Package 'ivdtools'

September 11, 2026

Title Statistical Tools for Evaluation of in Vitro Diagnostic Reagents

Version 0.2.5

Description Provides statistical workflows used in the evaluation of in vitro diagnostic reagents. Facilities include method comparison, commutability assessment, Bland-Altman and receiver operating characteristic analysis, qualitative agreement, C5 and C95 estimation, precision and variance-component analysis, linearity, interference, dilution and spiking studies, high-dose hook assessment, measurement uncertainty, reference-material bias, reference intervals, stability studies, quality-control charts, curve fitting, analytical sensitivity, outlier and normality assessment, and sample-size calculations. For methodological details, see Bland and Altman (1986) <doi:10.1016/S0140-6736(86)90837-8>, Passing and Bablok (1983) <doi:10.1515/cclm.1983.21.11.709>, Linnet (1993) <doi:10.1093/clinchem/39.3.424>, Hawkins and Kraker (2026) <doi:10.1093/jalm/jfaf183>, Hanley and McNeil (1982) <doi:10.1148/radiology.143.1.7063747>, Horn et al. (1998) <doi:10.1093/clinchem/44.3.622>, Westgard et al. (1981) <doi:10.1093/clinchem/27.3.493>, and Lu et al. (2016) <doi:10.1515/ijb-2015-0039>.

License MIT + file LICENSE

URL https://github.com/hiox-tech/ivdtools

BugReports https://github.com/hiox-tech/ivdtools/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports ggplot2, ggrepel, minpack.lm, nloptr, nls2, nortest, ppwdeming, rlang, stats, utils, VCA, VFP

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author hiox-tech [cph, aut, cre]

Maintainer hiox-tech <GeorgeBinDragon@outlook.com>

Repository CRAN

Date/Publication 2026-09-11 08:30:02 UTC

Contents

`arrhenius` *arrhenius — Arrhenius accelerated stability analysis*

Description

Estimates degradation rates at multiple elevated temperatures, fits the Arrhenius equation, and extrapolates to the target storage temperature to predict shelf life.

Usage

```
arrhenius(
  data,
  temperature,
  time,
  value,
  target_temp,
  order = c("auto", "zero", "first"),
  direction = c("auto", "decrease", "increase"),
  limit,
  bias_type = c("relative", "absolute"),
  temp_unit = "C",
  time_unit = "d",
  conf.level = 0.95
)
```

Arguments

`data`	A data frame.
`temperature`	Column name for storage temperature (numeric).
`time`	Column name for time point (numeric).
`value`	Column name for measurement value (numeric).
`target_temp`	Target storage temperature.
`order`	`"auto"`, `"zero"`, or `"first"`.
`direction`	`"auto"`, `"decrease"`, or `"increase"`.
`limit`	A positive number for symmetric allowable bias.
`bias_type`	`"relative"` (percent) or `"absolute"`.
`temp_unit`	Temperature unit: `"C"` or `"K"`.
`time_unit`	Time unit: m, min, h, d, month, or y.
`conf.level`	Confidence level, default 0.95.

Value

An S3 object of class "arrhenius".

Examples

```
temp_df <- data.frame(temp = rep(c(25,30,37), each = 3),
                      time = rep(c(0,1,3), 3),
                      val = c(100,98,95, 100,96,90, 100,94,85))
arrhenius(temp_df, "temp", "time", "val", target_temp = 5, limit = 10)
```

---

auc.roc *Compute ROC curves and AUC for each evaluation column.*

Description

Compute ROC curves and AUC for each evaluation column.

Usage

```
## S3 method for class 'roc'
auc(x, cols = NULL, ...)
```

Arguments

x	roc object
cols	column names to analyze; default is all
...	reserved arguments

Value

Updated roc object with results stored in $auc_list and $auc_table

Examples

```
obj <- roc(ivd_roc_example, cols = c("x1", "x2"), reference = "ref")
obj <- auc(obj)
obj <- auc(obj, cols = "x1")
```

`auc_compare.roc` *Compare two paired ROC AUCs*

Description

Compare two ROC curves evaluated on the common complete cases stored in a single `roc()` object. A curve can be a marker previously processed by `auc()` or a named multivariate model produced by `mlr()`.

Usage

```
## S3 method for class 'roc'
auc_compare(
  x,
  curves,
  method = c("ep24", "delong"),
  conf.level = 0.95,
  rating_method = c("pearson", "kendall"),
  name = NULL,
  ...
)
```

Arguments

`x`	A `roc` object.
`curves`	Character vector naming exactly two marker or MLR curves. The reported difference is `curves[1] - curves[2]`.
`method`	Paired comparison method: `"ep24"` (default) or `"delong"`.
`conf.level`	Confidence level for the normal-approximation interval.
`rating_method`	Correlation method for EP24 paired scores: `"pearson"` for interval-scale laboratory results (default) or `"kendall"` for ordinal ratings. Ignored for DeLong.
`name`	Optional unique name for storing the comparison. By default it is generated from the two curve names and method.
`...`	Additional arguments (currently unused).

Details

`method = "ep24"` implements the paired Hanley–McNeil approximation in CLSI EP24-A2. It combines the two single-curve Hanley–McNeil standard errors with the correlation between the paired AUCs. The latter is obtained by linear interpolation of EP24-A2 Table 5 from the average within-group score correlation and average AUC. Coordinates outside the published table are bounded to its range and recorded in the result.

`method = "delong"` uses the nonparametric DeLong covariance estimator and handles ties with half credit. Both methods test the two-sided null hypothesis that the AUC difference is zero.

Value

The updated roc object, invisibly. The comparison is stored in x$auc_comparisons[[name]] as a roc_auc_comparison object.

Examples

```
obj <- roc(ivd_roc_example, c("x1", "x2"), "ref")
obj <- auc(obj)
obj <- auc_compare(obj, c("x1", "x2"), method = "ep24")
obj$auc_comparisons[[1L]]

obj <- mlr(obj, name = "combined")
obj <- auc_compare(obj, c("x1", "combined"), method = "delong")
```

---

bias.mcr *Bias analysis (S3 method) – estimate bias at given medical decision levels*

Description

Compute bias at specified medical decision levels (MDL) based on stored regression results: bias = predicted_candidate - reference = (intercept + slope * x0) - x0 = intercept + (slope - 1) * x0

Usage

```
## S3 method for class 'mcr'
bias(
  x,
  mdl,
  level = 0.95,
  interval = c("", "confidence", "prediction", "both"),
  ci_method = c("auto", "analytic", "jackknife", "rank", "bootstrap_percentile",
    "bootstrap_standard"),
  bootstrap_replicates = 5000L,
  seed = NULL,
  ...
)
```

Arguments

x	mcr object (must call regression() first)
mdl	Medical decision levels (numeric vector) on the reference/X scale
level	Confidence level, default 0.95
interval	Interval type: "" (point estimate, default), "confidence" (confidence interval), "prediction" (prediction interval), "both" (confidence + prediction)
ci_method	Parameter-uncertainty method. "auto" uses percentile Bootstrap for Passing-Bablok and otherwise reuses the regression method.

bootstrap_replicates
: Number of paired Bootstrap resamples; at least 5000 when a Bootstrap method is used.

seed
: Optional integer seed.

...
: Additional arguments (currently unused).

Details

Bias follows the conventional candidate-minus-reference direction; a positive value indicates that the candidate method is higher.

Value

Updated mcr object (invisible), results stored in $bias

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
obj <- regression(obj, method = "ols")
bias(obj, mdl = c(200, 400, 600))
bias(obj, mdl = c(200, 400, 600), interval = "both")
```

---

bland_altman.mcr *Bland-Altman analysis (S3 method)*

Description

Calculate Limits of Agreement between two measurement methods, supporting three y-axis metrics: difference, ratio, and percent difference.

Usage

```
## S3 method for class 'mcr'
bland_altman(
  x,
  type = c("difference", "ratio", "percent"),
  x_axis = c("mean", "candidate", "reference"),
  percent_denominator = c("reference", "mean", "candidate"),
  agree.level = 0.95,
  conf.level = 0.95,
  method = c("parametric", "nonparametric", "hodges_lehmann"),
  median_ci_method = c("order_statistic", "interpolated"),
  bootstrap_replicates = 5000L,
  seed = NULL,
  ...
)
```

Arguments

x	mcr object
type	Y-axis metric: "difference" (default), "ratio" or "percent"
x_axis	X-axis: "mean" (average of the two methods, default), "candidate" (test method values), "reference" (reference method values)
percent_denominator	Denominator used when type = "percent": "reference" (default), "mean" (average of candidate and reference), or "candidate". This is independent of x_axis.
agree.level	Agreement level for limits of agreement, default 0.95 (i.e., 95% LoA)
conf.level	Confidence level for LoA confidence intervals, default 0.95
method	Estimation method: "parametric" uses the mean, its t confidence interval, the standard deviation, and parametric limits of agreement; "nonparametric" uses the median, an exact sign-test order-statistic interval, empirical limits, and percentile Bootstrap confidence intervals for the limits. "hodges_lehmann" uses the Walsh-average pseudomedian and an uncorrected Wilcoxon/Tukey interval; its LoA remain empirical quantiles with percentile Bootstrap intervals.
median_ci_method	For the ordinary nonparametric median, either the conservative order-statistic interval (default) or the EP09c continuous-rank normal approximation with linear interpolation between adjacent order statistics.
bootstrap_replicates	Number of Bootstrap resamples for the nonparametric method, default 5000. Must be at least 100.
seed	Optional integer random seed for reproducible Bootstrap intervals.
...	Additional arguments

Value

Updated mcr object (invisible), with results stored in $bland_altman. The result records the selected method, its centre estimate and center_ci. Method-specific aliases are mean_diff_ci for the parametric mean and median_diff_ci for the nonparametric median. The original mean_diff and sd_diff fields are retained for compatibility.

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
bland_altman(obj)
bland_altman(obj, type = "ratio")
bland_altman(obj, type = "percent", x_axis = "reference")
bland_altman(obj, type = "percent", x_axis = "mean",
             percent_denominator = "mean")
```

`bottle_anova` *bottle_anova constructor*

Description

Create a bottle/batch ANOVA analysis object. Supports one-way and nested designs, with automatic assumption checks, missing value reporting, and ANOVA table output.

