Package 'halfmoon'

March 4, 2026

Title Techniques to Build Better Balance

Version 0.2.0

Description Build better balance in causal inference models. 'halfmoon' helps you assess propensity score models for balance between groups using metrics like standardized mean differences and visualization techniques like mirrored histograms. 'halfmoon' supports both weighting and matching techniques.

License MIT + file LICENSE

URL `https://github.com/r-causal/halfmoon`, `https://r-causal.github.io/halfmoon/`

BugReports `https://github.com/r-causal/halfmoon/issues`

Depends R (>= 4.1.0)

Imports cli, dplyr, ggplot2, gtsummary (>= 2.1.0), propensity, purrr, rlang, scales, smd, tibble, tidyr, tidyselect, tidysmd, vctrs

Suggests cards, cardx (>= 0.2.3), cobalt, covr, mgcv, survey, testthat (>= 3.0.0), vdiffr, withr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Malcolm Barrett [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0003-0299-5825`>)

Maintainer Malcolm Barrett <`malcolmbarrett@gmail.com`>

Repository CRAN

Date/Publication 2026-03-04 17:20:02 UTC

Contents

`add_ess_header` *Add ESS Table Header*

Description

This function replaces the counts in the default header of `gtsummary::tbl_svysummary()` tables to counts representing the Effective Sample Size (ESS). See `ess()` for details.

Usage

```
add_ess_header(
  x,
  header = "**{level}**  \nESS = {format(n, digits = 1, nsmall = 1)}"
)
```

Arguments

`x`	(`tbl_svysummary`) Object of class `'tbl_svysummary'` typically created with `gtsummary::tbl_svysummary()`.
`header`	(`string`) String specifying updated header. Review `gtsummary::modify_header()` for details on use.

Value

a 'gtsummary' table

Examples

```
svy <- survey::svydesign(~1, data = nhefs_weights, weights = ~ w_ate)

gtsummary::tbl_svysummary(svy, include = c(age, sex, smokeyrs)) |>
  add_ess_header()
hdr <- paste0(
  "**{level}**  \n",
  "N = {n_unweighted}; ESS = {format(n, digits = 1, nsmall = 1)}"
)
gtsummary::tbl_svysummary(svy, by = qsmk, include = c(age, sex, smokeyrs)) |>
  add_ess_header(header = hdr)
```

`autoplot-halfmoon` *Autoplot Methods for halfmoon Objects*

Description

These methods provide automatic plot generation for halfmoon data objects using ggplot2's autoplot interface. Each method dispatches to the appropriate plot_*() function as follows:

Usage

```
## S3 method for class 'halfmoon_balance'
autoplot(object, ...)

## S3 method for class 'halfmoon_ess'
autoplot(object, ...)

## S3 method for class 'halfmoon_calibration'
autoplot(object, ...)

## S3 method for class 'halfmoon_roc'
autoplot(object, ...)

## S3 method for class 'halfmoon_auc'
autoplot(object, ...)

## S3 method for class 'halfmoon_qq'
autoplot(object, ...)
```

Arguments

`object`	A halfmoon data object with appropriate class
`...`	Additional arguments passed to the underlying plot_*() function

Details

- `autoplot.halfmoon_balance` calls `plot_balance()`
- `autoplot.halfmoon_ess` calls `plot_ess()`
- `autoplot.halfmoon_calibration` calls `plot_model_calibration()`
- `autoplot.halfmoon_roc` calls `plot_model_roc_curve()`
- `autoplot.halfmoon_auc` calls `plot_model_auc()`
- `autoplot.halfmoon_qq` calls `plot_qq()`

Value

A ggplot2 object

bal_corr *Balance Weighted or Unweighted Pearson Correlation*

Description

Calculates the Pearson correlation coefficient between two numeric vectors, with optional case weights. Uses the standard correlation formula for unweighted data and weighted covariance for weighted data.

Usage

```
bal_corr(.x, .y, .weights = NULL, na.rm = FALSE)
```

Arguments

`.x`	A numeric vector containing the first variable.
`.y`	A numeric vector containing the second variable. Must have the same length as `.x`.
`.weights`	An optional numeric vector of case weights. If provided, must have the same length as `.x` and `.y`. All weights must be non-negative.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values result in `NA` output.

Value

A numeric value representing the correlation coefficient between -1 and 1. Returns `NA` if either variable has zero variance.

See Also

`check_balance()` for computing multiple balance metrics at once

Other balance functions: `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
bal_corr(nhefs_weights$age, nhefs_weights$wt71)
```

`bal_energy` *Balance Energy Distance*

Description

Computes the energy distance as a multivariate measure of covariate balance between groups. Energy distance captures the similarity between distributions across the entire joint distribution of .covariates, making it more comprehensive than univariate balance measures.

Usage

```
bal_energy(
  .covariates,
  .exposure,
  .weights = NULL,
  estimand = NULL,
  .focal_level = NULL,
  use_improved = TRUE,
  standardized = TRUE,
  na.rm = FALSE
)
```

Arguments

`.covariates`	A data frame or matrix containing the .covariates to compare.
`.exposure`	A vector (factor or numeric) indicating group membership. For binary and multi-category treatments, must have 2+ unique levels. For continuous treatments, should be numeric.
`.weights`	An optional numeric vector of weights. If provided, must have the same length as rows in `.covariates`. All weights must be non-negative.
`estimand`	Character string specifying the estimand. Options are: • NULL (default): Pure between-group energy distance comparing distributions • "ATE": Energy distance weighted to reflect balance for estimating average treatment effects across the entire population • "ATT": Energy distance weighted to reflect balance for the treated .exposure, measuring how well controls match the treated distribution • "ATC": Energy distance weighted to reflect balance for the control .exposure, measuring how well treated units match the control distribution For continuous treatments, only NULL is supported.
`.focal_level`	The treatment level for ATT/ATC. If `NULL` (default), automatically determined based on estimand.
`use_improved`	Logical. Use improved energy distance for ATE? Default is TRUE. When TRUE, adds pairwise treatment comparisons for better group separation.

`standardized`	Logical. For continuous treatments, return standardized distance correlation? Default is TRUE.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values result in an error (energy distance cannot be computed with missing data).

Details

Energy distance is based on the energy statistics framework (Székely & Rizzo, 2004) and implemented following Huling & Mak (2024) and Huling et al. (2024). The calculation uses a quadratic form: $w^T Pw + q^T w + k$, where the components depend on the estimand.

For binary variables in the .covariates, variance is calculated as p(1-p) rather than sample variance to prevent over-weighting.

For continuous treatments, the function uses distance correlation instead of traditional energy distance, measuring independence between treatment and .covariates.

Value

A numeric value representing the energy distance between groups. Lower values indicate better balance, with 0 indicating perfect balance (identical distributions). For continuous treatments, returns the distance correlation coefficient (0 = independence, 1 = perfect dependence).

References

Huling, J. D., & Mak, S. (2024). Energy Balancing of Covariate Distributions. Journal of Causal Inference, 12(1) . Huling, J. D., Greifer, N., & Chen, G. (2023). Independence weights for causal inference with continuous treatments. *Journal of the American Statistical Association*, 0(ja), 1–25. doi:10.1080/01621459.2023.2213485

Székely, G. J., & Rizzo, M. L. (2004). Testing for equal distributions in high dimension. InterStat, 5.

Examples

```
# Binary treatment
bal_energy(
  .covariates = dplyr::select(nhefs_weights, age, wt71, smokeyrs),
  .exposure = nhefs_weights$qsmk
)

# With weights
bal_energy(
  .covariates = dplyr::select(nhefs_weights, age, wt71, smokeyrs),
  .exposure = nhefs_weights$qsmk,
  .weights = nhefs_weights$w_ate
)

# ATT estimand
bal_energy(
  .covariates = dplyr::select(nhefs_weights, age, wt71, smokeyrs),
  .exposure = nhefs_weights$qsmk,
  .weights = nhefs_weights$w_att,
  estimand = "ATT"
)
```

bal_ess *Calculate Effective Sample Size for Single Weight Vector*

Description

Computes the effective sample size (ESS) for a single weighting scheme. This is a wrapper around `ess()` that follows the bal_*() naming convention for API consistency.

Usage

```
bal_ess(.weights, na.rm = FALSE)
```

Arguments

`.weights`	A numeric vector of weights or a single weight column from a data frame.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.

Details

The effective sample size (ESS) is calculated using the classical formula: $ESS = (\sum w)^2 / \sum(w^2)$.

ESS reflects how many observations you would have if all were equally weighted. When weights vary substantially, the ESS can be much smaller than the actual number of observations, indicating that a few observations carry disproportionately large weights.

Diagnostic Value:

- A large discrepancy between ESS and the actual sample size indicates that a few observations carry disproportionately large weights
- A small ESS signals that weighted estimates are more sensitive to a handful of observations, inflating the variance and standard errors
- If ESS is much lower than the total sample size, consider investigating why some weights are extremely large or small

Value

A single numeric value representing the effective sample size.

See Also

ess() for the underlying implementation, check_ess() for computing ESS across multiple weighting schemes

Other balance functions: bal_corr(), bal_ks(), bal_model_auc(), bal_model_roc_curve(), bal_qq(), bal_smd(), bal_vr(), check_balance(), check_ess(), check_model_auc(), check_model_roc_curve(), check_qq(), plot_balance()

Examples

```
# ESS for ATE weights
bal_ess(nhefs_weights$w_ate)

# ESS for ATT weights
bal_ess(nhefs_weights$w_att)

# With missing values
weights_with_na <- nhefs_weights$w_ate
weights_with_na[1:5] <- NA
bal_ess(weights_with_na, na.rm = TRUE)
```

---

bal_ks *Balance Kolmogorov-Smirnov (KS) Statistic for Two Groups*

Description

Computes the two-sample KS statistic comparing empirical cumulative distribution functions (CDFs) between two groups. For binary variables, returns the absolute difference in proportions. For continuous variables, computes the maximum difference between empirical CDFs.

Usage

```
bal_ks(
  .covariate,
  .exposure,
  .weights = NULL,
  .reference_level = NULL,
  na.rm = FALSE
)
```

Arguments

.covariate	A numeric vector containing the covariate values to compare.
.exposure	A vector (factor or numeric) indicating group membership. Must have exactly two unique levels.
.weights	An optional numeric vector of case weights. If provided, must have the same length as other input vectors. All weights must be non-negative.

`.reference_level`
: The reference group level for comparisons. Can be either a group level value or a numeric index. If `NULL` (default), uses the first level.

`na.rm`
: A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.

Details

The Kolmogorov-Smirnov statistic measures the maximum difference between empirical cumulative distribution functions of two groups:

$$KS = \max_x |F_1(x) - F_0(x)|$$

where $F_1(x)$ and $F_0(x)$ are the empirical CDFs of the treatment and control groups.

For binary variables, this reduces to the absolute difference in proportions. For continuous variables, the statistic captures differences in the entire distribution shape, not just means or variances.

The KS statistic ranges from 0 (identical distributions) to 1 (completely separate distributions). Smaller values indicate better distributional balance between groups.

Value

A numeric value representing the KS statistic. Values range from 0 to 1, with 0 indicating identical distributions and 1 indicating completely separate distributions.

See Also

`check_balance()` for computing multiple balance metrics at once

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Binary exposure
bal_ks(nhefs_weights$age, nhefs_weights$qsmk)

# With weights
bal_ks(nhefs_weights$wt71, nhefs_weights$qsmk,
       .weights = nhefs_weights$w_ate)

# Categorical exposure (returns named vector)
bal_ks(nhefs_weights$age, nhefs_weights$alcoholfreq_cat)

# Specify reference level
bal_ks(nhefs_weights$age, nhefs_weights$alcoholfreq_cat,
       .reference_level = "none")

# With categorical weights
bal_ks(nhefs_weights$wt71, nhefs_weights$alcoholfreq_cat,
       .weights = nhefs_weights$w_cat_ate)
```

`bal_model_auc` *Calculate Single AUC for Model Balance Assessment*

Description

Computes the Area Under the ROC Curve (AUC) for a single weighting scheme or unweighted data. In causal inference, an AUC around 0.5 indicates good balance between treatment groups.

