Package: ezrsurvey (via r-universe)

November 5, 2026

Type Package

Title Single-Line Helpers for Consumer Survey Analysis

Version 0.7.1

Description Simplifies the day-to-day work of research and insights professionals who analyse consumer and customer survey data but have limited coding experience. Wraps common 'tidyverse' patterns into single-line functions for the tasks these teams repeat constantly: loading and combining survey exports, turning questions into percentages and summaries, recoding rating and Net Promoter scales, deriving demographics such as age cohorts and regions, modelling the drivers of satisfaction, checking sampling precision, selecting illustrative open-text comments, and producing presentation-ready charts and reports. A simulated consumer-survey dataset is included for learning and examples. Weighting rakes to target margins as in Deming and Stephan (1940) <doi:10.1214/aoms/1177731829> and reports the design effect of Kish (1965, ISBN:978-0-471-10949-5); driver models use the relative weights of Johnson (2000) <doi:10.1207/S15327906MBR3501_1> or the random forests of Breiman (2001) <doi:10.1023/A:1010933404324>; the Net Promoter Score follows Reichheld (2003) <https://hbr.org/2003/12/the-one-number-you-need-to-grow>; and diverse comments are chosen by maximal marginal relevance as in Carbonell and Goldstein (1998) <doi:10.1145/290941.291025>.

License MIT + file LICENSE

URL https://mashud37.github.io/ezr-research/ezr-survey/, https://github.com/mashud37/ezr-research

BugReports https://github.com/mashud37/ezr-research/issues

Encoding UTF-8

Language en-GB

LazyData true

Depends R (>= 4.1.0), dplyr (>= 1.1.0), ggplot2, tidyr (>= 1.2.0), tibble, readr, stringr, purrr

Imports rlang (>= 1.0.0), withr

Suggests rwa, randomForest, treemapify, ggrepel, scales, officer (>= 0.6.0), flextable, openxlsx2, xml2, systemfonts, quarto, knitr, rmarkdown, svglite, writexl, yaml, stopwords, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation no

Author Andreas Schellewald [aut, cre]

Maintainer Andreas Schellewald <aschellewald@acm.org>

Config/pak/sysreqs libicu-dev libx11-dev

Repository https://aschellewald.r-universe.dev

Date/Publication 2026-11-04 21:40:02 UTC

RemoteUrl https://github.com/cran/ezrsurvey

RemoteRef HEAD

RemoteSha 9064194909fd2b27e2f32fc239337e446333f7e9

Contents

add_currency 4
add_region 6
annotate_bands 7
apply_order 8
bands_rating_3 9
bin_numeric 10
brand_info 11
calc_importance 12
calc_nps 13
calc_percentage 14
calc_percentage_batch 16
calc_percentage_multi 17
calc_summary 19
clean_label 20
clear_brand 21
clear_checkpoints 22
clear_weights_cache 22
compare_values 23
convert_currency 24
country_region 25
crosstab 26
crosstab_banner 28
currency_rates 31
diagnose 32

drop_items . . . 33
edit_ezrsurvey_profile . . . 34
ensure_numeric . . . 35
example_report . . . 36
export_summary_xlsx . . . 37
export_xlsx . . . 39
ezrsurvey_options . . . 40
ezrsurvey_profile_paths . . . 42
generation_scheme . . . 42
get_dataset . . . 43
get_order . . . 44
get_weights . . . 45
ipm_model . . . 46
label_pct . . . 47
list_currencies . . . 48
list_orders . . . 48
list_report_templates . . . 49
load_ezrsurvey_profile . . . 50
margin_of_error . . . 51
mark_value . . . 52
na_blank . . . 53
nice_max . . . 54
nps_group . . . 55
order_for . . . 56
pal_brand . . . 56
pal_rating . . . 57
parse_filename . . . 58
plot_bars . . . 60
plot_diff . . . 62
plot_gauges . . . 63
plot_ipm . . . 64
plot_nps . . . 65
plot_nps_gauge . . . 66
plot_quotes_tree . . . 67
plot_rating_grid . . . 68
plot_stacked_rating . . . 69
podracing_survey . . . 71
precision_summary . . . 72
read_folder . . . 73
recode_age . . . 75
recode_generation . . . 76
recode_likert . . . 77
recode_region . . . 78
register_order . . . 80
register_order_presets . . . 81
remove_order . . . 82
report_add_plot . . . 82
report_add_slide . . . 83

report_add_table . . . 85
report_add_text . . . 86
report_deck . . . 86
report_layouts . . . 88
report_new . . . 89
report_save . . . 90
report_section . . . 91
report_slide . . . 92
report_title_slide . . . 93
rescale_bands . . . 94
reset_ezrsurvey_options . . . 95
rse . . . 96
rse_rating . . . 97
sample_comments . . . 98
sample_comments_diverse . . . 99
save_data . . . 101
save_ezrsurvey_profile . . . 102
save_output . . . 103
save_plot . . . 104
scaffold_report . . . 105
scale_fill_brand . . . 106
scale_fill_nps . . . 107
scale_fill_rating . . . 108
scale_y_pct . . . 109
se_mean . . . 110
se_prop . . . 111
select_prefix . . . 112
select_suffix . . . 113
set_weights . . . 114
shopping_survey . . . 115
split_multi . . . 116
theme_ezrsurvey . . . 117
theme_transparent . . . 118
use_brand . . . 119
use_dataset . . . 120
use_ezrsurvey_profile . . . 121
weight_vector . . . 122

Index **124**

`add_currency` *Add a converted-currency column to a data frame*

Description

Convenience wrapper around `convert_currency()` that appends a converted column to `data`. The source currency can be a fixed code or a per-row column.

Usage

```
add_currency(data = NULL, amount, from, to = "USD", into = NULL, rates = NULL)
```

Arguments

data	A data frame.
amount	The amount column to convert (unquoted).
from	Source currency: either a bare column name holding per-row codes (e.g. currency) or a quoted constant code (e.g. "EUR").
to	Target currency code. Defaults to "USD".
into	Name of the new column. Defaults to "<amount>_<to>".
rates	Passed to convert_currency().

Details

The from argument is clever about whether you mean a column or a constant: a bare name (from = currency) is read as a per-row source-currency column, while a quoted string (from = "EUR") is a fixed code applied to every row. This is the common case where each respondent reported spend in their own currency. The new column defaults to <amount>_<to> (e.g. spend_usd); override with into. Conversion uses convert_currency(), so the same rates, aliases and warnings apply.

Value

data with the converted column added.

See Also

convert_currency().

Other currency: convert_currency(), list_currencies()

Examples

```
df <- tibble::tibble(spend = c(100, 200, 50),
                     currency = c("EUR", "GBP", "JPY"))

# per-row source currency (a column)
add_currency(df, spend, from = currency, to = "USD")

# a fixed source currency (a constant), custom output name
add_currency(df, spend, from = "EUR", to = "USD", into = "spend_usd")
```

add_region *Add region (and subregion) columns from a country column*

Description

Convenience wrapper that appends a `region` column (and optionally a `subregion`) to a data frame by looking up an existing country column with `recode_region()`.

Usage

```
add_region(
  data = NULL,
  country,
  region_to = "region",
  subregion = FALSE,
  quiet = FALSE
)
```

Arguments

`data`	A data frame.
`country`	The country column (unquoted).
`region_to`	Name of the region column to add. Defaults to `"region"`.
`subregion`	If `TRUE`, also add a `subregion` column. Defaults to `FALSE`.
`quiet`	Passed to `recode_region()`.

Details

This is the data-frame-friendly form of `recode_region()`: point it at an existing country column and it appends a `region` column (named via `region_to`) and, optionally, a `subregion`. Handy right after `read_folder()` to enrich raw exports before grouping by region. The lookup, matching and unmatched-country warning behave exactly as in `recode_region()`.

Value

`data` with the new column(s) added.

See Also

`recode_region()`, country_region.

Other recode: `bin_numeric()`, `clean_label()`, `drop_items()`, `ensure_numeric()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_likert()`, `recode_region()`, `split_multi()`

Examples

```
df <- tibble::tibble(demo_country = c("Germany", "Japan", "Brazil"))
add_region(df, demo_country, subregion = TRUE)
```

`annotate_bands` *Annotate a plot with labelled decision bands*

Description

Programmatically adds the BAD / OK / GOOD style decision markers survey charts carry: for each band a thin coloured marker line runs along one axis, with a bold centred label. This turns the copy-pasted `geom_rect() + annotate("text", x = median(seq(from, to, .1)), ...)` blocks into one call, so any chart can be given consistent "where's good, where's bad" guidance.

Usage

```
annotate_bands(
  plot,
  bands,
  axis = c("x", "y"),
  at = NULL,
  label_offset = NULL,
  labels = TRUE,
  text_colour = NULL,
  text_size = NULL,
  linewidth = 1
)
```

Arguments

`plot`	A ggplot object.
`bands`	A band specification: a data frame (or one of `bands_rating_3()`, `bands_rating_5()`, `bands_nps()`) with columns `from`, `to`, `label` and optionally `colour`.
`axis`	Which axis the bands run along: `"x"` (default) or `"y"`.
`at`	Position on the *opposite* axis at which to draw the marker line and labels (e.g. the top of a bar chart). If `NULL`, inferred from the plot's panel range.
`label_offset`	Distance to nudge labels away from the marker line, in data units of the opposite axis. If `NULL` (default), 4% of the opposite axis range, so labels sit clear of the line instead of on top of it.
`labels`	Whether to draw the band labels. Defaults to `TRUE`.
`text_colour, text_size`	Label appearance. `text_colour = NULL` (default) uses each band's own colour; `text_size = NULL` (default) matches the theme's base font size (11 pt) instead of a fixed geom size.
`linewidth`	Marker line width. Defaults to `1`.

Details

Bands turn a chart into a decision aid: a thin coloured marker line runs along one axis for each band, with a bold centred label, so a reader instantly sees where "good" and "bad" lie. Supply a band spec (a data frame with `from`, `to`, `label` and optional `colour`, or one of the `bands_rating_3()` presets), the `axis` the bands run along, and `at` – the position on the opposite axis for the marker line (usually the top of the panel; inferred from the plot when omitted).

Value

The plot with the band layers added.

See Also

`mark_value()` to add a single "you are here" marker.

Other decisions: `bands_rating_3()`, `mark_value()`, `rescale_bands()`

Examples

```
# the performance axis of an IPM chart is the 1-5 rating scale the
# bands describe, so "where is good" lands where a reader expects it
ipm_model(podracing_survey, nps_value, "ratings_") %>%
  plot_ipm() %>%
  annotate_bands(bands_rating_3(), axis = "x")
```

`apply_order` *Apply a registered order to a vector*

Description

Turns a vector into an ordered factor using a registered order, selected either by order `name` or by looking up the order linked to a variable name.

Usage

```
apply_order(x, name = NULL, var = NULL)
```

Arguments

`x`	A vector.
`name`	Order name (see `register_order()`). Takes precedence over `var`.
`var`	A variable name to look up via `order_for()`.

Value

A factor with the registered levels, or `x` unchanged if no order is found.

See Also

register_order(), order_for().

Other config: edit_ezrsurvey_profile(), ezrsurvey_options(), ezrsurvey_profile_paths(), get_dataset(), get_order(), list_orders(), load_ezrsurvey_profile(), order_for(), register_order(), register_order_presets(), remove_order(), reset_ezrsurvey_options(), save_ezrsurvey_profile(), use_dataset(), use_ezrsurvey_profile()

Examples

```
register_order("size", c("S", "M", "L"))
apply_order(c("L", "S", "M"), name = "size")
remove_order("size")
```

---

bands_rating_3 *Built-in decision-band presets*

Description

Ready-made band specifications for annotate_bands().

Usage

```
bands_rating_3(from = NULL, to = NULL)

bands_rating_5(from = NULL, to = NULL)

bands_nps(from = NULL, to = NULL)

bands_nps_score(from = NULL, to = NULL)
```

Arguments

from, to	Optional axis limits. A band reaching past them is trimmed and a band entirely outside them is dropped, so a preset can be reused on a chart that shows only part of the scale.

Details

- bands_rating_3() – BAD / OK / GOOD over a 1-5 rating axis.
- bands_rating_5() – per-point labels (1 Very bad .. 5 Very good) over 1-5.
- bands_nps() – Detractor / Passive / Promoter over a 0-10 answer axis.
- bands_nps_score() – Needs work / Good / Great / Excellent over a -100..100 Net Promoter Score axis (the scale a headline NPS sits on, not the 0-10 answers it is computed from).

Pass from / to when the chart's panel is narrower than the full scale. Plotting a 2-5 range with the untrimmed bands_rating_3() would open the BAD band at 1, off the left edge, and stretch the panel to reach it.

Value

A tibble with `from`, `to`, `label` and `colour`.

See Also

Other decisions: `annotate_bands()`, `mark_value()`, `rescale_bands()`

Examples

```
bands_rating_3()

# only the part of the scale a 2-5 panel actually shows
bands_rating_3(from = 2)

bands_nps_score()
```

`bin_numeric` *Bin a numeric vector into labelled groups*

Description

A thin, survey-friendly wrapper around `base::cut()` that returns a factor in band order and uses left-closed, right-open intervals by default so that age bands like 18-21 behave intuitively. Generalises the `case_when(age %in% seq(...))` age-grouping pattern.

Usage

```
bin_numeric(x, breaks, labels, right = FALSE, quiet = FALSE)
```

Arguments

`x`	A numeric vector.
`breaks`	Numeric vector of cut points. With `n` labels you need `n + 1` breaks. Use `Inf` for an open-ended top band.
`labels`	Character vector of group labels, length `length(breaks) - 1`.
`right`	If `TRUE`, intervals are closed on the right; if `FALSE` (default) closed on the left. Left-closed matches "18 to 21" style bands.
`quiet`	If `TRUE`, do not report values that fell outside `breaks`.

Details

Bands are built with `base::cut()` using `include.lowest = TRUE`, so the very lowest break is included. With `right = FALSE` (the default) a band runs from its lower break up to *but not including* the next – i.e. `[18, 25)` – which is what you want for age groups like "18 to 24". Text input is salvaged with `ensure_numeric()`, so `"27 years"` bins correctly. The result is a factor whose levels follow `labels`, so tables and charts built from it list the bands low to high ("40-70k" before "100-150k") rather than alphabetically. Wrap it in `as.character()` if you need plain text.

A value below the first break or above the last becomes NA and drops out of every chart built from the result, so the function says how many did and what their range was. A closed top band such as 35 to 40.5 under a label of "35 to 40+" is the usual cause; Inf is what that label means.

Value

A factor whose levels are labels in the order given (values outside the range or NA become NA).

See Also

recode_age() for a ready-made age-band wrapper, ensure_numeric().

Other recode: add_region(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), na_blank(), nps_group(), recode_age(), recode_generation(), recode_likert(), recode_region(), split_multi()

Examples

```
bin_numeric(c(15, 19, 27, 41),
            breaks = c(0, 18, 25, 35, Inf),
            labels = c("<18", "18-24", "25-34", "35+"))
```

---

brand_info *Show the active brand settings*

Description

Show the active brand settings

Usage

```
brand_info()
```

Value

An ezrsurvey_brand list with elements colors, color_primary, font_major, font_minor, template_pptx, template_docx (each NULL when unset).

See Also

use_brand(), clear_brand().

Other brand: clear_brand(), pal_brand(), scale_fill_brand(), use_brand()

Examples

```
brand_info()
```

`calc_importance` *Driver importance*

Description

Ranks how much each predictor drives an outcome such as the NPS rating, as importance scores summing to 100. Relative weights analysis is the default; a random forest and a plain correlation are available as alternatives.

Usage

```
calc_importance(
  data = NULL,
  outcome,
  predictors,
  method = c("rwa", "forest", "correlation"),
  recode = TRUE,
  likert_levels = c("Very bad", "Bad", "Ok", "Good", "Very good")
)
```

Arguments

`data`	A data frame.
`outcome`	The outcome column (unquoted), e.g. `nps_value`.
`predictors`	The predictor columns, using tidyselect (e.g. `starts_with("ratings_")`).
`method`	How to measure importance. `"rwa"` (default) is relative weights analysis via `rwa::rwa()`; `"forest"` is random-forest permutation importance via `randomForest::randomForest()`; `"correlation"` is the absolute correlation with the outcome. See Details.
`recode`	If `TRUE` (default), character predictor columns are mapped to 1-5 with `recode_likert()`; numeric columns are left as-is.
`likert_levels`	Scale wordings passed to `recode_likert()` when recoding.

Details

All three methods answer the same question and are scaled the same way, so each feature's `importance` reads as "this feature accounts for X% of what drives the outcome" and any of them can be fed to `plot_ipm()`. They differ in what they can see:

- `"rwa"`, relative weights analysis (Johnson's epsilon), shares out the linear model's explained variance between correlated predictors, which ordinary regression coefficients handle poorly. This is the default because rating batteries are almost always heavily correlated.
- `"forest"` grows a random forest and measures how much prediction error rises when each predictor is permuted, so it picks up non-linear effects and interactions that relative weights cannot. It is the useful cross-check: if a feature ranks high here and low under `"rwa"`, its effect is not linear. Being a random method, it moves a little between runs; call `set.seed()` first for a figure you intend to publish.

- "correlation" is the crude one, ignoring the other predictors entirely, so correlated features double-count. Its merit is that it needs no suggested package and cannot fail to converge, which makes it a reasonable sanity check on the other two.

Only complete cases are used. Worded rating columns are recoded to 1-5 automatically (recode = TRUE). Negative importances (which permutation importance can produce for a predictor that is pure noise) are floored at zero before rescaling. Most users call ipm_model(), which pairs this with performance for the importance/performance matrix.

Value

A tibble with feature and importance, the latter rescaled to sum to 100 whichever method produced it, strongest driver first.

See Also

ipm_model(), plot_ipm().

Other modelling: calc_nps(), ipm_model()

Examples

```
calc_importance(podracing_survey, nps_value, starts_with("ratings_"))

# a cross-check that can see non-linear effects
set.seed(1)
calc_importance(podracing_survey, nps_value, starts_with("ratings_"),
                method = "forest")
```

---

calc_nps *Net Promoter Score*

Description

Computes NPS from a 0-10 recommendation question. The score is % promoters - % detractors, equivalently 100 * mean(nps_group(x)) over the signed -1/0/1 coding (see nps_group()).

Usage

```
calc_nps(data = NULL, value, by = NULL, weights = NULL)
```

Arguments

data	A data frame.
value	The 0-10 recommendation column (unquoted).
by	Optional grouping column(s); see calc_percentage().
weights	Survey weighting: NULL (default) uses the session scheme from set_weights() if set; FALSE forces unweighted; or pass an ad-hoc scheme. When weighting is active nps is the weighted score (n stays the unweighted base).

Details

Respondents are bucketed by `nps_group()` into detractors (0–6), passives (7–8) and promoters (9–10); the score is the percentage of promoters minus the percentage of detractors, which equals `100 * mean()` of the signed `-1/0/1` coding. Missing or out-of-range scores are dropped before the mean. Text answers are coerced with `ensure_numeric()`. Use `plot_nps()` for the full 0–10 distribution chart and `plot_nps_gauge()` for a single-number gauge.

The three group counts and shares are reported alongside the score because a single NPS hides how it was reached: +20 from 40% promoters and 20% detractors is a different picture from +20 with 25% and 5%. Counts are always unweighted respondent counts; the shares follow `nps`, so they are weighted when a weighting scheme is active. All three shares and `nps` are rounded to whole numbers independently, so `pct_promoters - pct_detractors` can land one point away from `nps`; the identity holds exactly before rounding.

Value

A tibble with `n` (valid responses), `nps` (an integer from -100 to 100), the three group shares (`pct_detractors`, `pct_passives`, `pct_promoters`) and the counts behind them (`detractors`, `passives`, `promoters`), one row per group when `by` is given. The shares come first because they are what a report quotes.

See Also

`nps_group()`, `plot_nps()`, `plot_nps_gauge()`.

Other modelling: `calc_importance()`, `ipm_model()`

Examples

```
calc_nps(podracing_survey, nps_value)

calc_nps(podracing_survey, nps_value, by = region)
```

---

calc_percentage *Count a categorical question as percentages*

Description

The headline helper: replaces the ubiquitous `count(x) %>% mutate(pct = round(n / sum(n) * 100)) %>% select(-n)` dance with a single call. Returns both the raw counts and the percentages, optionally grouped by one or more variables and optionally pivoted to a wide cross-tabulation.

Usage

```
calc_percentage(
  data = NULL,
  column,
  by = NULL,
  sort = c("none", "desc", "asc"),
  levels = NULL,
```

```
  digits = 0,
  wide = FALSE,
  na_rm = TRUE,
  drop = NULL,
  weights = NULL
)
```

Arguments

data	A data frame. If omitted, the session default set with use_dataset() is used.
column	The categorical column to tabulate (unquoted).
by	Optional grouping column(s). Percentages are computed *within* each group so they sum to ~100 per group. Pass one bare name (by = year) or several with c() (by = c(year, game)).
sort	Level ordering of column: "none" (default, data order), "desc" (largest percentage first) or "asc".
levels	Optional character vector giving an explicit level order for column (the "custom" ordering mode). Overrides sort. If omitted and sort = "none", a registered order for this variable is applied automatically (see register_order()), and failing that a factor keeps its own level order.
digits	Decimal places for the percentage. Defaults to 0.
wide	If TRUE, pivot to one row per by group and one column per answer (dropping n), the shape a wide summary table needs. Defaults to FALSE (tidy long form).
na_rm	If TRUE (default), blanks and "Prefer not to answer" responses (see na_blank()) are dropped before counting.
drop	Optional character vector of answer values to remove before counting (e.g. c("Other", "Don't know")), so the kept answers re-base to ~100%. Matching is case-insensitive (see drop_items()). Defaults to the drop_answers option (NULL = drop nothing; see ezrsurvey_options()).
weights	Survey weighting for this call: NULL (default) uses the session scheme from set_weights() if one is set; FALSE forces unweighted; or pass an ad-hoc scheme (any form set_weights() accepts) to weight just this call. When weighting is active a wpct column (weighted percentage) is added beside n and pct.

Details

Percentages are computed *within* each group, so they sum to about 100 per group (subject to rounding). The level order of column is decided in this order of precedence: an explicit levels argument; then a non-"none" sort; then a registered order for the variable (see register_order()); then the column's own levels, when it is a factor; otherwise data order. When by is omitted, the default_by option is used if set (see ezrsurvey_options()), so you can apply a standard breakdown without repeating it. wide = TRUE pivots to one row per group with a column per answer – the shape you want for a slide table or an Excel tab.

Value

A tibble with column, n and pct (and wpct when weighting is active) in long form, or one row per group with an answer column each (wide form).

See Also

calc_percentage_multi() for check-all-that-apply questions, calc_percentage_batch() for many questions at once, calc_summary() for numeric variables, register_order() for reusable level orders.

Other summaries: calc_percentage_batch(), calc_percentage_multi(), calc_summary(), clear_checkpoints(), crosstab(), crosstab_banner()

Examples

```
calc_percentage(podracing_survey, demo_gender)

# largest first
calc_percentage(podracing_survey, demo_gender, sort = "desc")

# grouped and pivoted to a wide cross-tab
calc_percentage(podracing_survey, satis_return, by = region, wide = TRUE)
```

---

calc_percentage_batch *Percentages for a batch of questions at once*

Description

Runs calc_percentage() over several columns and stacks the results into one tidy table with a variable column identifying the source question and a shared answer column – handy for tabulating a whole block of questions (e.g. every demo_ variable) in a single call.

Usage

```
calc_percentage_batch(
  data = NULL,
  ...,
  by = NULL,
  sort = c("none", "desc", "asc"),
  digits = 0,
  na_rm = TRUE,
  drop = NULL,
  weights = NULL,
  clean_names = FALSE,
  prefix = NULL
)
```

Arguments

data	A data frame.
...	Columns to tabulate, using tidyselect (e.g. starts_with("demo_") or demo_gender, demo_edu).

by, sort, digits, na_rm, drop, weights
: Passed to calc_percentage() (so a wpct column appears when weighting is active, and drop removes unwanted answers from every question).

clean_names
: If TRUE, tidy each variable with clean_label() instead of reporting the raw column name.

prefix
: Optional question prefix to strip from variable, e.g. "ratings_". Implies clean_names.