Usage

```
bottle_anova(data, formula, conf.level = 0.95)
```

Arguments

`data`	A data frame
`formula`	A formula, e.g. `value ~ batch` or `value ~ batch/vial`
`conf.level`	Confidence level (default 0.95)

Value

An S3 object of class `"bottle_anova"` with components:

`call`	Function call
`data`	Original data
`formula`	Input formula
`conf.level`	Confidence level
`response_name`	Response variable name
`design_type`	Design type (`"one_way"` or `"nested"`)
`rhs_vars`	Right-hand side grouping variable names
`term_labels`	Model term labels
`n_total`	Total sample size
`n_complete`	Number of complete cases
`missing_rows`	Indices of excluded missing rows
`cc_data`	Complete-case data frame
`aov_fit`	`aov` fitted object
`aov_table`	ANOVA table
`desc_stats`	Descriptive statistics table
`desc_group_col`	Grouping column label
`assumptions`	Assumption test results (Bartlett + Shapiro-Wilk)
`group_means`	Group means
`tukey_results`	Tukey HSD results (NULL initially, filled by `tukey()`)

Examples

```
# One-way design
df1 <- data.frame(
  bottle = rep(c("A", "B", "C"), each = 5),
  value  = c(rnorm(5, 10, 1), rnorm(5, 12, 1), rnorm(5, 11, 1))
)
obj <- bottle_anova(df1, value ~ bottle)

# Nested design
df2 <- data.frame(
  lot   = rep(c("L1", "L2"), each = 10),
  vial  = rep(c("V1", "V2"), each = 5, times = 2),
  value = c(rnorm(5, 10, 1), rnorm(5, 10.5, 1),
            rnorm(5, 12, 1), rnorm(5, 12.5, 1))
)
obj2 <- bottle_anova(df2, value ~ lot/vial)
```

---

c5_c95 *Estimate C5 and C95 from an internal continuous response*

Description

Estimates response values associated with selected probabilities of a qualitative result. The response SD is modeled as a function of the expected response using a VFP precision profile. For `direction = "geq"`, the event is `result >= cutoff`; for `direction = "leq"`, the event is `result <= cutoff`. No acceptance decision is made.

Usage

```
c5_c95(
  data,
  result = NULL,
  sample = NULL,
  cutoff,
  direction = c("geq", "leq"),
  probabilities = c(0.05, 0.95),
  component = NULL,
  model.no = 1:10,
  K = 2,
  search_range = NULL,
  tol = 1e-07
)
```

Arguments

`data`	A data frame of replicate continuous responses, or a `precision` object containing a fitted precision profile.
`result`	Name of the numeric response column. Required for a data frame.

sample	Name of the sample or level column. Required for a data frame.
cutoff	One or more finite decision cutoffs.
direction	Direction defining the condition of interest: "geq" means response greater than or equal to the cutoff; "leq" means response less than or equal to the cutoff.
probabilities	Probabilities to estimate, default c(0.05, 0.95).
component	Precision-profile component used when data is a precision object. Defaults to "total" when available.
model.no	Candidate VFP model numbers for raw data.
K	Fixed exponent supplied to applicable VFP models.
search_range	Optional two-element finite numeric range for root finding. By default, the observed mean range is expanded automatically.
tol	Positive numeric root-finding tolerance.

Value

An object of class c5_c95. Its estimates element contains the estimated response, fitted SD, achieved probability, extrapolation flag, and root-search diagnostics for each cutoff and probability.

Examples

```
set.seed(1203)
cx_data <- data.frame(
  level = rep(paste0("L", 1:7), each = 20),
  response = unlist(lapply(seq(34, 46, length.out = 7), function(mu)
    stats::rnorm(20, mean = mu, sd = 0.7 + 0.01 * mu)))
)
cx <- c5_c95(cx_data, result = "response", sample = "level",
             cutoff = 40, model.no = 1)
cx$estimates
plot(cx)
```

---

ci.precision *S3 confidence interval method*

Description

Compute confidence intervals for each variance component (SD and %CV) based on Satterthwaite approximation. Requires variance() to be run first to populate the $results slot.

Usage

```
## S3 method for class 'precision'
ci(x, digits = 4, ...)
```

Arguments

x	precision object
digits	Number of decimal places, default 4
...	Reserved arguments

Details

This method reports intervals for the fitted model components and the total component. EP05 semantic combinations such as multisite within-laboratory precision require the day, run, and site mapping supplied to report.precision and are therefore constructed by report(x, ..., table = "CI").

Value

Updated precision object with results stored in $ci

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- variance(obj)
obj <- ci(obj)
```

---

coef.fit_equation *Extract coefficients from an equation fit*

Description

Extract coefficients from an equation fit

Usage

```
## S3 method for class 'fit_equation'
coef(object, ...)
```

Arguments

object	A fit_equation object.
...	Additional arguments (currently unused).

Value

A named numeric vector of model coefficients.

Examples

```
df <- data.frame(x = 1:5, y = c(2.1, 4.0, 6.2, 7.9, 10.1))
f <- fit_equation("E01", df, "x", "y")
coef(f)
```

commutability *Assess reference-material commutability using four standard approaches*

Description

Implements the statistical calculations for CLSI EP14-A3 Deming-regression prediction intervals, CLSI EP30-A prediction intervals and relative residuals, the IFCC Part 2 difference-in-bias approach, or the IFCC Part 3 calibration-effectiveness approach. The function expects long-format data and does not assess specimen collection, run randomization, reagent lots, or other experimental conduct.

Usage

```
commutability(
  data,
  sample_id,
  material_type = NULL,
  procedure,
  result,
  position = NULL,
  approach = c("ep14", "ep30", "ifcc", "calibration"),
  scale = c("raw", "log10", "ln"),
  reference_procedure = NULL,
  procedure_pairs = NULL,
  conf.level = 0.95,
  criterion = NULL,
  bias_model = c("constant_global", "constant_local", "linear_local"),
  local_n = NULL,
  coverage_factor = 1.9,
  rm_position = c("pooled", "material_specific"),
  rm_set_size = 1L,
  relative_residual_limit = 2,
  calibration_stage = NULL,
  calibration_replicate_summary = c("mean", "median", "error"),
  exclude_procedures = NULL,
  w3_reference_procedure = NULL,
  material_type_map = NULL
)
```

Arguments

`data`	A data frame in long format.
`sample_id`, `material_type`, `procedure`, `result`	Column names identifying the material, material type, measurement procedure, and numeric result. `material_type` may be `NULL` for `approach = "calibration"`.

`position` Optional column identifying RM positions within a run; required for `approach = "ifcc"`.

`approach` One of `"ep14"`, `"ep30"`, `"ifcc"` (IFCC Part 2), or `"calibration"` (IFCC Part 3).

`scale` Analysis scale: `"raw"`, `"log10"`, or `"ln"`. EP14 permits raw or log10; EP30 permits all three scales; IFCC Part 2 permits raw or natural-log analysis; calibration effectiveness requires raw results.

`reference_procedure` Optional reference measurement procedure. When supplied, it is compared with every other procedure.

`procedure_pairs` Optional list of two-element procedure vectors.

`conf.level` Prediction-interval level for EP14 and EP30.

`criterion` Predefined IFCC Part 2 criterion on the selected analysis scale, or maximum IMPBR percentage for calibration effectiveness. It is mandatory for the two IFCC-derived approaches and is never estimated from data.

`bias_model` IFCC clinical-sample bias model: `"constant_global"`, `"constant_local"`, or `"linear_local"`.

`local_n` Number of clinical samples around each RM for a local IFCC model. Ignored for a global model.

`coverage_factor` IFCC uncertainty coverage factor; default 1.9.

`rm_position` IFCC position-mean uncertainty mode. `"pooled"` pools position-mean variances across reference materials within each procedure, as in the IFCC Part 2 worked example. `"material_specific"` uses each reference material's own position-mean variance.

`rm_set_size` Number of related reference materials used as a set for the EP30 normal critical-value adjustment. Independent materials use 1.

`relative_residual_limit` Positive EP30 relative-residual limit; default 2.

`calibration_stage` Column identifying `"before"` and `"after"` candidate-RM recalibration results; required for `approach = "calibration"`.

`calibration_replicate_summary` How multiple calibration results in the same sample, procedure, and stage are reduced to one reported result: `"mean"` (default), `"median"`, or `"error"` to reject duplicates.

`exclude_procedures` Optional procedures explicitly excluded after investigation in a calibration-effectiveness analysis. Full-set results are retained.

`w3_reference_procedure` Procedure whose SD/W(3) ratio scales the robust W(3) statistics. The first observed procedure is used when `NULL`.

material_type_map

Optional named character vector that explicitly maps observed material-type values to "clinical" or "rm", for example c("patient sample" = "clinical", "candidate RM" = "rm"). When supplied, this mapping is used as given. When NULL, the function automatically recognizes "clinical", "patient", and "native" as clinical samples, and "rm" and "reference_material" as reference materials. If automatic recognition fails, the error lists the unrecognized values and requests this argument.

Details

Sample-size, replicate-count, position-matching, and method-count targets from CLSI and IFCC publications are reported in design_checks; they are not treated as proof of compliance and do not stop a calculation that is statistically defined. Conditions required by the formula itself still produce an error. A local result that cannot be calculated is returned as not_evaluable, while an unavailable secondary statistic is returned as NA.

Value

An object of class commutability_result containing material-level results, model estimates, design checks, IFCC position diagnostics, and the transformed analysis data. Design checks report whether the supplied study meets the sample-size and replication recommendations of the named method; unmet checks do not prevent otherwise computable analyses.

Examples

```
# Long-format replicate results for eight clinical samples.
clinical <- expand.grid(
  sample = paste0("CS", 1:8), procedure = c("MP1", "MP2"),
  replicate = 1:3, stringsAsFactors = FALSE
)
level <- setNames(seq(10, 80, length.out = 8), paste0("CS", 1:8))
clinical$result <- level[clinical$sample] +
  ifelse(clinical$procedure == "MP2", 1, 0) +
  rep(c(-0.2, 0, 0.2), each = 16)
clinical$material_type <- "clinical"
clinical$position <- NA_integer_

# One reference material measured at three positions, in triplicate.
rm <- expand.grid(
  sample = "RM1", procedure = c("MP1", "MP2"),
  position = 1:3, replicate = 1:3, stringsAsFactors = FALSE
)
rm$result <- 45 + ifelse(rm$procedure == "MP2", 1, 0) +
  rep(c(-0.1, 0, 0.1), each = 6)
rm$material_type <- "rm"
columns <- c("sample", "material_type", "procedure", "result",
             "position", "replicate")
example_data <- rbind(clinical[columns], rm[columns])

ep14 <- commutability(
```

```
  example_data, "sample", "material_type", "procedure", "result",
  approach = "ep14"
)
ep14$results