Usage

```
bal_model_auc(
  .data,
  .exposure,
  .fitted,
  .weights = NULL,
  na.rm = TRUE,
  .focal_level = NULL
)
```

Arguments

`.data`	A data frame containing the variables.
`.exposure`	The treatment/outcome variable (unquoted).
`.fitted`	The propensity score or fitted values (unquoted).
`.weights`	Optional single weight variable (unquoted). If NULL, computes unweighted AUC.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.

Details

The AUC provides a single metric for assessing propensity score balance. When propensity scores achieve perfect balance, the weighted distribution of scores should be identical between treatment groups, resulting in an AUC of 0.5 (chance performance).

AUC values significantly different from 0.5 indicate systematic differences in propensity score distributions between groups, suggesting inadequate balance.

Value

A numeric value representing the AUC. Values around 0.5 indicate good balance, while values closer to 0 or 1 indicate poor balance.

See Also

check_model_auc() for computing AUC across multiple weights, bal_model_roc_curve() for the full ROC curve

Other balance functions: bal_corr(), bal_ess(), bal_ks(), bal_model_roc_curve(), bal_qq(), bal_smd(), bal_vr(), check_balance(), check_ess(), check_model_auc(), check_model_roc_curve(), check_qq(), plot_balance()

Examples

```
# Unweighted AUC
bal_model_auc(nhefs_weights, qsmk, .fitted)

# Weighted AUC
bal_model_auc(nhefs_weights, qsmk, .fitted, w_ate)
```

bal_model_roc_curve *Calculate Single ROC Curve for Model Balance Assessment*

Description

Computes the Receiver Operating Characteristic (ROC) curve for a single weighting scheme or unweighted data. In causal inference, an ROC curve near the diagonal indicates good balance between treatment groups.

Usage

```
bal_model_roc_curve(
  .data,
  .exposure,
  .fitted,
  .weights = NULL,
  na.rm = TRUE,
  .focal_level = NULL
)
```

Arguments

.data	A data frame containing the variables.
.exposure	The treatment/outcome variable (unquoted).
.fitted	The propensity score or fitted values (unquoted).
.weights	Optional single weight variable (unquoted). If NULL, computes unweighted ROC curve.
na.rm	A logical value indicating whether to remove missing values before computation. If FALSE (default), missing values in the input will produce NA in the output.

`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.

Details

The ROC curve plots sensitivity (true positive rate) against 1-specificity (false positive rate) across all possible threshold values. When propensity scores achieve perfect balance, the ROC curve should lie close to the diagonal line from (0,0) to (1,1), indicating that the propensity scores have no discriminatory ability between treatment groups.

ROC curves that bow significantly above the diagonal indicate that the propensity scores can still distinguish between treatment groups, suggesting inadequate balance.

Value

A tibble with columns:

`threshold`	Numeric. The decision threshold.
`sensitivity`	Numeric. True positive rate at the threshold.
`specificity`	Numeric. True negative rate at the threshold.

See Also

`check_model_roc_curve()` for computing ROC curves across multiple weights, `bal_model_auc()` for the area under the curve summary

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Unweighted ROC curve
bal_model_roc_curve(nhefs_weights, qsmk, .fitted)

# Weighted ROC curve
bal_model_roc_curve(nhefs_weights, qsmk, .fitted, w_ate)
```

`bal_prognostic_score` *Compute Prognostic Scores for Balance Assessment*

Description

Calculates prognostic scores by fitting an outcome model on the control group and generating predictions for all observations. Prognostic scores represent the expected outcome under control conditions and are useful for assessing whether balance on treatment predictors ensures balance on outcome-relevant variables.

Usage

```
bal_prognostic_score(
  .data,
  outcome = NULL,
  .covariates = everything(),
  .exposure,
  formula = NULL,
  .reference_level = NULL,
  family = gaussian(),
  .weights = NULL,
  na.rm = FALSE,
  ...
)
```

Arguments

`.data`	A data frame containing the variables for analysis
`outcome`	The outcome variable. Can be specified as a bare name or character string. Ignored if `formula` is provided.
`.covariates`	Variables to include in the outcome model. Defaults to `everything()` which includes all variables except the outcome and treatment. Supports tidyselect syntax. Ignored if `formula` is provided.
`.exposure`	The treatment variable. Can be specified as a bare name or character string.
`formula`	Optional formula for the outcome model. If provided, `outcome` and `.covariates` arguments are ignored. The formula should not include the treatment variable.
`.reference_level`	The level of treatment that represents the control group. If NULL (default), the first level is used as control.
`family`	A family object for the GLM. Defaults to `gaussian()` for continuous outcomes. Use `binomial()` for binary outcomes.
`.weights`	Optional weights for the outcome model. Can be a numeric vector or bare name of a variable in `.data`.
`na.rm`	Logical. Should missing values be removed? Defaults to FALSE.
`...`	Additional arguments passed to `glm()`.

Details

The prognostic score method, introduced by Stuart et al. (2013), provides a way to assess balance that focuses on variables predictive of the outcome rather than just the treatment assignment. The procedure:

1. Fits an outcome model using only control group observations
2. Generates predictions (prognostic scores) for all observations
3. Returns these scores for balance assessment

This approach is particularly useful when:

- The outcome model includes non-linearities or interactions
- You want to ensure balance on outcome-relevant variables
- Traditional propensity score balance checks may miss important imbalances

The returned prognostic scores can be used with existing balance functions like `bal_smd()`, `bal_vr()`, or `check_balance()` to assess balance between treatment groups.

Value

A numeric vector of prognostic scores with the same length as the number of rows in `.data` (after NA removal if `na.rm = TRUE`).

References

Stuart EA, Lee BK, Leacy FP. Prognostic score-based balance measures can be a useful diagnostic for propensity score methods in comparative effectiveness research. J Clin Epidemiol. 2013;66(8):S84-S90.

Examples

```
# Using tidyselect interface
prog_scores <- bal_prognostic_score(
  nhefs_weights,
  outcome = wt82_71,
  .exposure = qsmk,
  .covariates = c(age, sex, race, wt71)
)

# Using formula interface
prog_scores_formula <- bal_prognostic_score(
  nhefs_weights,
  .exposure = qsmk,
  formula = wt82_71 ~ age + sex + race + wt71 + I(age^2)
)

# Add to data and check balance
nhefs_with_prog <- nhefs_weights
nhefs_with_prog$prog_score <- prog_scores
check_balance(nhefs_with_prog, prog_score, qsmk, .weights = w_ate)

# For binary outcome
prog_scores_binary <- bal_prognostic_score(
  nhefs_weights,
  outcome = death,
  .exposure = qsmk,
  family = binomial()
)
```

bal_qq *Compute QQ Data for Single Variable and Weight*

Description

Calculate quantile-quantile data comparing the distribution of a variable between treatment groups for a single weighting scheme (or unweighted). This function computes the quantiles for both groups and returns a data frame suitable for plotting or further analysis.

Usage

```
bal_qq(
  .data,
  .var,
  .exposure,
  .weights = NULL,
  quantiles = seq(0.01, 0.99, 0.01),
  .reference_level = NULL,
  na.rm = FALSE
)
```

Arguments

`.data`	A data frame containing the variables.
`.var`	Variable to compute quantiles for (unquoted).
`.exposure`	Column name of treatment/group variable (unquoted).
`.weights`	Optional single weight variable (unquoted). If NULL, computes unweighted quantiles.
`quantiles`	Numeric vector of quantiles to compute. Default is `seq(0.01, 0.99, 0.01)` for 99 quantiles.
`.reference_level`	The reference group level for comparisons. Can be either a group level value or a numeric index. If `NULL` (default), uses the first level.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.

Details

This function computes the data needed for quantile-quantile plots by calculating corresponding quantiles from two distributions. The computation uses the inverse of the empirical cumulative distribution function (ECDF). For weighted data, it first computes the weighted ECDF and then inverts it to obtain quantiles.

When the distributions of a variable are similar between treatment groups (indicating good balance), the QQ plot points will lie close to the diagonal line y = x.

Value

A tibble with columns:

`quantile`	Numeric. The quantile probability (0-1).
`exposed_quantiles`	Numeric. The quantile value for the exposed group.
`unexposed_quantiles`	Numeric. The quantile value for the unexposed group.

See Also

`check_qq()` for computing QQ data across multiple weights, `plot_qq()` for visualization

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Unweighted QQ data
bal_qq(nhefs_weights, age, qsmk)

# Weighted QQ data
bal_qq(nhefs_weights, age, qsmk, .weights = w_ate)

# Custom quantiles
bal_qq(nhefs_weights, age, qsmk, .weights = w_ate,
       quantiles = seq(0.1, 0.9, 0.1))
```

`bal_smd` *Balance Standardized Mean Difference (SMD)*

Description

Calculates the standardized mean difference between two groups using the smd package. This is a common measure of effect size for comparing group differences while accounting for variability.

Usage

```
bal_smd(
  .covariate,
  .exposure,
  .weights = NULL,
  .reference_level = NULL,
  na.rm = FALSE
)
```

Arguments

`.covariate`	A numeric vector containing the covariate values to compare.
`.exposure`	A vector (factor or numeric) indicating group membership. Must have exactly two unique levels.
`.weights`	An optional numeric vector of case weights. If provided, must have the same length as other input vectors. All weights must be non-negative.
`.reference_level`	The reference group level for comparisons. Can be either a group level value or a numeric index. If `NULL` (default), uses the first level.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.