Details

Each selected column is tabulated with calc_percentage() and the results are row-bound, with a variable column naming the source question and the answers collected into a shared character answer column (factor levels differ between questions, so they are coerced to character when stacked). This is the tidy long shape you want for faceted plots or a single export of a whole question block. To send each question to its own Excel tab instead, see export_xlsx().

variable is the raw column name by default. Set clean_names = TRUE (or name a prefix) to get the same tidied wording calc_percentage_multi() and ipm_model() produce, which is what lets a batch table join to a model table on the question name.

Value

A tibble with variable, answer, n, pct (plus any by columns).

See Also

calc_percentage(), clean_label(), export_xlsx().

Other summaries: calc_percentage(), calc_percentage_multi(), calc_summary(), clear_checkpoints(), crosstab(), crosstab_banner()

Examples

```
calc_percentage_batch(podracing_survey, demo_gender, demo_job)

calc_percentage_batch(podracing_survey, starts_with("demo_"))

# tidy question names, with the block's prefix removed
calc_percentage_batch(podracing_survey, starts_with("ratings_"),
                      prefix = "ratings_")
```

---

calc_percentage_multi *Tabulate a check-all-that-apply (multi-select) question as percentages*

Description

Multi-select questions arrive as a block of columns sharing a prefix, each holding the chosen option (or blank). This computes, per option, the share of respondents who selected it – so the percentages can (and usually do) sum to more than 100. The denominator is the number of distinct respondents who selected at least one option.

Usage

```
calc_percentage_multi(
  data = NULL,
  prefix,
  id = NULL,
  by = NULL,
  sort = c("none", "desc", "asc"),
  digits = 0,
  drop = NULL,
  clean_names = TRUE,
  split = "auto"
)
```

Arguments

data	A data frame.
prefix	Common column-name prefix identifying the option block, e.g. "motivations_".
id	Optional respondent identifier column (unquoted) used as the distinct-respondent denominator. If omitted, each row is treated as one respondent.
by	Optional grouping column(s); percentages are computed within group.
sort, digits	As in calc_percentage(). Sorting acts on the option labels.
drop	Optional character vector of option labels to remove before counting (matched against the cleaned labels, case-insensitive). Defaults to the drop_answers option. See drop_items().
clean_names	If TRUE (default), tidy the option labels by stripping prefix and un-mangling exporter artefacts ("A...B" -> "A / B").
split	How to handle a question that arrives packed into a single cell per respondent ("Speed; Drivers"). "auto" (default) detects ";", "|" or "," and splits on it; pass a string to force a delimiter, or FALSE never to split. Only ever applies when prefix selects exactly one column.

Details

The crucial difference from calc_percentage() is the denominator: it is the number of distinct respondents who picked *any* option in the block, not the number of ticks. So if 700 of 1,000 respondents selected at least one motivation, each option's percentage is "out of 700", and the percentages can (and usually do) sum to more than 100. Supply id to count distinct respondents exactly; without it, each row counts as one respondent. With clean_names = TRUE the option labels are stripped of prefix and exporter artefacts like "A...B" are turned back into "A / B".

Two export shapes are handled. The usual one is a block of columns, one per option, named with a shared prefix; that is what prefix selects. The other is a single column holding every answer a respondent ticked, joined by a delimiter, which is what Google Forms and most spreadsheet exports produce. If prefix matches exactly one column and a delimiter is found in it, that column is unpacked automatically and the answer text itself becomes the option label (clean_names is not applied, since the answers are real wording rather than mangled column names). Behaviour on a column block is unchanged. Use split_multi() when you want those unpacked columns kept for other work.

Value

A tibble with option, n (respondents choosing it) and pct.

See Also

calc_percentage(), split_multi(), plot_bars().

Other summaries: calc_percentage(), calc_percentage_batch(), calc_summary(), clear_checkpoints(), crosstab(), crosstab_banner()

Examples

```
calc_percentage_multi(podracing_survey, "motivations_",
                      id = respondent_id, sort = "desc")

# a spreadsheet export that packs every answer into one cell
packed <- data.frame(
  respondent = 1:4,
  motivations = c("Speed; Drivers", "Speed", "", "Betting; Speed")
)
calc_percentage_multi(packed, "motivations", id = respondent, sort = "desc")
```

---

calc_summary *Summarise a numeric question (mean / median / sd)*

Description

The numeric counterpart to calc_percentage(): a one-liner for the summarise(mean, median, sd) blocks used on age and other continuous variables, with optional grouping.

Usage

```
calc_summary(data = NULL, column, by = NULL, na_rm = TRUE, weights = NULL)
```

Arguments

data	A data frame.
column	Numeric column to summarise (unquoted).
by	Optional grouping column(s); see calc_percentage().
na_rm	Drop NA before summarising. Defaults to TRUE.
weights	Survey weighting: NULL (default) uses the session scheme from set_weights() if set; FALSE forces unweighted; or pass an ad-hoc scheme. When weighting is active the mean, median and sd become their weighted versions (n stays the unweighted base).

Details

The column is passed through `ensure_numeric()` first, so a text age column like `"25 years"` still summarises. `n` counts non-missing values (after that coercion). When a weighting scheme is active the statistics are weighted (and missing values are dropped); `n` remains the unweighted respondent count. For a measure of how precise the `mean` is, pair this with `diagnose()` or `se_mean()`.

Value

A tibble with `n` (non-missing count), `mean`, `median` and `sd`, one row per group when `by` is supplied.

See Also

`calc_percentage()` for categorical variables, `diagnose()` for precision.

Other summaries: `calc_percentage()`, `calc_percentage_batch()`, `calc_percentage_multi()`, `clear_checkpoints()`, `crosstab()`, `crosstab_banner()`

Examples

```
calc_summary(podracing_survey, demo_age)

calc_summary(podracing_survey, demo_age, by = region)
```

---

clean_label *Tidy an exported column name into a readable label*

Description

Turns the mangled column names survey tools produce back into the wording a reader expects: `"Broadcast.quality...audio"` becomes `"Broadcast quality / audio"`. Drop a question prefix at the same time with `prefix`, so a block of `ratings_*` columns becomes plain feature names.

Usage

```
clean_label(x, prefix = NULL)
```

Arguments

`x`	A character vector of column names.
`prefix`	Optional prefix to strip from the start of each name before tidying, e.g. `"ratings_"`. Names that do not start with it are left alone.

Details

Exporters replace every character they cannot put in a column name with a dot, so a slash surrounded by spaces arrives as three dots and a space arrives as one. This restores both, in that order, then squishes the result. It is the same cleaning `calc_percentage_multi()` and `ipm_model()` apply to their own output, exported so a hand-built table can match them: labels that disagree will not join, which is what silently empties a comparison chart.

Value

A character vector of labels, the same length as `x`.

See Also

`calc_percentage_batch()`, `calc_percentage_multi()`.

Other recode: `add_region()`, `bin_numeric()`, `drop_items()`, `ensure_numeric()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_likert()`, `recode_region()`, `split_multi()`

Examples

```
clean_label("Broadcast.quality...audio")

clean_label(c("ratings_Camera.work", "ratings_Talent...analysis"),
            prefix = "ratings_")
```

---

clear_brand *Clear the active brand*

Description

Resets every `brand_*` option so charts and reports return to the neutral ezrsurvey defaults.

Usage

```
clear_brand()
```

Value

Invisibly `TRUE`.

See Also

`use_brand()`.

Other brand: `brand_info()`, `pal_brand()`, `scale_fill_brand()`, `use_brand()`

Examples

```
clear_brand()
```

clear_checkpoints *Delete saved cross-tab checkpoints*

Description

Removes the checkpoint files that `crosstab_banner(checkpoint = TRUE)` keeps under `tools::R_user_dir()`. A finished run clears its own checkpoint, so anything left is from a run that was interrupted and never repeated; this empties the folder in one call.

Usage

```
clear_checkpoints()
```

Details

Only checkpoints the package named for itself are stored there. A checkpoint you pointed somewhere yourself (`checkpoint = "my-run.rds"`) is your file and is never touched, by this function or by a completed run.

Value

Invisibly the number of files removed.

See Also

`crosstab_banner()`, `clear_weights_cache()`.

Other summaries: `calc_percentage()`, `calc_percentage_batch()`, `calc_percentage_multi()`, `calc_summary()`, `crosstab()`, `crosstab_banner()`

Examples

```
withr::with_envvar(c(R_USER_CACHE_DIR = tempdir()), clear_checkpoints())
```

clear_weights_cache *Forget cached survey weights*

Description

Empties the session's cache of computed weight vectors. Weighting the same rows under the same scheme always gives the same numbers, so ezrsurvey works them out once and reuses them; a table crossing every question against every group would otherwise re-run the raking for each cell.

Usage

```
clear_weights_cache()
```

Details

The cache keys on the weighting columns' own contents, so editing the data produces different weights without any action from you. Clearing it by hand is only needed to reclaim memory, or after changing a weighting variable in place inside an object the cache has already seen. `clear_weights()` clears it too.

Value

`TRUE`, invisibly.

See Also

`set_weights()`, `weight_vector()`.

Other weighting: `get_weights()`, `set_weights()`, `weight_vector()`

Examples

```
clear_weights_cache()
```

`compare_values` *Compare a metric between two datasets (e.g. two events or waves)*

Description

Joins a `current` and a `previous` table on a key column and computes the difference of a metric – for example feature `performance` from two `ipm_model()` outputs, or `pct` from two `calc_percentage()` tables for two events. The result feeds straight into `plot_diff()`.

Usage

```
compare_values(current, previous, by = "feature", value = "performance")
```

Arguments

`current`, `previous`	Data frames sharing a key column and a metric column.
`by`	Name of the key column to join on. Default `"feature"`.
`value`	Name of the metric column to compare. Default `"performance"`.

Details

The join is a left join keyed on `by` with `previous` on the left, so the row set follows the previous period and `difference` is `current - previous` (a positive number means it went up). Keys present only in `current` are dropped; keys missing from `current` get an `NA` difference. The inputs are deliberately generic two-column-or-more tables, so the same function compares feature `performance` from two `ipm_model()` runs, `pct` from two `calc_percentage()` tables for two events, or any other keyed metric.

Value

A tibble with the key column, `previous`, `current` and `difference` (`current - previous`).

See Also

`plot_diff()`, `ipm_model()`, `calc_percentage()`.

Other compare: `plot_diff()`

Examples

```
a <- data.frame(feature = c("price", "quality"), performance = c(3.1, 4.2))
b <- data.frame(feature = c("price", "quality"), performance = c(2.8, 4.4))
compare_values(current = b, previous = a)
```

`convert_currency` *Convert amounts between currencies*

Description

Converts monetary amounts from one currency to another using a table of exchange rates. By default it uses the bundled currency_rates snapshot (units per US dollar); pass your own `rates` for up-to-date or custom figures. Because rates share the USD base, any pair is converted as a cross-rate through USD automatically (e.g. EUR to CNY goes EUR -> USD -> CNY in one call). `amount`, `from` and `to` are vectorised and recycled, so you can convert a whole column with per-row source currencies at once.

Usage

```
convert_currency(amount, from, to = "USD", rates = NULL)
```

Arguments

`amount`	Numeric amounts (text like `"$1,200"` is salvaged with `ensure_numeric()`).
`from`	Source currency code(s), e.g. `"EUR"` (case-insensitive). Common aliases resolve to their ISO code (e.g. `"RMB"` -> `"CNY"`).
`to`	Target currency code(s). Defaults to `"USD"`.
`rates`	Exchange rates: `NULL` (default) for the bundled snapshot, a named numeric vector of units-per-USD (e.g. `c(EUR = 0.92, GBP = 0.79)`), or a data frame with `currency` and `per_usd` columns.

Details

Each rate is stored as units of the currency per 1 US dollar, so a conversion is `amount / rate(from) * rate(to)` – which means any pair is handled as a cross-rate through USD without you doing anything. The bundled currency_rates table is a dated, approximate snapshot meant for convenience, not live trading; for accuracy pass your own `rates` (a named vector or a `currency`/`per_usd` data frame). Amounts are coerced with `ensure_numeric()`, codes are upper-cased and resolved through a small alias map (`"RMB" -> "CNY"`, `"EURO" -> "EUR"`, ...), and any unknown code yields `NA` with a warning rather than a silent wrong number.

Value

A numeric vector of converted amounts; unknown currency codes yield `NA` with a warning.

See Also

`add_currency()`, `list_currencies()`, currency_rates.

Other currency: `add_currency()`, `list_currencies()`

Examples

```
convert_currency(100, "EUR", "USD")

convert_currency(c(100, 50), from = c("EUR", "GBP"), to = "USD")

convert_currency(100, "EUR", "CNY")          # cross-rate via USD
convert_currency(100, "EUR", "RMB")          # alias -> CNY, same result
```

---

country_region *Country to region lookup*

Description

A tidy lookup table mapping country names to a world `region` and a finer `subregion`, used by `recode_region()` / `add_region()` to turn a country column into regions. Each country also carries its ISO 3166-1 codes, so a result can be reported against the standard.

Usage

```
country_region
```

Format

A tibble with 255 rows and 5 columns, one per entry in ISO 3166-1 that has a UN M49 grouping, plus Kosovo:

country Country name.

iso2 ISO 3166-1 alpha-2 code (e.g. `"DE"`). `NA` for Kosovo, which has no code of its own.

iso3 ISO 3166-1 alpha-3 code (e.g. `"DEU"`).

region World region (Africa, Asia, Europe, Middle East, North America, Oceania, South America).

subregion Finer subregion (e.g. Western Europe, East Asia).

Note that Namibia's alpha-2 code is the two letters `"NA"`, not a missing value, which is worth knowing before filtering on the column.

Source

Countries and codes are ISO 3166-1. The regions are the package's own grouping, which follows the UN M49 sub-regions but treats the Middle East as a region and splits the Americas north and south, because that is how consumer research reports them. See `data-raw/make_country_region.R`.

See Also

`recode_region()`, `add_region()`.

Other data: `currency_rates`, `podracing_survey`, `shopping_survey`

Examples

```
head(country_region)
recode_region("Germany")
```

`crosstab` *Cross-tabulate two survey questions*

Description

Builds a crosstab from two categorical columns: `x` forms the rows and `y` the columns. The cell contents are chosen with `cell` – counts or row/column/total percentages – or, when a numeric `value` column is supplied, an aggregate of that column (e.g. the mean spend) for each `x` by `y` combination. Registered orders (see `register_order()`) are applied to `x` and `y` automatically, and a factor column keeps its own level order when no order is registered.

Usage

```
crosstab(
  data = NULL,
  x,
  y,
  cell = c("count", "row_pct", "col_pct", "total_pct"),
  value = NULL,
  fn = mean,
  wide = TRUE,
  digits = NULL,
  na_rm = TRUE,
  drop = NULL,
  weights = NULL
)
```

Arguments

data	A data frame.
x	Row variable (unquoted).
y	Column variable (unquoted).
cell	What to put in each cell when value is not given: "count" (default), "row_pct", "col_pct" or "total_pct".
value	Optional numeric column (unquoted) to aggregate per cell instead of counting.
fn	Aggregation function used with value. Default mean().
wide	If TRUE (default), return one row per x with a column per y level; if FALSE, return the tidy long form.
digits	Decimal places for percentage / aggregated cells. Default 0 for percentages, 2 when value is supplied.
na_rm	Drop blanks / non-answers in x, y (and NA in value). Default TRUE.
drop	Optional character vector of answer values to remove from both x and y before tabulating (e.g. c("Other")), matched case-insensitively. Defaults to the drop_answers option. See drop_items().
weights	Survey weighting: NULL (default) uses the session scheme from set_weights() if set; FALSE forces unweighted; or pass an ad-hoc scheme. When weighting is active the cells default to weighted values – weighted counts, weighted percentages, or (for a numeric value) the weighted mean.

Details

Choose what the cells mean with cell: "row_pct" makes each **row** sum to 100 (the distribution of y within each x), "col_pct" makes each **column** sum to 100, "total_pct" is the share of the whole table, and "count" is the raw frequency. Supplying a numeric value switches the cells to an aggregate of that column – by default the mean – which answers questions like "what is the average NPS for each region by gender?". Blanks and non-answers in x/y are dropped when na_rm = TRUE. Any registered orders (register_order()) set the row/column ordering automatically; a factor without one keeps its own level order; anything else stays in data order.

Value

A tibble: wide (x plus one column per y level) or long (x, y, value). The long form refuses an x or y that is itself called value, since the two columns would share a name.

See Also

calc_percentage(), compare_values(), register_order().

Other summaries: calc_percentage(), calc_percentage_batch(), calc_percentage_multi(), calc_summary(), clear_checkpoints(), crosstab_banner()

Examples

```
# counts
crosstab(podracing_survey, demo_gender, region)

# row percentages: gender split within each region (each row ~ 100)
crosstab(podracing_survey, region, demo_gender, cell = "row_pct")

# mean NPS for each region x gender cell
crosstab(podracing_survey, region, demo_gender, value = nps_value, fn = mean)
```

`crosstab_banner` *Build a master banner (cross-tab) table*

Description

Produces the market-research "banner" table: one master table with a stack of question variables down the side (`rows`) and one or more grouping variables across the top (`cols`), plus an **Overall** column for the whole sample. Categorical questions become column-percentage blocks (each banner column sums to ~100 within the block); numeric questions become a mean / median / sd / quartile block. It generalises `crosstab()` from a single pair to a whole table, and picks the cell content per question automatically.

Usage

```
crosstab_banner(
  data = NULL,
  rows,
  cols,
  cell = c("auto", "pct", "count", "mean", "diff"),
  stats = c("mean", "median", "sd", "p25", "p75"),
  total = TRUE,
  digits = NULL,
  na_rm = TRUE,
  drop = NULL,
  weights = NULL,
  max_levels = 20,
  long = FALSE,
  flextable = FALSE,
  checkpoint = NULL,
  confirm = NULL
)
```

Arguments

`data`	A data frame. If omitted, the session default (`use_dataset()`) is used.
`rows`	Stub variables, down the side (tidyselect, e.g. `c(satis_return, demo_edu)` or `starts_with("ratings_")`). If omitted, every eligible variable is used (see `max_levels` and Details).

cols	Banner / grouping variables, across the top (tidyselect). Pass several to get several spanning column groups. If omitted, every eligible variable is used, giving a full variable-by-variable banner.
cell	What each body cell holds: "auto" (default) uses column percentages for categorical questions and a statistics block for numeric ones; "pct" forces column percentages; "count" uses (weighted) counts; "mean" forces a numeric mean (errors on a categorical question); "diff" shows each cell's difference from the Overall column (percentage points, or mean difference for numeric questions).
stats	For numeric questions, which statistics form the block. Any of "mean", "median", "sd", "p25", "p75". Default all five.
total	Include the whole-sample **Overall** column. Default TRUE.
digits	Decimal places. NULL (default) uses 0 for percentages / counts and 2 for numeric statistics; a value overrides both.
na_rm	Drop blanks / non-answers in the questions and banner variables. Default TRUE.
drop	Answer values to remove before tabulating (see drop_items()), passed to the underlying crosstab() calls. Defaults to the drop_answers option.
weights	Survey weighting: NULL (default) uses the session scheme from set_weights() if set; FALSE forces unweighted; or pass an ad-hoc scheme. When weighting is active the cells are weighted (weighted percentages, counts, means and quartiles).
max_levels	When rows / cols are omitted, the largest number of distinct answers a categorical variable may have to be used automatically (default 20). Bigger categorical variables, identifier columns and free-text are skipped; numeric scales are always kept as stub rows, and binary-encoded columns (an option label or blank) as banner groups.
long	If TRUE, return the tidy long form (variable, item, group, group_item, value) instead of the wide master table. Default FALSE.
flextable	If TRUE, return a flextable with the two-row banner header ready for a slide or Word report. Default FALSE. Requires the suggested flextable package.
checkpoint	Save each question as it finishes, so re-running the same call picks up where an interrupted run stopped. TRUE keeps the file under tools::R_user_dir() and deletes it once the run completes; a file path puts it where you choose and leaves it there for you to manage. NULL (default) or FALSE writes nothing. See Details and clear_checkpoints().
confirm	When rows / cols are left to the automatic selection, show which variables were chosen and which were skipped, and wait for a yes before the run starts. NULL (default) follows the confirm option, which asks in an interactive session and never asks in a script; TRUE or FALSE force it either way. Answering no returns NULL and computes nothing.

Details

A tibble has a single header row, so the two-row banner header (each grouping variable's name spanning its items) cannot be stored in the data itself: the wide return carries the grouping in the "banner_spanners" attribute and flextable = TRUE renders the real spanning header. Column

percentages are computed *within* each banner column, so every column (Overall included) sums to about 100 down each question block. Registered orders (`register_order()`) set the item and banner-column ordering automatically. Passing the same selection to `rows` and `cols` gives the full every-question-by-every-question matrix (a variable is never crossed with itself). Supplying only the data frame does this automatically: every variable with at most `max_levels` distinct answers becomes both a stub and a banner group, numeric scales are added as stub statistics blocks, and identifier / free-text columns are skipped (and named in a message). That selection decides the whole table, so an interactive session prints it and waits for a yes before starting the run (see `confirm`). Check-all-that-apply blocks (columns sharing a prefix that each hold one option or blank, such as `motivations_*`) are recognised as a single multi-select stub question and tabulated with `calc_percentage_multi()`, whose base is the respondents who picked any option (so those rows can sum past 100); those stub rows are unweighted. Across the top, a binary-encoded column gives one column of the respondents who ticked it, and the columns of a multi-select block share one spanner named after the block. Those columns overlap, since a respondent can tick several. A lone binary column is used across the top only, never as a stub.

A full every-variable banner is one `crosstab()` per question per grouping variable, so a wide survey takes minutes rather than seconds. In an interactive session it reports which question it is on, with an estimate of the time left; `ezrsurvey_options(progress = FALSE)` turns that off, and scripts are silent by default. `checkpoint = TRUE` additionally saves each question as it completes, so re-running the identical call after an interruption resumes instead of starting again. The managed file is named after the run's own fingerprint, so a call finds its own interrupted run and a call with different data or arguments never sees it, and it is deleted as soon as the table is built. Pass a path instead if you would rather choose the location; that file is yours, so the package neither deletes it nor counts it as one of its own. `clear_checkpoints()` empties the managed folder if a run was interrupted and never repeated.

Value

By default a wide tibble: `variable`, `item`, `Overall`, then one column per banner item. The banner grouping (which column belongs to which top-row variable) is stored in the `"banner_spanners"` attribute, which `flextable = TRUE` turns into a spanning two-row header. With `long = TRUE`, the tidy long form.

See Also

`crosstab()` for a single pair, `calc_percentage_batch()` for stacked one-variable percentages, `export_summary_xlsx()`, `register_order()`.

Other summaries: `calc_percentage()`, `calc_percentage_batch()`, `calc_percentage_multi()`, `calc_summary()`, `clear_checkpoints()`, `crosstab()`

Examples

```
# gender and region banner over two questions (column percentages)
crosstab_banner(podracing_survey,
                rows = c(satis_return, demo_edu),
                cols = c(demo_gender, region))

# numeric questions become a mean / median / sd / quartile block
crosstab_banner(podracing_survey,
```

```
                rows = c(nps_value, demo_age),
                cols = demo_gender)

# each cell as its difference from the Overall column
crosstab_banner(podracing_survey, rows = satis_return,
                cols = region, cell = "diff")


# full variable-by-variable banner: just pass the data frame
crosstab_banner(podracing_survey)
```

`currency_rates` *Currency exchange-rate snapshot*

Description

A dated, approximate reference table of exchange rates used by `convert_currency()` / `add_currency()` when no custom rates are supplied. Rates are expressed as units of the currency per 1 US dollar (base `USD`). These are a convenience snapshot, **not** live rates – pass your own `rates` to the conversion helpers when accuracy matters.

Usage

```
currency_rates
```

Format

A tibble with one row per currency and columns:

currency ISO 4217 code (e.g. `"EUR"`).

name Currency name.

per_usd Units of the currency per 1 US dollar.

The snapshot date is stored in `attr(currency_rates, "snapshot_date")`.

Source

Approximate reference snapshot. See `data-raw/make_currency_rates.R`.

See Also

`convert_currency()`, `add_currency()`, `list_currencies()`.

Other data: `country_region`, `podracing_survey`, `shopping_survey`

Examples

```
head(currency_rates)
attr(currency_rates, "snapshot_date")
```

`diagnose` *Survey precision diagnostics for one or more columns*

Description

The headline diagnostics wrapper: for each selected column it auto-detects whether it is a point estimate (numeric, e.g. a 1-5 rating) or a proportion (categorical), and returns a tidy table of sample size, estimate, standard error, relative standard error and margin of error – i.e. the whole appendix "Data info" block as one call.