# Explicitly map nonstandard material-type labels.
mapped_data <- example_data
mapped_data$material_type <- ifelse(
  mapped_data$material_type == "clinical", "patient_sample", "candidate_rm"
)
ep14_mapped <- commutability(
  mapped_data, "sample", "material_type", "procedure", "result",
  approach = "ep14",
  material_type_map = c(patient_sample = "clinical", candidate_rm = "rm")
)
ep14_mapped$results

ifcc <- commutability(
  example_data, "sample", "material_type", "procedure", "result",
  position = "position", approach = "ifcc", criterion = 2,
  bias_model = "constant_global"
)
ifcc$results
```

`compare_equation` *Fit and compare multiple equations*

Description

Fit and compare multiple equations

Usage

```
compare_equation(data, x = "x", y = "y", eqs = NULL, ...)
```

Arguments

`data`	a data frame
`x`	x column name
`y`	y column name
`eqs`	equation name/ID vector; NULL = all Response equations
`...`	additional arguments passed to fit_equation

Value

An object of class "compare_equation" with components:

`table`	data.frame sorted by AIC: Eq, Name, Formula, nPar, AIC, deltaAIC, R2, RSE

fits	list of fit_equation objects, named by eq_id
failed	character vector of eq IDs that failed to fit
data, x, y	input data reference

Examples

```
df <- data.frame(
  x = c(0.1, 0.3, 1, 3, 10, 30, 100),
  y = c(12, 25, 48, 72, 88, 96, 99)
)
compare_equation(df, "x", "y", eqs = c("E01", "E06", "E07"))
```

---

continuous_to_binary *Convert quantitative columns to binary based on cutoff values*

Description

Creates new binary columns (0/1) from existing quantitative columns using specified cutoff values, appending them to the original data frame. New columns are named <col>_binary.

Usage

```
continuous_to_binary(data, cols, cutoff)
```

Arguments

data	data frame
cols	character vector of column names to binarize
cutoff	numeric vector of cutoff values; recycled to length(cols) if a single value is given

Value

data frame with original columns plus new <col>_binary columns

Examples

```
df <- data.frame(x = 1:10, y = rnorm(10))
continuous_to_binary(df, "x", 5)
continuous_to_binary(df, c("x", "y"), c(5, 0))
```

correlation.mcr *Correlation analysis (S3 method)*

Description

Perform correlation analysis between the reference method (X) and candidate method (Y), supporting Pearson, Spearman, and Kendall methods.

Usage

```
## S3 method for class 'mcr'
correlation(x, method = c("pearson", "spearman", "kendall"), ...)
```

Arguments

x	mcr object
method	Correlation method: "pearson" (default), "spearman", or "kendall"
...	Additional arguments passed to cor.test

Value

Updated mcr object (invisible), results stored in $correlation

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
correlation(obj)
correlation(obj, method = "spearman")
```

counts_to_table *Construct a 2x2 contingency table from TP / FP / TN / FN frequencies (silent construction)*

Description

When raw data are not available and only the four diagnostic test frequencies are known, use this function to directly construct a result object with the same structure as raw_to_table(), compatible with print() for displaying the fourfold table.

Usage

```
counts_to_table(
  tp,
  fp,
  tn,
  fn,
  candidate = "candidate",
  reference = "reference",
  candidate_levels = c("positive", "negative"),
  reference_levels = c("positive", "negative")
)
```

Arguments

tp	True positives (candidate=level1, reference=level1)
fp	False positives (candidate=level1, reference=level2)
tn	True negatives (candidate=level2, reference=level2)
fn	False negatives (candidate=level2, reference=level1)
candidate	Candidate method name (character)
reference	Reference method name (character)
candidate_levels	Two levels of the candidate method, default c("positive", "negative")
reference_levels	Two levels of the reference method, default c("positive", "negative")

Value

A fourfold_table object (S3 class "fourfold_table") with the same structure as raw_to_table(): - $freq_raw Long-format frequency data.frame - $table 3x3 matrix with margins - $n Total sample size - $print Print slot - $candidate_levels Levels of the candidate method - $reference_levels Levels of the reference method Use print() directly to display the fourfold table.

Examples

```
result <- counts_to_table(tp = 45, fp = 12, tn = 80, fn = 5,
                          candidate = "new method", reference = "gold standard")
```

---

cutoff.roc *Optimal Cutoff Analysis (S3 method)*

Description

For each evaluation column, compute diagnostic metrics at all cutoff values, and identify the optimal cutoff (maximum Youden index, closest to (0,1) corner).

Usage

```
## S3 method for class 'roc'
cutoff(x, cols = NULL, ...)
```

Arguments

x	roc object
cols	column names to analyze; default is all
...	reserved arguments

Value

Updated roc object with results stored in $cutoff_table

Examples

```
obj <- roc(ivd_roc_example, cols = c("x1", "x2"), reference = "ref")
obj <- auc(obj)
cutoff(obj)
cutoff(obj, cols = "x1")
```

---

describe *Describe an analysis object*

Description

Describe an analysis object

Usage

```
describe(x, ...)

correlation(x, ...)

regression(x, ...)

bias(x, ...)

bland_altman(x, ...)

report(x, ...)

profile(object, ...)

normal(x, ...)

ci(x, ...)
```

```
variance(x, ...)

vc(x, ...)

diagnostics(object, ...)

kappa(x, ...)

mcnemar(x, ...)

cutoff(x, ...)

mlr(x, ...)

auc(x, ...)

auc_compare(x, ...)

tukey(x, ...)
```

Arguments

x	An S3 analysis object.
...	Additional arguments passed to a method.
object	An S3 analysis object.

Value

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

The value returned by the dispatched method.

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
describe(obj)
```

`describe.fourfold_table` *Descriptive Statistics: Raw Data Overview (S3 Method)*

Description

Print variable description information for a fourfold_table object (from raw_to_table). If the object comes from counts_to_table (no raw data), indicates unavailability.

Usage

```
## S3 method for class 'fourfold_table'
describe(x, ...)
```

Arguments

`x`	a fourfold_table object
`...`	reserved arguments

Value

Updated fourfold_table object, with $describe slot filled with description results

Examples

```
df <- data.frame(
  new    = c("positive","positive","negative","negative","positive","negative"),
  gold   = c("positive","negative","positive","negative","positive","positive"),
  age    = c(45, 52, 38, 61, 47, 55),
  gender = c("M","F","F","M","M","F"),
  id     = c("S001","S002","S003","S004","S005","S006")
)
tab <- raw_to_table(df, "new", "gold", id = "id")
tab <- describe(tab)
```

describe.mcr *Compute descriptive statistics and store results*

Description

Computes data overview (rows/columns/ID duplicates), per-column summaries, and a side-by-side comparison table of candidate, reference, and Diff. Results are stored in `x$describe` (class `mcr_describe`) and displayed via `print.mcr_describe()` (e.g., when calling `summary()`).

Usage

```
## S3 method for class 'mcr'
describe(x, cols = NULL, digits = 4L, ...)
```

Arguments

`x`	mcr object
`cols`	Column name vector to analyze; by default, analyzes all columns except id, candidate, and reference. Supports exclusion with - prefix, e.g., cols = -c("age", "sex").
`digits`	Number of decimal places, default 4
`...`	Additional arguments

Value

Updated `mcr` object (invisible), results stored in `x$describe`

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
describe(obj)
```

describe.roc *Descriptive Statistics (S3 method)*

Description

Print distribution summaries for evaluation columns and the reference column (if numeric). When the reference is binary, evaluation column distributions are shown grouped by positive/negative.

Usage

```
## S3 method for class 'roc'
describe(x, cols = NULL, digits = 4L, ...)
```

Arguments

x	roc object
cols	columns to describe; defaults to all evaluation columns + reference column (if numeric)
digits	number of decimal places, default 4
...	reserved arguments

Value

Updated roc object with results stored in $describe

Examples

```
obj <- roc(ivd_roc_example, cols = "x1", reference = "ref")
describe(obj)
```

---

diagnostics.fourfold_table

Diagnostic performance metrics: calculate sensitivity, specificity, likelihood ratios, etc. with confidence intervals for a fourfold_table

---

Description

Diagnostic performance metrics: calculate sensitivity, specificity, likelihood ratios, etc. with confidence intervals for a fourfold_table

Usage

```
## S3 method for class 'fourfold_table'
diagnostics(
  object,
  conf.level = 0.95,
  ci.method = "wilson",
  prevalence = NULL,
  ...
)
```

Arguments

object	a fourfold_table object
conf.level	Confidence level, default 0.95
ci.method	Proportion confidence interval method, default "wilson". Supports "wald", "wald-cc", "wilson", "wilson-cc", "agresti-coull", "jeffreys", "clopper-pearson"
prevalence	User-specified prevalence. If provided, PPV/NPV will be calculated based on this prevalence via Bayes' theorem (rather than the observed prevalence in the data).
...	Additional arguments passed to print

Value

Updated fourfold_table object, with diagnostic performance metrics stored in $diagnostics: - $tp / $fp / $fn / $tn / $n Four-fold frequencies and total sample size - $sensitivity Sensitivity with CI - $specificity Specificity with CI - $ppv Positive predictive value with CI - $npv Negative predictive value with CI - $accuracy Accuracy with CI - $prevalence Prevalence with CI - $lr_positive Positive likelihood ratio with CI - $lr_negative Negative likelihood ratio with CI - $odds_ratio Odds ratio with CI - $conf_level Confidence level used - $ci_method Method name used - $prev_specified Whether user specified prevalence

Examples

```
tab <- raw_to_table(ivd_qualitative_example, "new", "gold",
                    positive = "positive", id = "id")
tab <- diagnostics(tab)
tab <- diagnostics(tab, prevalence = 0.3)
tab <- diagnostics(tab, conf.level = 0.90)
tab <- diagnostics(tab, ci.method = "clopper-pearson")
```

---

dilution_recovery *Calculate dilution recovery*

Description

Summarizes replicate measurements and calculates target concentration, dilution-corrected concentration, and recovery for each specimen and dilution factor. Across-specimen mean recovery and its Student t confidence interval are calculated for every dilution factor. No acceptance criterion or suitability decision is applied.

Usage

```
dilution_recovery(
  data,
  sample,
  result,
  dilution_factor,
  assigned = NULL,
  diluent_value = 0,
  conf.level = 0.95
)
```

Arguments

data	Data frame in long format, with one row per measurement.
sample	Sample-identification column.
result	Numeric measurement-result column.

dilution_factor
: Numeric dilution-factor column. Undiluted material is represented by 1, a 1/2 dilution by 2, and a 1/20 dilution by 20.

assigned
: Optional column containing one assigned original-sample concentration per sample. When omitted, the mean undiluted result is used.

diluent_value
: Measurand concentration in the diluent. The default is 0.

conf.level
: Confidence level for the across-sample mean recovery.

