrap_cols = 8L)
```

---

tab_xl *Write a table to an Excel workbook*

Description

The Excel exporter behind tab_export: tab_export(x, format = "xl") calls this. Colours follow the same palettes as the console and the HTML output, so a table looks the same wherever it is read; change them with set_color_style and set_color_breaks.

Usage

```
tab_xl(
  tabs,
  path = NULL,
  replace = FALSE,
  open = rlang::is_interactive(),
  lang = NULL,
  colnames_rotation = 0,
  remove_tab_vars = TRUE,
  colwidth = "auto",
  color_legend = TRUE,
  sheets = "auto",
  titles,
  caption = NULL,
  font_text = NULL,
  font_num = NULL,
  font_num_stars = NULL,
  text_size = 10,
  text_size_headers = 9,
  text_size_subtext = 9,
  theme = NULL,
  color = TRUE,
  transpose = FALSE,
  var_names = NULL,
  wrap_rows = 35,
  wrap_cols = 15,
  ratio_cells = NULL,
  check = FALSE,
  data = NULL,
  print_color_legend = lifecycle::deprecated(),
  ...
)
```

Arguments

`tabs` A table made with `tab` or `tab_reg`, or a `list` of tab. A list of tables sharing the same `col_vars` (and no `tab_vars`) is merged into one; any other list — several `row_vars` and/or `tab_vars` — is rendered one table after another, each keeping its own sub-tables.

`path, replace, open`
The name, and possibly the path, of the Excel file to create (the `.xlsx` extension is optional). Defaults to a temporary directory; set the global option `"tabxplor.export_dir"` with `options` to change it. `replace` defaults to `TRUE` when `path` is given and `FALSE` otherwise; set it to `TRUE` to overwrite an existing file. Use `open = FALSE` not to open the workbook straight away in Excel (or whatever opens `.xlsx` files).

`lang` Colour-legend language: `NULL` (auto from the R/OS locale, English fallback), `"en"` or `"fr"`.

`colnames_rotation`
Rotate the names of columns to an angle (in degrees).

`remove_tab_vars`
By default, `tab_vars` columns are removed to gain space — the sub-table's Total row names it. Ignored where several `row_vars` are stacked: the level column alone is then not a complete row index, so the column stays. Set to `FALSE` to keep them.

`colwidth` Column widths. `"auto"` (the default) fits every column to what its cells actually show, so a number column is exactly as wide as its widest figure and a text column wraps instead of growing past a cap. Give a number instead to force that fixed width on every numeric column (a mean's `sd` sibling then takes a proportionally narrower one). Widths are set per *sheet*, so several tables written to one sheet all fit.

`color_legend` Print the colour legend below the table (with the subtext). `TRUE` by default, and a no-op on a table that carries no colours.

`sheets` The Excel sheets options :

- `"tabs"`: a new sheet is created for each table
- `"unique"`: all tables are on the same sheet
- `"auto"`: subsequent tables with the same column vars are printed on the same sheets

`titles` The titles of the different tables, as a character vector. When missing titles are given based on the names of the variables.

`caption` A single caption; a shortcut that fills `titles` (an explicit `titles` still wins). Unified name across all exporters.

`font_text, font_num, font_num_stars`
Fonts for text (labels, headers) and for numbers. The number font is chosen **per table**: `font_num` (default `"DejaVu Sans"`) when the table shows no significance stars, and `font_num_stars` (default `"Cascadia Mono"`, a **monospace** font) when it does — monospace aligns the stars and `(n=...)` composites, which a proportional font cannot. Defaults from `options(tabxplor.xl_font_text)` / `options(tabxplor.xl_font_num)` / `options(tabxplor.xl_font_num_stars)`.

Note that xlsx, unlike CSS, has **no font-fallback list**: only one name is recorded, so if it is missing on the machine opening the workbook Excel substitutes by its own rules. Set the options to a font you know is installed.

`text_size`, `text_size_headers`, `text_size_subtext`
: Font sizes of text elements.