Details

The standardized mean difference (SMD) is calculated as:

$$SMD = \frac{\bar{x}_1 - \bar{x}_0}{\sqrt{(s_1^2 + s_0^2)/2}}$$

where $\bar{x}_1$ and $\bar{x}_0$ are the means of the treatment and control groups, and s_1^2 and s_0^2 are their variances.

In causal inference, SMD values of 0.1 or smaller are often considered indicative of good balance between treatment groups.

Value

A numeric value representing the standardized mean difference. Positive values indicate the comparison group has a higher mean than the reference group.

See Also

`check_balance()` for computing multiple balance metrics at once

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Binary exposure
bal_smd(nhefs_weights$age, nhefs_weights$qsmk)

# With weights
bal_smd(nhefs_weights$wt71, nhefs_weights$qsmk,
        .weights = nhefs_weights$w_ate)

# Categorical exposure (returns named vector)
bal_smd(nhefs_weights$age, nhefs_weights$alcoholfreq_cat)
```

```
# Specify reference level
bal_smd(nhefs_weights$age, nhefs_weights$alcoholfreq_cat,
        .reference_level = "daily")

# With categorical weights
bal_smd(nhefs_weights$wt71, nhefs_weights$alcoholfreq_cat,
        .weights = nhefs_weights$w_cat_ate)
```

---

bal_vr *Balance Variance Ratio for Two Groups*

Description

Calculates the ratio of variances between two groups: var(comparison) / var(reference). For binary variables, uses the p*(1-p) variance formula. For continuous variables, uses Bessel's correction for weighted sample variance.

Usage

```
bal_vr(
  .covariate,
  .exposure,
  .weights = NULL,
  .reference_level = NULL,
  na.rm = FALSE
)
```

Arguments

`.covariate`	A numeric vector containing the covariate values to compare.
`.exposure`	A vector (factor or numeric) indicating group membership. Must have exactly two unique levels.
`.weights`	An optional numeric vector of case weights. If provided, must have the same length as other input vectors. All weights must be non-negative.
`.reference_level`	The reference group level for comparisons. Can be either a group level value or a numeric index. If `NULL` (default), uses the first level.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.

Details

The variance ratio compares the variability of a covariate between treatment groups. It is calculated as:

$$VR = \frac{s_1^2}{s_0^2}$$

where s_1^2 and s_0^2 are the variances of the treatment and control groups.

For binary variables (0/1), variance is computed as $p(1-p)$ where p is the proportion of 1s in each group. For continuous variables, the weighted sample variance is used with Bessel's correction when weights are provided.

Values close to 1.0 indicate similar variability between groups, which is desirable for balance. Values substantially different from 1.0 suggest imbalanced variance.

Value

A numeric value representing the variance ratio. Values greater than 1 indicate the comparison group has higher variance than the reference group.

See Also

`check_balance()` for computing multiple balance metrics at once

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Binary exposure
bal_vr(nhefs_weights$age, nhefs_weights$qsmk)

# With weights
bal_vr(nhefs_weights$wt71, nhefs_weights$qsmk,
       .weights = nhefs_weights$w_ate)

# Categorical exposure (returns named vector)
bal_vr(nhefs_weights$age, nhefs_weights$alcoholfreq_cat)

# Specify reference level
bal_vr(nhefs_weights$age, nhefs_weights$alcoholfreq_cat,
       .reference_level = "2_3_per_week")

# With categorical weights
bal_vr(nhefs_weights$wt71, nhefs_weights$alcoholfreq_cat,
       .weights = nhefs_weights$w_cat_ate)
```

`check_balance` *Check Balance Across Multiple Metrics*

Description

Computes balance statistics for multiple variables across different groups and optional weighting schemes. This function generalizes balance checking by supporting multiple metrics (SMD, variance ratio, Kolmogorov-Smirnov, weighted correlation) and returns results in a tidy format.

Usage

```
check_balance(
  .data,
  .vars,
  .exposure,
  .weights = NULL,
  .metrics = c("smd", "vr", "ks", "energy"),
  include_observed = TRUE,
  .reference_level = 1L,
  na.rm = FALSE,
  make_dummy_vars = TRUE,
  squares = FALSE,
  cubes = FALSE,
  interactions = FALSE
)
```

Arguments

`.data`	A data frame containing the variables to analyze.
`.vars`	Variables for which to calculate metrics. Can be unquoted variable names, a character vector, or a tidyselect expression.
`.exposure`	Grouping variable, e.g., treatment or exposure group.
`.weights`	Optional weighting variables. Can be unquoted variable names, a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes.
`.metrics`	Character vector specifying which metrics to compute. Available options: "smd" (standardized mean difference), "vr" (variance ratio), "ks" (Kolmogorov-Smirnov), "correlation" (for continuous exposures), "energy" (multivariate energy distance). Defaults to c("smd", "vr", "ks", "energy").
`include_observed`	Logical. If using `.weights`, also calculate observed (unweighted) metrics? Defaults to TRUE.
`.reference_level`	The reference group level to use for comparisons. Defaults to 1 (first level).

`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.
`make_dummy_vars`	Logical. Transform categorical variables to dummy variables using `model.matrix()`? Defaults to TRUE. When TRUE, categorical variables are expanded into separate binary indicators for each level.
`squares`	Logical. Include squared terms for continuous variables? Defaults to FALSE. When TRUE, adds squared versions of numeric variables.
`cubes`	Logical. Include cubed terms for continuous variables? Defaults to FALSE. When TRUE, adds cubed versions of numeric variables.
`interactions`	Logical. Include all pairwise interactions between variables? Defaults to FALSE. When TRUE, creates interaction terms for all variable pairs, excluding interactions between levels of the same categorical variable and between squared/cubed terms.

Details

This function serves as a comprehensive balance assessment tool by computing multiple balance metrics simultaneously. It automatically handles different variable types and can optionally transform variables (dummy coding, polynomial terms, interactions) before computing balance statistics.

The function supports several balance metrics:

- **SMD (Standardized Mean Difference)**: Measures effect size between groups, with values around 0.1 or smaller generally indicating good balance
- **Variance Ratio**: Compares group variances, with values near 1.0 indicating similar variability between groups
- **Kolmogorov-Smirnov**: Tests distributional differences between groups, with smaller values indicating better balance
- **Correlation**: For continuous exposures, measures linear association between covariate and exposure
- **Energy Distance**: Multivariate test comparing entire distributions

When multiple weighting schemes are provided, the function computes balance for each method, enabling comparison of different approaches (e.g., ATE vs ATT weights). The `include_observed` parameter controls whether unweighted ("observed") balance is included in the results.

Value

A tibble with columns:

`variable`	Character. The variable name being analyzed.
`group_level`	Character. The non-reference group level.
`method`	Character. The weighting method ("observed" or weight variable name).
`metric`	Character. The balance metric computed ("smd", "vr", "ks").
`estimate`	Numeric. The computed balance statistic.

See Also

bal_smd(), bal_vr(), bal_ks(), bal_corr(), bal_energy() for individual metric functions, plot_balance() for visualization

Other balance functions: bal_corr(), bal_ess(), bal_ks(), bal_model_auc(), bal_model_roc_curve(), bal_qq(), bal_smd(), bal_vr(), check_ess(), check_model_auc(), check_model_roc_curve(), check_qq(), plot_balance()

Examples

```
# Basic usage with binary exposure
check_balance(nhefs_weights, c(age, wt71), qsmk, .weights = c(w_ate, w_att))

# With specific metrics only
check_balance(nhefs_weights, c(age, wt71), qsmk, .metrics = c("smd", "energy"))

# Categorical exposure
check_balance(nhefs_weights, c(age, wt71), alcoholfreq_cat,
              .weights = c(w_cat_ate, w_cat_att_2_3wk))

# Specify reference group for categorical exposure
check_balance(nhefs_weights, c(age, wt71, sex), alcoholfreq_cat,
              .reference_level = "daily", .metrics = c("smd", "vr"))

# Exclude observed results
check_balance(nhefs_weights, c(age, wt71), qsmk, .weights = w_ate,
              include_observed = FALSE)

# Use correlation for continuous exposure
check_balance(mtcars, c(mpg, hp), disp, .metrics = c("correlation", "energy"))

# With dummy variables for categorical variables (default behavior)
check_balance(nhefs_weights, c(age, sex, race), qsmk)

# Without dummy variables for categorical variables
check_balance(nhefs_weights, c(age, sex, race), qsmk, make_dummy_vars = FALSE)
```

check_ess *Check Effective Sample Size*

Description

Computes the effective sample size (ESS) for one or more weighting schemes, optionally stratified by treatment groups. ESS reflects how many observations you would have if all were equally weighted.

Usage

```
check_ess(
  .data,
  .weights = NULL,
  .exposure = NULL,
  include_observed = TRUE,
  n_tiles = 4,
  tile_labels = NULL
)
```

Arguments

`.data`	A data frame containing the variables to analyze.
`.weights`	Optional weighting variables. Can be unquoted variable names, a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes.
`.exposure`	Optional exposure variable. When provided, ESS is calculated separately for each exposure level. For continuous variables, groups are created using quantiles.
`include_observed`	Logical. If using `.weights`, also calculate observed (unweighted) metrics? Defaults to TRUE.
`n_tiles`	For continuous `.exposure` variables, the number of quantile groups to create. Default is 4 (quartiles).
`tile_labels`	Optional character vector of labels for the quantile groups when `.exposure` is continuous. If NULL, uses "Q1", "Q2", etc.

Details

The effective sample size (ESS) is calculated using the classical formula: $ESS = (\sum w)^2 / \sum(w^2)$.

When weights vary substantially, the ESS can be much smaller than the actual number of observations, indicating that a few observations carry disproportionately large weights.

When `.exposure` is provided, ESS is calculated separately for each exposure level:

- For binary/categorical exposures: ESS is computed within each treatment level
- For continuous exposures: The variable is divided into quantiles (using `dplyr::ntile()`) and ESS is computed within each quantile

The function returns results in a tidy format suitable for plotting or further analysis.

Value

A tibble with columns:

`method`	Character. The weighting method ("observed" or weight variable name).
`group`	Character. The exposure level (if `.exposure` is provided).
`n`	Integer. The number of observations in the group.

`ess`	Numeric. The effective sample size.
`ess_pct`	Numeric. ESS as a percentage of the actual sample size.