Usage

```
diagnose(
  data = NULL,
  ...,
  type = c("auto", "mean", "prop"),
  by = NULL,
  z = 1.96,
  digits = 2
)
```

Arguments

`data`	A data frame.
`...`	Columns to diagnose, using tidyselect (e.g. `starts_with("ratings_")`, `demo_gender`).
`type`	Force the estimate type: `"auto"` (default, detect per column), `"mean"` or `"prop"`.
`by`	Optional grouping column(s); diagnostics are computed within group.
`z`	Margin-of-error multiplier passed to `margin_of_error()`. Defaults to `1.96` (95%).
`digits`	Decimal places for the numeric output columns. Defaults to `2`.

Details

For categorical columns the reported estimate is the **largest answer category's** share, which gives a concrete `+/-` percentage-point margin to quote in a report footnote.

Each column is classified (or forced via `type`) as a point estimate or a proportion. For numeric columns the estimate is the mean and the error is in rating points; for categorical columns the estimate is the **largest answer category's** share, which gives a single concrete `+/-` percentage-point margin to quote (the largest category is also the worst case among the observed categories, so it is a sensible headline). The `precision` column is the five-band rating from `rse_rating()`. This is the per-variable form of the report-appendix "Data info" block; for the narrative bullet-point version see `precision_summary()`.

Value

A tibble with one row per column (per group), with columns `variable`, `type`, `unit`, `n`, `estimate`, `se`, `rse`, `moe` and `precision`. For `type == "mean"`, `unit` is `"points"`; for proportions it is `"ppt"` (percentage points) and `estimate` is a percentage.

See Also

`precision_summary()`, `rse_rating()`, `se_mean()`, `se_prop()`.

Other diagnostics: `margin_of_error()`, `precision_summary()`, `rse()`, `rse_rating()`, `se_mean()`, `se_prop()`

Examples

```
diagnose(podracing_survey, demo_gender, nps_value)

diagnose(podracing_survey, starts_with("ratings_"))
```

---

drop_items *Drop unwanted answer categories from a question*

Description

Removes specific answer values – typically catch-all or uninformative options such as "Other", "Don't know" or "Not applicable" – by turning them into `NA`, so they fall out of counts, percentages and charts and the remaining percentages re-base on the answers you keep. The companion to `na_blank()` (which targets blanks and standard non-answers): use `drop_items()` for the substantive categories you simply do not want to show.

Usage

```
drop_items(x, items, trim = TRUE)
```

Arguments

`x`	A character (or factor) vector.
`items`	Character vector of exact answer values to drop. Matching is case-insensitive.
`trim`	Whether to trim surrounding whitespace before comparing. Defaults to `TRUE`.

Details

`x` is coerced to character (so factors are handled) and, by default, whitespace-trimmed; any value equal (ignoring case) to one of `items` becomes `NA`. The summary helpers (`calc_percentage()`, `calc_percentage_multi()`, `calc_percentage_batch()`) and `crosstab()` take a `drop =` argument that applies this for you *before* counting, so the kept answers re-base to ~100%; set a session-wide default with `ezrsurvey_options(drop_answers = ...)`.

Value

A character vector the same length as `x`, with any value matching `items` replaced by `NA`.

See Also

`na_blank()`, `calc_percentage()`.

Other recode: `add_region()`, `bin_numeric()`, `clean_label()`, `ensure_numeric()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_likert()`, `recode_region()`, `split_multi()`

Examples

```
drop_items(c("Yes", "No", "Other", "Don't know"),
           items = c("Other", "Don't know"))
```

`edit_ezrsurvey_profile` *Open the ezrsurvey profile for editing*

Description

Opens your ezrsurvey YAML profile in the editor (like `usethis::edit_r_profile()` for `.Rprofile`), creating it from a template first if it does not exist. After saving, reload it with `load_ezrsurvey_profile()` or by restarting the session.

Usage

```
edit_ezrsurvey_profile(path = NULL)
```

Arguments

`path`	Profile path. If `NULL` (default), the first existing profile among `ezrsurvey_profile_paths()` is used, or the per-user config file is created.

Value

Invisibly the path opened.

See Also

`use_ezrsurvey_profile()`, `save_ezrsurvey_profile()`, `load_ezrsurvey_profile()`.

Other config: `apply_order()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
tmp <- tempfile(fileext = ".yml")
edit_ezrsurvey_profile(tmp)
```

---

ensure_numeric *Coerce text to numeric, salvaging embedded numbers*

Description

A defensive helper for the common survey-export headache where a column that should be numeric arrives as text – `"25 years"`, `"8 - very likely"`, `"3.5/5"`. It returns `x` unchanged if it is already numeric; otherwise it pulls the first number out of each value (via a digit pattern) and coerces to numeric, leaving genuinely unparseable entries as `NA`. The ezrsurvey functions that need numbers (e.g. `calc_nps()`, `calc_summary()`) call this for you, so they "just work" on lightly messy data.

Usage

```
ensure_numeric(x, name = NULL, quiet = FALSE)
```

Arguments

`x`	A vector. Numeric input is returned as-is.
`name`	Optional column/argument name, used only to make the message clearer.
`quiet`	If `FALSE` (default), emit a one-line message when coercion happens (and note any values that could not be parsed). Set `TRUE` to silence it.

Details

The first numeric token in each value is extracted with the pattern `-?[0-9]*\.?[0-9]+`, which captures negatives and decimals but takes only the *first* number it finds – so `"8 - very likely"` becomes `8`, not `-8`. A value that is genuinely numeric already is returned untouched (no copy, no message). When `quiet = FALSE` a single informative message reports that coercion happened and how many values could not be parsed; the internal callers (`calc_nps()`, `calc_summary()`, `ipm_model()`, `bin_numeric()`, ...) pass `quiet = TRUE` so they don't spam your console. Truly blank entries (`""`/`NA`) are not counted as parse failures.

Value

A numeric vector the same length as `x`.

See Also

`recode_age()`, `recode_likert()`, `na_blank()`.

Other recode: `add_region()`, `bin_numeric()`, `clean_label()`, `drop_items()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_likert()`, `recode_region()`, `split_multi()`

Examples

```
ensure_numeric(c("25 years", "31", "forty"), quiet = TRUE)

ensure_numeric(c("8 - very likely", "10", "3.5/5"), quiet = TRUE)

ensure_numeric(c(1, 2, 3))    # already numeric, returned as-is
```

`example_report` *Copy the worked example report into a folder*

Description

Drops the package's complete worked example next to your data: a full Quarto report over the bundled `podracing_survey` (every chart type, real narrative) plus the matching slide-deck script on the officer path. Render / run them as they are, then swap in your own data.

Usage

```
example_report(dir, overwrite = FALSE)
```

Arguments

`dir`	Destination folder, created if missing.
`overwrite`	Overwrite existing files of the same names. Defaults to `FALSE`.

Details

Two files are copied: `podracing-report.qmd` (render with `quarto render podracing-report.qmd`) and `podracing-deck.R` (run with `Rscript podracing-deck.R`; writes a 16:9 deck to `ezrsurvey-outputs/`). Unlike the blank `scaffold_report()` skeletons, the example ships finished narrative over the bundled data, so you can see every visual and slide element in a finished state before adapting it.

Value

Invisibly the paths written.

See Also

`scaffold_report()` for blank skeletons, `report_deck()`.

Other reporting: `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_new()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
dir <- file.path(tempdir(), "ezrsurvey-example")
example_report(dir)
list.files(dir)
```

`export_summary_xlsx` *Export a per-question summary workbook (table + chart per sheet)*

Description

Writes one Excel workbook with a worksheet per question, each holding that question's summary table and its chart. Numeric questions get a `calc_summary()` table and a histogram; categorical questions get a `calc_percentage()` table and a `plot_bars()` chart; check-all-that-apply blocks get a `calc_percentage_multi()` table (based on the respondents who picked any option) and a bar chart. This is the "hand the client a tabbed workbook" counterpart to the slide/Word builders (`report_deck()`). Requires the suggested `openxlsx2` package (the only xlsx writer here that can embed images).

Usage

```
export_summary_xlsx(
  data = NULL,
  ...,
  path,
  by = NULL,
  chart = TRUE,
  sort = c("none", "desc", "asc"),
  digits = 0,
  weights = NULL,
  width = 6,
  height = 3.4,
  confirm = NULL
)
```

Arguments

`data`	A data frame. If omitted, the session default (`use_dataset()`) is used.
`...`	Variables to summarise, using tidyselect (e.g. `starts_with("ratings_")` or `demo_gender, satis_return`). One worksheet per question. If omitted, every question is written: each eligible column plus any multi-select block, skipping identifier and free-text columns.
`path`	Output `.xlsx` path. A bare file name is written to the working directory, or into the folder set by the `output_dir` option.
`by`	Optional grouping column(s) for the tables (passed to `calc_percentage()` / `calc_summary()` / `calc_percentage_multi()`). The chart is always the ungrouped distribution.
`chart`	Include the chart beneath each table. Default `TRUE`.
`sort`	Level ordering for categorical tables, passed to `calc_percentage()`: `"none"` (default, respecting any registered order), `"desc"` or `"asc"`. The chart always auto-lays-out its bars.

digits	Decimal places for percentages. Default 0.
weights	Survey weighting, passed to the table and chart helpers (see calc_percentage()). NULL (default) uses the session scheme if set. Multi-select blocks are always unweighted.
width, height	Chart size in inches. Default 6 x 3.4.
confirm	When no variables are named, show which questions were chosen and which were skipped, and wait for a yes before writing. NULL (default) follows the confirm option, which asks in an interactive session and never asks in a script; TRUE or FALSE force it either way. Answering no writes nothing and returns NULL.

Details

Worksheets are named from each question (cleaned to valid, unique Excel names). The table is written from cell A1; the chart, when included, is rendered to a temporary PNG and embedded a few rows below it. A numeric column (after ensure_numeric()) is treated as a scale question (summary + histogram); columns that share a prefix and each hold a single option or blank (e.g. motivations_*) are treated as one check-all-that-apply question; anything else is categorical (percentages + bar chart). With no ..., every question in the data is written and identifier / free-text columns are skipped (and named in a message). For a single table or a pre-named list of tables in one workbook without charts, see export_xlsx() / save_data().

Value

Invisibly the path written.

See Also

export_xlsx(), save_data(), report_deck(), calc_percentage_batch().

Other save: export_xlsx(), save_data(), save_output(), save_plot()

Examples

```
# named questions
tmp <- tempfile(fileext = ".xlsx")
export_summary_xlsx(podracing_survey, demo_gender, satis_return, nps_value,
                    path = tmp)

# or every question at once (identifier / free-text columns are skipped)
tmp2 <- tempfile(fileext = ".xlsx")
export_summary_xlsx(podracing_survey, path = tmp2)
```

`export_xlsx` *Export several tables to one Excel workbook, a tab each*

Description

A quick way to drop a handful of summaries into a single `.xlsx`, one per worksheet – e.g. `export_xlsx(calc_percentage(d, a), calc_percentage(d, b), path = "out.xlsx")`. Tabs are named from the argument names you give, else from each table's question/first column, else `Sheet1`, `Sheet2`, ...

Usage

```
export_xlsx(..., path, sheet_names = NULL)
```

Arguments

`...`	Data frames to write, one per tab. Name them to set tab names (e.g. `gender = calc_percentage(d, demo_gender)`).
`path`	Output `.xlsx` path. A bare file name is written to the working directory, or into the folder set by the `output_dir` option.
`sheet_names`	Optional character vector of tab names (overrides argument names).

Details

Each argument becomes one worksheet. Tab names are taken from the argument names you give (`gender = ...`), otherwise from each table's `variable` column or first column, otherwise `Sheet1`, `Sheet2`, ...; they are then cleaned to valid, unique Excel names (<= 31 characters, with `[ ] : * ? / \\` stripped). This is the quick way to drop a set of summaries into one workbook for a colleague. Requires the suggested `writexl` package.

[ ]: R:%20

Value

Invisibly `path`.

See Also

`save_data()` for a single table or a pre-named list.

Other save: `export_summary_xlsx()`, `save_data()`, `save_output()`, `save_plot()`

Examples

```
tmp <- tempfile(fileext = ".xlsx")
export_xlsx(
  gender = calc_percentage(podracing_survey, demo_gender),
  calc_percentage(podracing_survey, demo_job),
  path = tmp
)
file.exists(tmp)
```

`ezrsurvey_options` *Get or set ezrsurvey defaults*

Description

ezrsurvey reads a handful of defaults from R options (prefixed `ezrsurvey.`), so you can tune behaviour once instead of repeating arguments. Call with no arguments to see the current effective values; call with `name = value` pairs to set them for the session. Persist them across sessions with a YAML profile (see `use_ezrsurvey_profile()`).

Usage

```
ezrsurvey_options(...)
```

Arguments

`...`	Either nothing (to read all values) or named `option = value` pairs to set.

Details

Recognised options:

`pct_axis_unit` Rounding step for percentage y-axes (default `25`).

`pct_axis_max` Fixed percentage y-axis maximum, e.g. `100`; `NULL` (default) means dynamic via `nice_max()`.

`bar_cols_max_items`, `bar_cols_max_label` Thresholds that switch `plot_bars()` (`orientation = "auto"`) from vertical columns to horizontal bars: more than `bar_cols_max_items` bars (`6`) or any label longer than `bar_cols_max_label` characters (`12`).

`bar_wrap_cols`, `bar_wrap_bars` `stringr::str_wrap()` widths used by `plot_bars()` for column (`12`) and bar (`28`) labels.

`bar_label_size`, `bar_size_step`, `bar_size_min` Data-label text sizing in `plot_bars()`: base size (`3.5`), reduction per item past the item threshold (`0.15`) and the floor (`2.4`).

`bar_width`, `bar_ref_items` Bar thickness. Bars are drawn at `bar_width` (`0.66`) of a category slot when a chart has `bar_ref_items` (`6`) bars, and the fraction scales with the bar count so the *drawn* thickness stays the same on every chart. Raise `bar_ref_items` for thinner bars throughout, lower it for chunkier ones.

`age_breaks`, `age_labels` Default bands used by `recode_age()`.

`na_answers` Strings treated as non-answers by `na_blank()`.

`drop_answers` Answer values dropped by the `drop =` argument of `calc_percentage()` and friends when `drop` is not given; `NULL` (default) means drop nothing. See `drop_items()`.

`generation_scheme` Default scheme for `recode_generation()`.

`current_year` Reference year for age-to-cohort conversion; `NULL` uses the system year.

`default_by` Default grouping column name(s) applied by `calc_percentage()` / `calc_percentage_multi()` when `by` is omitted; `NULL` (default) means no default grouping.

`brand_colors`, `brand_color_primary` Organisation brand colours: a vector of accent hexes and the primary accent used as the default single-series fill. Usually set by `use_brand()`, but can be set by hand or in a profile. `NULL` (default) keeps the neutral ezrsurvey look.

`brand_font_major`, `brand_font_minor` Brand heading and body typefaces. `brand_font_minor` becomes the default `base_family` of `theme_ezrsurvey()` when the font is installed on this machine.

`brand_fonts_enabled` Set `FALSE` to keep brand colours but ignore brand fonts (default `TRUE`).

`brand_template_pptx`, `brand_template_docx` Paths to the brand PowerPoint / Word template used as the default reference document by `report_new()`, `report_deck()` and `scaffold_report()`. These are machine-specific absolute paths – if you persist them, prefer the project-level `.ezrsurvey.yml` profile over the user-level one.

`progress` Whether the slow helpers report which item they are on. `"auto"` (default) reports in an interactive session and stays silent in scripts, vignettes and `R CMD check`; `TRUE` or `FALSE` force it either way. See `crosstab_banner()`.

`confirm` Whether a call that picks its own variables shows the selection and waits for a yes before a long run. `"auto"` (default) asks in an interactive session and never asks in a script; `TRUE` or `FALSE` force it either way. See `crosstab_banner()`.

`output_dir` Folder that a bare output file name is written into, created on demand. The default `"."` writes it where it is named: in the working directory. Set a folder, say `"outputs"`, and `save_plot(p, "chart.png")` writes `outputs/chart.png`, so a script's results collect in one place. A path that names a directory (`"./chart.png"`, `"charts/chart.png"`, anything absolute) is always written exactly where it says.

Options are ordinary R options under the `ezrsurvey.` prefix, so anything you can do with `options()` works here too, and a setting lasts for the session. Reading (`ezrsurvey_options()` with no arguments) returns every recognised option's current effective value; setting returns the previous values invisibly, so you can restore them. To make settings permanent, write them to a YAML profile – edit it with `edit_ezrsurvey_profile()` or dump the current session with `save_ezrsurvey_profile()` – which is loaded automatically at package start-up.

Value

When reading, a named list of all current values. When setting, the previous values, invisibly.

See Also

`use_ezrsurvey_profile()`, `load_ezrsurvey_profile()`, `reset_ezrsurvey_options()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
ezrsurvey_options()                      # view all defaults
ezrsurvey_options(pct_axis_max = 100)
ezrsurvey_options()$pct_axis_max         # 100
reset_ezrsurvey_options()                # back to defaults
```

ezrsurvey_profile_paths *Profile file locations ezrsurvey looks in*

Description

At load time ezrsurvey reads a YAML profile from each of these paths in turn (later paths override earlier ones), if present: the per-user config directory, your home directory, then the current project.

Usage

```
ezrsurvey_profile_paths()
```

Value

A character vector of candidate profile paths.

See Also

`use_ezrsurvey_profile()`, `load_ezrsurvey_profile()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
ezrsurvey_profile_paths()
```

generation_scheme *Generational-cohort definitions*

Description

Returns the birth-year boundaries for a named generational scheme, for use by `recode_generation()`. The default `"pew"` scheme follows the widely cited Pew Research Center cohort definitions.

Usage

```
generation_scheme(scheme = c("pew"))
```

Arguments

`scheme`	Scheme name. Currently `"pew"`.

Details

The "pew" scheme uses the Pew Research Center boundaries: Silent (1928–45), Baby Boomer (1946–64), Gen X (1965–80), Millennial (1981–96), Gen Z (1997–2012) and Gen Alpha (2013+). To use different cut-offs, build your own tibble with from and label columns and pass it to recode_generation()'s scheme argument.

Value

A tibble with label, from (first birth year) and to (last birth year; Inf for the open-ended youngest cohort).

See Also

recode_generation().

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), na_blank(), nps_group(), recode_age(), recode_generation(), recode_likert(), recode_region(), split_multi()

Examples

```
generation_scheme("pew")
```

---

get_dataset *Get, check or clear the default dataset*

Description

Companions to use_dataset() for inspecting and resetting the session's default dataset.

Usage

```
get_dataset()

has_dataset()

clear_dataset()

dataset_vars()
```

Details

has_dataset() is the safe way to check before calling get_dataset(), which errors when nothing is set. clear_dataset() removes the default so that subsequent calls require an explicit data again – useful at the end of a script or between analyses of different datasets.

Value

`get_dataset()` returns the default data frame (error if none is set); `has_dataset()` returns a logical; `clear_dataset()` returns `TRUE` invisibly; `dataset_vars()` returns a character vector of column names.

See Also

`use_dataset()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
has_dataset()

use_dataset(podracing_survey)
has_dataset()

nrow(get_dataset())

clear_dataset()
```

---

get_order *Get the levels of a registered order*

Description

Get the levels of a registered order

Usage

```
get_order(name)
```

Arguments

`name`	Order name.

Value

The character vector of levels.

See Also

register_order().

Other config: apply_order(), edit_ezrsurvey_profile(), ezrsurvey_options(), ezrsurvey_profile_paths(), get_dataset(), list_orders(), load_ezrsurvey_profile(), order_for(), register_order(), register_order_presets(), remove_order(), reset_ezrsurvey_options(), save_ezrsurvey_profile(), use_dataset(), use_ezrsurvey_profile()

Examples

```
register_order("yesno", c("No", "Yes"))
get_order("yesno")
remove_order("yesno")
```

---

get_weights *Inspect or clear the session weighting scheme*

Description

Companions to set_weights(): read the stored scheme, test whether one is set, or remove it.

Usage

```
get_weights()

has_weights()

clear_weights()
```

Details

has_weights() is the safe check before get_weights(), which errors when no scheme is set. clear_weights() turns weighting off again, so the helpers return purely unweighted results.

Value

get_weights() returns the scheme (a named list of target vectors; error if none is set); has_weights() returns a logical; clear_weights() returns TRUE invisibly.

See Also

set_weights(), weight_vector().

Other weighting: clear_weights_cache(), set_weights(), weight_vector()

Examples

```
has_weights()
set_weights(c(variable = "region", Europe = 0.5, "North America" = 0.5))
get_weights()
clear_weights()
```

ipm_model *Build an importance / performance model*

Description

Combines driver **importance** (against `outcome`) with feature **performance** (mean rating) into the tidy table consumed by `plot_ipm()`. Worded rating columns are recoded to 1-5 automatically.

Usage

```
ipm_model(
  data = NULL,
  outcome,
  rating_prefix,
  method = c("rwa", "forest", "correlation"),
  recode = TRUE,
  likert_levels = c("Very bad", "Bad", "Ok", "Good", "Very good")
)
```

Arguments

`data`	A data frame.
`outcome`	The outcome column (unquoted), e.g. `nps_value`.
`rating_prefix`	Column-name prefix identifying the rating block, e.g. `"ratings_"`.
`method`	How to measure importance, passed to `calc_importance()`. Defaults to `"rwa"`.
`recode`	If `TRUE` (default), character rating columns are mapped to 1-5 with `recode_likert()`; numeric columns are left as-is.
`likert_levels`	Scale wordings passed to `recode_likert()` when recoding.

Details

An importance/performance model answers two questions per feature at once: *how much does it drive the outcome* (importance, via `calc_importance()`) and *how well are we doing on it* (performance, the mean 1–5 rating). Plotting one against the other (`plot_ipm()`) reveals priorities: high-importance, low-performance features are where to invest. Worded rating columns are mapped to 1–5 with `recode_likert()` automatically (`recode = TRUE`); the prefix is stripped from feature names; and `perf_class` buckets performance into a 1–5 factor by its integer part (a 3.57 mean is band 3, not 4) for colouring. Importance comes from `calc_importance()`, so `method` chooses between relative weights (the default, requiring the suggested `rwa` package), a random forest and a plain correlation.

Value

A tibble with `feature`, `importance`, `performance` and `perf_class` (the performance band as a 1-5 factor, from its integer part so a 3.57 mean sits in band 3, used for colouring).

See Also

calc_importance(), plot_ipm(), compare_values().

Other modelling: calc_importance(), calc_nps()

Examples

```
ipm_model(podracing_survey, nps_value, "ratings_")
```

---

label_pct *Percent label formatter*

Description

Returns a function that turns numbers into percent strings, e.g. 42 becomes "42%". The input is assumed to already be on a 0-100 scale (as produced by calc_percentage()), not a 0-1 proportion.

Usage

```
label_pct(digits = 0, suffix = "%")
```

Arguments

digits	Number of decimal places to keep. Defaults to 0.
suffix	String appended after the number. Defaults to "%".

Details

label_pct() returns a *function* (a closure), not formatted strings – this is the form ggplot2 scales expect for their labels argument. The number is rounded to digits decimal places with formatC() and suffix appended. Because the input is assumed to be on a 0–100 scale, label_pct()(42) is "42%", not "4200%".

Value

A function suitable for the labels argument of a ggplot2 scale or for use with geom_text().

See Also

scale_y_pct().

Other scales: nice_max(), scale_y_pct()

Examples

```
f <- label_pct()
f(c(0, 33.4, 100))

label_pct(1)(33.45)
```

list_currencies *List the currencies in a rate table*

Description

List the currencies in a rate table

Usage

```
list_currencies(rates = NULL)
```

Arguments

rates	Rate table; NULL (default) for the bundled currency_rates.

Details

A quick way to see which codes convert_currency() will accept from the bundled snapshot (or from a rates table you pass). Aliases such as "RMB" are *not* listed – they resolve to their ISO code (here "CNY").

Value

A character vector of currency codes.

See Also

currency_rates, convert_currency().

Other currency: add_currency(), convert_currency()

Examples

```
list_currencies()
```

list_orders *List registered orders*

Description

List registered orders

Usage

```
list_orders()
```

Value

A tibble with `name`, `n_levels`, `vars` and `prefixes` (the last two comma-joined).

See Also

`register_order()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
register_order("yesno", c("No", "Yes"), vars = "agree")
list_orders()
remove_order("yesno")
```

`list_report_templates` *Available Quarto report templates*

Description

Available Quarto report templates

Usage

```
list_report_templates()
```

Value

A character vector of the output formats ezrsurvey ships a Quarto skeleton for.