`theme`
: By default (`"light"`) a white table with black text; set to `"dark"` for a black table with white text (the colours follow the theme). The black-and-white **publication** palettes render a table for a page that has no colour: `"print_ready"` picks the right one per table, or name it yourself – `"print_marks"`, `"print_emphasis"`, `"print_minimalistic"` (`"bw"`). See `tab_css` for what each of them says.

`color`
: Set to `FALSE` to render the table without colours (monochrome).

`transpose`
: Set to `TRUE` to transpose each table before export (rows become columns) – the col-percentages-with-several-row-variables use case.

`var_names`
: Which variable names to write beside the table: `"both"` (the default), `"rows"`, `"cols"` or `"none"`. The row-variable name is the leading column a table with several `row_vars` uses to name each block (written once per block); the column-variable names are the spanning row above their level columns. Level headers always keep their name.

`wrap_rows`
: Row labels are wrapped past this width (35 characters by default).

`wrap_cols`
: Column headers are wrapped past this width (15 characters by default), at the seams a compound name is built from (`_`, `.`, `*`, a camelCase boundary). A col_var's spanning name is measured against the width its own columns leave it first: past that it wraps, and past what wrapping can do it is shown from the prefix it shares with the block before it (`MUS_CONCERT_CLASSIQUE`, then `_ROCK`), and held to this width in the last resort — never while there is room for the whole name.

`ratio_cells`
: What a ratio / odds-ratio cell holds in the workbook. Excel cannot compute inside a number format, so a cell storing `0.83` cannot be made to print `÷1.2` the way the console does. `"fold"` (the default) stores the **reading value** instead — the fold, signed by its direction (`x` at or above the neutral, `-1/x` below it) — which prints as `×1.20` and `÷1.20`, `2.11` and `1/2.11`. The cell stays a real number: it sorts and filters in the direction it is read, and takes the reader's own decimal separator. `"raw"` stores the untransformed ratio (printed `×0.83`); `"text"` writes the exact display string, which reads perfectly but is no longer a number. Option twin: `tabxplor.xl_ratio_cells`.

`check`
: Model-check plots to draw under each `tab_reg()` table: `FALSE` (the default), `"auto"`, or a vector of check keys — the same values `reg_check_plots` takes, which is what draws them. Each grid is written as a picture below the table it belongs to. Needs `ggplot2` and `gridExtra`; a crosstab takes none.

`data`
: The data frame the models were fitted on. Only needed when `check` is on AND the `tab_reg` call cannot be replayed from the name it was written with (a `%>%` pipeline, a subset expression) — an ordinary `tab_reg(gss, ...)` recovers it by itself.

`print_color_legend`
: **[Deprecated]** Renamed to `color_legend`.

... Retired arguments, accepted and ignored with a deprecation message since 2.0.0 (`color_type`, `html_24_bit`, `n_min`, `hide_near_zero`): colour is a channel of `color =`, Excel is always 24-bit, and the other two are `tab()`'s business — the small-base filter and the display template. Anything else is an error naming the argument you meant, as it already was in `tab()`.

Value

The table(s) with formatting and colors in an Excel file, as a side effect. Invisibly returns `tabs`.

Recovering the raw ratio in Excel

A ratio or odds-ratio cell holds its **reading value**: the fold, signed by its direction. The sign IS the marker — negative means the cell reads ÷ (or `1/`) — so one formula gives the raw ratio back, with no macro and no add-in:

```
=IF(A2<0, -1/A2, A2)     the ratio itself
=ABS(A2)                how many times, whichever way it goes
```

Sorting and filtering need neither: the stored value is monotone in the direction it is read, so "at least twice as likely" is `>2` and "at least twice as unlikely" is `<-2`. Use `ratio_cells = "raw"` when the untransformed ratio matters more than the reading.