See Also

`ess()` for the underlying ESS calculation, `plot_ess()` for visualization

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Overall ESS for different weighting schemes
check_ess(nhefs_weights, .weights = c(w_ate, w_att, w_atm))

# ESS by treatment group (binary exposure)
check_ess(nhefs_weights, .weights = c(w_ate, w_att), .exposure = qsmk)

# ESS by treatment group (categorical exposure)
check_ess(nhefs_weights, .weights = w_cat_ate, .exposure = alcoholfreq_cat)

# ESS by quartiles of a continuous variable
check_ess(nhefs_weights, .weights = w_ate, .exposure = age, n_tiles = 4)

# Custom labels for continuous groups
check_ess(nhefs_weights, .weights = w_ate, .exposure = age,
          n_tiles = 3, tile_labels = c("Young", "Middle", "Older"))

# Without unweighted comparison
check_ess(nhefs_weights, .weights = w_ate, .exposure = qsmk,
          include_observed = FALSE)
```

`check_model_auc` *Check Balance Using Weighted ROC Curves*

Description

Computes weighted ROC curves and AUC for evaluating propensity score balance. In causal inference, a weighted ROC curve near the diagonal (AUC around 0.5) indicates good balance between treatment groups.

Usage

```
check_model_auc(
  .data,
  .exposure,
  .fitted,
  .weights,
  include_observed = TRUE,
  na.rm = TRUE,
  .focal_level = NULL
)
```

Arguments

`.data`	A data frame containing the variables.
`.exposure`	The treatment/outcome variable.
`.fitted`	The propensity score or fitted values.
`.weights`	Weighting variables (supports tidyselect).
`include_observed`	Logical. If using `.weights`, also calculate observed (unweighted) metrics? Defaults to TRUE.
`na.rm`	A logical value indicating whether to remove missing values before computation. If `FALSE` (default), missing values in the input will produce `NA` in the output.
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.

Details

The Area Under the ROC Curve (AUC) provides a single metric for assessing propensity score balance. When propensity scores achieve perfect balance, the weighted distribution of scores should be identical between treatment groups, resulting in an AUC of 0.5 (chance performance).

AUC values significantly different from 0.5 indicate systematic differences in propensity score distributions between groups, suggesting inadequate balance. Values closer to 0.5 indicate better balance achieved by the weighting scheme.

This approach complements traditional balance diagnostics by focusing specifically on the propensity score overlap and balance.

Value

A tibble with columns:

`method`	Character. The weighting method ("observed" or weight variable name).
`auc`	Numeric. The ROC AUC value.

See Also

`bal_model_auc()` for single AUC values, `plot_model_auc()` for visualization, `check_balance()` for other balance metrics

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_roc_curve()`, `check_qq()`, `plot_balance()`

Examples

```
# Check balance for propensity scores
check_model_auc(nhefs_weights, qsmk, .fitted, c(w_ate, w_att))

# Without observed results
check_model_auc(nhefs_weights, qsmk, .fitted, w_ate, include_observed = FALSE)
```

`check_model_calibration` *Compute calibration data for binary outcomes*

Description

`check_model_calibration()` summarizes predicted probabilities and observed outcomes, computing mean prediction, observed rate, counts, and confidence intervals. Calibration represents the agreement between predicted probabilities and observed outcomes. Supports multiple methods for calibration assessment.

Usage

```
check_model_calibration(
  data,
  .fitted,
  .exposure,
  .focal_level = NULL,
  method = c("breaks", "logistic", "windowed"),
  bins = 10,
  binning_method = c("equal_width", "quantile"),
  smooth = TRUE,
  conf_level = 0.95,
  window_size = 0.1,
  step_size = window_size/2,
  k = 10,
  na.rm = FALSE
)
```

Arguments

`data`	A data frame containing the data.
`.fitted`	Column name of predicted probabilities (numeric between 0 and 1). Can be unquoted (e.g., `p`) or quoted (e.g., `"p"`).
`.exposure`	Column name of treatment/exposure variable. Can be unquoted (e.g., `g`) or quoted (e.g., `"g"`).
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.

`method`	Character; calibration method. One of: "breaks", "logistic", or "windowed".
`bins`	Integer > 1; number of bins for the "breaks" method.
`binning_method`	"equal_width" or "quantile" for bin creation (breaks method only).
`smooth`	Logical; for "logistic" method, use GAM smoothing via the mgcv package.
`conf_level`	Numeric in (0,1); confidence level for CIs (default = 0.95).
`window_size`	Numeric; size of each window for "windowed" method.
`step_size`	Numeric; distance between window centers for "windowed" method.
`k`	Integer; the basis dimension for GAM smoothing when method = "logistic" and smooth = `TRUE`. Default is 10.
`na.rm`	Logical; if `TRUE`, drop `NA` values before summarizing.

Value

A tibble with columns:

- For "breaks" method:
  - `.bin`: integer bin index
  - `predicted_rate`: mean predicted probability in bin
  - `observed_rate`: observed treatment rate in bin
  - `count`: number of observations in bin
  - `lower`: lower bound of CI for `observed_rate`
  - `upper`: upper bound of CI for `observed_rate`
- For "logistic" and "windowed" methods:
  - `predicted_rate`: predicted probability values
  - `observed_rate`: calibrated outcome rate
  - `lower`: lower bound of CI
  - `upper`: upper bound of CI

Examples

```
# Using the included `nhefs_weights` dataset
# `.fitted` contains propensity scores, and `qsmk` is the treatment variable
check_model_calibration(nhefs_weights, .fitted, qsmk)

# Logistic method with smoothing
check_model_calibration(nhefs_weights, .fitted, qsmk, method = "logistic")

# Windowed method
check_model_calibration(nhefs_weights, .fitted, qsmk, method = "windowed")
```

check_model_roc_curve *Check ROC Curves for Multiple Weights*

Description

Computes ROC curves (weighted or unweighted) for evaluating propensity score balance. In causal inference, an ROC curve near the diagonal (AUC around 0.5) indicates good balance between treatment groups.

Usage

```
check_model_roc_curve(
  .data,
  .exposure,
  .fitted,
  .weights = NULL,
  include_observed = TRUE,
  na.rm = TRUE,
  .focal_level = NULL
)
```

Arguments

`.data`	A data frame containing the variables.
`.exposure`	The treatment/outcome variable (unquoted).
`.fitted`	The propensity score or covariate (unquoted).
`.weights`	Optional weighting variables (unquoted, can be multiple).
`include_observed`	Include unweighted results? Default TRUE.
`na.rm`	Remove missing values? Default TRUE.
`.focal_level`	The level of `.exposure` to consider as the treatment/event. Default is NULL, which uses the second level.

Value

A tibble with class "halfmoon_roc" containing ROC curve data.

See Also

`check_model_auc()` for AUC summaries, `bal_model_roc_curve()` for single ROC curves

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_qq()`, `plot_balance()`

Examples

```
# Check ROC curves for propensity scores with multiple weights
roc_data <- check_model_roc_curve(
  nhefs_weights,
  qsmk,
  .fitted,
  c(w_ate, w_att)
)

# Check ROC curve for a single weight without observed
check_model_roc_curve(
  nhefs_weights,
  qsmk,
  .fitted,
  w_ate,
  include_observed = FALSE
)

# Specify a different focal level
check_model_roc_curve(
  nhefs_weights,
  qsmk,
  .fitted,
  w_ate,
  .focal_level = 0  # Use 0 as the treatment level instead of 1
)
```

---

check_qq *Check QQ Data for Multiple Weights*

Description

Calculate quantile-quantile data comparing the distribution of a variable between treatment groups. This function computes the quantiles for both groups and returns a tidy data frame suitable for plotting or further analysis.

Usage

```
check_qq(
  .data,
  .var,
  .exposure,
  .weights = NULL,
  quantiles = seq(0.01, 0.99, 0.01),
  include_observed = TRUE,
  .reference_level = NULL,
  na.rm = FALSE
)
```

Arguments

`.data`	A data frame containing the variables.
`.var`	Variable to compute quantiles for. Supports tidyselect syntax.
`.exposure`	Column name of treatment/group variable. Supports tidyselect syntax.
`.weights`	Optional weighting variable(s). Can be unquoted variable names (supports tidyselect syntax), a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes. Default is NULL (unweighted).
`quantiles`	Numeric vector of quantiles to compute. Default is `seq(0.01, 0.99, 0.01)` for 99 quantiles.
`include_observed`	Logical. If using `.weights`, also compute observed (unweighted) quantiles? Defaults to TRUE.
`.reference_level`	The reference treatment level to use for comparisons. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.
`na.rm`	Logical; if TRUE, drop NA values before computation.

Details

This function computes the data needed for quantile-quantile plots by calculating corresponding quantiles from two distributions. The computation uses the inverse of the empirical cumulative distribution function (ECDF). For weighted data, it first computes the weighted ECDF and then inverts it to obtain quantiles.

Value

A tibble with class "halfmoon_qq" containing columns:

`method`	Character. The weighting method ("observed" or weight variable name).
`quantile`	Numeric. The quantile probability (0-1).
`exposed_quantiles`	Numeric. The quantile value for the exposed group.
`unexposed_quantiles`	Numeric. The quantile value for the unexposed group.

See Also

`bal_qq()` for single weight QQ data, `plot_qq()` for visualization

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `plot_balance()`

Examples

```
# Basic QQ data (observed only)
check_qq(nhefs_weights, age, qsmk)

# With weighting
check_qq(nhefs_weights, age, qsmk, .weights = w_ate)

# Compare multiple weighting schemes
check_qq(nhefs_weights, age, qsmk, .weights = c(w_ate, w_att))
```

`ess` *Calculate the Effective Sample Size (ESS)*

Description

This function computes the effective sample size (ESS) given a vector of weights, using the classical $(\sum w)^2 / \sum(w^2)$ formula (sometimes referred to as "Kish's effective sample size").

Usage

```
ess(wts, na.rm = FALSE)
```

Arguments

`wts`	A numeric vector of weights (e.g., from survey or inverse-probability weighting).
`na.rm`	Logical. Should missing values be removed? Default is FALSE.

Details

The effective sample size (ESS) reflects how many observations you would have if all were equally weighted. If the weights vary substantially, the ESS can be much smaller than the actual number of observations. Formally:

$$\text{ESS} = \frac{\left(\sum_i w_i\right)^2}{\sum_i w_i^2}.$$

Diagnostic Value:

- **Indicator of Weight Concentration**: A large discrepancy between ESS and the actual sample size indicates that a few observations carry disproportionately large weights, effectively reducing the usable information in the dataset.
- **Variance Inflation**: A small ESS signals that weighted estimates are more sensitive to a handful of observations, inflating the variance and standard errors.
- **Practical Guidance**: If ESS is much lower than the total sample size, it is advisable to investigate why some weights are extremely large or small. Techniques like weight trimming or stabilized weights might be employed to mitigate the issue

Value

A single numeric value representing the effective sample size.

Examples

```
# Suppose we have five observations with equal weights
wts1 <- rep(1.2, 5)
# returns 5, because all weights are equal
ess(wts1)

# If weights vary more, smaller than 5
wts2 <- c(0.5, 2, 2, 0.1, 0.8)
ess(wts2)
```

---

geom_calibration *Geom for calibration plot with confidence intervals*

Description

geom_calibration() creates calibration plots to assess the agreement between predicted probabilities and observed binary outcomes. It supports three methods: binning ("breaks"), logistic regression ("logistic"), and windowed ("windowed"), all computed with check_model_calibration().

Usage

```
geom_calibration(
  mapping = NULL,
  data = NULL,
  method = "breaks",
  bins = 10,
  binning_method = "equal_width",
  smooth = TRUE,
  conf_level = 0.95,
  window_size = 0.1,
  step_size = window_size/2,
  .focal_level = NULL,
  k = 10,
  show_ribbon = TRUE,
  show_points = TRUE,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mapping (must supply `.fitted` and `.exposure` if not inherited). `.fitted` should be propensity scores/predicted probabilities, `.exposure` should be treatment variable.
`data`	Data frame or tibble; if NULL, uses ggplot default.
`method`	Character; calibration method - "breaks", "logistic", or "windowed".
`bins`	Integer >1; number of bins for the "breaks" method.
`binning_method`	"equal_width" or "quantile" for bin creation (breaks method only).
`smooth`	Logical; for "logistic" method, use GAM smoothing if available.
`conf_level`	Numeric in (0,1); confidence level for CIs (default = 0.95).
`window_size`	Numeric; size of each window for "windowed" method.
`step_size`	Numeric; distance between window centers for "windowed" method.
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.
`k`	Integer; the basis dimension for GAM smoothing when method = "logistic" and smooth = TRUE. Default is 10.
`show_ribbon`	Logical; show confidence interval ribbon.
`show_points`	Logical; show points (only for "breaks" and "windowed" methods).
`position`	Position adjustment.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.
`...`	Other arguments passed on to layer().