See Also

`scaffold_report()`.

Other reporting: `example_report()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_new()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
list_report_templates()
```

load_ezrsurvey_profile *Load ezrsurvey defaults from a YAML profile*

Description

Reads a YAML file of `option: value` pairs and applies them with `ezrsurvey_options()`. Called automatically when the package loads (over the standard `ezrsurvey_profile_paths()`); call it manually to load a specific file. Requires the suggested `yaml` package; without it, the function is a no-op.

Usage

```
load_ezrsurvey_profile(path = NULL, quiet = TRUE)
```

Arguments

`path`	Profile path(s). If `NULL` (default), the standard locations are tried in order (later files override earlier ones).
`quiet`	If `TRUE` (default), load silently; if `FALSE`, message which file(s) were loaded.

Value

Invisibly `TRUE` if any profile was applied, else `FALSE`.

See Also

`use_ezrsurvey_profile()` to create a template.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
tmp <- tempfile(fileext = ".yml")
writeLines("pct_axis_max: 100", tmp)
load_ezrsurvey_profile(tmp, quiet = FALSE)
ezrsurvey_options()$pct_axis_max
reset_ezrsurvey_options()
```

`margin_of_error` *Margin of error from a standard error*

Description

`z * se`, the half-width of a confidence interval. The default `z = 1.96` gives an approximate 95% interval; the survey reports used the `2 * se` rule of thumb, which is `z = 2`.

Usage

```
margin_of_error(se, z = 1.96)
```

Arguments

`se`	A standard error (scalar or vector).
`z`	The critical value / multiplier. Defaults to `1.96` (95%). Use `2` to reproduce the report's rule of thumb.

Details

A confidence interval is the estimate plus or minus this margin. `z = 1.96` gives the standard 95% interval; `z = 2` reproduces the "two standard errors" rule of thumb often quoted in survey reports; `z = 2.58` gives 99%. The result is on the same scale as `se`, so for a proportion error in percentage points, pass `se_prop(p, n, pctp = TRUE)`.

Value

The `+/-` margin of error, on the same scale as `se`.

See Also

`se_mean()`, `se_prop()`, `diagnose()`.

Other diagnostics: `diagnose()`, `precision_summary()`, `rse()`, `rse_rating()`, `se_mean()`, `se_prop()`

Examples

```
margin_of_error(se_prop(0.33, 1184, pctp = TRUE))   # in percentage points

margin_of_error(0.02, z = 2)
```

mark_value *Mark a single value on a plot*

Description

Adds a reference line (and optional label) at a single value – the "you are here" marker used on the NPS and quality gauges.

Usage

```
mark_value(
  plot,
  value,
  axis = c("x", "y"),
  colour = "black",
  linewidth = 1,
  label = NULL,
  ...
)
```

Arguments

plot	A ggplot object.
value	Numeric position of the marker.
axis	Axis the value is on: "x" (default, draws a vertical line) or "y" (horizontal line).
colour, linewidth	Line appearance.
label	Optional text label drawn at the marker.
...	Passed to ggplot2::annotate() for the label; overrides the default justification, which keeps the label inside the panel next to the marker line.

Value

The plot with the marker added.

See Also

Other decisions: annotate_bands(), bands_rating_3(), rescale_bands()

Examples

```
# a target line across the percentage axis of a bar chart
calc_percentage(podracing_survey, demo_gender) %>%
  plot_bars() %>%
  mark_value(20, axis = "y", label = "Target 20%")
```

na_blank *Convert blank and boilerplate non-answers to* NA

Description

Survey exports are littered with empty strings and standard "non-answer" options such as "Prefer not to answer". This helper turns those into real NA values so they drop out of counts and summaries, replacing the repetitive `filter(x != "", x != "Prefer not to answer")` idiom.

Usage

```
na_blank(x, also = ezrsurvey_default("na_answers"), trim = TRUE)
```

Arguments

x	A character (or factor) vector.
also	Additional exact strings to treat as missing. Defaults to the na_answers option ("Prefer not to answer"; see ezrsurvey_options()). Set to character(0) to only blank out "".
trim	Whether to trim surrounding whitespace before comparing. Defaults to TRUE.

Details

The input is coerced to character (so factors are handled), optionally whitespace-trimmed, and any value equal to "" or to one of also becomes NA. The default also reads the na_answers option, so you can change what counts as a non-answer package-wide with ezrsurvey_options(na_answers = ...). Most of the calc_* helpers call na_blank() internally when na_rm = TRUE, so you usually get this clean-up for free; call it directly when you want to blank a column in a pipeline.

Value

A character vector the same length as x, with blanks and non-answers replaced by NA.

See Also

ensure_numeric(), ezrsurvey_options().

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), nps_group(), recode_age(), recode_generation(), recode_likert(), recode_region(), split_multi()

Examples

```
na_blank(c("Yes", "", "Prefer not to answer", "No"))

# add your own non-answers
na_blank(c("a", "n/a", "N/A"), also = c("n/a", "N/A"))
```

`nice_max` *Round an axis maximum up to a "nice" value*

Description

Picks the smallest multiple of `unit` that is greater than or equal to `max(x)`. This reproduces the dynamic y-axis trick used throughout the original survey reports, where bar charts are given a tidy ceiling in units of 25 (percentages) or 5 (NPS counts) so that data labels never collide with the top of the panel.

Usage

```
nice_max(x, unit = 25, pad = 0)
```

Arguments

`x`	A numeric vector. `NA` values are ignored.
`unit`	Size of the rounding step (e.g. `25` for percentages, `5` for smaller counts). Must be a positive number.
`pad`	Extra headroom added to the result, in the same units as `x`. Useful to leave space for annotations above the tallest bar. Defaults to `0`.

Details

The ceiling is `(floor(max(x) / unit) + 1) * unit + pad`: the *next* multiple of `unit` strictly above `max(x)`, always leaving headroom even when the max already sits exactly on a multiple (e.g. `nice_max(75)` is `100`, not `75`). Use `pad` for additional headroom on top of that, e.g. for annotations above the tallest bar. An all-`NA` logical column (a common shape for an empty survey field) is tolerated and returns `NA`. This is the engine behind `scale_y_pct()` and the percentage plot wrappers, which is why their bars never quite touch the top of the panel.

Value

A single numeric value: the padded, rounded-up ceiling. Returns `NA` if `x` is empty or all `NA`.

See Also

`scale_y_pct()`, `label_pct()`.

Other scales: `label_pct()`, `scale_y_pct()`

Examples

```
nice_max(c(12, 63, 40))

nice_max(80, unit = 25)        # already need >75, so jumps to 100

nice_max(75, unit = 25)        # exact multiple still advances, for headroom
```

```
nice_max(c(8, 17), unit = 5, pad = 10)   # next multiple of 5 (20) + 10
```

---

nps_group *Classify Net Promoter Score answers into groups*

Description

Collapses an 0-10 "how likely to recommend" question into the three standard NPS groups: detractors (0-6), passives (7-8) and promoters (9-10).

Usage

```
nps_group(x, labels = FALSE)
```

Arguments

x	A numeric vector of 0-10 ratings (character input is coerced).
labels	If FALSE (default) returns the signed integer coding -1 / 0 / 1 used by calc_nps(). If TRUE returns the labels "Detractor" / "Passive" / "Promoter".

Details

The signed -1/0/1 coding is deliberate: the mean of it, times 100, is exactly the Net Promoter Score (% promoters minus % detractors), which is how calc_nps() computes the headline number. Inputs are run through ensure_numeric() first, so worded answers like "9 - very likely" are classified correctly. Values outside 0–10 (and NA) return NA.

Value

Either an integer vector (-1/0/1) or a character vector, the same length as x. Out-of-range or missing values become NA.

See Also

calc_nps(), plot_nps().

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), na_blank(), recode_age(), recode_generation(), recode_likert(), recode_region(), split_multi()

Examples

```
nps_group(c(0, 6, 7, 8, 9, 10))

nps_group(c(3, 8, 10), labels = TRUE)
```

order_for *Find the order that applies to a variable*

Description

Looks up the registered order for a column: an exact `vars` match wins, otherwise the first matching `prefixes` entry. Used internally by `calc_percentage()` to apply orders automatically; exposed so you can check what would be applied.

Usage

```
order_for(var)
```

Arguments

`var`	A column name.

Value

The matching level vector, or `NULL` if none is registered.

See Also

`register_order()`, `apply_order()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
register_order("edu", c("low", "mid", "high"), prefixes = "edu_")
order_for("edu_level")
remove_order("edu")
```

pal_brand *Brand colour palette*

Description

Returns the organisation's accent colours set by `use_brand()` (or the `brand_colors` option), ready for manual scales or direct use. Without a brand, it falls back to pal_sequential_blue so it always returns something usable.

Usage

```
pal_brand(n = NULL)
```

Arguments

n	Number of colours wanted. NULL (default) returns the full accent vector as-is; when n exceeds the available colours the palette is interpolated with grDevices::colorRampPalette().

Details

Only the *categorical* brand accents live here. The semantic palettes (pal_rating, pal_nps) keep their red-amber-green vocabulary regardless of brand – recolouring "bad to good" with corporate accents would destroy the meaning the colours carry.

Value

A character vector of hex colours.

See Also

use_brand(), scale_fill_brand().

Other brand: brand_info(), clear_brand(), scale_fill_brand(), use_brand()

Examples

```
pal_brand()      # sequential blue until a brand is set
pal_brand(2)
```

---

pal_rating *ezrsurvey colour palettes*

Description

A small set of named, semantic palettes for survey reporting. Each is a plain character vector of hex colours (some named) that you can use directly or via the scale_*_rating() / scale_fill_nps() helpers.

Usage

```
pal_rating

pal_nps

pal_sequential_blue

pal_neutral
```

Format

Each palette is a character vector of hex colour strings:

`pal_rating` Five-point red -> amber -> green scale for 1-5 ratings, named `"1"`..`"5"`.

`pal_nps` Three NPS-group colours, named `"-1"`, `"0"`, `"1"` (detractor / passive / promoter).

`pal_sequential_blue` Four-step sequential blue, light to dark, for ordered brand questions.

`pal_neutral` A single neutral grey used for unbranded single-series bars.

Details

These are the semantic colour vocabularies behind the plot helpers: red -> amber -> green for ratings (worse to better), a matching three-colour NPS scale, a sequential blue for ordered brand questions, and a neutral grey for plain single-series bars. Use them directly, or via the `scale_fill_rating()` / `scale_fill_nps()` ggplot2 scales.

`pal_rating` follows the same thresholds as `bands_rating_3()`: ratings of 1 and 2 are BAD (red), 3 is OK (amber), 4 and 5 are GOOD (green). Keeping the two in step matters – a point coloured amber sitting inside a red band is the kind of contradiction that quietly misleads a reader.

Value

Each palette is a character vector of hex colour strings, named by the value it colours where that matters (`pal_rating` by rating `"1"` to `"5"`, `pal_nps` by NPS group `"-1"`, `"0"`, `"1"`), ready to pass as `values` to a ggplot2 manual scale.

See Also

Other themes: `scale_fill_nps()`, `scale_fill_rating()`, `theme_ezrsurvey()`, `theme_transparent()`

Examples

```
pal_rating

scales::show_col(pal_rating)
```

---

parse_filename *Split a filename column into metadata columns*

Description

Survey exports often encode metadata in the filename (e.g. `"podracing_wave1_NA_2026.csv"`). This splits a filename column on a separator into named metadata columns, dropping the file extension first. Replaces the brittle `str_split(file, "_")[[1]][n]` indexing this saves you.

Usage

```
parse_filename(
  data = NULL,
  col = "file",
  into,
  sep = "_",
  drop_ext = TRUE,
  remove = FALSE
)
```

Arguments

data	A data frame.
col	Name of the column holding the filename (string or unquoted). Defaults to "file".
into	Character vector of new column names, one per field you expect after splitting.
sep	Separator to split on (a fixed string, not a regex). Defaults to "_".
drop_ext	If TRUE (default), strip a trailing file extension before splitting.
remove	If TRUE, drop the original filename column. Defaults to FALSE so you keep the source reference.

Details

Pairs naturally with read_folder(), whose file column carries the source filename: split it once and you have event, wave, locale, year, etc. as real columns to group by. The file extension is stripped first (drop_ext), the split is on a fixed string (not a regex), and rows with fewer fields than into are padded with NA so a stray filename never derails the parse.

Value

The data frame with the new metadata columns added.

See Also

read_folder().

Other import: read_folder(), select_prefix(), select_suffix()

Examples

```
df <- tibble::tibble(file = c("podracing_wave1_NA_2026.csv"))
parse_filename(df, into = c("survey", "wave", "locale", "year"))
```

plot_bars *Bar chart of a percentage table (auto-laid-out)*

Description

Plots the output of `calc_percentage()` (or `calc_percentage_multi()`) as a labelled bar chart, with a tidy auto-scaled axis (`nice_max()`) and the ezrsurvey theme. By default the **layout is chosen for you**: vertical columns for a few short labels, horizontal bars for many or long ones, with long labels wrapped, the text size stepped down as bars multiply, and the bars ordered so the longest sits at the top (bars) or on the left (cols).

Usage

```
plot_bars(
  data,
  label = NULL,
  value = pct,
  orientation = c("auto", "cols", "bars"),
  sort = c("auto", "none", "asc", "desc"),
  wrap = NULL,
  flip = NULL,
  avg_line = FALSE,
  axis_labels = FALSE,
  unit = ezrsurvey_default("pct_axis_unit"),
  max = ezrsurvey_default("pct_axis_max"),
  fill = NULL,
  title = NULL,
  label_size = NULL
)
```

Arguments

`data`	A data frame with a category column and a value column.
`label`	Category column (unquoted). If `NULL` (default), the first non-`n`, non-value column is used.
`value`	Value column (unquoted). Defaults to `pct`.
`orientation`	`"auto"` (default), `"cols"` (vertical) or `"bars"` (horizontal). `"auto"` picks horizontal bars when there are more than `bar_cols_max_items` items or a label longer than `bar_cols_max_label` characters (see `ezrsurvey_options()`).
`sort`	`"auto"` (default), `"none"`, `"asc"` or `"desc"`. `"auto"` orders bars by value so the longest is on top (bars) or on the left (cols), but leaves an intentional ordinal scale (an ordered factor, e.g. from a registered order) untouched.
`wrap`	Label wrap width in characters (`stringr::str_wrap()`). `NULL` (default) uses `bar_wrap_cols` / `bar_wrap_bars` for the chosen orientation.
`flip`	Deprecated back-compat shortcut: `TRUE` forces `orientation = "bars"`, `FALSE` forces `"cols"`. `NULL` (default) defers to `orientation`.

avg_line	If TRUE, add a reference line at the mean value. Defaults to FALSE.
axis_labels	If TRUE, show the percentage axis; if FALSE (default) hide it (the bars carry their own data labels).
unit	Axis rounding step passed to nice_max(). Defaults to the pct_axis_unit option (25; see ezrsurvey_options()).
max	Optional fixed y-axis maximum (e.g. 100). Defaults to the pct_axis_max option (NULL = dynamic).
fill	Bar fill colour. NULL (default) uses the brand primary colour when a brand is set (see use_brand()), else pal_neutral.
title	Optional plot title.
label_size	Data-label text size. NULL (default) steps down from bar_label_size as the number of bars grows (floored at bar_size_min).

Details

Expects a summary table (from calc_percentage() and friends), not raw survey rows. The label column is auto-detected as the first non-n, non-value column, so a piped percentage table just works. The layout decisions all read from the bar_* options (ezrsurvey_options()): orientation = "auto" switches to horizontal bars once there are many or long labels, labels are wrapped to bar_wrap_cols / bar_wrap_bars, the data-label size steps down with the bar count, and sort = "auto" puts the longest bar at the top (bars) or on the left (cols) – unless the label column is an ordered factor (a registered or explicit order), which is preserved. The axis ceiling is chosen by nice_max() / the pct_axis_* options so data labels never collide with the panel top, and the % axis is hidden unless axis_labels = TRUE. Add decision guidance with annotate_bands().

Bars are drawn to a constant thickness regardless of how many there are (bar_width / bar_ref_items, see ezrsurvey_options()), so a three-answer chart and a ten-answer chart look like they belong in the same deck instead of the first one's bars turning into slabs.

Value

A ggplot object.

See Also

calc_percentage(), scale_y_pct(), annotate_bands().

Other plots: plot_gauges(), plot_ipm(), plot_nps(), plot_nps_gauge(), plot_quotes_tree(), plot_rating_grid(), plot_stacked_rating()

Examples

```
# auto layout: few short labels -> vertical columns, largest on the left
calc_percentage(podracing_survey, demo_gender) %>% plot_bars()

# many long labels -> horizontal bars, wrapped, largest on top
calc_percentage(podracing_survey, fav_driver) %>% plot_bars()

# an ordinal scale keeps its order; force horizontal with orientation
```

```
calc_percentage(podracing_survey, demo_edu) %>%
  plot_bars(orientation = "bars")
```

`plot_diff` *Diverging bar chart of differences*

Description

Plots a column of differences (e.g. from `compare_values()`) as a sorted, diverging horizontal bar chart: positives in green, negatives in amber, signed value labels, and a zero reference line. This is the standard "what improved, what slipped" change chart.

Usage

```
plot_diff(
  data,
  label = feature,
  difference = difference,
  limits = NULL,
  digits = 2,
  pos_colour = "#A7C23D",
  neg_colour = "#FFCB3E",
  title = NULL
)
```

Arguments

`data`	A data frame with a label column and a difference column.
`label`	Category/label column (unquoted). Default `feature`.
`difference`	Difference column (unquoted). Default `difference`.
`limits`	Numeric length-2 y-axis limits. If `NULL` (default), a symmetric range around zero is chosen from the data.
`digits`	Decimal places for the value labels. Default `2`.
`pos_colour, neg_colour`	Bar colours for positive / negative differences.
`title`	Optional plot title.

Details

Bars are sorted by the difference and drawn horizontally (largest gain at the top), coloured green for positive and amber for negative changes, with signed value labels (`+0.20` / `-0.30`) nudged to sit just outside each bar and a zero reference line. When `limits` is `NULL` the y-axis is made symmetric around zero from the data so gains and losses are visually comparable; pass `limits = c(-1, 1)` (say) to fix it. Feed it the output of `compare_values()`, or any data frame with a label column and a numeric difference column.

Value

A ggplot object.

See Also

compare_values().

Other compare: compare_values()

Examples

```
a <- data.frame(feature = c("price", "quality", "service"),
                performance = c(3.1, 4.2, 3.5))
b <- data.frame(feature = c("price", "quality", "service"),
                performance = c(2.8, 4.4, 3.9))
compare_values(b, a) %>% plot_diff()
```

---

plot_gauges *Stacked score gauges (NPS over average quality, etc.)*

Description

Draws two or more scores as thin banded gauge bars stacked one above another, each on its own scale with a "you are here" marker. The headline summary slide: the Net Promoter Score on top and the average feature quality rating below it, so both key numbers sit together instead of one fat bar filling the slide.

Usage

```
plot_gauges(
  scores,
  scales = NULL,
  title = NULL,
  height = 0.5,
  label_size = NULL
)
```

Arguments

scores	A **named** numeric vector; each name labels a gauge and each value is placed on its scale. The first is drawn at the top.
scales	Which band scale each gauge uses: "nps" (a -100..100 Net Promoter scale) or "rating" (a 1..5 quality scale). Length 1 (recycled) or one per score. NULL (default) infers "rating" for a value in 1..5 and "nps" otherwise.
title	Optional title. NULL (default) draws none, so a slide title can carry the wording instead.
height	Bar thickness in plot units (row spacing is 1). Defaults to 0.5, keeping each bar deliberately thin so several share the panel.
label_size	Band-label text size. NULL (default) is a compact 10 / .pt.

Details

Each gauge is normalised to its own scale, so an NPS of `+23` and a quality rating of `3.4` line up on a shared 0..1 panel with the band boundaries drawn where each scale puts them (the NPS gauge reuses the `plot_nps_gauge()` bands; the rating gauge uses `bands_rating_3()`). The score for each gauge is printed beside its label, and a black marker shows where it lands. Because the bars are thin and stacked, this is the summary-slide companion to the detailed `plot_nps()` distribution and the `plot_ipm()` driver matrix.

Since the bars share one panel but not one scale, no axis can serve them both. Each bar therefore carries its own band boundaries as small numbers underneath, so an NPS bar reads `-100 0 30 70 100` and a rating bar reads `1 3 4 5`; without them a marker two thirds along says nothing about the score it marks. A band too narrow to hold its name is left unlabelled rather than having the text run over its neighbours, and those numbers are what still place it.

Value

A ggplot object.

See Also

`plot_nps_gauge()`, `calc_nps()`, `ipm_model()`.

Other plots: `plot_bars()`, `plot_ipm()`, `plot_nps()`, `plot_nps_gauge()`, `plot_quotes_tree()`, `plot_rating_grid()`, `plot_stacked_rating()`

Examples

```
nps <- calc_nps(podracing_survey, nps_value)$nps
quality <- 3.4
plot_gauges(c("Net Promoter Score" = nps,
              "Average quality rating" = quality))
```

`plot_ipm` *Importance / performance matrix*

Description

Plots an `ipm_model()` table as a scatter of feature performance (x) vs. importance (y), coloured by performance band, with decision bands across the top: the importance / performance slide of a survey deck, in one call.

Usage

```
plot_ipm(model, title = NULL, repel = TRUE, bands = bands_rating_3())
```

Arguments

model	An ipm_model() output (feature, importance, performance, perf_class).
title	Optional title; defaults to the average performance.
repel	Use ggrepel for non-overlapping labels when available. Defaults to TRUE.
bands	Decision-band spec drawn across the top and used to colour the points. Defaults to bands_rating_3() (BAD 1-3, OK 3-4, GOOD 4-5).

Details

Each feature is a point: performance on the x-axis (mean 1–5 rating, with a reference line at the average) and importance on the y-axis (relative weight, as a percentage). Each point takes the colour of the decision band it falls in, so a feature averaging 2.4 is red because it sits in the BAD band – the point and the band behind it can never disagree. Read it by quadrant – high-importance, low-performance features (upper left) are the priorities to fix, while high-importance, high-performance features (upper right) are strengths to protect. Feed it an ipm_model() table; uses ggrepel for non-overlapping labels when available.

Value

A ggplot object.

See Also

ipm_model(), compare_values().

Other plots: plot_bars(), plot_gauges(), plot_nps(), plot_nps_gauge(), plot_quotes_tree(), plot_rating_grid(), plot_stacked_rating()

Examples

```
ipm_model(podracing_survey, nps_value, "ratings_") %>% plot_ipm()
```

---

plot_nps *NPS distribution slide (0-10 scale with group labels)*

Description

The canonical Net Promoter Score chart: the full 0-10 recommendation distribution as labelled bars coloured by NPS group, with the detractor / passive / promoter shares called out in coloured boxes across the top and the overall NPS in the title: the NPS slide of a survey deck, in one call.

Usage

```
plot_nps(data = NULL, value, title = NULL)
```

Arguments

data	A data frame. If omitted, the session default (use_dataset()) is used.
value	The 0-10 recommendation column (unquoted). Text such as "8 - likely" is salvaged with ensure_numeric().
title	Optional title; defaults to "Net Promoter Score of: <score>".

Details

Unlike plot_nps_gauge() (which takes a single score), this reads the raw 0–10 column and draws the full distribution: one labelled bar per score, coloured red/amber/green by NPS group, with the detractor/passive/promoter shares called out across the top and the overall score in the title. It is the detailed companion slide to the gauge.

Value

A ggplot object.

See Also

calc_nps(), plot_nps_gauge().

Other plots: plot_bars(), plot_gauges(), plot_ipm(), plot_nps_gauge(), plot_quotes_tree(), plot_rating_grid(), plot_stacked_rating()

Examples

```
plot_nps(podracing_survey, nps_value)
```

---

plot_nps_gauge *Score gauge with decision bands and a value marker*

Description

A horizontal gauge that places a single score (NPS or mean rating) onto a banded scale, with a "you are here" marker: the NPS and quality gauges a summary slide carries.

Usage