Examples

```
# openxlsx2 is Suggests-only and tab_xl() stops without it, so guard the example: \donttest{}
# does NOT exempt it from R CMD check --as-cran, which CRAN also runs without Suggests.
if (requireNamespace("openxlsx2", quietly = TRUE)) {
  forcats::gss_cat |>
    tab(marital, race, pct = "row", color = "difference") |>
    tab_xl()
}
```

Index

∗ **datasets**
car_arrests, 3
car_salaries, 4
facto_tea, 6
questionr_hdv, 39

as.factor(), *45*
as.matrix.tabxplor_tab (tabxplor-base-coercion), 62
as.table.tabxplor_tab (tabxplor-base-coercion), 62
as_refcol (fmt_attributes), 17
as_refrow (fmt_fields), 21
as_totcol (fmt_attributes), 17
as_totrow (fmt_fields), 21
as_tottab (fmt_fields), 21
attr, *8*

car_arrests, 3
car_salaries, 4
conf_level_to_z, 5, *47*

dplyr::filter, *61*, *86*
dplyr::select, *59*

facto_tea, 6
fct_recode_helper, 7
fmt, 8, *9*, *61*, *64*, *66*, *91*, *94*
fmt(), *16*, *20*, *23*, *70*, *74*
fmt_attr, 16
fmt_attr(), *17*, *20*, *73*, *74*
fmt_attr<- (fmt_attr), 16
fmt_attributes, *14*, 17, *23*
fmt_fields, *14*, *20*, 21
fmt_get_color_code, 23
forcats::fct_na_value_to_level(), *45*
forcats::fct_recode(), *7*
forest_plot, 24, *60*, *80*
forest_plot(), *41*, *62*, *98*, *103*
format(), *88*

get_bars (set_bars), 45
get_caption (set_caption), 46
get_ci_method (fmt_attributes), 17
get_col_group (fmt_attributes), 17
get_col_var (fmt_attributes), 17
get_col_var(), *16*
get_color (fmt_attributes), 17
get_color_bg (fmt_attributes), 17
get_color_breaks (set_color_palette), 47
get_color_breaks(), *47*
get_color_signif (fmt_attributes), 17
get_color_style (set_color_palette), 47
get_color_style(), *47*
get_comp_all, *19*
get_comp_all (fmt_attributes), 17
get_digits (fmt_fields), 21
get_display(), *66*
get_footer_tabs (set_footer_tabs), 51
get_legend_words (set_legend_words), 52
get_model_family (fmt_attributes), 17
get_num, *63*
get_num (fmt_fields), 21
get_pct_type (fmt_attributes), 17
get_pct_type(), *73*
get_pvalue (fmt_fields), 21
get_ref_type (fmt_attributes), 17
get_ref_type(), *20*
get_row_kind, *22*
get_row_kind (fmt_fields), 21
get_row_kind(), *13*
get_scale (fmt_attributes), 17
get_scale(), *16*, *72*, *73*
get_subtext (set_subtext), 53
get_test, 26
get_type (tabxplor-type), 72
gss_cat_data_formatting, 27

is_fmt (fmt), 8
is_refcol (fmt_attributes), 17
is_refrow (fmt_fields), 21

is_tab, 28
is_totcol (fmt_attributes), 17
is_totcol(), *16*
is_totrow (fmt_fields), 21
is_tottab (fmt_fields), 21

jmvtab, 28
jmvtabreg, 33

MASS::polr(), *42*

new_grouped_tab (new_tab), 37
new_tab, 37
new_tab(), *46*, *51*
nnet::multinom(), *42*

options, *109*
options(), *68*

questionr_hdv, 39

record, *8*
reg_check_plots, *24*, *26*, *38*, 40, *110*
reg_check_plots(), *71*, *102, 103*
reg_formulas, 42
reg_formulas(), *44*, *95*, *103*
reg_measures, 43
reg_measures(), *42*, *97*, *103*

score_from_lv1, 44
score_from_lv1(), *6*
set_bars, *39*, 45
set_caption, *38*, 46, *51*, *60*
set_col_var (fmt_attributes), 17
set_color (fmt_attributes), 17
set_color(), *67*