Details

This geom provides a ggplot2 layer for creating calibration plots with confidence intervals. The geom automatically computes calibration statistics using the specified method and renders appropriate geometric elements (points, lines, ribbons) to visualize the relationship between predicted and observed rates.

The three methods offer different approaches to calibration assessment:

- **"breaks"**: Discrete binning approach, useful for understanding calibration across prediction ranges with sufficient sample sizes
- **"logistic"**: Regression-based approach that can include smoothing for continuous calibration curves
- **"windowed"**: Sliding window approach providing smooth curves without requiring additional packages

Value

A ggplot2 layer or list of layers

See Also

`check_model_calibration()` for computing calibration statistics

Other ggplot2 functions: `geom_ecdf()`, `geom_mirror_density()`, `geom_mirror_histogram()`, `geom_qq2()`, `geom_roc()`

Examples

```
library(ggplot2)

# Basic calibration plot using nhefs_weights dataset
# .fitted contains propensity scores, qsmk is the treatment variable
ggplot(nhefs_weights, aes(.fitted = .fitted, .exposure = qsmk)) +
  geom_calibration() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(x = "Propensity Score", y = "Observed Treatment Rate")

# Using different methods
ggplot(nhefs_weights, aes(.fitted = .fitted, .exposure = qsmk)) +
  geom_calibration(method = "logistic") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(x = "Propensity Score", y = "Observed Treatment Rate")

# Specify treatment level explicitly
ggplot(nhefs_weights, aes(.fitted = .fitted, .exposure = qsmk)) +
  geom_calibration(.focal_level = "1") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(x = "Propensity Score", y = "Observed Treatment Rate")

# Windowed method with custom parameters
ggplot(nhefs_weights, aes(.fitted = .fitted, .exposure = qsmk)) +
  geom_calibration(method = "windowed", window_size = 0.2, step_size = 0.1) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(x = "Propensity Score", y = "Observed Treatment Rate")
```

`geom_ecdf` *Calculate weighted and unweighted empirical cumulative distributions*

Description

The empirical cumulative distribution function (ECDF) provides an alternative visualization of distribution. `geom_ecdf()` is similar to `ggplot2::stat_ecdf()` but it can also calculate weighted ECDFs.

Usage

```
geom_ecdf(
  mapping = NULL,
  data = NULL,
  geom = "step",
  position = "identity",
  ...,
  n = NULL,
  pad = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following: • A `Geom` ggproto subclass, for example `GeomPoint`. • A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`. • For more information and other ways to specify the geom, see the layer geom documentation.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.

	• For more information and other ways to specify the position, see the layer position documentation.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept. • The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
`n`	if NULL, do not interpolate. If not NULL, this is the number of points to interpolate with.
`pad`	If `TRUE`, pad the ecdf with additional points (-Inf, 0) and (Inf, 1)
`na.rm`	If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Details

ECDF plots show the cumulative distribution function $F(x) = P(X \leq x)$, displaying what proportion of observations fall below each value. When comparing treatment groups, overlapping ECDF curves indicate similar distributions and thus good balance.

ECDF plots are closely related to quantile-quantile (QQ) plots (see `geom_qq2()`). While ECDF plots show $F(x)$ for each group, QQ plots show the inverse relationship by plotting $F_1^{-1}(p)$ vs $F_2^{-1}(p)$. Both visualize the same distributional information:

- ECDF plots: Compare cumulative probabilities at each value
- QQ plots: Compare values at each quantile

Choose ECDF plots when you want to see the full cumulative distribution or when comparing multiple groups simultaneously. Choose QQ plots when you want to directly compare two groups with an easy-to-interpret 45-degree reference line.

Value

a geom

Aesthetics

In addition to the aesthetics for `ggplot2::stat_ecdf()`, `geom_ecdf()` also accepts:

- weights

See Also

- `geom_qq2()` for an alternative visualization using quantile-quantile plots
- `ggplot2::stat_ecdf()` for the unweighted version

Other ggplot2 functions: `geom_calibration()`, `geom_mirror_density()`, `geom_mirror_histogram()`, `geom_qq2()`, `geom_roc()`

Examples

```
library(ggplot2)

ggplot(
  nhefs_weights,
  aes(x = smokeyrs, color = qsmk)
) +
  geom_ecdf(aes(weights = w_ato)) +
  xlab("Smoking Years") +
  ylab("Proportion <= x")
```

`geom_mirror_density` *Create mirrored density plots*

Description

Create mirrored density plots

Usage

```
geom_mirror_density(
  mapping = NULL,
  data = NULL,
  stat = "density",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  outline.type = "upper"
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`stat`	The statistical transformation to use on the data for this layer. This should always be "density" (the default).
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`. • For more information and other ways to specify the position, see the layer position documentation.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth`

= 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`orientation`	The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.
`outline.type`	Type of the outline of the area; `"both"` draws both the upper and lower lines, `"upper"`/`"lower"` draws the respective lines only. `"full"` draws a closed polygon around the area.

Value

a geom

See Also

Other ggplot2 functions: `geom_calibration()`, `geom_ecdf()`, `geom_mirror_histogram()`, `geom_qq2()`, `geom_roc()`

Examples

```
library(ggplot2)
ggplot(nhefs_weights, aes(.fitted)) +
  geom_mirror_density(
    aes(group = qsmk),
    bw = 0.02
  ) +
  geom_mirror_density(
    aes(fill = qsmk, weight = w_ate),
    bw = 0.02,
    alpha = 0.5
  ) +
  scale_y_continuous(labels = abs)
```

`geom_mirror_histogram` *Create mirrored histograms*

Description

Create mirrored histograms

Usage

```
geom_mirror_histogram(
  mapping = NULL,
  data = NULL,
  position = "stack",
  ...,
  binwidth = NULL,
  bins = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`. • For more information and other ways to specify the position, see the layer position documentation.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept. • The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
`binwidth`	The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data. The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.
`bins`	Number of bins. Overridden by `binwidth`. Defaults to 30.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`orientation`	The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be

	given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

a geom

See Also

Other ggplot2 functions: `geom_calibration()`, `geom_ecdf()`, `geom_mirror_density()`, `geom_qq2()`, `geom_roc()`

Examples

```
library(ggplot2)
ggplot(nhefs_weights, aes(.fitted)) +
  geom_mirror_histogram(
    aes(group = qsmk),
    bins = 50
  ) +
  geom_mirror_histogram(
    aes(fill = qsmk, weight = w_ate),
    bins = 50,
    alpha = 0.5
  ) +
  scale_y_continuous(labels = abs)
```

`geom_qq2` *Create 2-dimensional QQ geometries*

Description

`geom_qq2()` is a geom for creating quantile-quantile plots with support for weighted comparisons. QQ plots compare the quantiles of two distributions, making them useful for assessing distributional balance in causal inference. As opposed to `geom_qq()`, this geom does not compare a variable against a theoretical distribution, but rather against two group's distributions, e.g., treatment vs. control.

Usage

```
geom_qq2(
  mapping = NULL,
  data = NULL,
  stat = "qq2",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  quantiles = seq(0.01, 0.99, 0.01),
  .reference_level = NULL,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings. Required aesthetics are `sample` (variable) and `treatment` (group). The `treatment` aesthetic can be a factor, character, or numeric. Optional aesthetics include `weight` for weighting.
`data`	Data frame to use. If not specified, inherits from the plot.
`stat`	Statistical transformation to use. Default is "qq2".
`position`	Position adjustment. Default is "identity".
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.
`quantiles`	Numeric vector of quantiles to compute. Default is `seq(0.01, 0.99, 0.01)` for 99 quantiles.
`.reference_level`	The reference treatment level to use for comparisons. If `NULL` (default), uses the first level for factors or the minimum value for numeric variables.
`...`	Other arguments passed on to layer().

Details

Quantile-quantile (QQ) plots visualize how the distributions of a variable differ between treatment groups by plotting corresponding quantiles against each other. If the distributions are identical, points fall on the 45-degree line (y = x). Deviations from this line indicate differences in the distributions.

QQ plots are closely related to empirical cumulative distribution function (ECDF) plots (see `geom_ecdf()`). While ECDF plots show $F(x) = P(X \leq x)$ for each group, QQ plots show $F_1^{-1}(p)$ vs $F_2^{-1}(p)$, essentially the inverse relationship. Both approaches visualize the same information about distributional differences, but QQ plots make it easier to spot deviations through a 45-degree reference line.

Value

A ggplot2 layer.

See Also

- `geom_ecdf()` for an alternative visualization of distributional differences
- `plot_qq()` for a complete plotting function with reference line and labels
- `check_qq()` for the underlying data computation

Other ggplot2 functions: `geom_calibration()`, `geom_ecdf()`, `geom_mirror_density()`, `geom_mirror_histogram()`, `geom_roc()`

Examples

```
library(ggplot2)

# Basic QQ plot
ggplot(nhefs_weights, aes(sample = age, treatment = qsmk)) +
  geom_qq2() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")

# With weighting
ggplot(nhefs_weights, aes(sample = age, treatment = qsmk, weight = w_ate)) +
  geom_qq2() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")

# Compare multiple weights using long format
long_data <- tidyr::pivot_longer(
  nhefs_weights,
  cols = c(w_ate, w_att),
  names_to = "weight_type",
  values_to = "weight"
)

ggplot(long_data, aes(color = weight_type)) +
  geom_qq2(aes(sample = age, treatment = qsmk, weight = weight)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")
```

`geom_roc` *ROC Curve Geom for Causal Inference*

Description

A ggplot2 geom for plotting ROC curves with optional weighting. Emphasizes the balance interpretation where AUC around 0.5 indicates good balance.