Value

An object of class `dilution_recovery` containing `results`, `summary`, the retained data, and excluded row indices.

References

CLSI. Establishing and Verifying an Extended Measuring Interval Through Specimen Dilution and Spiking. EP34, 1st ed. 2018.

Examples

```
dilution_data <- data.frame(
  sample = rep(c("S1", "S2"), each = 6),
  factor = rep(rep(c(1, 2, 4), each = 2), 2),
  result = c(99, 101, 49, 51, 24, 26,
             199, 201, 99, 101, 49, 51)
)
dilution <- dilution_recovery(
  dilution_data, "sample", "result", "factor"
)
dilution$summary
```

---

fit_equation *General-purpose equation fitting*

Description

General-purpose equation fitting

Usage

```
fit_equation(
  eq,
  data,
  x = "x",
  y = "y",
  start = NULL,
  lower = NULL,
  upper = NULL,
```

```
  constraints = NULL,
  weights = NULL,
  ...
)
```

Arguments

eq	equation identifier: ID("E07"), name("4PLC"), or a custom formula string ("y ~ a*exp(-b*x)")
data	a data frame
x	x variable column name (default "x")
y	y variable column name (default "y")
start	named list of starting values (NULL=auto)
lower	named list/vector of lower bounds (e.g. list(Vmax=0, Km=0))
upper	named list/vector of upper bounds
constraints	list of constraint functions: list(ineq = f, eq = g) f(par) returns a vector; all values must be non-positive g(par) returns a vector; must satisfy all(g(par) = 0)
weights	weights: NULL(equal) / numeric vector / method name string methods: "equal", "1/y", "1/y^2", "1/x", "1/x^2"
...	additional arguments passed to nlsLM() / nloptr()

Value

an S3 object of class "fit_equation"

Examples

```
# ==== Response curve fitting ====

# 1) Linear
df <- data.frame(x = 1:5, y = c(2.1, 4.0, 6.2, 7.9, 10.1))
f1 <- fit_equation("E01", df, "x", "y")
print(f1)

# 2) Logarithmic
f2 <- fit_equation("E06", df, "x", "y")
print(f2)

# 3) 4PLC (dose-response, descending)
df4plc <- data.frame(
  conc = c(0.1, 0.3, 1, 3, 10, 30, 100),
  resp = c(99, 96, 88, 72, 48, 25, 12)
)
f3 <- fit_equation("E07", df4plc, "conc", "resp")
print(f3)

# 4) Custom formula
f4 <- fit_equation("a + b * log(x)", df4plc, "conc", "resp")
```

```
print(f4)

# 5) Box constraints (parameter bounds)
f5 <- fit_equation("E07", df4plc, "conc", "resp",
  lower = list(D = 0), upper = list(D = 5))
print(f5)

# 6) Inequality constraints (parameter relationship)

# require D > 1 (Hill slope cannot be too shallow)
f6 <- fit_equation("E07", df4plc, "conc", "resp",
  constraints = list(ineq = function(p) 1 - p["D"]))
print(f6)


# 7) Equality constraints (fixed parameter relationship)

# fix b = 0.5 in exponential decay y = a*exp(-b*x)
df_exp <- data.frame(x = 0:5, y = c(10, 6.1, 3.7, 2.2, 1.4, 0.8))
f7 <- fit_equation("E04", df_exp, "x", "y",
  constraints = list(eq = function(p) p["b"] - 0.5))
print(f7)


# 8) Weighted fitting via replicate_to_mean

# data with replicate measurements (8 dose levels, 3 replicates each)
df_rep <- data.frame(
  dose = rep(c(0.3, 0.6, 1.5, 4, 10, 25, 60, 150), each = 3),
  resp = c(4.1, 3.8, 3.9, 8.5, 8.9, 8.6,
           18.2, 17.9, 18.5,
           34.6, 35.1, 34.8,
           54.3, 53.9, 54.7,
           73.0, 73.6, 73.2,
           87.5, 87.1, 87.8,
           97.9, 98.3, 98.0)
)
# first summarize, generate inverse-variance weights
rep <- replicate_to_mean(df_rep, "dose", "resp", weights = "1/sd^2")

# fit 4PLC with summarized means and explicit start values
f8 <- fit_equation("E07", rep, x = "x", y = "y_mean",
  weights = rep$weight,
  start = list(A = 2, B = 100, C = 8, D = -1.5))
print(f8)
```

---

hook_effect *Summarize a high-dose hook-effect experiment*

Description

Summarizes raw response by known concentration and identifies the first post-peak measured concentration whose mean response is at or below the raw response associated with the ULoQ. The reported `hook_concentration` is the immediately preceding measured concentration; no interpolation is used. No hook-effect or extended-interval acceptability decision is made.

Usage

```
hook_effect(
  data,
  concentration,
  response,
  series = NULL,
  uloq,
  uloq_response = NULL,
  conf.level = 0.95
)
```

Arguments

`data`	Data frame containing known concentrations and raw responses.
`concentration`	Numeric known-concentration column.
`response`	Numeric raw-response column.
`series`	Optional column identifying independent hook experiments.
`uloq`	Finite concentration representing the upper limit of quantitation.
`uloq_response`	Optional raw response at the ULoQ. It may be one finite value for all series or one value per series. When omitted, each series must contain measurements at `concentration == uloq`.
`conf.level`	Confidence level for mean raw response at each concentration.

Value

An object of class `hook_effect` containing concentration-level summaries and one threshold result per series.

References

CLSI. Establishing and Verifying an Extended Measuring Interval Through Specimen Dilution and Spiking. EP34, 1st ed. 2018.

Examples

```
hook_data <- data.frame(
  concentration = rep(c(100, 200, 400, 800), each = 2),
  response = c(48, 52, 98, 102, 148, 152, 88, 92)
)
hook <- hook_effect(
  hook_data, "concentration", "response", uloq = 200
```

```
)
hook$results
```

`interference_dose_response` *Analyze an EP07 interference dose-response experiment*

Description

Summarizes replicate results at each interferent concentration relative to an observed baseline concentration. The recommended point-to-point method is the default; a straight-line regression can be requested explicitly.

Usage

```
interference_dose_response(
  data,
  result,
  concentration,
  by = NULL,
  baseline = 0,
  method = c("point_to_point", "linear"),
  conf.level = 0.95
)
```

Arguments

`data`	Long-format data frame.
`result`	Numeric measurement-result column.
`concentration`	Numeric interferent-concentration column.
`by`	Optional stratification columns, typically interferent name and measurand level.
`baseline`	Observed interferent concentration used to estimate M0.
`method`	`"point_to_point"` or `"linear"`.
`conf.level`	Confidence level for regression coefficients.

Value

An object of class `interference_dose_response`.

Examples

```
dose_data <- expand.grid(
  interferent = c("A", "B"), concentration = c(0, 10, 20, 30, 40),
  replicate = 1:5, KEEP.OUT.ATTRS = FALSE
)
dose_data$result <- 100 + ifelse(dose_data$interferent == "A", 0.5, -0.3) *
```

```
  dose_data$concentration + stats::rnorm(nrow(dose_data), 0, 0.5)
dose_fit <- interference_dose_response(
  dose_data, "result", "concentration", by = "interferent")
dose_fit$summary
```

`interference_paired` *Analyze paired-difference interference data*

Description

Supports the standard EP07 test/control preparation and two laboratory extensions: analyte spiking in normal/interferent matrices and direct measurement of interferent dissolved in a blank. One row represents one replicate result. Multiple interferents or measurand levels are handled by supplying their columns in `by`.

Usage

```
interference_paired(
  data,
  result,
  design = c("ep07", "analyte_spike", "blank"),
  by = NULL,
  condition = NULL,
  test_level = "test",
  control_level = "control",
  matrix = NULL,
  normal_level = "normal",
  interferent_level = "interferent",
  spike = NULL,
  unspiked_level = "unspiked",
  spiked_level = "spiked",
  conf.level = 0.95
)
```

Arguments

`data`	Data frame in long format.
`result`	Numeric measurement-result column.
`design`	`"ep07"`, `"analyte_spike"`, or `"blank"`.
`by`	Optional character vector of stratification columns.
`condition`	Test/control condition column for `ep07`, or optional condition column for `blank`.
`test_level`	Value identifying the test/interferent condition.
`control_level`	Value identifying the solvent control condition.
`matrix`	Matrix-type column for `analyte_spike`.
`normal_level`	Value identifying the normal matrix.

interferent_level	Value identifying the interferent-containing matrix.
spike	Spike-status column for analyte_spike.
unspiked_level	Value identifying measurements before analyte spiking.
spiked_level	Value identifying measurements after analyte spiking.
conf.level	Confidence level for intervals.

Value

An S3 object of class interference_paired containing effects, cell_summary, issues, and excluded row indices.

Examples

```
ep07_data <- data.frame(
  interferent = rep(c("bilirubin", "hemoglobin"), each = 10),
  condition = rep(rep(c("control", "test"), each = 5), 2),
  result = c(0.45, 0.56, 0.48, 0.54, 0.47,
             0.24, 0.28, 0.35, 0.37, 0.26,
             5.02, 4.98, 5.08, 4.95, 5.00,
             5.20, 5.12, 5.18, 5.10, 5.15)
)
interference_paired(ep07_data, "result", design = "ep07",
                    condition = "condition", by = "interferent")

spike_data <- expand.grid(
  matrix = c("normal", "interferent"), spike = c("before", "after"),
  replicate = 1:4, KEEP.OUT.ATTRS = FALSE
)
spike_data$result <- with(spike_data,
  ifelse(matrix == "normal", 10, 12) +
  ifelse(spike == "after", ifelse(matrix == "normal", 10, 8), 0))
interference_paired(spike_data, "result", design = "analyte_spike",
                    matrix = "matrix", spike = "spike",
                    unspiked_level = "before", spiked_level = "after")
```

---

interference_patient *Evaluate interference with patient specimens*

Description

Averages replicates within specimen and method, then calculates the evaluated procedure result minus the comparative procedure result. Potential interferent concentrations can be supplied as multiple numeric columns and are modeled separately.

Usage

```
interference_patient(
  data,
  id,
  group,
  method,
  result,
  test_group,
  control_group,
  evaluated_method,
  comparative_method,
  interferent_cols = NULL,
  conf.level = 0.95
)
```

Arguments

data	Long-format patient-result data.
id	Specimen identifier column.
group	Patient test/control group column.
method	Measurement-procedure column.
result	Numeric result column.
test_group	Value identifying the selected/test patient group.
control_group	Value identifying the control patient group.
evaluated_method	Value identifying the evaluated procedure.
comparative_method	Value identifying the comparative procedure.
interferent_cols	Optional numeric interferent-concentration columns.
conf.level	Confidence level for summaries and regression coefficients.