```
plot_nps_gauge(
  score,
  scale = c("nps", "rating"),
  title = NULL,
  height = 0.5,
  label_size = NULL
)
```

Arguments

score	The score to mark (NPS on -100..100, or a mean rating on 1..5).
scale	"nps" (default) or "rating", selecting the band layout and limits.
title	Optional title; a sensible default is generated from score.
height	Bar thickness in plot units. Defaults to 0.5.
label_size	Band-label text size. NULL (default) matches the theme's base font size (11 pt).

Details

Takes a single number (not a dataset) and places it on a coloured band scale with a black marker, for the headline "where do we stand" slide. "nps" uses a -100..100 scale banded from needs-work to excellent; "rating" uses a 1..5 scale with BAD/OK/GOOD bands. Pair it with calc_nps() for the score.

Value

A ggplot object.

See Also

calc_nps(), plot_nps().

Other plots: plot_bars(), plot_gauges(), plot_ipm(), plot_nps(), plot_quotes_tree(), plot_rating_grid(), plot_stacked_rating()

Examples

```
nps <- calc_nps(podracing_survey, nps_value)$nps
plot_nps_gauge(nps)

plot_nps_gauge(3.8, scale = "rating")
```

---

plot_quotes_tree *Treemap of selected quotes*

Description

A treemap where each tile is a comment sized by length – the quote slide of a survey deck. Requires the suggested treemapify package.

Usage

```
plot_quotes_tree(data, label = comment, area = length, colour = "black")
```

Arguments

data	A data frame of quotes, e.g. from sample_comments().
label	Text column (unquoted). Defaults to comment.
area	Tile-size column (unquoted). Defaults to length.
colour	Tile border colour. Defaults to "black".

Details

Lays selected verbatims out as a treemap, each tile sized by comment length, so a slide can show real customer voice at a glance. Pair it with sample_comments() or sample_comments_diverse(), which produce the comment/length columns it expects. Requires the suggested treemapify package.

Value

A ggplot object.

See Also

sample_comments(), sample_comments_diverse().

Other plots: plot_bars(), plot_gauges(), plot_ipm(), plot_nps(), plot_nps_gauge(), plot_rating_grid(), plot_stacked_rating()

Examples

```
sample_comments(podracing_survey, nps_com, show_com, n = 5) %>%
  plot_quotes_tree()
```

---

plot_rating_grid *A stacked rating chart for a whole block of questions*

Description

The one-line form of plot_stacked_rating() for the common case: a block of columns sharing a prefix, each asking the same rating question about a different feature or brand. Tabulates the block, tidies the question names, numbers the answers by their position on the scale and stacks the result.

Usage

```
plot_rating_grid(data = NULL, prefix, levels = NULL, digits = 2, ...)
```

Arguments

data	A data frame. If omitted, the session default (use_dataset()).
prefix	The block's shared column prefix, e.g. "ratings_". It is stripped from the feature labels.
levels	The scale's answer wordings, worst first. NULL (default) looks the block up in the order registry (register_order()), falling back to the recode_likert() default five-point scale.
digits	Decimal places kept in the underlying percentages. Default 2.
...	Passed to plot_stacked_rating() (palette, label_min, show_average).

Details

Answers are numbered by their position in levels, which is what orders the features by their weighted mean and colours the segments worst-to-best. A wording that is not on the scale cannot be numbered, so it is reported rather than being silently dropped into an unlabelled segment: check it against the raw export, since exports often differ from a questionnaire by a typo. Registering the block once with register_order("likeability", levels, prefixes = "partner_likeability_") removes the levels argument from every later call.

Value

A ggplot object.

See Also

plot_stacked_rating(), calc_percentage_batch(), register_order().

Other plots: plot_bars(), plot_gauges(), plot_ipm(), plot_nps(), plot_nps_gauge(), plot_quotes_tree(), plot_stacked_rating()

Examples

```
plot_rating_grid(podracing_survey, "ratings_")

plot_rating_grid(podracing_survey, "partner_likeability_",
                 levels = c("Very unlikeable", "Unlikeable",
                            "Likeable", "Very likeable"))
```

---

plot_stacked_rating *Stacked rating bars with weighted-average ordering*

Description

Draws a 100%-stacked bar per feature across an ordinal rating scale, labels each segment, and orders features by their weighted mean rating – the likeability, purchase and feature-rating charts a survey deck is built from.

Usage

```
plot_stacked_rating(
  data,
  feature,
  level,
  value = pct,
  palette = NULL,
  label_min = 1,
  show_average = TRUE
)
```

Arguments

data	Long data with one row per feature x rating level.
feature	Feature column (unquoted).
level	Ordered rating-level column (unquoted), e.g. a factor like "1 - Very bad" .. "5 - Very good". The leading digit is used as the numeric weight.
value	Percentage column (unquoted). Defaults to pct.
palette	Fill colours, named by the level labels. Defaults to NULL, which derives a red -> amber -> green rating palette from the level labels themselves (mapping each by its leading digit), so a very-bad..very-good scale is coloured worse-to-better automatically. Pass a named vector to override.
label_min	Hide segment labels below this percentage. Defaults to 1.
show_average	Append the weighted mean to each feature label. Defaults to TRUE.

Details

Each feature becomes a 100%-stacked bar across the rating scale, and features are ordered by their weighted-mean rating (the leading digit of each level is the weight), so the best-rated feature sits at the top. The weighted mean is appended to each feature label when show_average = TRUE. Feed it a long table with one row per feature x rating level; the partner_* likeability and purchase questions in the source reports are the canonical use.

Value

A ggplot object.

See Also

calc_percentage(), scale_fill_rating().

Other plots: plot_bars(), plot_gauges(), plot_ipm(), plot_nps(), plot_nps_gauge(), plot_quotes_tree(), plot_rating_grid()

Examples

```
rating_long <- podracing_survey %>%
  select(starts_with("ratings_")) %>%
  tidyr::pivot_longer(everything(),
                      names_to = "feature", values_to = "level") %>%
  filter(level != "") %>%
  mutate(level = paste0(recode_likert(level), " - ", level)) %>%
  count(feature, level) %>%
  group_by(feature) %>%
  mutate(pct = n / sum(n) * 100) %>%
  ungroup()
plot_stacked_rating(rating_long, feature, level)
```

---

podracing_survey *Simulated pod-racing fan survey*

Description

A fictional consumer-feedback survey from a Boonta Eve-style pod-racing meeting – an affectionate Star Wars parody used throughout the ezrsurvey examples and tests (the package's literal answer to `starwars`, but survey-shaped). A latent enjoyment drives correlated attribute ratings and the recommend score, the demographics use international-standard categories (ISO/IEC 5218 sex, ISCED education, ILO labour-force status, ISIC sectors, ISO 3166 countries), and the multi-select, sponsor and open-text columns mirror the shapes the helpers consume. The open-text comments deliberately name many drivers, places and races, so they exercise the text / NER helpers. Blanks and "Prefer not to answer" responses are sprinkled in so cleaning helpers have something to do.

Usage

```
podracing_survey
```

Format

A tibble with 1,000 rows and 32 columns:

respondent_id Unique respondent id (character).

collector Survey source: email, panel, socials or in_app.

demo_age Age in years (integer, 16-70).

demo_gender Sex/gender: Male, Female, Non-binary (ISO/IEC 5218, extended), some blank / "Prefer not to answer".

demo_edu Highest education level (ISCED 2011 broad categories).

demo_country Country of residence (ISO 3166 names).

region World region derived from `demo_country`.

demo_job Labour-force status (ILO categories).

demo_sector Industry sector (ISIC Rev.4 sections).

race_attended Which meeting they attended (e.g. "Boonta Eve Classic"); long labels that exercise the auto bar layout.

fav_driver Favourite pod-racer (e.g. "Anakin Skywalker", "Sebulba").

nps_value 0-10 "how likely to recommend attending" rating (integer).

satis_return Likelihood of returning next season (Very unlikely .. Very likely).

ratings_atmosphere, ratings_commentary, ratings_safety, ratings_speed, ratings_value, ratings_venue Worded 1-5 attribute ratings (Very bad .. Very good).

motivations_speed, motivations_drivers, motivations_betting, motivations_social, motivations_tradition Multi-select reasons for attending; each holds its option text when chosen, otherwise *""*.

partner_recall_PodTech, partner_recall_BanthaBrew, partner_recall_JawaJuice Sponsor recall per brand (Sponsor / Not a sponsor / Don't know this brand).

partner_likeability_PodTech, partner_likeability_BanthaBrew, partner_likeability_JawaJuice Sponsor likeability (Very likeable .. Very unlikeable).

nps_com, show_com Open-text comments (mostly blank).

Source

Simulated. See `data-raw/make_podracing_survey.R` for the generator.

See Also

Other data: `country_region`, `currency_rates`, `shopping_survey`

Examples

```
calc_percentage(podracing_survey, demo_gender, sort = "desc")
calc_nps(podracing_survey, nps_value)
```

---

precision_summary *Plain-language survey precision summary*

Description

Distils a whole survey's sampling precision into a few bullet points – the narrative "Data info" paragraph from the report appendix, generated automatically. It scans the dataset, assesses the numeric (point-scale) and categorical (proportion) variables, and reports the typical sampling margins and an overall reliability verdict.

Usage

```
precision_summary(data = NULL, ..., z = 1.96)
```

Arguments

data	A data frame. If omitted, the session default (use_dataset()) is used.
...	Optional columns to restrict the assessment to (tidyselect). If omitted, all suitable columns are used: every numeric column, and every categorical column that is not free text or an identifier.
z	Confidence multiplier for the margins of error. Default 1.96 (95%).

Details

Rather than make you choose which variables to quote precision for, this picks them: numeric columns become point estimates (margins in rating points) and categorical columns become proportions (margins in percentage points), with free-text and id-like columns skipped. It then reports the **typical** and **worst** margins across each group, plus an overall verdict from the median relative standard error via rse_rating(). For proportions it also gives the worst-case +/- percentage-point margin at a 50/50 split, which is the widest any percentage in the study can be. See vignette("ezrsurvey") for the rationale behind the relative-standard-error reliability bands.

Value

An object of class ezrsurvey_precision (a list with n, the per-variable table, overall_rse, rating and the bullets). Printing it shows the bullet points.

See Also

diagnose(), rse_rating().

Other diagnostics: diagnose(), margin_of_error(), rse(), rse_rating(), se_mean(), se_prop()

Examples

```
precision_summary(podracing_survey)
```

---

read_folder *Read and stack every CSV in a folder*

Description

Loads all CSV files matching pattern in path, reading each as character (so survey codes never get silently coerced), tags every row with its source filename, and row-binds the lot into one tibble. Generalises the list.files() %>% map(read.csv) %>% bind_rows() opening of a survey script.

Usage

```
read_folder(
  path,
  pattern = "\\.csv$",
  id = "file",
  all_character = TRUE,
  question_row = NULL,
  locale = readr::locale(encoding = "UTF-8"),
  ...
)
```

Arguments

path	Directory to read from.
pattern	Regular expression matching the files to load. Defaults to "\\.csv$" (all CSVs).
id	Name of the column in which to record each row's source filename. Defaults to "file". Set to NULL to omit it.
all_character	If TRUE (default), every column is read as character. This keeps ragged survey exports stackable; recode types afterwards with the recode_* helpers. If FALSE, types are guessed per file.
question_row	What to do with a second header row of question wording. NULL (default) keeps it and warns when a file looks like it has one; TRUE drops the first row of every file; FALSE keeps it silently.
locale	A readr::locale() controlling encoding etc. Defaults to a UTF-8 locale; pass readr::locale(encoding = "windows-1252") for legacy exports.
...	Further arguments passed to readr::read_csv().

Details

Every matching file is read and the results are row-bound, so a folder of monthly or per-event exports becomes one dataset. By default every column is read as text (all_character = TRUE); this is deliberate, because survey exports are ragged (a column that is numeric in one file may carry "Prefer not to answer" in another), and stacking text columns never fails. Recode the types you need afterwards with recode_*() / ensure_numeric(). The id column records which file each row came from – split it into metadata with parse_filename(). For legacy encodings pass a locale, e.g. readr::locale(encoding = "windows-1252").

Survey platforms commonly write the question wording as a second header row, which reads back as respondent number one and quietly skews every count by one. A file whose first row is long, spaced prose in most columns at once and numeric in none is reported as such; question_row = TRUE drops that row from every file, and FALSE keeps it without comment.

Value

A single tibble of all files stacked together. If columns differ across files, missing columns are filled with NA.

See Also

select_prefix(), parse_filename().

Other import: parse_filename(), select_prefix(), select_suffix()

Examples

```
# build a tiny folder of CSVs, then read them back
dir <- tempfile(); dir.create(dir)
readr::write_csv(head(podracing_survey, 3), file.path(dir, "a.csv"))
readr::write_csv(head(podracing_survey, 2), file.path(dir, "b.csv"))
read_folder(dir)[, c("file", "respondent_id")]
```

---

recode_age *Recode age into standard survey bands*

Description

Convenience wrapper over bin_numeric() using the standard survey age bands. Also tolerates messy inputs such as "25 years" by extracting the first run of digits.

Usage

```
recode_age(
  x,
  breaks = ezrsurvey_default("age_breaks"),
  labels = ezrsurvey_default("age_labels"),
  quiet = FALSE
)
```

Arguments

x	A numeric or character vector of ages.
breaks, labels	Override the default bands if needed; the defaults come from the age_breaks / age_labels options (see ezrsurvey_options()).
quiet	If FALSE (default), report answers that held no number and answers that fell outside the bands. Set TRUE to silence both.

Details

Ages are first passed through ensure_numeric(), so messy entries like "22 years" or "age: 31" are handled, then binned with bin_numeric() using left-closed bands. An answer with no number in it at all ("young", "prefer not to say") becomes NA and is counted in a message, so a column that was never numeric does not pass for a column of missing ages. The default bands come from the age_breaks / age_labels options, so you can set your study's standard cohorts once with ezrsurvey_options(age_breaks = ..., age_labels = ...) (or a profile) instead of passing them on every call. For *generational* cohorts (Gen Z, Millennial, ...) use recode_generation() instead.

Value

A factor of age-band labels, levels in band order.

See Also

bin_numeric(), recode_generation(), ezrsurvey_options().

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), na_blank(), nps_group(), recode_generation(), recode_likert(), recode_region(), split_multi()

Examples

```
recode_age(c("17", "22 years", "31", "47"))
```

---

recode_generation *Recode age or birth year into a generational cohort*

Description

Maps either an **age** (converted to a birth year using a reference year) or a **birth year** directly onto a generational cohort such as Millennial or Gen Z. Text input like `"34 years"` is salvaged with ensure_numeric().

Usage

```
recode_generation(x, input = c("age", "year"), year = NULL, scheme = NULL)
```

Arguments

x	A numeric (or coercible) vector of ages or birth years.
input	What x represents: `"age"` (default) or `"year"` (birth year).
year	Reference year for the age-to-birth-year conversion. If `NULL` (default), uses the `current_year` option (see ezrsurvey_options()) or the system year.
scheme	Either a scheme name passed to generation_scheme() (default from the `generation_scheme` option) or your own data frame with `from` and `label` columns.

Details

When `input = "age"`, the birth year is `year - age`, where `year` defaults to the `current_year` option or the system year – so set `ezrsurvey_options(current_year = 2026)` to keep results stable across runs. When `input = "year"`, `x` is treated as the birth year directly. The birth year is then bucketed into the scheme's cohorts (see generation_scheme()); birth years before the earliest cohort return `NA`. Use this for generational reporting; for simple age bands use recode_age().

Value

A factor of cohort labels, levels from the oldest cohort to the youngest; values outside the scheme's range become NA.

See Also

generation_scheme(), recode_age(), ezrsurvey_options().

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), na_blank(), nps_group(), recode_age(), recode_likert(), recode_region(), split_multi()

Examples

```
recode_generation(c(1990, 2001, 1968), input = "year")

# from age, with an explicit reference year
recode_generation(c(36, 25), input = "age", year = 2026)
```

---

recode_likert *Recode a worded rating scale to integers*

Description

Maps the worded answers of an ordinal rating question (e.g. "Very bad" ... "Very good") onto integers 1:length(levels). Matching is case-insensitive and tolerant of common synonyms supplied via synonyms, in place of a case_when(str_detect("Very bad") ...) recoding block.

Usage

```
recode_likert(
  x,
  levels = c("Very bad", "Bad", "Ok", "Good", "Very good"),
  synonyms = NULL
)
```

Arguments

x	A character (or factor) vector of worded answers.
levels	Character vector of the scale's answer wordings in ascending order. The first element maps to 1, the last to length(levels). Defaults to a 5-point bad-to-good scale.
synonyms	Optional named list mapping a canonical level (one of levels) to a character vector of alternative spellings that should map to the same integer. Matching is by case-insensitive substring.

Details

Matching happens in three passes, in order: (1) an exact, case-insensitive, trimmed match against `levels`; (2) a substring fallback, so prefixed exports like `"4 - Good"` still resolve; (3) any `synonyms` you supply. This makes the function robust to the small wording differences between survey tools (e.g. "Dissatisfied" vs "Bad"). Turning ratings into integers is the first step of `ipm_model()` and lets you average a scale with `calc_summary()`.

The substring passes try the longest wording first, so a scale whose levels nest inside one another ("Likeable" inside "Very likeable", "Good" inside "Very good") resolves to the level the answer actually names. Without that, any answer the exact pass misses, such as one carrying an emoji or a stray character, would quietly land on the shorter level.

Value

An integer vector the same length as `x`; unmatched values become `NA`.

See Also

`nps_group()`, `ipm_model()`, `calc_summary()`.

Other recode: `add_region()`, `bin_numeric()`, `clean_label()`, `drop_items()`, `ensure_numeric()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_region()`, `split_multi()`

Examples

```
recode_likert(c("Very bad", "Ok", "Good", "Very good"))

# substring fallback copes with numbered labels
recode_likert("4 - Good")

# nested wordings resolve to the level the answer names, not the shorter one
recode_likert(c("\U0001F642 Very good", "\U0001F610 Good"))

# map another tool's wording onto the same scale
recode_likert(c("Dissatisfied", "Satisfied"),
              synonyms = list(Bad = "Dissatisfied", Good = "Satisfied"),
              levels = c("Very bad", "Bad", "Ok", "Good", "Very good"))
```

---

recode_region *Look up the region or subregion for a country*

Description

Vectorised lookup from country name to world `region` or finer `subregion`, using the bundled country_region table. Matching is case-insensitive and whitespace-tolerant, and understands the spellings people type as well as ISO 3166-1 alpha-2 and alpha-3 codes, so `"USA"`, `"U.S."`, `"us"` and `"United States"` all resolve alike; blanks and non-answers (see `na_blank()`) become `NA`, and unmatched countries become `NA` with a one-line warning so spelling mismatches are easy to spot.

Usage

```
recode_region(x, which = c("region", "subregion"), quiet = FALSE)

recode_subregion(x, quiet = FALSE)
```

Arguments

x	A character vector of country names.
which	"region" (default) or "subregion".
quiet	If FALSE (default), warn about unmatched countries. Set TRUE to silence the warning.

Details

A country column that people typed themselves is rarely tidy: in one real open salary survey, nine thousand respondents wrote "United States" and eight thousand wrote "USA". So the name is folded down before it is looked up – case, stray punctuation, accents and a leading "the" are all dropped – and then tried against the bundled country_region table (182 countries), then against a table of the spellings people actually use. That table carries endonyms ("Deutschland", "Brasil"), the constituent countries of the United Kingdom, and ISO 3166-1 alpha-2 and alpha-3 codes ("US", "USA", "DE", "DEU"), so a column already coded to the standard needs no preparation at all.

A two-letter code is only read as a code when it was typed in capitals, because "NO" is Norway but "no" is an answer to a different question. Blanks and non-answers are blanked with na_blank() first, and anything still unmatched returns NA with a warning listing the offenders, so a spelling the table does not carry is easy to spot and add. Set quiet = TRUE inside pipelines where you have already checked the coverage. region is the coarse level (e.g. "Europe"); subregion is finer (e.g. "Western Europe").

Value

A character vector of regions (or subregions), NA where unmatched.

See Also

add_region(), country_region.

Other recode: add_region(), bin_numeric(), clean_label(), drop_items(), ensure_numeric(), generation_scheme(), na_blank(), nps_group(), recode_age(), recode_generation(), recode_likert(), split_multi()

Examples

```
recode_region(c("Germany", "Japan", "Brazil"))

# the spellings a free-text country question actually collects
recode_region(c("USA", "U.S.", "England", "Deutschland", "Holland"))

# a column already coded to ISO 3166-1
recode_region(c("US", "GBR", "DE", "JPN"))
```

```
recode_subregion(c("Germany", "Japan"))

# unmatched -> NA (warning suppressed here)
recode_region(c("Germany", "Atlantis"), quiet = TRUE)
```

`register_order` *Register a reusable ordinal level order*

Description

Survey scales (education, Likert, frequency, ...) have a natural order that is tedious to retype as `levels = c(...)` every time. Register an order once, link it to the variable names and/or column prefixes it applies to, and the tabulating helpers (e.g. `calc_percentage()`) will look it up and apply it automatically. Persist your orders in a profile with `save_ezrsurvey_profile()`.

Usage

```
register_order(name, levels, vars = NULL, prefixes = NULL, overwrite = TRUE)
```

Arguments

`name`	Short name for the order (e.g. `"education"`).
`levels`	Character vector of the levels in ascending order.
`vars`	Optional exact column names this order applies to (e.g. `"demo_edu"`).
`prefixes`	Optional column-name prefixes this order applies to (e.g. `"edu_"`). A variable matches if its name starts with any prefix.
`overwrite`	Overwrite an existing order of the same name. Default `TRUE`.

Details

An order links a set of `levels` to the columns it applies to, by exact name (`vars`) and/or by prefix (`prefixes`). Once registered, `calc_percentage()`, `calc_percentage_multi()` and `crosstab()` look it up automatically (via `order_for()`) whenever you don't pass an explicit `levels`/`sort`, so a question always comes out in the right order without retyping it. Register interactively for the session, or store orders in a profile with `save_ezrsurvey_profile()` so they load every session.

Value

Invisibly the order `name`.

See Also

`order_for()`, `apply_order()`, `list_orders()`, `register_order_presets()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
register_order(
  "education",
  levels = c("Primary or less", "Lower secondary", "Upper secondary",
             "Short-cycle tertiary", "Bachelor or equivalent",
             "Master or equivalent", "Doctoral or equivalent"),
  vars = "demo_edu"
)
calc_percentage(podracing_survey, demo_edu)   # rows now in education order
remove_order("education")
```

`register_order_presets` *Register a set of common ordinal scales*

Description

A convenience that registers a handful of frequently used orders, linked to sensible default variable names/prefixes, so common surveys work out of the box. Override any of them afterwards with `register_order()`.

Usage

```
register_order_presets()
```

Details

Registers: `likert_bad_good`, `likert_agree`, `frequency`, `likelihood` and `education_isced`.

Value

Invisibly the names registered.