Usage

```
geom_roc(
  mapping = NULL,
  data = NULL,
  stat = "roc",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  linewidth = 0.5,
  .focal_level = NULL,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings. Must include `estimate` (propensity scores/predictions) and `exposure` (treatment/outcome variable). If specified, inherits from the plot.
`data`	Data frame to use. If not specified, inherits from the plot.
`stat`	Statistical transformation to use. Default is "roc".
`position`	Position adjustment. Default is "identity".
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.
`linewidth`	Width of the ROC curve line. Default is 0.5.
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.
`...`	Other arguments passed on to layer().

Value

A ggplot2 layer.

See Also

`check_model_auc()` for computing AUC values, `stat_roc()` for the underlying stat

Other ggplot2 functions: `geom_calibration()`, `geom_ecdf()`, `geom_mirror_density()`, `geom_mirror_histogram()`, `geom_qq2()`

Examples

```
# Basic usage
library(ggplot2)
ggplot(nhefs_weights, aes(estimate = .fitted, exposure = qsmk)) +
  geom_roc() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")

# With grouping by weight
long_data <- tidyr::pivot_longer(
  nhefs_weights,
  cols = c(w_ate, w_att),
  names_to = "weight_type",
  values_to = "weight"
)

ggplot(long_data, aes(estimate = .fitted, exposure = qsmk, weight = weight)) +
  geom_roc(aes(color = weight_type)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")
```

`nhefs_weights` *NHEFS with various propensity score weights*

Description

A dataset containing various propensity score weights for `causaldata::nhefs_complete`, including weights for both binary (smoking cessation) and categorical (alcohol frequency) exposures.

Usage

```
nhefs_weights
```

Format

A data frame with 1566 rows and 30 variables:

qsmk Quit smoking (binary exposure)

alcoholfreq Alcohol frequency (numeric 0-5)

alcoholfreq_cat Alcohol frequency as categorical (factor with levels: none, lt_12_per_year, 1_4_per_month, 2_3_per_week, daily); NA for unknown alcohol frequency

race Race

age Age

sex Sex

education Education level

smokeintensity Smoking intensity

smokeyrs Number of smoke-years

exercise Exercise level

active Daily activity level

wt71 Participant weight in 1971 (baseline)

wt82_71 Weight change from 1971 to 1982

death Death indicator

wts Simple inverse probability weight for binary exposure

w_ate ATE weight for binary exposure

w_att ATT weight for binary exposure

w_atc ATC weight for binary exposure

w_atm ATM weight for binary exposure

w_ato ATO weight for binary exposure

w_cat_ate ATE weight for categorical exposure (NA for unknown alcohol frequency)

w_cat_att_none ATT weight with "none" as focal category (NA for unknown alcohol frequency)

w_cat_att_lt12 ATT weight with "lt_12_per_year" as focal category (NA for unknown alcohol frequency)

w_cat_att_1_4mo ATT weight with "1_4_per_month" as focal category (NA for unknown alcohol frequency)

w_cat_att_2_3wk ATT weight with "2_3_per_week" as focal category (NA for unknown alcohol frequency)

w_cat_att_daily ATT weight with "daily" as focal category (NA for unknown alcohol frequency)

w_cat_atu_none ATU weight with "none" as focal category (NA for unknown alcohol frequency)

w_cat_ato ATO weight for categorical exposure (NA for unknown alcohol frequency)

w_cat_atm ATM weight for categorical exposure (NA for unknown alcohol frequency)

.fitted Propensity score for binary exposure

`plot-halfmoon` *Plot Methods for halfmoon Objects*

Description

These methods provide standard plot generation for halfmoon data objects. They create the plot using autoplot() and then print it.

Usage

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

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

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

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

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

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

Arguments

`x`	A halfmoon data object with appropriate class
`...`	Additional arguments passed to autoplot()

Value

Invisibly returns the ggplot2 object after printing

`plot_balance` *Create balance plot from check_balance output*

Description

Create a Love plot-style visualization to assess balance across multiple metrics computed by `check_balance()`. This function wraps `geom_love()` to create a comprehensive balance assessment plot.

Usage

```
plot_balance(
  .df,
  abs_smd = TRUE,
  facet_scales = "free",
  linewidth = 0.8,
  point_size = 1.85,
  vline_xintercept = 0.1,
  vline_color = "grey70",
  vlinewidth = 0.6
)
```

Arguments

`.df`	A data frame produced by `check_balance()`
`abs_smd`	Logical. Take the absolute value of SMD estimates? Defaults to TRUE. Does not affect other metrics which are already non-negative.

`facet_scales`	Character. Scale specification for facets. Defaults to "free" to allow different scales for different metrics. Options are "fixed", "free_x", "free_y", or "free".
`linewidth`	The line size, passed to `ggplot2::geom_line()`.
`point_size`	The point size, passed to `ggplot2::geom_point()`.
`vline_xintercept`	The X intercept, passed to `ggplot2::geom_vline()`.
`vline_color`	The vertical line color, passed to `ggplot2::geom_vline()`.
`vlinewidth`	The vertical line size, passed to `ggplot2::geom_vline()`.

Details

This function visualizes the output of `check_balance()`, creating a plot that shows balance statistics across different variables, methods, and metrics. The plot uses faceting to separate different metrics and displays the absolute value of SMD by default (controlled by `abs_smd`).

For categorical exposures (>2 levels), the function automatically detects multiple group level comparisons and uses `facet_grid()` to display each comparison in a separate row, with metrics in columns. For binary exposures, the standard `facet_wrap()` by metric is used.

Different metrics have different interpretations:

- **SMD**: Standardized mean differences, where values near 0 indicate good balance. Often displayed as absolute values.
- **Variance Ratio**: Ratio of variances between groups, where values near 1 indicate similar variability.
- **KS**: Kolmogorov-Smirnov statistic, where smaller values indicate better distributional balance.
- **Correlation**: For continuous exposures, measures association with covariates.
- **Energy**: Multivariate balance metric applied to all variables simultaneously.

Value

A ggplot2 object

See Also

`check_balance()` for computing balance metrics, `geom_love()` for the underlying geom

Other balance functions: `bal_corr()`, `bal_ess()`, `bal_ks()`, `bal_model_auc()`, `bal_model_roc_curve()`, `bal_qq()`, `bal_smd()`, `bal_vr()`, `check_balance()`, `check_ess()`, `check_model_auc()`, `check_model_roc_curve()`, `check_qq()`

Examples

```
# Compute balance metrics
balance_data <- check_balance(
  nhefs_weights,
  c(age, education, race),
  qsmk,
  .weights = c(w_ate, w_att)
)

# Create balance plot
plot_balance(balance_data)

# Without absolute SMD values
plot_balance(balance_data, abs_smd = FALSE)

# With fixed scales across facets
plot_balance(balance_data, facet_scales = "fixed")

# Customize threshold lines
plot_balance(balance_data, vline_xintercept = 0.05)

# Categorical exposure example
# Automatically uses facet_grid to show each group comparison
balance_cat <- check_balance(
  nhefs_weights,
  c(age, wt71, sex),
  alcoholfreq_cat,
  .weights = w_cat_ate,
  .metrics = c("smd", "vr")
)
plot_balance(balance_cat)
```

plot_ess *Plot Effective Sample Size*

Description

Creates a bar plot visualization of effective sample sizes (ESS) for different weighting schemes. ESS values are shown as percentages of the actual sample size, with a reference line at 100% indicating no loss of effective sample size.

Usage

```
plot_ess(
  .data,
  .weights = NULL,
  .exposure = NULL,
  include_observed = TRUE,
  n_tiles = 4,
  tile_labels = NULL,
  fill_color = "#0172B1",
  alpha = 0.8,
  show_labels = TRUE,
  label_size = 3,
  percent_scale = TRUE,
  reference_line_color = "gray50",
  reference_line_type = "dashed"
)
```

Arguments

`.data`	A data frame, either: • Output from `check_ess()` containing ESS calculations • Raw data to compute ESS from (requires `.weights` to be specified)
`.weights`	Optional weighting variables. Can be unquoted variable names, a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes.
`.exposure`	Optional exposure variable. When provided, ESS is calculated separately for each exposure level. For continuous variables, groups are created using quantiles.
`include_observed`	Logical. If using `.weights`, also calculate observed (unweighted) metrics? Defaults to TRUE.
`n_tiles`	For continuous `.exposure` variables, the number of quantile groups to create. Default is 4 (quartiles).
`tile_labels`	Optional character vector of labels for the quantile groups when `.exposure` is continuous. If NULL, uses "Q1", "Q2", etc.
`fill_color`	Color for the bars when `.exposure` is not provided. Default is "#0172B1".
`alpha`	Transparency level for the bars. Default is 0.8.
`show_labels`	Logical. Show ESS percentage values as text labels on bars? Default is TRUE.
`label_size`	Size of text labels. Default is 3.
`percent_scale`	Logical. Display ESS as percentage of sample size (TRUE) or on original scale (FALSE)? Default is TRUE.
`reference_line_color`	Color for the 100% reference line. Default is "gray50".
`reference_line_type`	Line type for the reference line. Default is "dashed".

Details

This function visualizes the output of `check_ess()` or computes ESS directly from the provided data. The plot shows how much "effective" sample size remains after weighting, which is a key diagnostic for assessing weight variability.

When `.exposure` is not provided, the function displays overall ESS for each weighting method. When `.exposure` is provided, ESS is shown separately for each exposure level using dodged bars.

For continuous grouping variables, the function automatically creates quantile groups (quartiles by default) to show how ESS varies across the distribution of the continuous variable.

Lower ESS percentages indicate:

- Greater weight variability

- More extreme weights
- Potentially unstable weighted estimates
- Need for weight trimming or alternative methods

Value

A ggplot2 object.

See Also

`check_ess()` for computing ESS values, `ess()` for the underlying calculation

Examples

```
# Overall ESS for different weighting schemes
plot_ess(nhefs_weights, .weights = c(w_ate, w_att, w_atm))

# ESS by treatment group (binary exposure)
plot_ess(nhefs_weights, .weights = c(w_ate, w_att), .exposure = qsmk)

# ESS by treatment group (categorical exposure)
plot_ess(nhefs_weights, .weights = w_cat_ate, .exposure = alcoholfreq_cat)

# ESS by age quartiles
plot_ess(nhefs_weights, .weights = w_ate, .exposure = age)

# Customize quantiles for continuous variable
plot_ess(nhefs_weights, .weights = w_ate, .exposure = age,
         n_tiles = 5, tile_labels = c("Youngest", "Young", "Middle", "Older", "Oldest"))