Value

An object of class `interference_patient`.

Examples

```
patient_data <- expand.grid(
  id = paste0("S", 1:12), method = c("evaluated", "comparative"),
  replicate = 1:2, KEEP.OUT.ATTRS = FALSE
)
patient_data$group <- ifelse(as.integer(sub("S", "", patient_data$id)) <= 6,
                             "control", "test")
patient_data$bilirubin <- 2 * (as.integer(sub("S", "", patient_data$id)) - 1)
truth <- 50 + as.integer(sub("S", "", patient_data$id))
```

```
patient_data$result <- truth +
  ifelse(patient_data$method == "evaluated", 0.08 * patient_data$bilirubin, 0)
patient_fit <- interference_patient(
  patient_data, "id", "group", "method", "result",
  test_group = "test", control_group = "control",
  evaluated_method = "evaluated", comparative_method = "comparative",
  interferent_cols = "bilirubin")
patient_fit$group_summary
```

`interference_replicates` *Calculate replicate counts for an EP07 paired-difference experiment*

Description

Calculates the number of replicate measurements needed in each of the test and control samples. `sd` and `detectable_difference` must use the same scale: both absolute units, or both relative units (SD/CV and percent). The calculation is vectorized and reports design parameters only.

Usage

```
interference_replicates(
  sd,
  detectable_difference,
  alpha = 0.05,
  power = 0.9,
  alternative = c("two.sided", "one.sided"),
  min_replicates = 5L,
  label = NULL
)
```

Arguments

`sd`	Assumed repeatability SD, or repeatability CV when a relative calculation is intended.
`detectable_difference`	Smallest interference difference the experiment is designed to detect, in the same units as `sd`.
`alpha`	Type I error probability.
`power`	Desired statistical power.
`alternative`	`"two.sided"` or `"one.sided"`.
`min_replicates`	Minimum returned replicate count per test/control sample.
`label`	Optional labels, one per calculation.

Value

A data frame of class `interference_replicates`.

References

CLSI. Interference Testing in Clinical Chemistry. EP07, 3rd ed.

Examples

```
interference_replicates(sd = c(5, 5),
                        detectable_difference = c(7, 10),
                        label = c("7 percent", "10 percent"))
```

`ivd_bottle_example` *Small deterministic one-way ANOVA example*

Description

Small deterministic one-way ANOVA example

Usage

```
ivd_bottle_example
```

Format

A data frame with 12 observations from three bottles.

`ivd_mcr_example` *Small deterministic method-comparison example*

Description

Small deterministic method-comparison example

Usage

```
ivd_mcr_example
```

Format

A data frame with 12 rows and columns `sid`, `test`, and `ref`.

ivd_qualitative_example *Small deterministic qualitative-method example*

Description

Small deterministic qualitative-method example

Usage

```
ivd_qualitative_example
```

Format

A data frame with paired binary results and sample identifiers.

ivd_roc_example *Small deterministic ROC example*

Description

Small deterministic ROC example

Usage

```
ivd_roc_example
```

Format

A data frame with 12 rows, two markers, and a binary reference.

kappa.fourfold_table *Cohen's Kappa / PABAK Agreement Analysis (2x2 table only)*

Description

Calculate Cohen's Kappa coefficient or PABAK (Prevalence-Adjusted Bias-Adjusted Kappa) and its confidence interval, used to evaluate agreement between two binary classification methods.

Usage

```
## S3 method for class 'fourfold_table'
kappa(x, prevalence = NULL, conf.level = 0.95, ...)
```

Arguments

x	a fourfold_table object
prevalence	Patient prevalence. If provided, calculates PABAK instead of Cohen's Kappa.
conf.level	Confidence level, default 0.95
...	Additional arguments (currently unused).

Details

When data prevalence is extreme, Cohen's Kappa may be low even when observed agreement is high. In such cases, specify the prevalence parameter to compute PABAK as a correction. PABAK = 2 x p_obs - 1, which assumes expected agreement = 0.5.

Value

Updated fourfold_table object, with Kappa results stored in $kappa: - $kappa Kappa / PABAK coefficient - $se Standard error - $lower / $upper Lower/upper confidence interval bounds - $p_obs Observed agreement - $p_exp Expected agreement - $n Total sample size - $conf_level Confidence level used - $method Method name used

Examples

```
tab <- counts_to_table(tp = 85, fp = 3, tn = 90, fn = 2,
                       candidate = "new method", reference = "gold standard")
tab <- kappa(tab)                        # Cohen's Kappa
tab <- kappa(tab, prevalence = 0.5)      # PABAK
```

---

linearity *Evaluate linearity using straight-line or polynomial regression*

Description

linearity() is a parameter-driven analysis engine rather than a separate validation/verification workflow. The straight-line calculations implement the approach in CLSI EP06-Ed2. Polynomial regression reproduces the superseded EP6-A procedure and is reported as exploratory. Straight-line models are fitted to sample-level means; EP6-A polynomial models are fitted to the retained replicate observations.

Usage

```
linearity(
  data,
  sample,
  result,
  x,
  method = c("linear", "polynomial"),
  intercept = TRUE,
```

```
  weights = c("equal", "level_variance", "pooled", "precision_profile"),
  variance_group = NULL,
  max_degree = 3L,
  poly_alpha = 0.05,
  percent_base = c("predicted", "x", "mean"),
  conf.level = NULL,
  multiplicity = c("familywise", "none"),
  adl = NULL,
  adl_abs = NULL,
  adl_rel = NULL,
  profile_exclude_low = TRUE,
  profile_intercept = FALSE
)
```

Arguments

`data`	Data frame in long format, with one row per replicate result.
`sample`	Sample-level identifier column name.
`result`	Numeric measurement-result column name.
`x`	Numeric expected value, assigned value, relative concentration, or HIGH-proportion column name. Its value must be constant within a sample.
`method`	`"linear"` or `"polynomial"`.
`intercept`	Include an intercept in every fitted model?
`weights`	One of `"equal"`, `"level_variance"`, `"pooled"`, or `"precision_profile"`; alternatively, a numeric vector with one weight per observation or one weight per sample level (in increasing x order).
`variance_group`	When `weights = "pooled"`, a single character string naming the column that assigns every sample level to an explicitly chosen contiguous variance group. The value must be constant within a sample, and every group must contain at least two sample levels. It must be `NULL` for other weighting methods.
`max_degree`	Highest polynomial degree, 2 or 3.
`poly_alpha`	Significance level for EP6-A nonlinear coefficients.
`percent_base`	Denominator for percentage residuals and polynomial deviation from linearity: the straight-line prediction, x value, or observed sample mean. EP06-Ed2 examples use the prediction.
`conf.level`	Optional overall or pointwise confidence level for straight-line residuals. Use `NULL` to omit confidence intervals.
`multiplicity`	`"familywise"` adjusts intervals so their joint coverage is `conf.level`; `"none"` applies `conf.level` to each level.
`adl`	Optional direct absolute ADL, scalar or one value per level.
`adl_abs`	Optional fixed absolute ADL.
`adl_rel`	Optional relative ADL in percent. If more than one ADL form is supplied, the largest allowable value is used at each level.

profile_exclude_low
: Exclude the lowest-x level from the precision profile and use its observed SD directly?

profile_intercept
: Include an intercept in the SD-on-mean precision profile? EP06 uses a zero intercept.

Details

With `weights = "pooled"`, variance groups must be supplied explicitly. Within group g, the pooled variance is `sum((n[j] - 1) * s[j]^2) / sum(n[j] - 1)`, and the level-mean regression weight is `n[j] / pooled_variance[g]`. The function does not infer groups from the number of concentration levels because EP06 grouping also depends on the observed repeatability pattern.

Value

An S3 object of class `linearity`.

Examples

```
set.seed(2020)
linearity_data <- data.frame(
  level = rep(1:6, each = 4),
  expected = rep(c(10, 30, 50, 70, 90, 110), each = 4)
)
linearity_data$result <- 2 + 0.98 * linearity_data$expected +
  stats::rnorm(nrow(linearity_data), sd = 1.5)

# Parameter-driven straight-line analysis.
fit_linear <- linearity(
  linearity_data, sample = "level", result = "result", x = "expected",
  method = "linear", intercept = TRUE, weights = "level_variance",
  adl_rel = 5
)
fit_linear$model_comparison
fit_linear$level_summary

# EP6-A polynomial analysis (reported as exploratory under EP06-Ed2).
fit_polynomial <- linearity(
  linearity_data, sample = "level", result = "result", x = "expected",
  method = "polynomial", max_degree = 3
)
fit_polynomial$coefficients
```

`linearity_endpoints` *Calculate target LOW and HIGH concentrations for verification*

Description

Implements the equations in Appendix J of CLSI EP06-Ed2 for studies in which results outside the measuring interval cannot be obtained.

Usage

```
linearity_endpoints(
  lloq = NULL,
  uloq = NULL,
  cv_high = NULL,
  cv_at_lloq = NULL,
  cv_at_k_lloq = NULL,
  k = 3,
  cv_scale = c("percent", "proportion")
)

## S3 method for class 'linearity_endpoints'
print(x, ...)
```

Arguments

`lloq`	Lower limit of quantitation. Supply with `cv_at_lloq` and `cv_at_k_lloq` to calculate the LOW target.
`uloq`	Upper limit of quantitation. Supply with `cv_high` to calculate the HIGH target.
`cv_high`	Repeatability CV near the upper limit.
`cv_at_lloq`	CV at the LLoQ.
`cv_at_k_lloq`	CV at `k * lloq`.
`k`	Concentration multiple for `cv_at_k_lloq`; EP06 uses 3.
`cv_scale`	Whether CV inputs are percentages or proportions.
`x`	A `linearity_endpoints` object.
`...`	Reserved arguments.

Value

A list of class `linearity_endpoints`.

The unchanged `x`, invisibly.

Examples

```
linearity_endpoints(
  lloq = 10, uloq = 1000,
  cv_high = 5, cv_at_lloq = 20, cv_at_k_lloq = 15
)
```

`linearity_panel` *Design a HIGH/LOW linearity panel*

Description

Design a HIGH/LOW linearity panel

Usage

```
linearity_panel(
  proportion_high,
  high,
  low = 0,
  total_volume = NULL,
  sample = NULL
)

## S3 method for class 'linearity_panel'
print(x, ...)
```

Arguments

`proportion_high`	Proportion of the HIGH material at each level, between 0 and 1.
`high`	Assigned or measured value of the HIGH material.
`low`	Assigned or measured value of the LOW material; use zero for a HIGH/blank design.
`total_volume`	Optional total volume prepared at each level.
`sample`	Optional sample labels.
`x`	A `linearity_panel` object.
`...`	Reserved arguments.