See Also

`register_order()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
register_order_presets()
list_orders()
```

remove_order *Remove a registered order*

Description

Remove a registered order

Usage

```
remove_order(name)
```

Arguments

name	Order name.

Value

Invisibly TRUE.

See Also

register_order().

Other config: apply_order(), edit_ezrsurvey_profile(), ezrsurvey_options(), ezrsurvey_profile_paths(), get_dataset(), get_order(), list_orders(), load_ezrsurvey_profile(), order_for(), register_order(), register_order_presets(), reset_ezrsurvey_options(), save_ezrsurvey_profile(), use_dataset(), use_ezrsurvey_profile()

Examples

```
register_order("yesno", c("No", "Yes"))
remove_order("yesno")
```

report_add_plot *Add a plot to a report*

Description

Renders a ggplot to an image and places it in the current slide's content placeholder (pptx) or as a new figure (docx).

Usage

```
report_add_plot(doc, plot, width = NULL, height = NULL, dpi = 150)
```

Arguments

doc	A document from report_new().
plot	A ggplot object (any grid grob also works, e.g. an aligned plot from report_deck()'s alignment step).
width, height	Image size in inches. NULL (default) sizes the image to the slide's content placeholder (pptx) or 6 x 3.5 (docx), so charts fill whatever template the deck is built on.
dpi	Raster resolution. Defaults to 150.

Details

On slides the image is rendered at exactly the content placeholder's size, so nothing is stretched or letterboxed; when the template does not define placeholder geometry the slide size (minus margins) is used instead.

Value

The updated document.

See Also

report_add_table().

Other reporting: example_report(), list_report_templates(), report_add_slide(), report_add_table(), report_add_text(), report_deck(), report_layouts(), report_new(), report_save(), report_section(), report_slide(), report_title_slide(), scaffold_report()

Examples

```
p <- calc_percentage(podracing_survey, demo_gender) %>% plot_bars()
doc <- report_new("pptx") %>%
  report_add_slide("Gender") %>%
  report_add_plot(p)
class(doc)
```

---

report_add_slide *Add a slide (PowerPoint) or heading (Word) to a report*

Description

For a pptx document this starts a new slide and sets its title; for a docx document it adds a heading paragraph.

Usage

```
report_add_slide(
  doc,
  title = NULL,
  layout = NULL,
  master = NULL,
  heading_level = 1
)
```

Arguments

doc	A document from report_new().
title	Optional slide title / heading text.
layout, master	PowerPoint layout and master names. NULL (default) picks the template's best content layout automatically (by inspecting placeholders, so corporate templates with renamed layouts work). Name a layout from report_layouts() to override.
heading_level	Word heading level (1-3). Defaults to 1.

Details

Automatic selection scores every layout of the template: a title placeholder plus a single content placeholder is the ideal, and the layout's *placeholders* decide – not its (language-dependent) name. If the chosen layout has no title placeholder, the title is skipped with a warning rather than failing the build.

Value

The updated document.

See Also

report_new(), report_layouts().

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_table(), report_add_text(), report_deck(), report_layouts(), report_new(), report_save(), report_section(), report_slide(), report_title_slide(), scaffold_report()

Examples

```
doc <- report_new("pptx") %>% report_add_slide("Audience")
class(doc)
```

`report_add_table` *Add a table to a report*

Description

Adds a data frame as a **flextable** – in the current slide's content placeholder (pptx) or as a new table (docx).

Usage

```
report_add_table(doc, data, font_size = 12, autofit = TRUE)
```

Arguments

`doc`	A document from `report_new()`.
`data`	A data frame / tibble.
`font_size`	Cell font size in points. Defaults to `12`.
`autofit`	Word only: auto-size columns to content. Defaults to `TRUE`. On slides the table is always sized to fill the content placeholder.

Details

On a slide the table is styled (a clean banded header, centred cells, a readable font) and its columns are widened to span the content placeholder – roughly in proportion to each column's contents – so it fills the slide meaningfully instead of sitting tiny in a corner. In Word it is added as a plain auto-fitted **flextable**.

Value

The updated document.

See Also

`report_add_plot()`.

Other reporting: `example_report()`, `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_new()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
tbl <- calc_percentage(podracing_survey, demo_gender)
doc <- report_new("docx") %>% report_add_table(tbl)
class(doc)
```

report_add_text *Add a paragraph of text to a report*

Description

Add a paragraph of text to a report

Usage

```
report_add_text(doc, text, ...)
```

Arguments

doc	A document from report_new().
text	Text to add; a character vector becomes one paragraph (bullet) per element on slides.
...	Passed to officer::body_add_par() (docx only).

Value

The updated document.

See Also

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_slide(), report_add_table(), report_deck(), report_layouts(), report_new(), report_save(), report_section(), report_slide(), report_title_slide(), scaffold_report()

Examples

```
doc <- report_new("docx") %>% report_add_text("Key findings follow.")
class(doc)
```

report_deck *Build a slide deck from a list of plots and tables in one call*

Description

Turns a named list of ggplots / data frames into a titled-slide deck (or Word document) and saves it. Names become slide titles; each item lands on its own slide, sized to the template's content placeholder.

Usage

```
report_deck(
  items,
  path,
  format = c("pptx", "docx"),
  title = NULL,
  template = NULL,
  style = c("elevated", "plain")
)
```

Arguments

items	A named list; names become slide titles. Each element is either a ggplot (added as a plot) or a data frame (added as a table).
path	Output file path. A bare file name is written to the working directory, or into the folder set by the output_dir option.
format	"pptx" (default) or "docx".
title	Optional title-slide / document-title text.
template	Optional reference document; NULL (default) uses the brand template registered by use_brand(), see report_new().
style	Built-in template to fall back on: "elevated" (default, the styled deck) or "plain" (undecorated white). See report_new().

Details

Every chart is rendered at the exact size of the template's content placeholder and centred in it, and plot_bars() draws bars to a constant thickness whatever the category count, so the deck reads consistently as you flick through it.

For a deck built a slide at a time, with section dividers and per-slide control over the layout, use report_new() and the report_add_*() family instead.

Value

Invisibly path.

See Also

report_new(), use_brand().

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_slide(), report_add_table(), report_add_text(), report_layouts(), report_new(), report_save(), report_section(), report_slide(), report_title_slide(), scaffold_report()

Examples

```
tmp <- tempfile(fileext = ".pptx")
report_deck(
```

```
  list(
    "Gender" = plot_bars(calc_percentage(podracing_survey, demo_gender)),
    "NPS"    = calc_nps(podracing_survey, nps_value)
  ),
  path = tmp
)
file.exists(tmp)
```

`report_layouts` *List the slide layouts of a PowerPoint template*

Description

Shows what a template offers the report builders: every layout, its master, and the placeholders it carries. Useful to decide which `layout` to name in `report_add_slide()`, or to sanity-check an organisation template before building a deck against it.

Usage

```
report_layouts(template = NULL, style = c("elevated", "plain"))
```

Arguments

`template`	Path to a `.pptx`, an existing document from `report_new()`, or `NULL` (default) for the brand template registered by `use_brand()` (falling back to a built-in template, see `style`).
`style`	Built-in template to inspect when no `template` and no brand template are set: `"elevated"` (default) or `"plain"`. See `report_new()`.

Details

The report builders pick layouts automatically by inspecting placeholders, so most decks never need this – but when a corporate template offers several content layouts, `report_layouts()` plus an explicit `report_add_slide(layout = ...)` gives you full control. Requires the suggested `officer` package.

Value

A tibble with one row per layout: `layout`, `master`, `has_title`, `n_body`, `body_width`, `body_height` (inches of the first body placeholder, `NA` when the layout has none).

See Also

`report_new()`, `report_add_slide()`, `use_brand()`.

Other reporting: `example_report()`, `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_new()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
report_layouts()
```

`report_new` *Start a new PowerPoint or Word report*

Description

Opens an **officer** document you can build up with the `report_add_*()` helpers and write out with `report_save()`. This is the "build slides/Word directly from R" path; for a Quarto-based workflow see `scaffold_report()`.

Usage

```
report_new(
  format = c("pptx", "docx"),
  template = NULL,
  style = c("elevated", "plain"),
  slide_numbers = TRUE,
  keep_slides = TRUE
)
```

Arguments

`format`	`"pptx"` (default) for PowerPoint or `"docx"` for Word.
`template`	Optional path to a `.pptx` / `.docx` to use as the style template (reference doc). `NULL` (default) uses the brand template registered by `use_brand()`, falling back to one of the package's built-in 16:9 templates (see `style`).
`style`	Which built-in template to fall back on when no `template` and no brand template are set. `"elevated"` (default) is the styled deck: a navy/gold identity with a full-bleed navy cover, full-bleed navy section dividers, a navy title over one slim rule on every content slide and slide numbers in the corner. `"plain"` is the same widescreen deck with the same palette but no decoration – plain white slides. Ignored when a template is supplied.
`slide_numbers`	If `TRUE` (default), every content slide added with `report_add_slide()` shows the template's slide-number placeholder in its usual corner. Requires the suggested `xml2` package (silently skipped without it, or when the template has no slide-number placeholder).
`keep_slides`	PowerPoint only. If `TRUE` (default), any slides already in the template stay and your first slide is added after them. Set `FALSE` to start from an empty deck, which is what you want when the template carries example or boilerplate slides you do not intend to ship.

Details

This is the "build it from R" path: you get an **officer** document and add to it slide by slide (pptx) or section by section (docx) with the `report_add_*()` helpers, then write it out with `report_save()`. Slide layouts are chosen by inspecting the template's placeholders, so any organisation template works – see `report_layouts()` for what yours contains. For a quick deck from a list of charts and tables, `report_deck()` does the whole thing in one call; for a document-driven workflow, `scaffold_report()` gives you a Quarto template instead. Requires the suggested `officer` package.

Value

An officer document object (`rpptx` or `rdocx`).

See Also

`report_add_slide()`, `report_add_plot()`, `report_add_table()`, `report_save()`, `report_deck()`, `report_layouts()`, `use_brand()`.

Other reporting: `example_report()`, `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
doc <- report_new("pptx")
class(doc)

# any .pptx of your own replaces the built-in template
own <- system.file("templates", "ezrsurvey-16x9-plain.pptx",
                   package = "ezrsurvey")
doc <- report_new("pptx", template = own)
```

---

report_save *Save a report to disk*

Description

Writes a deck or document built with `report_new()` and the `report_*` verbs, and returns the path it used, so a build pipeline can carry on from it.

Usage

```
report_save(doc, path)
```

Arguments

`doc`	A document from `report_new()`.
`path`	Output file path (`.pptx` or `.docx`). A bare file name is written to the working directory, or into the folder set by the `output_dir` option.

Details

The format follows the document, not the file name: a deck is written as PowerPoint and a Word document as Word, so give `path` the matching `.pptx` or `.docx` extension.

Value

Invisibly `path`.

See Also

Other reporting: `example_report()`, `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_new()`, `report_section()`, `report_slide()`, `report_title_slide()`, `scaffold_report()`

Examples

```
tmp <- tempfile(fileext = ".pptx")
report_new("pptx") %>% report_add_slide("Hi") %>% report_save(tmp)
file.exists(tmp)
```

---

report_section *Add a section-divider slide*

Description

A one-line section break: a slide on the template's "Section Header" layout (falling back to a plain titled slide when the template has none). Use a short, single-word label – `"DEMOGRAPHICS"`, `"RATINGS"`, `"APPENDIX"` – to chapter a deck.

Usage

```
report_section(doc, title, layout = "Section Header", master = NULL)
```

Arguments

`doc`	A document from `report_new()`.
`title`	Section label.
`layout`	Section layout name. Defaults to `"Section Header"`.
`master`	Optional master name.

Value

The updated document.

See Also

report_slide(), report_title_slide().

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_slide(), report_add_table(), report_add_text(), report_deck(), report_layouts(), report_new(), report_save(), report_slide(), report_title_slide(), scaffold_report()

Examples

```
doc <- report_new("pptx") %>% report_section("DEMOGRAPHICS")
class(doc)
```

---

report_slide *Add a whole slide in one call (title plus its content)*

Description

The one-line-per-slide wrapper: starts a new slide, sets its title, and places content on it in a single call, so a deck script reads as one line per slide. content is dispatched by type – a ggplot becomes a chart, a data frame a table, a character vector a bulleted text box.

Usage

```
report_slide(
  doc,
  title = NULL,
  content = NULL,
  layout = NULL,
  master = NULL,
  ...
)
```

Arguments

doc	A document from report_new().
title	Slide title (typically the survey question the slide answers).
content	A ggplot, a data frame / tibble, or a character vector. NULL (default) adds an empty titled slide.
layout, master	Passed to report_add_slide(); NULL auto-selects.
...	Passed on to report_add_plot(), report_add_table() or report_add_text() depending on content.

Details

Equivalent to report_add_slide() followed by the matching report_add_*() call, but as one pipe-friendly step so calc -> plot -> add collapses onto a single line: report_slide("How likely to recommend?", plot_nps(nps_value)).

Value

The updated document.

See Also

report_section(), report_title_slide(), report_add_slide().

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_slide(), report_add_table(), report_add_text(), report_deck(), report_layouts(), report_new(), report_save(), report_section(), report_title_slide(), scaffold_report()

Examples

```
doc <- report_new("pptx") %>%
  report_slide("Who follows pod racing?",
               plot_bars(calc_percentage(podracing_survey, demo_gender)))
class(doc)
```

---

report_title_slide *Add the opening title slide*

Description

A one-line title slide, placed on the template's title layout (the one with a centre-title placeholder).

Usage

```
report_title_slide(doc, title, subtitle = NULL, layout = NULL, master = NULL)
```

Arguments

doc	A document from report_new().
title	Deck title.
subtitle	Optional strapline placed in the layout's subtitle placeholder – the respondent base and fieldwork period, say. NULL (default) leaves it empty.
layout, master	Optional overrides; NULL auto-selects the title layout.

Value

The updated document.

See Also

report_slide(), report_section().

Other reporting: example_report(), list_report_templates(), report_add_plot(), report_add_slide(), report_add_table(), report_add_text(), report_deck(), report_layouts(), report_new(), report_save(), report_section(), report_slide(), scaffold_report()

Examples

```
doc <- report_new("pptx") %>%
  report_title_slide("Pod-Racing Fan Survey",
                     subtitle = "1,000 fans | Fieldwork 2026")
class(doc)
```

---

rescale_bands *Move a band specification onto another scale*

Description

The band presets are written on the scales survey work uses most: 1-5 for a rating, 0-10 for the Net Promoter answer, -100 to 100 for the score. Plenty of real questionnaires do not use those. Satisfaction is often collected on 0-100, and a seven-point agreement scale is common. `rescale_bands()` stretches a band specification onto the scale the chart is actually drawn on, so the thresholds can stay where they are instead of the data being divided to meet them.

Usage

```
rescale_bands(bands, to, from = NULL)
```

Arguments

bands	A band specification: one of bands_rating_3() and friends, or any data frame with `from` and `to` columns.
to	Length-2 numeric: the scale the chart uses, as `c(min, max)`.
from	Length-2 numeric: the scale `bands` is written on. Read from the band edges themselves when omitted, which is what you want for the presets.

Details

The mapping is linear and proportional: a band covering the top fifth of its own scale covers the top fifth of the new one. Rescale rather than divide the data, so the axis a reader sees is still the scale the question was asked on.

Value

`bands` with `from` and `to` moved onto the new scale. Labels and colours are untouched.

See Also

Other decisions: annotate_bands(), bands_rating_3(), mark_value()

Examples

```
# satisfaction collected 0-100, thresholds still where they belong
bands_rating_3() %>% rescale_bands(to = c(0, 100))

# a seven-point agreement scale
bands_rating_3() %>% rescale_bands(to = c(1, 7))
```

---

reset_ezrsurvey_options

Reset all ezrsurvey options to their built-in defaults

Description

Reset all ezrsurvey options to their built-in defaults

Usage

```
reset_ezrsurvey_options()
```

Value

Invisibly TRUE.

See Also

ezrsurvey_options().

Other config: apply_order(), edit_ezrsurvey_profile(), ezrsurvey_options(), ezrsurvey_profile_paths(), get_dataset(), get_order(), list_orders(), load_ezrsurvey_profile(), order_for(), register_order(), register_order_presets(), remove_order(), save_ezrsurvey_profile(), use_dataset(), use_ezrsurvey_profile()

Examples

```
ezrsurvey_options(pct_axis_max = 100)
reset_ezrsurvey_options()
ezrsurvey_options()$pct_axis_max          # NULL again
```

`rse` *Relative standard error*

Description

The standard error expressed as a percentage of the estimate, `se / estimate * 100`. A common rule of thumb treats an RSE below ~5% as very good precision.

Usage

```
rse(estimate, se)
```

Arguments

`estimate`	The point estimate (mean or proportion).
`se`	Its standard error, on the same scale as `estimate`.

Details

Dividing the standard error by the estimate makes precision comparable across measures on different scales (a 1–5 rating vs a percentage). A common rule of thumb treats an RSE under ~5% as very good precision, which is the threshold `diagnose()` uses for its `precision` flag.

Value

The relative standard error, in percent.

See Also

`se_mean()`, `se_prop()`, `margin_of_error()`, `diagnose()`.

Other diagnostics: `diagnose()`, `margin_of_error()`, `precision_summary()`, `rse_rating()`, `se_mean()`, `se_prop()`

Examples

```
rse(estimate = 4.1, se = se_mean(c(4, 5, 3, 4, 5)))
```

`rse_rating` *Rate precision from a relative standard error*

Description

Turns a relative standard error (RSE, in percent) into a plain-language reliability rating, using a five-band scheme.

Usage

```
rse_rating(rse)
```

Arguments

`rse`	Relative standard error(s), in percent (see `rse()`).

Details

The bands are: under 5% "high precision", under 10% "precise", under 15% "satisfactory", under 25% "use with caution", and 25% or more "likely reliability issues". Rating estimates by their relative standard error is how national statistical agencies signal whether a survey number is solid enough to publish: the Australian Bureau of Statistics treats an RSE of 25% or more as "not reliable for most purposes", flags 25–50% with an asterisk as estimates to use with caution, and suppresses anything above 50%. The 25% boundary here is that same one. The tighter bands below it are specific to consumer-survey work, where the question is usually which of several adequately precise numbers to lead a report with. This drives the `precision` column of `diagnose()` and the overall verdict of `precision_summary()`.

Value

A character vector of ratings.

See Also

`rse()`, `diagnose()`, `precision_summary()`.

Other diagnostics: `diagnose()`, `margin_of_error()`, `precision_summary()`, `rse()`, `se_mean()`, `se_prop()`

Examples

```
rse_rating(c(3, 8, 12, 20, 40))
```

`sample_comments` *Pick a random sample of open-text comments*

Description

Pulls a tidy, presentation-ready sample of comments out of one or more free-text columns, applying the usual clean-ups: drop blanks and very short answers, optionally exclude comments matching unwanted terms, and truncate to a maximum length. The result feeds straight into `plot_quotes_tree()`.

Usage

```
sample_comments(
  data = NULL,
  ...,
  n = 8,
  min_chars = 30,
  max_chars = 400,
  exclude = NULL,
  by_column = TRUE,
  by = NULL,
  seed = NULL
)
```

Arguments

`data`	A data frame.
`...`	One or more comment columns, using tidyselect (e.g. `nps_com`, `show_com`, or `ends_with("_com")`).
`n`	Number of comments to sample. With `by_column = TRUE`, this is per column.
`min_chars`	Drop comments with this many characters or fewer. Default `30`.
`max_chars`	Truncate longer comments to this length. Default `400`.
`exclude`	Optional character vector of (case-insensitive) regex terms; comments matching any are dropped (e.g. `c("friend", "recommend")`).
`by_column`	If `TRUE` (default), sample `n` from each column; if `FALSE`, sample `n` from the pooled comments.
`by`	Optional grouping column (unquoted), e.g. an NPS group. Sampling then takes `n` from each group, and the group is kept as a column of the result so a slide can filter on it.
`seed`	Optional integer for reproducible sampling (the RNG state is restored afterwards).

Details

Verbatims need clean-up before they go on a slide: this drops blanks and very short answers, optionally removes comments matching unwanted terms (`exclude`), and truncates the long ones. With `by_column = TRUE` you get `n` from each source column (e.g. an even split across `nps_com` and `show_com`); with `FALSE`, `n` from the pooled set. The result feeds `plot_quotes_tree()`. For a set that is varied rather than uniformly random, use `sample_comments_diverse()`.

Value

A tibble with `source`, `comment` and `length`, plus the by column when one is given.

See Also

`sample_comments_diverse()` for a diversity-aware sample, `plot_quotes_tree()`.

Other comments: `sample_comments_diverse()`

Examples

```
sample_comments(podracing_survey, nps_com, show_com, n = 3)

# three quotes per NPS group, with the group kept as a column
podracing_survey %>%
  mutate(group = nps_group(nps_value, labels = TRUE)) %>%
  filter(!is.na(group)) %>%
  sample_comments(nps_com, n = 3, by = group)
```

---

sample_comments_diverse

Pick a diverse, information-rich sample of comments

Description

A smarter alternative to `sample_comments()`: instead of sampling uniformly, it scores each comment and selects a set that is both **informative** and **varied**, so a quote slide isn't five ways of saying the same thing. Scoring is lexical – comments are turned into TF-IDF vectors, an information score is computed (Shannon entropy of the word distribution, or TF-IDF mass), and a randomised maximal-marginal-relevance pass trades off informativeness against similarity to already-picked comments.

Usage

```
sample_comments_diverse(
  data = NULL,
  ...,
  n = 8,
  min_chars = 30,
  max_chars = 400,
```

```
  exclude = NULL,
  lambda = 0.6,
  method = c("mmr", "entropy"),
  score = c("entropy", "tfidf"),
  stopwords = NULL,
  max_candidates = 500,
  seed = NULL
)
```

Arguments

data	A data frame.
...	One or more comment columns, using tidyselect (e.g. nps_com, show_com, or ends_with("_com")).
n	Number of comments to sample. With by_column = TRUE, this is per column.
min_chars	Drop comments with this many characters or fewer. Default 30.
max_chars	Truncate longer comments to this length. Default 400.
exclude	Optional character vector of (case-insensitive) regex terms; comments matching any are dropped (e.g. c("friend", "recommend")).
lambda	Diversity/informativeness trade-off in [0, 1] for method = "mmr": higher favours informative comments, lower favours dissimilar ones. Default 0.6.
method	"mmr" (default) for the diversity-aware greedy selection, or "entropy" to weight a random sample by each comment's information score.
score	By which to measure information: "entropy" (default, Shannon bits) or "tfidf" (summed TF-IDF weight).
stopwords	Stop-word handling: NULL (default) uses the stopwords package when installed (English), otherwise a small built-in fallback; a character vector supplies your own list; FALSE disables removal. Pass e.g. stopwords::stopwords("de") for another language.
max_candidates	Largest number of comments compared with one another (default 500). A bigger corpus is narrowed to a random shortlist of this size first, because the comparison grows with the square of the number kept. Raise it to widen the choice, at the cost of time and memory.
seed	Optional integer for reproducible sampling (the RNG state is restored afterwards).

Value

A tibble with source, comment, length and the info score, ordered by selection.

See Also

sample_comments(), plot_quotes_tree().

Other comments: sample_comments()

Examples

```
sample_comments_diverse(podracing_survey, nps_com, show_com, n = 5, seed = 1)
```

---

save_data *Quick-save a table to CSV, TSV or XLSX*

Description

A pipe-friendly one-liner for dropping a summary table to disk. The format is taken from the file extension. For `.xlsx`, pass a single data frame for one sheet, or a *named* list of data frames to write one sheet per element. Returns its input invisibly so it can sit mid-pipeline.

Usage

```
save_data(data, path, na = "", ...)
```

Arguments

`data`	A data frame / tibble, or (for `.xlsx`) a named list of them.
`path`	Output path; the extension sets the format (`.csv`, `.tsv`, `.xlsx`). A bare file name is written to the working directory, or into the folder set by the `output_dir` option.
`na`	String to write for missing values. Default `""`.
`...`	Passed to the underlying writer (`readr::write_csv()` / `readr::write_tsv()` / `writexl::write_xlsx()`).

Details

The format follows the file extension: `.csv`/`.tsv` via **readr**, `.xlsx` via the suggested `writexl` package. For an Excel workbook, pass one data frame for a single sheet, or a *named* list of data frames for one sheet each. Returns its input invisibly so it can sit mid-pipeline. To send several separate calls to their own tabs in one line, see `export_xlsx()`.

Value

`data`, invisibly.

See Also

`save_plot()`, `save_output()`, `export_xlsx()`.

Other save: `export_summary_xlsx()`, `export_xlsx()`, `save_output()`, `save_plot()`

Examples

```
tab <- calc_percentage(podracing_survey, demo_gender)
tmp <- tempfile(fileext = ".csv")
save_data(tab, tmp)
file.exists(tmp)
```

save_ezrsurvey_profile *Save current options and orders to a profile*

Description

Writes your current non-default `ezrsurvey_options()` and all registered orders (see `register_order()`) to a YAML profile, so they persist across sessions. Requires the suggested `yaml` package.

Usage

```
save_ezrsurvey_profile(
  path = NULL,
  include_options = TRUE,
  include_orders = TRUE
)
```

Arguments

`path`	Destination path. If `NULL` (default), the per-user config file is used.
`include_options`	Write changed options. Default `TRUE`.
`include_orders`	Write registered orders. Default `TRUE`.

Value

Invisibly the path written.