# Without percentage labels
plot_ess(nhefs_weights, .weights = c(w_ate, w_att), .exposure = qsmk,
         show_labels = FALSE)

# Custom styling
plot_ess(nhefs_weights, .weights = c(w_ate, w_att), .exposure = qsmk,
         alpha = 0.6, fill_color = "steelblue",
         reference_line_color = "red")

# Using pre-computed ESS data
ess_data <- check_ess(nhefs_weights, .weights = c(w_ate, w_att))
plot_ess(ess_data)

# Show ESS on original scale instead of percentage
plot_ess(nhefs_weights, .weights = c(w_ate, w_att), percent_scale = FALSE)
```

plot_mirror_distributions *Create mirror distribution plots*

Description

Create mirror distribution plots (histograms or density plots) to compare the distribution of variables between treatment groups before and after weighting. This function helps assess covariate balance by visualizing the distributions side-by-side with one group mirrored below the axis.

Usage

```
plot_mirror_distributions(
  .data,
  .var,
  .exposure,
  .weights = NULL,
  type = c("histogram", "density"),
  mirror_axis = "y",
  bins = 30,
  binwidth = NULL,
  bw = "nrd0",
  adjust = 1,
  include_unweighted = TRUE,
  alpha = 0.6,
  na.rm = FALSE,
  .reference_level = 1L,
  facet_scales = "fixed"
)
```

Arguments

`.data`	A data frame containing the variables.
`.var`	The variable to plot. Supports tidyselect syntax. Can be unquoted.
`.exposure`	Column name of treatment/group variable. Supports tidyselect syntax. Can be unquoted. For binary variables, must have exactly 2 levels. For categorical variables (>2 levels), creates pairwise comparisons against a reference group.
`.weights`	Optional weighting variable(s). Can be unquoted variable names, tidyselect syntax, a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes. Default is NULL (unweighted).
`type`	Character; type of plot - "histogram" or "density". Default is "histogram".
`mirror_axis`	Character; which axis to mirror - "y" (default) or "x".
`bins`	Integer; number of bins for histogram. Only used when type = "histogram".
`binwidth`	Numeric; width of bins for histogram. Only used when type = "histogram". If both bins and binwidth are specified, binwidth takes precedence.

`bw`	Bandwidth for density estimation. Only used when type = "density". Can be numeric or character (e.g., "nrd0", "sj").
`adjust`	Numeric; bandwidth adjustment factor for density. Only used when type = "density". Default is 1.
`include_unweighted`	Logical. If using `.weights`, also show unweighted distribution? Defaults to TRUE.
`alpha`	Numeric; transparency level for fills. Default is 0.6.
`na.rm`	Logical; if TRUE, drop NA values before plotting.
`.reference_level`	The reference group level for categorical exposures (>2 levels). Can be a string (group level) or numeric (position). Defaults to 1 (first level). Only used when .exposure has more than 2 levels.
`facet_scales`	Character. Scale specification for facets. Defaults to "fixed" to match ggplot2's default behavior. Options are "fixed", "free_x", "free_y", or "free". Use "free_y" to allow different y-axis scales across panels, which was the previous default behavior.

Details

Mirror distribution plots display the distribution of one group above the x-axis and the other group below (mirrored). This makes it easy to compare distributions and assess balance. The function supports both histogram and density plot types.

For categorical exposures (>2 levels), the function creates a grid of pairwise comparisons against a reference group. Each panel shows one non-reference level compared to the reference level using the mirror distribution format.

When using weights, the function can display both weighted and unweighted distributions for comparison. Multiple weighting schemes can be compared by providing multiple weight variables.

Value

A ggplot2 object.

See Also

- `geom_mirror_histogram()` for the underlying histogram geom
- `geom_mirror_density()` for the underlying density geom
- `plot_qq()` for QQ plots, another distributional comparison
- `geom_ecdf()` for ECDF plots

Examples

```
library(ggplot2)

# Basic histogram (unweighted)
plot_mirror_distributions(nhefs_weights, age, qsmk)
```

```
# Density plot instead of histogram
plot_mirror_distributions(nhefs_weights, age, qsmk, type = "density")

# With weighting
plot_mirror_distributions(nhefs_weights, age, qsmk, .weights = w_ate)

# Compare multiple weighting schemes
plot_mirror_distributions(nhefs_weights, age, qsmk, .weights = c(w_ate, w_att))

# Customize appearance
plot_mirror_distributions(
  nhefs_weights, age, qsmk,
  .weights = w_ate,
  type = "density",
  alpha = 0.7
)

# Without unweighted comparison
plot_mirror_distributions(
  nhefs_weights, age, qsmk,
  .weights = w_ate,
  include_unweighted = FALSE
)

# Categorical exposure - creates grid of comparisons
plot_mirror_distributions(
  nhefs_weights,
  age,
  alcoholfreq_cat,
  type = "density"
)

# Categorical with weights
plot_mirror_distributions(
  nhefs_weights,
  wt71,
  alcoholfreq_cat,
  .weights = w_cat_ate,
  .reference_level = "none"
)
```

`plot_model_auc` *Plot ROC AUC Values for Balance Assessment*

Description

Creates a visualization of AUC values from weighted ROC analysis. Values near 0.5 indicate good balance.

Usage

```
plot_model_auc(
  .data,
  ref_line = TRUE,
  ref_color = "red",
  point_size = 3,
  point_shape = 19
)
```

Arguments

`.data`	Output from `check_model_auc()`.
`ref_line`	Show reference line at AUC = 0.5? Default is TRUE.
`ref_color`	Color for reference line. Default is "red".
`point_size`	Size of the points. Default is 3.
`point_shape`	Shape of the points. Default is 19 (filled circle).

Value

A ggplot2 object.

Examples

```
# Compute AUC values
auc_data <- check_model_auc(
  nhefs_weights,
  qsmk,
  .fitted,
  c(w_ate, w_att)
)

# Create plot
plot_model_auc(auc_data)
```

`plot_model_calibration` *Create calibration plot*

Description

Create a calibration plot to assess the agreement between predicted probabilities and observed treatment rates. This function wraps `geom_calibration()`.

Usage

```
plot_model_calibration(x, ...)

## S3 method for class 'data.frame'
plot_model_calibration(
  x,
  .fitted,
  .exposure,
  .focal_level = NULL,
  method = "breaks",
  bins = 10,
  smooth = TRUE,
  conf_level = 0.95,
  window_size = 0.1,
  step_size = window_size/2,
  k = 10,
  include_rug = FALSE,
  include_ribbon = TRUE,
  include_points = TRUE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'glm'
plot_model_calibration(
  x,
  .focal_level = NULL,
  method = "breaks",
  bins = 10,
  smooth = TRUE,
  conf_level = 0.95,
  window_size = 0.1,
  step_size = window_size/2,
  k = 10,
  include_rug = FALSE,
  include_ribbon = TRUE,
  include_points = TRUE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'lm'
plot_model_calibration(
  x,
  .focal_level = NULL,
  method = "breaks",
  bins = 10,
  smooth = TRUE,
  conf_level = 0.95,
  window_size = 0.1,
  step_size = window_size/2,
  k = 10,
  include_rug = FALSE,
  include_ribbon = TRUE,
  include_points = TRUE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'halfmoon_calibration'
plot_model_calibration(
  x,
  include_rug = FALSE,
  include_ribbon = TRUE,
  include_points = TRUE,
  ...
)
```

Arguments

`x`	Either a fitted model object (lm or glm) or a data frame
`...`	Additional arguments passed to methods
`.fitted`	Column name of predicted probabilities (propensity scores). Can be unquoted (e.g., `.fitted`) or quoted (e.g., `".fitted"`).
`.exposure`	Column name of treatment/exposure variable. Can be unquoted (e.g., `qsmk`) or quoted (e.g., `"qsmk"`).
`.focal_level`	Value indicating which level of `.exposure` represents treatment. If NULL (default), uses the last level for factors or max value for numeric.
`method`	Character; calibration method - "breaks", "logistic", or "windowed".
`bins`	Integer >1; number of bins for the "breaks" method.
`smooth`	Logical; for "logistic" method, use GAM smoothing if available.
`conf_level`	Numeric in (0,1); confidence level for CIs (default = 0.95).
`window_size`	Numeric; size of each window for "windowed" method.
`step_size`	Numeric; distance between window centers for "windowed" method.
`k`	Integer; the basis dimension for GAM smoothing when method = "logistic" and smooth = TRUE. Default is 10.
`include_rug`	Logical; add rug plot showing distribution of predicted probabilities.
`include_ribbon`	Logical; show confidence interval ribbon.
`include_points`	Logical; show points (only for "breaks" and "windowed" methods).
`na.rm`	Logical; if TRUE, drop NA values before computation.

Details

Calibration plots visualize how well predicted probabilities match observed outcome rates. Since outcomes are binary (0/1), the "observed rate" represents the proportion of units with outcome = 1 within each prediction group. For example, among all units with predicted probability around 0.3, we expect approximately 30% to actually have the outcome. Perfect calibration occurs when predicted probabilities equal observed rates (points fall on the 45-degree line).

The plot supports three calibration assessment methods:

- **"breaks"**: Bins predictions into groups and compares mean prediction vs observed rate within each bin
- **"logistic"**: Fits a logistic regression of outcomes on predictions; perfect calibration yields slope=1, intercept=0
- **"windowed"**: Uses sliding windows across the prediction range for smooth calibration curves

The function supports two approaches:

- For regression models (lm/glm): Extracts fitted values and observed outcomes automatically
- For data frames: Uses specified columns for fitted values and treatment group

Value

A ggplot2 object.

See Also

- `geom_calibration()` for the underlying geom
- `check_model_calibration()` for numerical calibration metrics
- `plot_stratified_residuals()` for residual diagnostic plots
- `plot_model_roc_curve()` for ROC curves
- `plot_qq()` for QQ plots

Examples

```
library(ggplot2)

# Method 1: Using data frame
plot_model_calibration(nhefs_weights, .fitted, qsmk)

# With rug plot
plot_model_calibration(nhefs_weights, .fitted, qsmk, include_rug = TRUE)

# Different methods
plot_model_calibration(nhefs_weights, .fitted, qsmk, method = "logistic")
plot_model_calibration(nhefs_weights, .fitted, qsmk, method = "windowed")

# Specify treatment level explicitly
plot_model_calibration(nhefs_weights, .fitted, qsmk, .focal_level = "1")
```

```
# Method 2: Using model objects
# Fit a propensity score model
ps_model <- glm(qsmk ~ age + sex + race + education,
                data = nhefs_weights,
                family = binomial())

# Plot calibration from the model
plot_model_calibration(ps_model)
```

`plot_model_roc_curve` *Plot weighted ROC Curves for Balance Assessment*

Description

Creates a ggplot2 visualization of ROC curves for evaluating propensity score balance. In causal inference, weighted curves near the diagonal (AUC around 0.5) indicate good balance.

Usage

```
plot_model_roc_curve(
  .data,
  linewidth = 0.5,
  diagonal_color = "gray50",
  diagonal_linetype = "dashed"
)
```

Arguments

`.data`	Output from `check_model_roc_curve()` with class "halfmoon_roc".
`linewidth`	Width of the ROC curve lines. Default is 1.
`diagonal_color`	Color for the diagonal reference line. Default is "gray50".
`diagonal_linetype`	Line type for the diagonal. Default is "dashed".

Details

ROC curves for balance assessment plot the true positive rate (sensitivity) against the false positive rate (1 - specificity) when using propensity scores to classify treatment assignment. When weights achieve perfect balance, the propensity score distributions become identical between groups, yielding an ROC curve along the diagonal (chance performance).

Curves that deviate substantially from the diagonal indicate that propensity scores can still discriminate between treatment groups after weighting, suggesting residual imbalance. The closer the curve is to the diagonal, the better the balance achieved by the weighting scheme.

Value

A ggplot2 object.

Examples

```
roc_data <- check_model_roc_curve(
  nhefs_weights,
  qsmk,
  .fitted,
  c(w_ate, w_att)
)

plot_model_roc_curve(roc_data)
```

---

plot_qq *Create QQ plots for weighted and unweighted samples*

Description

Create quantile-quantile (QQ) plots to compare the distribution of variables between treatment groups before and after weighting. This function helps assess covariate balance by visualizing how well the quantiles align between groups.

Usage

```
plot_qq(.data, ...)