Value

A data frame containing proportions, expected values, relative concentrations, and (when requested) component volumes.

The unchanged `x`, invisibly.

Examples

```
linearity_panel(
  proportion_high = c(1, 0.75, 0.5, 0.25, 0),
  high = 210, low = 10, total_volume = 1
)
```

`linearity_replicates` *Calculate replicate counts for a linearity study*

Description

Calculates the minimum number of replicates from the imprecision and allowable deviation from linearity (ADL), following CLSI EP06-Ed2. Relative inputs use percentage units (for example, `5` means 5%). Absolute inputs must use the same measurement unit for `imprecision`, `adl`, and `true_deviation`.

Usage

```
linearity_replicates(
  imprecision,
  adl,
  type = c("relative", "absolute"),
  level_probability = 0.99,
  levels = NULL,
  family_error = NULL,
  true_deviation = 0,
  min_replicates = 2L
)

## S3 method for class 'linearity_replicates'
print(x, ...)
```

Arguments

`imprecision`	Repeatability CV (percent) or SD (absolute units).
`adl`	Allowable deviation from linearity.
`type`	Either `"relative"` or `"absolute"`.
`level_probability`	Desired central probability for an individual level. The EP06 default is 0.99.
`levels`	Optional number of concentration levels. Required when `family_error` is supplied.
`family_error`	Optional overall risk that at least one level is outside the ADL. When supplied, it replaces `level_probability`.
`true_deviation`	Expected true deviation from linearity. It is subtracted from `adl` before calculating the replicate count.
`min_replicates`	Smallest returned replicate count.
`x`	A `linearity_replicates` object.
`...`	Reserved arguments.

Value

A data frame of class `linearity_replicates`.

The unchanged `x`, invisibly.

Examples

```
# Relative ADL: both CV and ADL are expressed in percent.
linearity_replicates(imprecision = c(3, 5), adl = 5)

# Choose the per-level probability from a 10% family-wise failure risk.
linearity_replicates(5, 5, levels = 9, family_error = 0.10)
```

`list_equation` *Print the equation registry*

Description

Print the equation registry

Usage

```
list_equation(category = NULL, engine = NULL)
```

Arguments

`category`	filter category: "Response", or NULL (all)
`engine`	filter engine: "lm", "nls", or NULL (all)

Value

invisible data.frame (matching rows), prints a formatted table to the console

Examples

```
# All equations
list_equation()

# Response only
list_equation(category = "Response")

# lm engine only
list_equation(engine = "lm")
```

list_sadler *List Sadler precision profile model equations*

Description

List Sadler precision profile model equations

Usage

```
list_sadler()
```

Arguments

This function has no arguments.

Details

List the 10 candidate Sadler precision profile model formulas and types used in the VFP package. Includes: Constant SD, Constant CV, Linear (variance), Power model, Exponential model, etc.

Value

Invisibly returns `NULL`. The model table is printed to the console as a side effect.

Examples

```
list_sadler()
```

list_westgard *List Westgard QC rules*

Description

Prints commonly used Westgard multi-rule QC rules and their meanings. Rule names printed here can be passed to `qc_chart(rules = "...")`.

Usage

```
list_westgard(brief = FALSE)
```

Arguments

`brief`	Set to `TRUE` to suppress printing and return only the named character vector.

Value

A named character vector of rule descriptions (invisibly).

Examples

```
list_westgard()
```

`mcnemar.fourfold_table` *McNemar Test (for fourfold_table only)*

Description

Perform McNemar's test on a 2x2 paired contingency table to evaluate whether there is a systematic difference between two binary classification methods (i.e., whether marginal probabilities are equal).

Usage

```
## S3 method for class 'fourfold_table'
mcnemar(x, conf.level = 0.95, ...)
```

Arguments

`x`	a fourfold_table object
`conf.level`	Confidence level, default 0.95
`...`	Additional arguments (currently unused).

Details

When discordant pairs (b + c) are few (< 25), automatically uses the exact binomial test (binom.test); otherwise uses McNemar's chi-squared test with continuity correction.

Value

Updated fourfold_table object, with McNemar results stored in $mcnemar: - $chi_sq McNemar chi-squared statistic (NA for exact test) - $df Degrees of freedom (NA for exact test) - $p_value Test p-value - $method Test method used - $b Discordant pairs (positive, negative) count - $c Discordant pairs (negative, positive) count - $n_bc Total discordant pairs - $conf.level Confidence level used

Examples

```
tab <- counts_to_table(tp = 45, fp = 12, tn = 80, fn = 5,
                       candidate = "new method", reference = "gold standard")
tab <- mcnemar(tab)
```

mcr *mcr constructor*

Description

Create a method comparison regression (mcr) object containing one summarized row per sample ID. Raw replicate rows must first be summarized with `replicate_to_mean()` or by the user.

Usage

```
mcr(
  data,
  id,
  candidate,
  reference,
  weights = NULL,
  candidate_sd = NULL,
  reference_sd = NULL,
  candidate_n = NULL,
  reference_n = NULL
)
```

Arguments

`data`	Data frame with id, candidate, reference columns
`id`	ID variable name (character), used to identify samples
`candidate`	Candidate/test method variable name (character), used as the response (Y) throughout regression and plotting.
`reference`	Reference/comparative method variable name (character), used as the predictor (X) throughout regression and plotting.
`weights`	Optional column name (character) in data containing non-negative finite numeric weights for weighted regression. Default NULL (no weighting).
`candidate_sd`, `reference_sd`	Optional column names containing within-sample SDs for summarized replicate data. Supply both when `lambda = "replicates"` will be used with one mean pair per sample.
`candidate_n`, `reference_n`	Optional column names containing replicate counts for summarized data. Supply both or neither. Without them, equal replicate counts are assumed and cancel from the variance ratio.

Value

Returns an S3 "mcr" object containing:

`call`	Original function call

`data`	Complete data frame
`id` / `id_name`	ID vector and column name
`candidate` / `reference`	Test/reference method numeric vectors (complete pairs only)
`candidate_name` / `reference_name`	Method column names
`n`	Number of complete pairs
`complete_idx`	Row indices of complete pairs in the original data

and analysis result storage slots: $correlation, $regression, $outlier, $bland_altman

Examples

```
df <- data.frame(sid = 1:30,
                 test = rnorm(30, 50, 10),
                 ref  = rnorm(30, 50, 10))
obj <- mcr(df, "sid", "test", "ref")
```

`mkt` *mkt — Mean Kinetic Temperature*

Description

Calculates the mean kinetic temperature from one or more temperature probes. Supports equal-interval and time-weighted trapezoidal methods.

Usage

```
mkt(data, temp_cols, time = NULL, temp_unit = "C", ea = 83.144)
```

Arguments

`data`	A data frame.
`temp_cols`	One or more column names for temperature probes.
`time`	Optional time column; accepts numeric, Date, or POSIXct.
`temp_unit`	Temperature unit: `"C"` or `"K"`.
`ea`	Activation energy in kJ/mol; default 83.144.

Value

An S3 object of class `"mkt"`.

Examples

```
temp_df <- data.frame(t1 = c(25,26,27), t2 = c(24,25,26), time = 1:3)
mkt(temp_df, c("t1","t2"), "time")
```

mlr.roc *Multivariate Logistic Regression (S3 method)*

Description

Fit logistic regression using selected evaluation columns, compute AUC of predicted probabilities. Names must be unique; auto-generated as MLR01, MLR02 ... when not specified.

Usage

```
## S3 method for class 'roc'
mlr(x, cols = NULL, name = NULL, ...)
```

Arguments

x	roc object
cols	column names to fit; default is all
name	fit name (must be unique); auto-generated by default
...	additional arguments passed to glm

Value

Updated roc object with results stored in $mlr_results[name]

Examples

```
obj <- roc(ivd_roc_example, cols = c("x1", "x2"), reference = "ref")
obj <- auc(obj)
obj <- mlr(obj)                    # -> MLR01
obj <- mlr(obj)                    # -> MLR02
obj <- mlr(obj, name = "my_model")  # -> my_model
```

normal.precision *S3 normality test method*

Description

Perform normality tests on the response values for each sample, underlying call to `normal_test()`. Supports automatic test selection (Shapiro-Wilk, Anderson-Darling, Lilliefors, CVM).

Usage

```
## S3 method for class 'precision'
normal(
  x,
  method = c("auto", "shapiro", "sw", "ad", "lillie", "cvm"),
  level = 0.95,
  ...
)
```

Arguments

x	precision object
method	Test method: "auto" (default, automatic selection), "shapiro", "sw", "ad", "lillie", "cvm"
level	Confidence level, default 0.95
...	Reserved arguments

Value

Updated precision object with results stored in $normal. Each sample entry contains test_method, statistic_name, statistic, p_value, and n; W is retained as a legacy alias only for Shapiro-Wilk results.

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- normal(obj)
```

---

normal_test *Normality test*

Description

Performs normality tests on a single numeric column of a data frame. Plots are generated separately via plot().

Usage

```
normal_test(
  data,
  col,
  method = c("auto", "shapiro", "sw", "ad", "lillie", "cvm"),
  level = 0.95
)
```

Arguments

data	A data frame.
col	Column name to test (single string).
method	Test method: "auto" (default, chooses by sample size), "shapiro"/"sw", "ad", "lillie", "cvm".
level	Confidence level (default 0.95, used for QQ confidence bands).

Value

An S3 object of class "normal_test" containing:

call Function call.

data_name Data object name.

col Column name.

method User-specified method.

level Confidence level.

n_total Total number of rows.

n Number of finite values (non-missing).

missing Number of missing values.

test_method Actual test method used.

statistic Test statistic.

p_value p-value.

y Clean numeric vector, used by plot().

Examples

```
df <- data.frame(x = rnorm(50))
normal_test(df, "x")
normal_test(df, "x", method = "ad")
plot(normal_test(df, "x"))
plot(normal_test(df, "x"), type = "his")
plot(normal_test(df, "x"), type = c("qq", "his"))
```

---

outlier *Detect outliers in an analysis object*

Description

Detect outliers in an analysis object

Usage

```
outlier(x, ...)
```

Arguments

x	An S3 analysis object.
...	Additional arguments passed to a method.

Value

The value returned by the dispatched method.

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
outlier(obj, method = "grubbs")
```

---

outlier.mcr *Outlier detection (S3 method)*

Description

Perform outlier detection based on the difference or percent difference between two measurements. Reuses four methods from ../precision/outliers.R.