See Also

`load_ezrsurvey_profile()`, `edit_ezrsurvey_profile()`, `register_order()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `use_dataset()`, `use_ezrsurvey_profile()`

Examples

```
ezrsurvey_options(pct_axis_max = 100)
tmp <- tempfile(fileext = ".yml")
save_ezrsurvey_profile(tmp, include_orders = FALSE)
readLines(tmp)
reset_ezrsurvey_options()
```

save_output *Quick-save any ezrsurvey output (plot or table)*

Description

Convenience dispatcher: routes ggplots to `save_plot()` and data frames / lists to `save_data()`, choosing by the object type. Handy at the end of a pipe when you do not want to think about which saver to call.

Usage

```
save_output(x, path, ...)
```

Arguments

`x`	A ggplot, a data frame, or a (named) list of data frames.
`path`	Output path; the extension picks the format. A bare file name is written to the working directory, or into the folder set by the `output_dir` option.
`...`	Passed to `save_plot()` or `save_data()`.

Value

`x`, invisibly.

See Also

`save_plot()`, `save_data()`.

Other save: `export_summary_xlsx()`, `export_xlsx()`, `save_data()`, `save_plot()`

Examples

```
tmp_csv <- tempfile(fileext = ".csv")
calc_percentage(podracing_survey, demo_gender) %>% save_output(tmp_csv)


tmp_png <- tempfile(fileext = ".png")
plot_bars(calc_percentage(podracing_survey, demo_gender)) %>% save_output(tmp_png)
```

save_plot *Quick-save a plot to PNG, SVG or PDF*

Description

A thin, pipe-friendly wrapper around ggplot2::ggsave() that picks the device from the file extension and uses presentation-friendly defaults (transparent background, generous size). Returns the plot invisibly, so it drops into a pipeline without breaking it.

Usage

```
save_plot(
  plot,
  path,
  width = 8,
  height = 4.5,
  dpi = 300,
  bg = "transparent",
  ...
)
```

Arguments

plot	A ggplot object.
path	Output path; the extension sets the format (.png, .svg, .pdf, .jpg/.jpeg, .tiff). A bare file name is written to the working directory, or into the folder set by the output_dir option.
width, height	Size in inches. Default 8 x 4.5.
dpi	Raster resolution for PNG/JPG/TIFF. Default 300.
bg	Background fill. Default "transparent" (matches the ezrsurvey themes); use "white" for a solid background.
...	Passed to ggplot2::ggsave().

Details

The device is chosen from the file extension, and the plot is returned invisibly so the call slots into a pipeline without breaking it (p %>% save_plot("p.png") %>% print()). Defaults suit slides: a transparent background (matching the ezrsurvey themes) and a generous size; pass bg = "white" for a solid background. SVG output needs the suggested svglite package.

Value

The plot, invisibly.

See Also

save_data(), save_output().

Other save: export_summary_xlsx(), export_xlsx(), save_data(), save_output()

Examples

```
p <- plot_bars(calc_percentage(podracing_survey, demo_gender))
tmp <- tempfile(fileext = ".png")
save_plot(p, tmp)
file.exists(tmp)
```

---

scaffold_report *Scaffold a Quarto survey-report template*

Description

Copies a ready-to-render Quarto report skeleton into your project, wired up with ezrsurvey helpers against the bundled `podracing_survey` data: an executive summary, per-question sections with placeholder narrative, and a methodology appendix built from the precision diagnostics. Swap in your own data via the `data` parameter (or at the top of the file) and render with Quarto. This is the document-driven counterpart to the report_new() / report_deck() officer builders.

Usage

```
scaffold_report(
  format = c("pptx", "html", "pdf", "docx"),
  path,
  title = "Survey Report",
  author = NULL,
  reference_doc = NULL,
  overwrite = FALSE
)
```

Arguments

format	Output format: one of list_report_templates() ("pptx", "html", "pdf", "docx"). Defaults to "pptx".
path	Destination .qmd path. A bare file name is written to the working directory, or into the folder set by the output_dir option.
title	Title inserted into the template's YAML header.
author	Author name for the YAML header. NULL leaves a placeholder.
reference_doc	Path to a PowerPoint / Word template used as the Quarto reference-doc (pptx and docx formats only). NULL (default) uses the brand template registered by use_brand(), if any; pptx then falls back to the package's built-in 16:9 template, docx leaves the line commented in the scaffold.
overwrite	Overwrite path if it already exists. Defaults to FALSE.

Details

Each scaffold declares one Quarto parameter: `data`, the path to a CSV of your survey; empty means the bundled example data.

Corporate templates: Quarto/Pandoc fills a `reference-doc` by looking for the *standard* layout names ("Title Slide", "Title and Content", "Two Content", ...). A corporate template whose layouts keep those names works directly; one with renamed layouts will render broken or blank slides here – build the deck with `report_deck()` instead, which inspects placeholders and works with any template.

Value

Invisibly the path written.

See Also

`report_deck()` for building decks directly without Quarto; `use_brand()` to register a default reference document.

Other reporting: `example_report()`, `list_report_templates()`, `report_add_plot()`, `report_add_slide()`, `report_add_table()`, `report_add_text()`, `report_deck()`, `report_layouts()`, `report_new()`, `report_save()`, `report_section()`, `report_slide()`, `report_title_slide()`

Examples

```
tmp <- tempfile(fileext = ".qmd")
scaffold_report("html", path = tmp, title = "Q2 Customer Survey")
file.exists(tmp)
```

---

scale_fill_brand *Brand fill and colour scales*

Description

Discrete ggplot2 scales using the brand accent palette from `pal_brand()`. `scale_color_brand()` is an alias of `scale_colour_brand()`.

Usage

```
scale_fill_brand(...)

scale_colour_brand(...)

scale_color_brand(...)
```

Arguments

`...`	Passed to `ggplot2::discrete_scale()`.

Details

The palette function interpolates when a plot needs more colours than the brand defines, so the scale never runs out. Without a brand set, the scales fall back to the pal_sequential_blue defaults via `pal_brand()`.

Value

A ggplot2 scale.

See Also

`pal_brand()`, `use_brand()`.

Other brand: `brand_info()`, `clear_brand()`, `pal_brand()`, `use_brand()`

Examples

```
df <- calc_percentage(podracing_survey, demo_gender)
ggplot(df, aes(demo_gender, pct, fill = demo_gender)) +
  geom_col() +
  scale_fill_brand()
```

---

scale_fill_nps *NPS group fill scale*

Description

Manual fill scale mapping the NPS-group coding (`"-1"` / `"0"` / `"1"`) onto the pal_nps palette.

Usage

```
scale_fill_nps(...)
```

Arguments

`...`	Passed to `ggplot2::scale_fill_manual()`.

Details

Maps the NPS-group coding from `nps_group()` (`"-1"` detractor, `"0"` passive, `"1"` promoter) onto the pal_nps red/amber/green palette, so the colours carry their usual meaning. Used by `plot_nps()`.

Value

A ggplot2 scale.

See Also

pal_nps, nps_group(), plot_nps().

Other themes: pal_rating, scale_fill_rating(), theme_ezrsurvey(), theme_transparent()

Examples

```
df <- data.frame(score = 0:2, pct = c(20, 30, 50), group = c(-1, 0, 1))
ggplot(df, aes(score, pct, fill = factor(group))) +
  geom_col() +
  scale_fill_nps()
```

---

scale_fill_rating *Rating colour and fill scales*

Description

Manual ggplot2 scales mapping a 1-5 rating (as a factor or its character codes "1".."5") onto the pal_rating red-amber-green palette.

Usage

```
scale_fill_rating(distinct = FALSE, ...)

scale_colour_rating(distinct = FALSE, ...)

scale_color_rating(distinct = FALSE, ...)
```

Arguments

distinct	If TRUE, use a fully distinct 5-colour ramp where each point has its own shade; if FALSE (default) use pal_rating, where 1 and 2 share red and 4 and 5 share green. The distinct ramp suits stacked bars, where neighbouring segments need to be told apart.
...	Passed to ggplot2::scale_fill_manual() / ggplot2::scale_colour_manual().

Details

Maps the rating codes "1".."5" onto pal_rating, which uses the bands_rating_3() thresholds (1-2 bad, 3 ok, 4-5 good). Set distinct = TRUE for a fully distinct 5-colour ramp, which reads better on stacked bars where adjacent segments must be separable. scale_color_rating() is an alias of scale_colour_rating().

Value

A ggplot2 scale.

See Also

pal_rating, scale_fill_nps().

Other themes: pal_rating, scale_fill_nps(), theme_ezrsurvey(), theme_transparent()

Examples

```
df <- data.frame(feature = c("a", "b"), pct = c(60, 40), rating = c(4, 2))
ggplot(df, aes(feature, pct, fill = factor(rating))) +
  geom_col() +
  scale_fill_rating()
```

---

scale_y_pct *Continuous y-scale capped at a nice maximum with optional percent labels*

Description

A thin wrapper around ggplot2::scale_y_continuous() that sets the upper limit with nice_max() and, by default, formats the axis as percentages. Designed for the percentage bar charts produced by calc_percentage().

Usage

```
scale_y_pct(values = NULL, unit = 25, pad = 0, max = NULL, labels = TRUE, ...)
```

Arguments

values	Numeric vector used to determine the axis ceiling, typically the column being plotted. If NULL (default) the limit is left to ggplot2 and only the label formatting is applied.
unit, pad	Passed to nice_max().
max	Optional fixed upper limit. If supplied (e.g. 100), it overrides the dynamic nice_max() ceiling. Defaults to NULL.
labels	One of: • TRUE (default) – format breaks as "42%". • FALSE – hide axis labels entirely (labels = NULL). • a function – used directly as the labels argument.
...	Further arguments passed to ggplot2::scale_y_continuous().

Details

The upper limit is chosen in one of three ways, in order of precedence: a fixed max if you supply one; otherwise nice_max() of values; otherwise left to ggplot2. Labels default to whole-percent strings via label_pct(). plot_bars() calls this for you, reading its unit and max from the pct_axis_unit / pct_axis_max options (see ezrsurvey_options()), so you rarely add it by hand – but it is exported for custom charts.

Value

A ggplot2 scale you can add to a plot with +.

See Also

nice_max(), label_pct(), plot_bars().

Other scales: label_pct(), nice_max()

Examples

```
df <- calc_percentage(podracing_survey, demo_gender)
ggplot(df, aes(demo_gender, pct)) +
  geom_col() +
  scale_y_pct(df$pct)            # axis capped at a tidy multiple, "%" labels
```

---

se_mean *Standard error of a mean (point estimate)*

Description

sd(x) / sqrt(n), the sampling error of a mean such as an average 1-5 rating. Packages the inline sem calculation from the report appendix.

Usage

```
se_mean(x)
```

Arguments

x	A numeric vector. NA values are ignored.

Details

The standard error of the mean is the sample standard deviation divided by the square root of the (non-missing) sample size, sd(x) / sqrt(n). It shrinks as the sample grows, and is the basis of the margin of error you quote for an average rating. Fewer than two values gives NA (no spread to estimate).

Value

A single numeric standard error, or NA if fewer than two non-missing values are present.

See Also

se_prop(), rse(), margin_of_error(), diagnose().

Other diagnostics: diagnose(), margin_of_error(), precision_summary(), rse(), rse_rating(), se_prop()

Examples

```
se_mean(c(4, 5, 3, 4, 5, 2, 4))
```

`se_prop` *Standard error of a proportion*

Description

`sqrt(p * (1 - p) / n)`, the sampling error of a percentage. Packages the inline proportion-error calculation from the report appendix.

Usage

```
se_prop(p, n, pctp = FALSE)
```

Arguments

`p`	The proportion, either as a fraction in `[0, 1]` or as a percentage in `(1, 100]` (values above 1 are divided by 100 automatically).
`n`	The sample size (number of respondents).
`pctp`	Return the answer in percentage points instead of on the `[0, 1]` proportion scale. Defaults to `FALSE`.

Details

For a percentage from a survey, the sampling error is `sqrt(p * (1 - p) / n)`. It is largest when `p = 0.5` (a 50/50 split is the hardest to pin down) and shrinks towards the extremes. The result is on the same 0–1 scale as a fractional `p`; pass `pctp = TRUE` for percentage points, the form used in report footnotes and the one `margin_of_error()` needs if you want the margin in points. Inputs above 1 are treated as percentages and divided by 100, so `se_prop(33, n)` and `se_prop(0.33, n)` agree.

Value

The standard error of the proportion, on the same `[0, 1]` scale as a fractional `p`, or in percentage points when `pctp = TRUE`.

See Also

`se_mean()`, `rse()`, `margin_of_error()`, `diagnose()`.

Other diagnostics: `diagnose()`, `margin_of_error()`, `precision_summary()`, `rse()`, `rse_rating()`, `se_mean()`

Examples

```
se_prop(0.33, 1184)

se_prop(0.33, 1184, pctp = TRUE)
```

`select_prefix` *Select identifier and prefixed columns*

Description

Keeps a handful of id columns plus every column sharing a prefix – the `select(id, starts_with("demo_"))` idiom – in one tidy call.

Usage

```
select_prefix(data = NULL, prefix, keep = NULL)
```

Arguments

`data`	A data frame. If omitted, the session default (`use_dataset()`) is used.
`prefix`	One or more column-name prefixes to keep (e.g. `"demo_"`, or `c("demo_", "ratings_")`).
`keep`	Optional character vector of additional column names to retain in front of the prefixed block (e.g. an id column).

Details

Survey questionnaires are usually organised by prefix (`demo_`, `ratings_`, `partner_`, ...), so this is a quick way to grab one block plus a couple of id columns without naming every variable. Order is `keep` first, then the prefix-matching columns in their original order.

Value

A data frame with `keep` columns followed by all prefix-matching columns.

See Also

`read_folder()`, `calc_percentage_multi()`.

Other import: `parse_filename()`, `read_folder()`, `select_suffix()`

Examples

```
select_prefix(podracing_survey, "demo_", keep = "respondent_id")
select_prefix(podracing_survey, c("ratings_", "partner_"))
```

`select_suffix` *Select identifier and suffixed columns*

Description

The mirror of `select_prefix()`: keeps a handful of id columns plus every column sharing a suffix – the `select(id, ends_with("_com"))` idiom – in one tidy call.

Usage

```
select_suffix(data = NULL, suffix, keep = NULL)
```

Arguments

`data`	A data frame. If omitted, the session default (`use_dataset()`) is used.
`suffix`	One or more column-name suffixes to keep (e.g. `"_com"`, or `c("_com", "_score")`).
`keep`	Optional character vector of additional column names to retain in front of the suffixed block (e.g. an id column).

Details

Some questionnaire blocks are marked by a trailing tag rather than a leading one (`_com` open-text follow-ups, `_score` derived columns), so this grabs one such block plus a couple of id columns without naming every variable. Order is `keep` first, then the suffix-matching columns in their original order.

Value

A data frame with `keep` columns followed by all suffix-matching columns.

See Also

`select_prefix()`, `read_folder()`.

Other import: `parse_filename()`, `read_folder()`, `select_prefix()`

Examples

```
select_suffix(podracing_survey, "_com", keep = "respondent_id")
```

`set_weights` *Set a survey weighting scheme for the session*

Description

Defines target shares for one or more categorical variables (e.g. a known population split by gender or region). Once set, the summary helpers (`calc_percentage()`, `calc_nps()`, `calc_summary()`, `crosstab()`) weight their results automatically, so the weighted margins match your targets. Pass `weights = FALSE` to any of them to opt out for a single call.

Usage

```
set_weights(...)
```

Arguments

`...`	One scheme per variable. Two forms are accepted: • **vector form** (one argument per variable) carrying a `variable` entry and a share per category: `c(variable = "demo_gender", Male = 0.49, Female = 0.50, "Non-binary" = 0.01)`; • **named form**: `demo_gender = c(Male = 0.49, Female = 0.50)`, where the argument name is the column. Shares need not sum to 1 – they are normalised per variable (so raw percentages or counts work too).

Details

With a single variable the weights are exact post-stratification: `target_share / sample_share` for each category. With several variables the weights are found by raking (iterative proportional fitting) so every variable's weighted margin matches its targets. Weights are normalised to mean 1 (so the weighted base equals the sample size), and respondents whose weighting value is blank / a non-answer (see `na_blank()`) are left unadjusted. A category that appears in the data but is missing from your targets is an error – give every observed category a share. If a default dataset is set, this prints the Kish design effect and the effective sample size so you can see the precision cost.

Value

Invisibly, the parsed scheme (a named list of normalised targets).

See Also

`clear_weights()`, `get_weights()`, `weight_vector()`, `calc_percentage()`.

Other weighting: `clear_weights_cache()`, `get_weights()`, `weight_vector()`

Examples

```
set_weights(c(variable = "demo_gender",
              "Male" = 0.49, "Female" = 0.50,
              "Non-binary" = 0.01))
calc_percentage(podracing_survey, satis_return)   # gains a wpct column
clear_weights()
```

`shopping_survey` *Simulated historical shopping-behaviour survey*

Description

A fictional survey of patrons of a turn-of-the-century (c. 1905) general emporium, written in an Edwardian register for colour. It behaves like a real survey export – a latent satisfaction drives the worded attribute ratings and the recommend score together, demographics use standard sex/country categories alongside period social-class and occupation items, and the multi-select and open-text columns mirror the shapes the helpers consume. A companion to podracing_survey for examples that want a second, very different theme.

Usage

```
shopping_survey
```

Format

A tibble with 800 rows and 25 columns:

patron_id Unique patron id (character).

demo_age Age in years (integer, 18-80).

demo_gender Sex (Female / Male, some "Prefer not to answer").

demo_country Country of residence (ISO 3166 names).

region World region derived from `demo_country`.

social_class Edwardian social class (Upper class .. Poor).

occupation Period occupation (Clerk, Domestic servant, ...).

household_size Number in the household (integer).

weekly_spend Weekly spend at the emporium, in shillings (numeric).

payment How they pay: Cash, On account or Barter.

transport How they travel to the shop (On foot, Horse and cart, ...).

visit_frequency How often they visit (Never .. Always).

recommend 0-10 "would recommend to a neighbour" rating (integer).

ratings_goods, ratings_service, ratings_credit, ratings_delivery, ratings_value Worded 1-5 attribute ratings (Very bad .. Very good).

reasons_price, reasons_quality, reasons_credit, reasons_proximity, reasons_variety, reasons_service Multi-select reasons for patronage; each holds its option text when chosen, otherwise `""`.

comment Open-text comment (mostly blank).

Source

Simulated. See `data-raw/make_shopping_survey.R` for the generator.

See Also

Other data: `country_region`, `currency_rates`, `podracing_survey`

Examples

```
calc_percentage(shopping_survey, social_class, sort = "desc")
calc_summary(shopping_survey, weekly_spend, by = payment)
```

---

split_multi *Split a packed multi-select column into one column per answer*

Description

Turns the single cell a spreadsheet export gives a "tick all that apply" question (`"Speed; Drivers; Betting"`) into the one-column-per-option layout the rest of the package expects, so `calc_percentage_multi()`, `crosstab()` and the plots all work on it.

Usage

```
split_multi(data = NULL, column, split = "auto", prefix = NULL)
```

Arguments

`data`	A data frame. If omitted, the session default (`use_dataset()`) is used.
`column`	The packed column (unquoted).
`split`	The delimiter between answers inside a cell. `"auto"` (default) detects `";"`, `"|"` or `","`, in that order. Pass a string to force one.
`prefix`	Prefix for the new column names. Defaults to the column's own name followed by an underscore, e.g. `motivations_`.

Details

Google Forms, Google Sheets and most survey platforms export a multi-select question as one cell per respondent holding every answer they ticked, joined by a delimiter. That shape cannot be counted directly, because a respondent who ticked three options is one row, not three. This function unpacks it into the indicator layout the package's own datasets use.

The new columns are named `prefix` plus the answer text verbatim, which keeps the labels readable all the way through to a chart: `calc_percentage_multi()` strips the prefix back off and uses what remains as the option label. Answer text is rarely a syntactic R name, so refer to the new columns with backticks if you need them individually. Surrounding whitespace is trimmed and empty answers are dropped, so `"Speed; ; Drivers"` yields two options. When no cell contains

the delimiter, each cell is treated as a single answer, which is the right answer for a multi-select question everyone happened to answer once.

If you only want the percentages, skip this step: `calc_percentage_multi()` detects a packed column and splits it for you.

Value

The data frame with one new column per distinct answer, each holding the answer text for respondents who chose it and `""` for those who did not, ordered most-chosen first. The packed column is kept.

See Also

`calc_percentage_multi()`, `na_blank()`.

Other recode: `add_region()`, `bin_numeric()`, `clean_label()`, `drop_items()`, `ensure_numeric()`, `generation_scheme()`, `na_blank()`, `nps_group()`, `recode_age()`, `recode_generation()`, `recode_likert()`, `recode_region()`

Examples

```
packed <- data.frame(
  respondent = 1:4,
  motivations = c("Speed; Drivers", "Speed", "", "Betting; Speed")
)
split_multi(packed, motivations)

split_multi(packed, motivations) %>%
  calc_percentage_multi("motivations_", id = respondent, sort = "desc")
```

---

theme_ezrsurvey *ezrsurvey ggplot2 themes*

Description

A clean, presentation-oriented theme family for survey reporting. The base theme strips chart junk – no ticks, no gridlines, centred bold titles – which suits labelled bar charts that carry their own data labels. The _x / _y / _xy variants add back faint major gridlines on the named axes.

Usage

```
theme_ezrsurvey(base_size = 11, base_family = NULL, transparent = FALSE)

theme_ezrsurvey_x(base_size = 11, base_family = NULL, transparent = FALSE)

theme_ezrsurvey_y(base_size = 11, base_family = NULL, transparent = FALSE)

theme_ezrsurvey_xy(base_size = 11, base_family = NULL, transparent = FALSE)
```

Arguments

base_size	Base font size in points. Defaults to 11.
base_family	Base font family. NULL (default) uses the brand body font when one is set and installed (see use_brand()), else "sans".
transparent	If TRUE, make the plot, panel and legend backgrounds transparent (handy for slides). Defaults to FALSE.

Details

A clean, presentation-first look: no axis ticks, no chart junk, bold centred titles, and faint gridlines only where you ask for them. The base theme_ezrsurvey() strips gridlines entirely (best for bar charts that carry their own data labels); the _x / _y / _xy variants add back faint major gridlines on the named axes. transparent = TRUE makes the backgrounds transparent for slides. All the plot_*() helpers apply one of these for you.

Value

A ggplot2::theme() object to add to a plot with +.

See Also

Other themes: pal_rating, scale_fill_nps(), scale_fill_rating(), theme_transparent()

Examples

```
df <- calc_percentage(podracing_survey, demo_gender)
ggplot(df, aes(demo_gender, pct)) + geom_col() + theme_ezrsurvey()
ggplot(df, aes(demo_gender, pct)) + geom_col() + theme_ezrsurvey_y()
```

---

theme_transparent *Transparent-background theme fragment*

Description

A small ggplot2::theme() fragment that makes the plot, panel and legend backgrounds transparent. Add it to any theme for slide-friendly exports; the transparent argument of theme_ezrsurvey() applies it for you.

Usage

```
theme_transparent()
```

Value

A ggplot2::theme() object.

See Also

Other themes: pal_rating, scale_fill_nps(), scale_fill_rating(), theme_ezrsurvey()

Examples

```
p <- calc_percentage(podracing_survey, demo_gender) %>% plot_bars()
p + theme_transparent()
```

---

use_brand *Adopt an organisation brand from a PowerPoint or Word template*

Description

Reads the colour and font theme out of your organisation's .pptx / .docx template and makes it the session default for everything ezrsurvey produces: charts pick up the brand accent colours and body font, and the template file itself becomes the default reference document for report_new(), report_deck() and scaffold_report(). One call, and analysis output is on-brand.

Usage

```
use_brand(template = NULL, colors = NULL, fonts = NULL, quiet = FALSE)
```

Arguments

template	Path to a .pptx or .docx file. Its OOXML theme (accent colours, heading/body typefaces) is extracted, and the file is registered as the default reference document for reports of the matching format. NULL to set colours/fonts directly without a template.
colors	Optional character vector of hex colours overriding the extracted accents. A single colour sets only the primary; a vector sets the full accent palette (first entry = primary). Invalid entries are dropped with a warning.
fonts	Optional character vector overriding the extracted typefaces: either a single body font, or c(major = "...", minor = "...").
quiet	If TRUE, suppress the confirmation message.