## Default S3 method:
plot_qq(
  .data,
  .var,
  .exposure,
  .weights = NULL,
  quantiles = seq(0.01, 0.99, 0.01),
  include_observed = TRUE,
  .reference_level = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'halfmoon_qq'
plot_qq(.data, ...)
```

Arguments

`.data`	A data frame containing the variables or a halfmoon_qq object.
`...`	Arguments passed to methods (see Methods section).
`.var`	Variable to plot. Can be unquoted (e.g., `age`) or quoted (e.g., `"age"`).

`.exposure`	Column name of treatment/exposure variable. Can be unquoted (e.g., `qsmk`) or quoted (e.g., `"qsmk"`).
`.weights`	Optional weighting variable(s). Can be unquoted variable names, a character vector, or NULL. Multiple weights can be provided to compare different weighting schemes. Default is NULL (unweighted).
`quantiles`	Numeric vector of quantiles to compute. Default is `seq(0.01, 0.99, 0.01)` for 99 quantiles.
`include_observed`	Logical. If using `.weights`, also show observed (unweighted) QQ plot? Defaults to TRUE.
`.reference_level`	The reference treatment level to use for comparisons. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.
`na.rm`	Logical; if TRUE, drop NA values before computation.

Details

QQ plots display the quantiles of one distribution against the quantiles of another. Perfect distributional balance appears as points along the 45-degree line (y = x). This function automatically adds this reference line and appropriate axis labels.

For an alternative visualization of the same information, see `geom_ecdf()`, which shows the empirical cumulative distribution functions directly.

Value

A ggplot2 object.

Methods

`plot_qq.default` For data frames. Accepts all documented parameters.

`plot_qq.halfmoon_qq` For halfmoon_qq objects from `check_qq()`. Only uses `.data` and `...` parameters.

See Also

- `geom_ecdf()` for ECDF plots, an alternative distributional visualization
- `geom_qq2()` for the underlying geom used by this function
- `check_qq()` for computing QQ data without plotting

Examples

```
library(ggplot2)

# Basic QQ plot (observed)
plot_qq(nhefs_weights, age, qsmk)

# With weighting
plot_qq(nhefs_weights, age, qsmk, .weights = w_ate)
```

```
# Compare multiple weighting schemes
plot_qq(nhefs_weights, age, qsmk, .weights = c(w_ate, w_att))

# For propensity scores
plot_qq(nhefs_weights, .fitted, qsmk, .weights = w_ate)

# Without observed comparison
plot_qq(nhefs_weights, age, qsmk, .weights = w_ate, include_observed = FALSE)
```

`plot_stratified_residuals` *Create stratified residual diagnostic plots*

Description

Create diagnostic plots to assess differences between .exposure group after adjustment. This function plots residuals from an outcome model against propensity scores (or fitted values), stratified by .exposure group, to reveal model mis-specification.

Usage

```
plot_stratified_residuals(x, ...)

## S3 method for class 'lm'
plot_stratified_residuals(
  x,
  .exposure,
  ps_model = NULL,
  plot_type = c("color", "facet", "both"),
  smooth = TRUE,
  smooth_span = 1,
  alpha = 0.25,
  na.rm = FALSE,
  ...
)

## S3 method for class 'glm'
plot_stratified_residuals(
  x,
  .exposure,
  ps_model = NULL,
  plot_type = c("color", "facet", "both"),
  smooth = TRUE,
  smooth_span = 1,
  alpha = 0.25,
  na.rm = FALSE,
  ...
)

## S3 method for class 'data.frame'
plot_stratified_residuals(
  x,
  .exposure,
  residuals,
  x_var,
  plot_type = c("color", "facet", "both"),
  smooth = TRUE,
  smooth_span = 1,
  alpha = 0.25,
  na.rm = FALSE,
  ...
)
```

Arguments

`x`	Either a fitted model object (lm or glm) or a data frame
`...`	Additional arguments passed to methods
`.exposure`	A vector indicating .exposure group membership. Must have exactly two unique levels. For data frames, can be an unquoted column name.
`ps_model`	Optional propensity score model (glm object). If provided, uses propensity scores instead of fitted values.
`plot_type`	Character; type of plot - "color" (default), "facet", or "both". • "color": Single plot with points colored by .exposure • "facet": Separate facets for each .exposure group • "both": Both color and faceting
`smooth`	Logical; whether to add loess smoothing curves. Default is TRUE.
`smooth_span`	Numeric; span parameter for loess smoothing. Default is 1.
`alpha`	Numeric; transparency level for points. Default is 0.25.
`na.rm`	Logical; if TRUE, remove missing values before plotting.
`residuals`	Column containing residuals. Supports tidyselect syntax.
`x_var`	Column for x-axis values (fitted values or propensity scores). Supports tidyselect syntax.

Details

This diagnostic plot was originally suggested by Rosenbaum and Rubin (1983) and revisited by D'Agostino McGowan, D'Agostino, and D'Agostino (2023). The key idea is that plotting residuals against propensity scores or fitted values by .exposure group can reveal non-linear relationships or heterogeneous .exposure effects that might be obscured in standard residuals-vs-fitted plots.

The function supports two approaches:

- For regression models (lm/glm): Extracts residuals and fitted values automatically
- For data frames: Uses specified columns for residuals, .exposure, and x-axis values

Value

A ggplot2 object

References

- Rosenbaum, P. R., & Rubin, D. B. (1983). The central role of the propensity score in observational studies for causal effects. Biometrika, 70(1), 41-55.
- D'Agostino McGowan, L., D'Agostino, R. B, Sr., & D'Agostino, R. B, Jr. (2023). A Visual Diagnostic Tool for Causal Inference. Observational Studies, 9(1), 87-95.

See Also

- `plot_model_calibration()` for calibration plots
- `plot_model_roc_curve()` for ROC curves
- `plot_qq()` for QQ plots

Examples

```
## Not run:
library(ggplot2)

# Simulate data with .exposure effect heterogeneity
set.seed(8)
n <- 1000
x <- rnorm(n)
ps <- plogis(x)  # True propensity score
.exposure <- rbinom(n, 1, ps)
y1 <- 0.5 * x + rnorm(n)
y0 <- -0.5 * x + rnorm(n)
y <- .exposure * y1 + (1 - .exposure) * y0

# Method 1: Using model objects
# Fit misspecified model (missing interaction)
model_wrong <- lm(y ~ .exposure + x)

# Plot with fitted values
plot_stratified_residuals(
  model_wrong,
  .exposure = .exposure,
  plot_type = "both"
)

# Plot with propensity scores
ps_model <- glm(.exposure ~ x, family = binomial)

plot_stratified_residuals(
  model_wrong,
  .exposure = .exposure,
  ps_model = ps_model,
  plot_type = "color"
)

# Method 2: Using data frame
library(dplyr)
plot_data <- data.frame(
  .exposure = .exposure,
  residuals = residuals(model_wrong),
  fitted_values = fitted(model_wrong),
  propensity_score = fitted(ps_model)
)

plot_stratified_residuals(
  plot_data,
  .exposure = .exposure,
  residuals = residuals,
  x_var = propensity_score,
  plot_type = "facet"
)

## End(Not run)
```

stat_qq2 *QQ2 Plot Stat*

Description

Statistical transformation for QQ plots.

Usage

```
stat_qq2(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  quantiles = seq(0.01, 0.99, 0.01),
  .reference_level = NULL,
  include_observed = FALSE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings.
`data`	Data frame.

`geom`	Geometric object to use. Default is "point".
`position`	Position adjustment.
`na.rm`	Remove missing values? Default TRUE.
`show.legend`	Show legend? Default NA.
`inherit.aes`	Inherit aesthetics? Default TRUE.
`quantiles`	Numeric vector of quantiles to compute.
`.reference_level`	The reference treatment level to use for comparisons.
`include_observed`	For compatibility with qq(). When weights are present, this determines if an additional "observed" group is added. Default FALSE for stat_qq2 to avoid duplication when using facets/colors.
`...`	Additional arguments.

Value

A ggplot2 layer.

`stat_roc` *ROC Curve Stat*

Description

Statistical transformation for ROC curves.

Usage

```
stat_roc(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  .focal_level = NULL,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot.

`data`	The data to be displayed in this layer. If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
`geom`	Geometric object to use. Default is "path".
`position`	Position adjustment.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.
`.focal_level`	The level of the outcome variable to consider as the treatment/event. If `NULL` (default), uses the last level for factors or the maximum value for numeric variables.
`...`	Other arguments passed on to layer().

Value

A ggplot2 layer.

`weighted_quantile` *Compute weighted quantiles*

Description

Calculate quantiles of a numeric vector with associated weights. This function sorts the values and computes weighted cumulative distribution before interpolating the requested quantiles.

Usage

```
weighted_quantile(values, quantiles, .weights)
```

Arguments

`values`	Numeric vector of values to compute quantiles for.
`quantiles`	Numeric vector of probabilities with values between 0 and 1.
`.weights`	Numeric vector of non-negative weights, same length as `values`.

Value

Numeric vector of weighted quantiles corresponding to the requested probabilities.

Examples

```
# Equal weights (same as regular quantiles)
weighted_quantile(1:10, c(0.25, 0.5, 0.75), rep(1, 10))

# Weighted towards higher values
weighted_quantile(1:10, c(0.25, 0.5, 0.75), 1:10)
```

Index