Usage

```
## S3 method for class 'mcr'
outlier(
  x,
  method = c("grubbs", "esd", "dixon", "iqr"),
  type = c("difference", "percent"),
  percent_denominator = c("reference", "mean", "candidate"),
  alpha = 0.05,
  coef = 1.5,
  ...
)
```

Arguments

x	mcr object
method	Outlier detection method: "grubbs" (default), "esd", "dixon", "iqr"
type	Data type to compute: "difference" (default) or "percent" (percent difference)
percent_denominator	Denominator for percent differences: reference (default), pair mean, or candidate.
alpha	Significance level, default 0.05 (Grubbs/ESD/Dixon only)
coef	IQR multiplier, default 1.5 (IQR only)
...	Additional arguments passed to esd_test / dixon_test

Value

Updated mcr object (invisible), results stored in $outlier

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
outlier(obj, method = "grubbs")
outlier(obj, method = "iqr", type = "percent")
```

---

outlier.precision *S3 outlier detection method*

Description

Detect outliers in the response values for each sample, supporting Grubbs test and IQR method. Underlying call to `outliers_test()` (from `outliers-and-normal.R`).

Usage

```
## S3 method for class 'precision'
outlier(x, alpha = 0.05, method = c("grubbs", "iqr"), ...)
```

Arguments

`x`	`precision` object
`alpha`	Significance level, default `0.05` (Grubbs only)
`method`	Detection method: `"grubbs"` (default) or `"iqr"`
`...`	Additional arguments passed to `outliers_test()`

Value

Updated `precision` object with results stored in `$outlier`

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- outlier(obj, method = "iqr")
```

`outliers_test` *Outlier detection*

Description

Integrates Grubbs test, generalized ESD test, Dixon Q test, and IQR method under a consistent API with an S3 print method.

Usage

```
outliers_test(data, col, method = c("grubbs", "esd", "dixon", "iqr"), ...)
```

Arguments

`data`	A data frame.
`col`	Column name to test (single string).
`method`	Detection method: "grubbs" (default), "esd", "dixon", "iqr".
`...`	Additional arguments passed to internal methods:
	`alpha` Significance level for grubbs / esd / dixon (default 0.05). dixon only accepts 0.01 and 0.05.
	`r` Maximum number of outliers for ESD (default `min(5, n-2)`).
	`type` Tail to test for Dixon: `"both"` (default), `"min"`, `"max"`.
	`coef` IQR multiplier (default 1.5).

Value

An S3 object of class `"outliers_test"` containing:

`method` Method name used.

`data_name` Name of the input data.

`col` Column name tested.

`n_total` Total number of rows.

`n` Number of finite observations used.

`missing` Number of missing (non-finite) values.

`parameters` List of method parameters.

`n_outliers` Number of outliers detected.

`indices` Indices of outliers in the original vector (1-based).

`values` Outlier values.

`details` Method-specific details (statistics, critical values, bounds, etc.).

Examples

```
set.seed(123)
df <- data.frame(x = c(rnorm(20), 10, -8))
outliers_test(df, "x", "grubbs")
outliers_test(df, "x", "esd", r = 3)
outliers_test(df, "x", "dixon")
outliers_test(df, "x", "iqr", coef = 2)
```

---

plot.arrhenius *Plot Arrhenius stability analysis*

Description

Plot Arrhenius stability analysis

Usage

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

Arguments

x	An arrhenius object.
...	Reserved for the S3 generic.

Value

A named list of ggplot objects, invisibly. The kinetic-fit and Arrhenius-regression plots are also drawn.

Examples

```
d <- data.frame(
  temperature = rep(c(25, 30, 37), each = 3),
  time = rep(c(0, 1, 3), 3),
  value = c(100, 98, 95, 100, 96, 90, 100, 94, 85)
)
fit <- arrhenius(d, "temperature", "time", "value",
                 target_temp = 5, limit = 10)
plot(fit)
```

plot.c5_c95 *Plot C5 and C95 results*

Description

Plot C5 and C95 results

Usage

```
## S3 method for class 'c5_c95'
plot(x, type = c("probability", "profile"), ...)
```

Arguments

x	A c5_c95 object.
type	Plot type: "probability" or "profile".
...	Reserved arguments.

Value

A ggplot object.

Examples

```
set.seed(95)
d <- data.frame(level = rep(1:6, each = 12),
                y = stats::rnorm(72, rep(seq(36, 44, length.out = 6), each = 12), 1))
z <- c5_c95(d, "y", "level", cutoff = 40, model.no = 1)
plot(z, type = "probability")
```

plot.commutability_result *Plot a commutability assessment*

Description

Plot a commutability assessment

Usage

```
## S3 method for class 'commutability_result'
plot(x, method = NULL, ...)
```

Arguments

x	A commutability_result object.
method	For EP30, either "prediction" or "relative_residual". Ignored by the other approaches.
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
clinical <- expand.grid(
  sample = paste0("CS", 1:8), procedure = c("MP1", "MP2"),
  replicate = 1:3, stringsAsFactors = FALSE
)
level <- setNames(seq(10, 80, length.out = 8), paste0("CS", 1:8))
clinical$result <- level[clinical$sample] +
  ifelse(clinical$procedure == "MP2", 1, 0) +
  rep(c(-0.2, 0, 0.2), each = 16)
clinical$material_type <- "clinical"
clinical$position <- NA_integer_
rm <- expand.grid(
  sample = "RM1", procedure = c("MP1", "MP2"),
  position = 1:3, replicate = 1:3, stringsAsFactors = FALSE
)
rm$result <- 45 + ifelse(rm$procedure == "MP2", 1, 0) +
  rep(c(-0.1, 0, 0.1), each = 6)
rm$material_type <- "rm"
columns <- c("sample", "material_type", "procedure", "result",
             "position", "replicate")
example_data <- rbind(clinical[columns], rm[columns])
assessment <- commutability(
  example_data, "sample", "material_type", "procedure", "result"
)
plot(assessment)
```

---

plot.dilution_recovery

Plot dilution-recovery results

Description

Plot dilution-recovery results

Usage

```
## S3 method for class 'dilution_recovery'
plot(x, type = c("recovery", "summary", "observed"), ...)
```

Arguments

x	A dilution_recovery object.
type	"recovery", "summary", or "observed".
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
dilution_data <- data.frame(
  sample = rep(c("S1", "S2"), each = 6),
  factor = rep(rep(c(1, 2, 4), each = 2), 2),
  result = c(99, 101, 49, 51, 24, 26,
             199, 201, 99, 101, 49, 51)
)
dilution <- dilution_recovery(
  dilution_data, "sample", "result", "factor"
)
plot(dilution)
```

---

plot.fit_equation *Plot an equation fit*

Description

Plot an equation fit

Usage

```
## S3 method for class 'fit_equation'
plot(x, interval = "confidence", level = 0.95, frame = FALSE, ...)
```

Arguments

x	A fit_equation object.
interval	Interval type, either "confidence" or "prediction".
level	Confidence level.
frame	Whether to draw a frame.
...	Additional arguments (currently unused).

Value

x, invisibly.

Examples

```
df4plc <- data.frame(
  conc = c(0.1, 0.3, 1, 3, 10, 30, 100),
  resp = c(12, 25, 48, 72, 88, 96, 99)
)
f <- fit_equation("E07", df4plc, "conc", "resp")
plot(f)
plot(f, interval = "prediction", frame = TRUE)
```

`plot.hook_effect` *Plot a high-dose hook-effect experiment*

Description

Plot a high-dose hook-effect experiment

Usage

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

Arguments

`x`	A `hook_effect` object.
`...`	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
hook_data <- data.frame(
  series = rep("S1", 8),
  concentration = rep(c(10, 20, 40, 80, 160, 320, 640, 1280), each = 2),
  response = c(10, 11, 20, 21, 39, 40, 70, 71,
               95, 94, 80, 79, 55, 54, 30, 29)
)
hook <- hook_effect(hook_data, "concentration", "response",
                    series = "series", uloq = 320)
plot(hook)
```

plot.interference_dose_response

Plot interference dose-response results

Description

Plot interference dose-response results

Usage

```
## S3 method for class 'interference_dose_response'
plot(x, type = c("response", "absolute", "percent"), ...)
```

Arguments

x	An interference_dose_response object.
type	"response", "absolute", or "percent".
...	Unused.

Value

A ggplot object.

Examples

```
d <- expand.grid(conc = c(0, 10, 20), rep = 1:5)
d$result <- 100 + 0.5 * d$conc
plot(interference_dose_response(d, "result", "conc"), type = "percent")
```

plot.interference_paired

Plot paired interference estimates

Description

Plot paired interference estimates

Usage

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

Arguments

x	An interference_paired object.
...	Unused.

Value

A ggplot object.

Examples

```
d <- data.frame(group = rep(c("control", "test"), each = 5),
                result = c(10, 10.1, 9.9, 10.2, 9.8, 11, 11.1, 10.9, 11.2, 10.8))
plot(interference_paired(d, "result", "ep07", condition = "group"))
```

---

plot.interference_patient

Plot patient-specimen interference results

Description

Plot patient-specimen interference results

Usage

```
## S3 method for class 'interference_patient'
plot(x, type = c("difference", "interferent"), interferent = NULL, ...)
```

Arguments

x	An interference_patient object.
type	"difference" or "interferent".
interferent	Interferent concentration column to plot when type = "interferent".
...	Unused.

Value

A ggplot object.

Examples

```
d <- expand.grid(id = paste0("S", 1:8), method = c("test", "ref"), rep = 1:2)
d$group <- ifelse(as.integer(sub("S", "", d$id)) <= 4, "control", "selected")
d$bilirubin <- 2 * (as.integer(sub("S", "", d$id)) - 1)
d$result <- 20 + as.integer(sub("S", "", d$id)) +
  ifelse(d$method == "test", 0.1 * d$bilirubin, 0)
fit <- interference_patient(d, "id", "group", "method", "result",
  "selected", "control", "test", "ref", "bilirubin")
plot(fit, type = "difference")
```

plot.linearity *Plot a linearity analysis*

Description

Plot a linearity analysis

Usage

```
## S3 method for class 'linearity'
plot(
  x,
  type = c("fit", "replicates", "precision", "residual", "difference"),
  ...
)
```

Arguments

x	A linearity object.
type	One of "fit", "replicates", "precision", "residual", or "difference".
...	Reserved for future use.

Value

A ggplot object.

Examples