Details

Extraction reads theme1.xml inside the template (requires the suggested xml2 package): accent colours 1-6 become brand_colors (first accent = brand_color_primary, the default fill of plot_bars()), and the minor (body) typeface becomes the default base_family of theme_ezrsurvey() – but only when that font is actually installed on this machine, so charts never render with substituted glyph boxes. Set ezrsurvey_options(brand_fonts_enabled = FALSE) to keep brand colours but ignore brand fonts.

A brand font also needs a graphics device that reads the machine's fonts. The pdf and postscript devices do not: they carry their own short list of families, and a chart printed to one fails with "invalid font type". That is the default device in a plain Rscript run, so use_brand() says so when

it sets a font there. Charts written with save_plot() are unaffected, because it saves through a device that does read system fonts.

Everything lands in ordinary ezrsurvey_options() (brand_* keys), so you can equally set the values by hand or persist them in a YAML profile; a later use_brand() call simply overwrites them. Semantic palettes (pal_rating, pal_nps) deliberately keep their meaning-carrying colours and are not rebranded. Use clear_brand() to return to the neutral look.

Value

Invisibly a ezrsurvey_brand list (see brand_info()).

See Also

brand_info(), clear_brand(), pal_brand(), scale_fill_brand(), report_new(), scaffold_report().

Other brand: brand_info(), clear_brand(), pal_brand(), scale_fill_brand()

Examples

```
tmp <- tempfile(fileext = ".pptx")
print(officer::read_pptx(), target = tmp)
use_brand(tmp, quiet = TRUE)
brand_info()
clear_brand()

# Or set brand values directly, no template needed:
use_brand(colors = c("#0B5394", "#E69138"), fonts = "Georgia", quiet = TRUE)
clear_brand()
```

---

use_dataset *Set a default dataset for the session*

Description

Registers a data frame as the session's default, so the analysis helpers can be called without repeating data every time. Set it once at the top of a script or Quarto document and then omit data (or pass it by pipe).

Usage

```
use_dataset(data)
```

Arguments

data	A data frame to use as the default.

Details

With a default set you can drop `data` entirely and name the column positionally, the way you would in the tidyverse:

- `calc_percentage(demo_gender)` – data taken from the default;
- `podracing_survey %>% calc_percentage(demo_gender)` – explicit pipe;
- `calc_percentage(other_survey, demo_gender)` – an explicit data frame always wins over the default.

The helpers tell a column from a data frame by looking at the first argument: a data frame is used as the data; anything else (a bare column name) is taken as the first column and the data comes from the default. Helpers that select several columns through `...` (e.g. `diagnose()`, `calc_percentage_batch()`) work the same way – `diagnose(starts_with("ratings_"))` needs no `data`.

The default applies to the analysis helpers that read a raw survey (`calc_percentage()`, `calc_summary()`, `calc_nps()`, `diagnose()`, `crosstab()`, `ipm_model()`, `sample_comments()`, ...). It does **not** apply to plot/report helpers that consume an already-summarised table. The setting lives only in the current R session; clear it with `clear_dataset()`.

Value

`data`, invisibly (so it can sit in a pipe).

See Also

`get_dataset()`, `clear_dataset()`, `has_dataset()`.

Other config: `apply_order()`, `edit_ezrsurvey_profile()`, `ezrsurvey_options()`, `ezrsurvey_profile_paths()`, `get_dataset()`, `get_order()`, `list_orders()`, `load_ezrsurvey_profile()`, `order_for()`, `register_order()`, `register_order_presets()`, `remove_order()`, `reset_ezrsurvey_options()`, `save_ezrsurvey_profile()`, `use_ezrsurvey_profile()`

Examples

```
use_dataset(podracing_survey)
calc_percentage(demo_gender)   # data taken from the default
calc_nps(nps_value)
clear_dataset()
```

---

use_ezrsurvey_profile *Create a starter ezrsurvey profile*

Description

Writes a commented YAML profile (pre-filled with the current defaults) you can edit to set persistent options. By default it is written to the per-user config directory (`tools::R_user_dir()`), which is where ezrsurvey looks first.

Usage

```
use_ezrsurvey_profile(path = NULL, overwrite = FALSE)
```

Arguments

path	Destination path. If NULL (default), the per-user config file ezrsurvey.yml is used.
overwrite	Overwrite an existing file. Defaults to FALSE.

Value

Invisibly the path written.

See Also

load_ezrsurvey_profile(), ezrsurvey_options().

Other config: apply_order(), edit_ezrsurvey_profile(), ezrsurvey_options(), ezrsurvey_profile_paths(), get_dataset(), get_order(), list_orders(), load_ezrsurvey_profile(), order_for(), register_order(), register_order_presets(), remove_order(), reset_ezrsurvey_options(), save_ezrsurvey_profile(), use_dataset()

Examples

```
tmp <- tempfile(fileext = ".yml")
use_ezrsurvey_profile(tmp)
head(readLines(tmp))
```

---

weight_vector *Compute the per-respondent survey weights*

Description

Returns the weight vector a scheme implies for a dataset – the same numbers the summary helpers use internally – so you can inspect them, attach them as a column, or feed them to another tool.

Usage

```
weight_vector(data = NULL, weights = NULL)
```

Arguments

data	A data frame. If omitted, the session default (use_dataset()) is used.
weights	A weighting scheme (any form accepted by set_weights()). If NULL (default), the session scheme is used.

Details

This is the function behind the automatic weighting: each respondent's weight is `target_share / sample_share` for their category (raked across variables when there are several), normalised so the weighted base equals the unweighted one. Pair it with `set_weights()`'s reported design effect to judge how much precision the weighting costs.

Value

A numeric vector of length `nrow(data)`, normalised to mean 1.

See Also

`set_weights()`, `calc_percentage()`.

Other weighting: `clear_weights_cache()`, `get_weights()`, `set_weights()`

Examples

```
target <- c(variable = "demo_gender",
            "Male" = 0.49, "Female" = 0.50, "Non-binary" = 0.01)
w <- weight_vector(podracing_survey, target)

# The sample under-represents women, so their weights are above 1 and the
# weighted share moves towards the target. Categories the target does not
# mention keep a weight of 1, which is why their shares do not move.
podracing_survey %>%
  mutate(weight = w) %>%
  group_by(demo_gender) %>%
  summarise(
    unweighted_pct = round(n() / nrow(podracing_survey) * 100, 1),
    weighted_pct = round(sum(weight) / nrow(podracing_survey) * 100, 1)
  )
```

Index

∗ **brand**
brand_info, 11
clear_brand, 21
pal_brand, 56
scale_fill_brand, 106
use_brand, 119
∗ **comments**
sample_comments, 98
sample_comments_diverse, 99
∗ **compare**
compare_values, 23
plot_diff, 62
∗ **config**
apply_order, 8
edit_ezrsurvey_profile, 34
ezrsurvey_options, 40
ezrsurvey_profile_paths, 42
get_dataset, 43
get_order, 44
list_orders, 48
load_ezrsurvey_profile, 50
order_for, 56
register_order, 80
register_order_presets, 81
remove_order, 82
reset_ezrsurvey_options, 95
save_ezrsurvey_profile, 102
use_dataset, 120
use_ezrsurvey_profile, 121
∗ **currency**
add_currency, 4
convert_currency, 24
list_currencies, 48
∗ **datasets**
country_region, 25
currency_rates, 31
podracing_survey, 71
shopping_survey, 115
∗ **data**
country_region, 25
currency_rates, 31
podracing_survey, 71
shopping_survey, 115
∗ **decisions**
annotate_bands, 7
bands_rating_3, 9
mark_value, 52
rescale_bands, 94
∗ **diagnostics**
diagnose, 32
margin_of_error, 51
precision_summary, 72
rse, 96
rse_rating, 97
se_mean, 110
se_prop, 111
∗ **import**
parse_filename, 58
read_folder, 73
select_prefix, 112
select_suffix, 113
∗ **modelling**
calc_importance, 12
calc_nps, 13
ipm_model, 46
∗ **plots**
plot_bars, 60
plot_gauges, 63
plot_ipm, 64
plot_nps, 65
plot_nps_gauge, 66
plot_quotes_tree, 67
plot_rating_grid, 68
plot_stacked_rating, 69
∗ **recode**
add_region, 6
bin_numeric, 10
clean_label, 20

drop_items, 33
ensure_numeric, 35
generation_scheme, 42
na_blank, 53
nps_group, 55
recode_age, 75
recode_generation, 76
recode_likert, 77
recode_region, 78
split_multi, 116
∗ **reporting**
example_report, 36
list_report_templates, 49
report_add_plot, 82
report_add_slide, 83
report_add_table, 85
report_add_text, 86
report_deck, 86
report_layouts, 88
report_new, 89
report_save, 90
report_section, 91
report_slide, 92
report_title_slide, 93
scaffold_report, 105
∗ **save**
export_summary_xlsx, 37
export_xlsx, 39
save_data, 101
save_output, 103
save_plot, 104
∗ **scales**
label_pct, 47
nice_max, 54
scale_y_pct, 109
∗ **summaries**
calc_percentage, 14
calc_percentage_batch, 16
calc_percentage_multi, 17
calc_summary, 19
clear_checkpoints, 22
crosstab, 26
crosstab_banner, 28
∗ **themes**
pal_rating, 57
scale_fill_nps, 107
scale_fill_rating, 108
theme_ezrsurvey, 117
theme_transparent, 118
∗ **weighting**
clear_weights_cache, 22
get_weights, 45
set_weights, 114
weight_vector, 122

add_currency, 4
add_currency(), *25*, *31*, *48*
add_region, 6
add_region(), *11*, *21*, *25, 26*, *34, 35*, *43*, *53*, *55*, *76–79*, *117*
annotate_bands, 7
annotate_bands(), *9, 10*, *52*, *61*, *95*
apply_order, 8
apply_order(), *34*, *41, 42*, *44, 45*, *49, 50*, *56*, *80–82*, *95*, *102*, *121, 122*
as.character(), *10*

bands_nps *(*bands_rating_3*)*, 9
bands_nps(), *7*
bands_nps_score *(*bands_rating_3*)*, 9
bands_rating_3, 9
bands_rating_3(), *7, 8*, *52*, *58*, *64, 65*, *94, 95*, *108*
bands_rating_5 *(*bands_rating_3*)*, 9
bands_rating_5(), *7*
base::cut(), *10*
bin_numeric, 10
bin_numeric(), *6*, *21*, *34, 35*, *43*, *53*, *55*, *75–79*, *117*
brand_info, 11
brand_info(), *21*, *57*, *107*, *120*

calc_importance, 12
calc_importance(), *14*, *46, 47*
calc_nps, 13
calc_nps(), *13*, *35*, *47*, *55*, *64*, *66, 67*, *114*
calc_percentage, 14
calc_percentage(), *13*, *16–20*, *22–24*, *27*, *30*, *33, 34*, *37, 38*, *40*, *47*, *56*, *60, 61*, *70*, *80*, *109*, *114*, *123*
calc_percentage_batch, 16
calc_percentage_batch(), *16*, *19–22*, *27*, *30*, *33*, *38*, *69*, *121*
calc_percentage_multi, 17
calc_percentage_multi(), *16, 17*, *20–22*, *27*, *30*, *33*, *37*, *40*, *60*, *80*, *112*, *116, 117*

calc_summary, 19
calc_summary(), *16, 17*, *19*, *22*, *27*, *30*, *35*, *37*, *78*, *114*
clean_label, 20
clean_label(), *6*, *11*, *17*, *34, 35*, *43*, *53*, *55*, *76–79*, *117*
clear_brand, 21
clear_brand(), *11*, *57*, *107*, *120*
clear_checkpoints, 22
clear_checkpoints(), *16, 17*, *19, 20*, *27*, *29, 30*
clear_dataset *(*get_dataset*)*, 43
clear_dataset(), *121*
clear_weights *(*get_weights*)*, 45
clear_weights(), *23*, *114*
clear_weights_cache, 22
clear_weights_cache(), *22*, *45*, *114*, *123*
compare_values, 23
compare_values(), *27*, *47*, *62, 63*, *65*
convert_currency, 24
convert_currency(), *4, 5*, *31*, *48*
country_region, *6*, 25, *31*, *72*, *78, 79*, *116*
crosstab, 26
crosstab(), *16, 17*, *19, 20*, *22*, *28–30*, *33*, *80*, *114*, *116*
crosstab_banner, 28
crosstab_banner(), *16, 17*, *19, 20*, *22*, *27*, *41*
currency_rates, *24–26*, 31, *48*, *72*, *116*

dataset_vars *(*get_dataset*)*, 43
diagnose, 32
diagnose(), *20*, *51*, *73*, *96, 97*, *110, 111*, *121*
drop_items, 33
drop_items(), *6*, *11*, *15*, *18*, *21*, *27*, *29*, *35*, *40*, *43*, *53*, *55*, *76–79*, *117*

edit_ezrsurvey_profile, 34
edit_ezrsurvey_profile(), *9*, *41, 42*, *44, 45*, *49, 50*, *56*, *80–82*, *95*, *102*, *121, 122*
ensure_numeric, 35
ensure_numeric(), *6*, *10, 11*, *14*, *20, 21*, *24, 25*, *34*, *38*, *43*, *53*, *55*, *66*, *74–79*, *117*
example_report, 36
example_report(), *49*, *83–88*, *90–94*, *106*
export_summary_xlsx, 37
export_summary_xlsx(), *30*, *39*, *101*, *103*, *105*
export_xlsx, 39
export_xlsx(), *17*, *38*, *101*, *103*, *105*
ezrsurvey_options, 40
ezrsurvey_options(), *9*, *15*, *34*, *42*, *44, 45*, *49, 50*, *53*, *56*, *60, 61*, *75–77*, *80–82*, *95*, *102*, *109*, *120–122*
ezrsurvey_profile_paths, 42
ezrsurvey_profile_paths(), *9*, *34*, *41*, *44, 45*, *49, 50*, *56*, *80–82*, *95*, *102*, *121, 122*

flextable, *29*
formatC(), *47*

generation_scheme, 42
generation_scheme(), *6*, *11*, *21*, *34, 35*, *53*, *55*, *76–79*, *117*
geom_text(), *47*
get_dataset, 43
get_dataset(), *9*, *34*, *41, 42*, *45*, *49, 50*, *56*, *80–82*, *95*, *102*, *121, 122*
get_order, 44
get_order(), *9*, *34*, *41, 42*, *44*, *49, 50*, *56*, *80–82*, *95*, *102*, *121, 122*
get_weights, 45
get_weights(), *23*, *114*, *123*
ggplot2::annotate(), *52*
ggplot2::discrete_scale(), *106*
ggplot2::ggsave(), *104*
ggplot2::scale_colour_manual(), *108*
ggplot2::scale_fill_manual(), *107, 108*
ggplot2::scale_y_continuous(), *109*
ggplot2::theme(), *118*
grDevices::colorRampPalette(), *57*

has_dataset *(*get_dataset*)*, 43
has_dataset(), *121*
has_weights *(*get_weights*)*, 45

ipm_model, 46
ipm_model(), *13, 14*, *17*, *20*, *23, 24*, *35*, *64, 65*, *78*

label_pct, 47
label_pct(), *54*, *109, 110*
list_currencies, 48
list_currencies(), *5*, *25*, *31*
list_orders, 48
list_orders(), *9*, *34*, *41, 42*, *44, 45*, *50*, *56*, *80–82*, *95*, *102*, *121, 122*

list_report_templates, 49
list_report_templates(), *36*, *83–88*, *90–94*, *105*, *106*
load_ezrsurvey_profile, 50
load_ezrsurvey_profile(), *9*, *34*, *41*, *42*, *44*, *45*, *49*, *56*, *80–82*, *95*, *102*, *121*, *122*

margin_of_error, 51
margin_of_error(), *32*, *33*, *73*, *96*, *97*, *110*, *111*
mark_value, 52
mark_value(), *8*, *10*, *95*
mean(), *27*

na_blank, 53
na_blank(), *6*, *11*, *15*, *21*, *33–35*, *40*, *43*, *55*, *76–79*, *114*, *117*
nice_max, 54
nice_max(), *40*, *47*, *60*, *61*, *109*, *110*
nps_group, 55
nps_group(), *6*, *11*, *13*, *14*, *21*, *34*, *35*, *43*, *53*, *76–79*, *107*, *108*, *117*

officer::body_add_par(), *86*
options(), *41*
order_for, 56
order_for(), *8*, *9*, *34*, *41*, *42*, *44*, *45*, *49*, *50*, *80–82*, *95*, *102*, *121*, *122*

pal_brand, 56
pal_brand(), *11*, *21*, *106*, *107*, *120*
pal_neutral, *61*
pal_neutral *(*pal_rating*)*, 57
pal_nps, *57*, *107*, *108*, *120*
pal_nps *(*pal_rating*)*, 57
pal_rating, 57, *57*, *108*, *109*, *118–120*
pal_sequential_blue, *56*, *107*
pal_sequential_blue *(*pal_rating*)*, 57
parse_filename, 58
parse_filename(), *74*, *75*, *112*, *113*
plot_bars, 60
plot_bars(), *19*, *37*, *40*, *64–70*, *87*, *109*, *110*, *119*
plot_diff, 62
plot_diff(), *23*, *24*
plot_gauges, 63
plot_gauges(), *61*, *65–70*
plot_ipm, 64
plot_ipm(), *12*, *13*, *46*, *47*, *61*, *64*, *66–70*
plot_nps, 65
plot_nps(), *14*, *55*, *61*, *64*, *65*, *67–70*, *107*, *108*
plot_nps_gauge, 66
plot_nps_gauge(), *14*, *61*, *64–66*, *68–70*
plot_quotes_tree, 67
plot_quotes_tree(), *61*, *64–67*, *69*, *70*, *98–100*
plot_rating_grid, 68
plot_rating_grid(), *61*, *64–68*, *70*
plot_stacked_rating, 69
plot_stacked_rating(), *61*, *64–69*
podracing_survey, *26*, *31*, 71, *115*, *116*
precision_summary, 72
precision_summary(), *32*, *33*, *51*, *96*, *97*, *110*, *111*

randomForest::randomForest(), *12*
read_folder, 73
read_folder(), *6*, *59*, *112*, *113*
readr::locale(), *74*
readr::read_csv(), *74*
readr::write_csv(), *101*
readr::write_tsv(), *101*
recode_age, 75
recode_age(), *6*, *11*, *21*, *34*, *35*, *40*, *43*, *53*, *55*, *76–79*, *117*
recode_generation, 76
recode_generation(), *6*, *11*, *21*, *34*, *35*, *40*, *42*, *43*, *53*, *55*, *75*, *76*, *78*, *79*, *117*
recode_likert, 77
recode_likert(), *6*, *11*, *12*, *21*, *34*, *35*, *43*, *46*, *53*, *55*, *69*, *76*, *77*, *79*, *117*
recode_region, 78
recode_region(), *6*, *11*, *21*, *25*, *26*, *34*, *35*, *43*, *53*, *55*, *76–78*, *117*
recode_subregion *(*recode_region*)*, 78
register_order, 80
register_order(), *8*, *9*, *15*, *16*, *26*, *27*, *30*, *34*, *41*, *42*, *44*, *45*, *49*, *50*, *56*, *69*, *81*, *82*, *95*, *102*, *121*, *122*
register_order_presets, 81
register_order_presets(), *9*, *34*, *41*, *42*, *44*, *45*, *49*, *50*, *56*, *80*, *82*, *95*, *102*, *121*, *122*
remove_order, 82
remove_order(), *9*, *34*, *41*, *42*, *44*, *45*, *49*, *50*, *56*, *80*, *81*, *95*, *102*, *121*, *122*

report_add_plot, 82
report_add_plot(), *36*, *49*, *84–88*, *90–94*, *106*
report_add_slide, 83
report_add_slide(), *36*, *49*, *83*, *85–94*, *106*
report_add_table, 85
report_add_table(), *36*, *49*, *83, 84*, *86–88*, *90–94*, *106*
report_add_text, 86
report_add_text(), *36*, *49*, *83–85*, *87, 88*, *90–94*, *106*
report_deck, 86
report_deck(), *36–38*, *41*, *49*, *83–86*, *88*, *90–94*, *105, 106*, *119*
report_layouts, 88
report_layouts(), *36*, *49*, *83–87*, *90–94*, *106*
report_new, 89
report_new(), *36*, *41*, *49*, *83–88*, *90–94*, *105, 106*, *119, 120*
report_save, 90
report_save(), *36*, *49*, *83–90*, *92–94*, *106*
report_section, 91
report_section(), *36*, *49*, *83–88*, *90, 91*, *93, 94*, *106*
report_slide, 92
report_slide(), *36*, *49*, *83–88*, *90–92*, *94*, *106*
report_title_slide, 93
report_title_slide(), *36*, *49*, *83–88*, *90–93*, *106*
rescale_bands, 94
rescale_bands(), *8*, *10*, *52*
reset_ezrsurvey_options, 95
reset_ezrsurvey_options(), *9*, *34*, *41, 42*, *44, 45*, *49, 50*, *56*, *80–82*, *102*, *121, 122*
rse, 96
rse(), *33*, *51*, *73*, *97*, *110, 111*
rse_rating, 97
rse_rating(), *32, 33*, *51*, *73*, *96*, *110, 111*
rwa::rwa(), *12*

sample_comments, 98
sample_comments(), *68*, *99, 100*
sample_comments_diverse, 99
sample_comments_diverse(), *68*, *99*
save_data, 101
save_data(), *38, 39*, *103*, *105*
save_ezrsurvey_profile, 102
save_ezrsurvey_profile(), *9*, *34*, *41, 42*, *44, 45*, *49, 50*, *56*, *80–82*, *95*, *121, 122*
save_output, 103
save_output(), *38, 39*, *101*, *105*
save_plot, 104
save_plot(), *38, 39*, *101*, *103*, *120*
scaffold_report, 105
scaffold_report(), *36*, *41*, *49*, *83–94*, *119, 120*
scale_color_brand *(*scale_fill_brand*)*, 106
scale_color_rating *(*scale_fill_rating*)*, 108
scale_colour_brand *(*scale_fill_brand*)*, 106
scale_colour_rating *(*scale_fill_rating*)*, 108
scale_fill_brand, 106
scale_fill_brand(), *11*, *21*, *57*, *120*
scale_fill_nps, 107
scale_fill_nps(), *58*, *109*, *118, 119*
scale_fill_rating, 108
scale_fill_rating(), *58*, *70*, *108*, *118, 119*
scale_y_pct, 109
scale_y_pct(), *47*, *54*, *61*
se_mean, 110
se_mean(), *20*, *33*, *51*, *73*, *96, 97*, *111*
se_prop, 111
se_prop(), *33*, *51*, *73*, *96, 97*, *110*
select_prefix, 112
select_prefix(), *59*, *75*, *113*
select_suffix, 113
select_suffix(), *59*, *75*, *112*
set.seed(), *12*
set_weights, 114
set_weights(), *13*, *15*, *19*, *23*, *27*, *29*, *45*, *122, 123*
shopping_survey, *26*, *31*, *72*, 115
split_multi, 116
split_multi(), *6*, *11*, *18, 19*, *21*, *34, 35*, *43*, *53*, *55*, *76–79*
stringr::str_wrap(), *40*, *60*

theme_ezrsurvey, 117
theme_ezrsurvey(), *41*, *58*, *108, 109*, *118, 119*
theme_ezrsurvey_x *(*theme_ezrsurvey*)*, 117

theme_ezrsurvey_xy *(*theme_ezrsurvey*)*, 117
theme_ezrsurvey_y *(*theme_ezrsurvey*)*, 117
theme_transparent, 118
theme_transparent(), *58*, *108, 109*, *118*
tibble, *10*, *13–15*, *17*, *19, 20*, *24, 25*, *27*, *30, 31*, *33*, *43*, *46*, *49*, *71*, *74*, *88*, *99, 100*, *115*
tools::R_user_dir(), *22*, *29*, *121*

use_brand, 119
use_brand(), *11*, *21*, *41*, *56, 57*, *61*, *87–90*, *105–107*, *118*
use_dataset, 120
use_dataset(), *9*, *15*, *28*, *34*, *37*, *41–45*, *49, 50*, *56*, *66*, *69*, *73*, *80–82*, *95*, *102*, *112, 113*, *116*, *122*
use_ezrsurvey_profile, 121
use_ezrsurvey_profile(), *9*, *34*, *40–42*, *44, 45*, *49, 50*, *56*, *80–82*, *95*, *102*, *121*
usethis::edit_r_profile(), *34*

weight_vector, 122
weight_vector(), *23*, *45*, *114*
writexl::write_xlsx(), *101*