set_color_breaks, *5*, *26*, *38*, *50*, *58*, *61*, *108*
set_color_breaks (set_color_palette), 47
set_color_breaks(), *47*, *53*, *62*, *67*, *69*, *79*, *98*
set_color_palette, 47
set_color_palette(), *47*, *62*, *69*, *79*
set_color_signif (fmt_attributes), 17
set_color_style, *108*
set_color_style (set_color_palette), 47
set_comp_all (fmt_attributes), 17
set_diff_type (fmt_attributes), 17
set_digits (fmt_fields), 21
set_display, *25*, *60*
set_display (fmt_fields), 21
set_display(), *64*, *66, 67*, *101*
set_footer_tabs, *39*, 51, *89*, *92*
set_footer_tabs(), *54*, *68*, *90*, *93*
set_legend_words, *38*, 52, *67*
set_legend_words(), *54*, *68*, *82*
set_model_family (fmt_attributes), 17
set_num (fmt_fields), 21
set_pct_type (fmt_attributes), 17
set_pct_type(), *73*
set_pvalue (fmt_fields), 21
set_ref_type (fmt_attributes), 17
set_ref_type(), *20*
set_row_kind (fmt_fields), 21
set_scale (fmt_attributes), 17
set_scale(), *72, 73*
set_subtext, *38*, *52*, 53
set_subtext(), *53*, *66*, *68*, *81, 82*, *90*
set_type (tabxplor-type), 72
shape_numeric_var, 54, *57*
shape_numeric_var(), *100*
stats::glm(), *42*
survey::svydesign, *86*
survey::svydesign(), *95, 96*
survey::svyglm(), *42*, *96*

tab, *8*, *10, 11*, *19*, *24*, *28*, *38*, *50*, 56, *63*, *80*, *83*, *85*, *88*, *104*, *106*, *109*
tab(), *26, 27*, *55, 56*, *64*, *66*, *68*, *70–72*, *75, 76*, *78*, *85, 86*, *89–91*, *93–96*, *98*, *100, 101*, *103, 104*, *111*
tab_chi2, *38*
tab_columns, 73
tab_columns(), *16, 17*, *20*, *27*, *105*
tab_compact, 74, *106*
tab_counts, *63*, 75, *91*, *94*
tab_counts(), *62*, *70*
tab_css, *32*, *36*, *49*, 77, *79, 80*, *83–85*, *88, 89*, *110*
tab_css(), *69–71*
tab_export, *26*, *63*, 80, *82*, *87*, *106*, *108*
tab_export(), *77*
tab_footer_text, 81
tab_footer_text(), *54*, *68*
tab_get_vars(), *28*
tab_html, *49*, *51*, *80*, 82, *88*
tab_html(), *68–71*, *77–79*
tab_kable, *46*, *60*
tab_kable (tab_html), 82
tab_kable(), *62*

tab_many, 85
tab_md, *46*, *51*, *60*, *80*, 87, *92, 93*
tab_md(), *62*, *68–71*, *77–79*, *93*
tab_note, *39*, 89
tab_note(), *68*
tab_num, 90
tab_num(), *62*
tab_parallel_stop, 92
tab_parallel_stop(), *72*
tab_pct, *9*
tab_pipe, 92
tab_plain, *38*, 93, *104*
tab_plain(), *62*
tab_reg, *11, 12*, *19*, *24*, *28*, *38*, *44*, *47*, *57, 58*, *63*, *65*, *80*, *83*, *88*, *91*, 94, *104*, *109, 110*
tab_reg(), *40–45*, *54–56*, *62*, *64*, *66*, *68*, *70–72*, *74*
tab_spread, *11*, 104, *106*
tab_structure, *28*, 105
tab_structure(), *27, 28*, *62*, *73, 74*
tab_transpose, 106
tab_wrap_text, 107
tab_xl, *46*, *51*, *80*, 108
tab_xl(), *62*, *70, 71*
table, *62*
tabxplor-base-coercion, 62
tabxplor-display, *9*, *14*, *60*, *62*, 64, *101*, *103*
tabxplor-footer, 66
tabxplor-options, *47*, *56*, *62*, *66*, *68*, 68, *92*
tabxplor-type, 72
tabxplor.options *(*tabxplor-options*)*, 68
tibble, *38*
tidy-select, *7*, *57*, *61*, *76*, *86*, *91*, *94*, *104*

vctrs::field, *8*
vctrs::field(), *12*, *21*
vctrs::vec_data(), *12*