```
set.seed(6)
d <- data.frame(level = rep(1:5, each = 3), x = rep(1:5, each = 3))
d$y <- 1 + 2 * d$x + stats::rnorm(nrow(d), sd = 0.2)
fit <- linearity(d, "level", "y", "x")
plot(fit, type = "fit")
plot(fit, type = "residual")
```

plot.mcr *Plot (S3 method) – unified plotting interface*

Description

Draw one of four plot types for an mcr object.

Usage

```
## S3 method for class 'mcr'
plot(
  x,
  type = c("scatter", "regression", "bland_altman", "bias"),
  interval = c("none", "confidence", "prediction", "both"),
  conf.level = NULL,
  mdl = NULL,
  ...
)
```

Arguments

x	mcr object
type	Plot type: "scatter" (scatter + identity line, default), "regression" (regression line + confidence/prediction bands), "bland_altman" (Bland-Altman plot), "bias" (bias trend plot)
interval	Interval type (only for type="regression"): "none" (default), "confidence", "prediction", "both"
conf.level	Confidence level; if NULL, reads from analysis slot, otherwise uses this value
mdl	Medical decision levels (only for type="bias"); if NULL, tries to read from x$bias$mdl
...	Additional arguments

Value

Invisible ggplot object

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
plot(obj)
plot(obj, type = "scatter")
obj <- regression(obj)
plot(obj, type = "regression", interval = "confidence")
obj <- bland_altman(obj)
plot(obj, type = "bland_altman")
obj <- bias(obj, mdl = c(200, 400))
plot(obj, type = "bias")
```

---

plot.normal_test *Plot normality diagnostics*

Description

Plot normality diagnostics

Usage

```
## S3 method for class 'normal_test'
plot(x, type = c("qq", "his"), ...)
```

Arguments

x	A normal_test object.
type	One or both of "qq" and "his".
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly; requested plots are drawn as a side effect. For fewer than three observations no plot is drawn.

Examples

```
d <- data.frame(value = rnorm(30))
nt <- normal_test(d, "value")
plot(nt, type = c("qq", "his"))
```

---

plot.precision *S3 plot method*

Description

Draw one of five plot types for a precision object. dot (run-order scatter plot) and his (histogram + N(0,1) density) require no prior analysis; var (variance component bar chart) requires variance() to be run first; qq (normal Q-Q plot) requires normal() first; profile (precision profile) requires profile() first.

Usage

```
## S3 method for class 'precision'
plot(x, type = c("dot", "his", "var", "qq", "profile"), ...)
```

Arguments

x	precision object
type	Plot type: "dot" (run-order scatter plot, default preferred), "his" (histogram), "var" (variance component plot), "qq" (normal Q-Q plot), "profile" (precision profile)
...	Reserved arguments

Value

Invisible ggplot object

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
plot(obj, type = "dot")
plot(obj, type = "his")
```

---

plot.qc_chart *Plot a Levey-Jennings QC chart*

Description

Plot a Levey-Jennings QC chart

Usage

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

Arguments

x	A qc_chart object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly; the chart is drawn as a side effect.

Examples

```
set.seed(42)
d <- data.frame(value = rnorm(30, 100, 5), run = 1:30)
plot(qc_chart(d, "value", run = "run"))
```

---

plot.reference_interval *Plot reference interval results*

Description

One panel per method: jitter strip chart overlaid with reference interval limits, endpoints, and CI error bars.

Usage

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

Arguments

x	A "reference_interval" object.
...	Additional arguments (reserved for generic).

Value

A ggplot object, invisibly. The plot is also drawn as a side effect.

Examples

```
set.seed(11)
d <- data.frame(value = rnorm(120, 100, 15))
ri <- reference_interval(d, "value")
plot(ri)
```

---

plot.roc *Plot ROC or cutoff-dependent diagnostic metrics*

Description

Draw ROC curves for evaluation columns and/or MLR model predictions, or draw sensitivity and specificity as functions of the cutoff for one evaluation column. Optionally mark optimal cutoff points (requires cutoff.roc() first).

Usage

```
## S3 method for class 'roc'
plot(x, curves = NULL, cutpoint = NULL, type = c("roc", "cutoff"), ...)
```

Arguments

x	roc object
curves	ROC curve names to plot. Names can refer to original marker columns or stored MLR models. NULL (default) plots all original marker curves; "all" plots all original and MLR curves.
cutpoint	mark optimal cutoff points: NULL (default, none), "Youden", "Corner", or "all".
type	"roc" (default) draws the conventional ROC curve; "cutoff" draws sensitivity and specificity against finite cutoff values for exactly one evaluation column.
...	unused; supplying additional arguments is an error.

Value

invisible ggplot object

Examples

```
obj <- roc(ivd_roc_example, cols = c("x1", "x2"), reference = "ref")
obj <- auc(obj)
plot(obj)
plot(obj, curves = "x1")
# AUC calculation is not needed for the cutoff-dependent metric plot.
plot(roc(ivd_roc_example, cols = "x1", reference = "ref"),
     curves = "x1", type = "cutoff")
obj <- auc(obj); obj <- cutoff(obj); obj <- mlr(obj)
plot(obj, curves = "all", cutpoint = "Youden")
plot(obj, curves = "x1", type = "cutoff", cutpoint = "Youden")
```

---

plot.sensitivity *Plot an analytical-sensitivity analysis*

Description

Plot an analytical-sensitivity analysis

Usage

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

Arguments

x	A sensitivity-analysis result object.
...	Reserved for the S3 generic.

Value

A ggplot object.

Examples

```
d <- data.frame(sample = rep("blank", 10), result = seq(0.1, 1, length.out = 10))
fit <- sensitivity_lob(d, "result", "sample")
plot(fit)
```

plot.spike_recovery *Plot spike-recovery results*

Description

Plot spike-recovery results

Usage

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

Arguments

x	A spike_recovery object.
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
spike_data <- data.frame(
  sample = rep(c("S1", "S2"), each = 4),
  condition = rep(rep(c("spiked", "control"), each = 2), 2),
  result = c(20.1, 19.9, 10.0, 10.2, 30.2, 29.8, 20.1, 19.9)
)
recovery <- spike_recovery(
  spike_data, "sample", "result", "condition",
  analyte_level = "spiked", solvent_level = "control",
  added_concentration = 10
)
plot(recovery)
```

plot.stability_bias *Plot stability bias across storage conditions*

Description

Plot stability bias across storage conditions

Usage

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

Arguments

x	A stability_bias object.
...	Reserved for the S3 generic.

Value

A ggplot object, invisibly; the plot is also drawn.

Examples

```
d <- data.frame(
  condition = rep("2-8C", each = 8),
  time = rep(c(0, 1, 3, 7), 2),
  value = c(100, 99.8, 99.4, 98.9, 100.2, 99.9, 99.5, 98.8)
)
plot(stability_bias(d, "condition", "time", "value"))
```

---

plot.stability_regression

Plot a stability regression

Description

Plot a stability regression

Usage

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

Arguments

x	A stability_regression object.
...	Reserved for the S3 generic.

Value

A ggplot object, invisibly; the plot is also drawn.

Examples

```
d <- data.frame(
  condition = rep("2-8C", each = 8),
  time = rep(c(0, 1, 3, 7), 2),
  value = c(100, 99.8, 99.4, 98.9, 100.2, 99.9, 99.5, 98.8)
)
fit <- stability_regression(d, "time", "value", "condition")
plot(fit)
```

plot.stability_time *Plot stability limit-time predictions*

Description

Plot stability limit-time predictions

Usage

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

Arguments

x	A stability_time object.
...	Reserved for the S3 generic.

Value

A ggplot object, invisibly; the plot is also drawn.

Examples

```
d <- data.frame(
  condition = rep("2-8C", each = 8),
  time = rep(c(0, 1, 3, 7), 2),
  value = c(100, 99.8, 99.4, 98.9, 100.2, 99.9, 99.5, 98.8)
)
fit <- stability_regression(d, "time", "value", "condition")
limit_time <- stability_time(fit, limit = 5)
plot(limit_time)
```

plot.uncertainty_budget *Plot an uncertainty budget*

Description

Plot an uncertainty budget

Usage

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

Arguments

x	An uncertainty_budget object.
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
repeatability <- uncertainty_type_a(
  c(9.9, 10.0, 10.1, 10.0), "repeatability", quantity = "mean"
)
calibrator <- uncertainty_type_b(
  "calibrator", estimate = 10, uncertainty = 0.2, k = 2,
  distribution = "normal"
)
budget <- uncertainty_combine(repeatability, calibrator, value = 10)
plot(budget)
```

---

plot.uncertainty_propagation

Plot propagated uncertainty

Description

Plot propagated uncertainty

Usage

```
## S3 method for class 'uncertainty_propagation'
plot(x, type = c("distribution", "contribution"), ...)
```

Arguments

x	An uncertainty_propagation object.
type	"distribution" or "contribution".
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
numerator <- uncertainty_type_b(
  "numerator", estimate = 10, uncertainty = 0.2, k = 2,
  distribution = "normal"
)
denominator <- uncertainty_type_b(
  "denominator", estimate = 2, uncertainty = 0.04, k = 2,
  distribution = "normal"
)
propagated <- uncertainty_propagate(
  function(numerator, denominator) numerator / denominator,
  list(numerator, denominator), method = "delta"
)
plot(propagated, type = "contribution")
```

---

plot.uncertainty_traceability

Plot cumulative uncertainty through a traceability chain

Description

Plot cumulative uncertainty through a traceability chain

Usage

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

Arguments

x	An uncertainty_traceability object.
...	Reserved arguments.

Value

A ggplot object, invisibly.

Examples

```
calibration <- uncertainty_type_b(
  "calibration", estimate = 100, uncertainty = 1, k = 2,
  distribution = "normal"
)
transport <- uncertainty_type_b(
  "transport", estimate = 100, lower = 99.5, upper = 100.5,
  distribution = "rectangular"
)
chain <- uncertainty_traceability(
  list(calibration = list(calibration), transport = list(transport)),
```

```
  value = 100
)
plot(chain)
```

`plot.youden_plot` *Plot a Youden analysis*

Description

Plot a Youden analysis

Usage

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

Arguments

`x`	A `youden_plot` object.
`...`	Reserved for the S3 generic.

Value

The unchanged `x`, invisibly; the plot is drawn as a side effect.

Examples

```
set.seed(42)
d <- data.frame(m1 = rnorm(30, 100, 5), m2 = rnorm(30, 98, 5))
plot(youden_plot(d, "m1", "m2"))
```

`precision` *Precision variance component analysis*

Description

For each sample (grouped by `by`), estimate variance components via `VCA::anovaVCA`, compute SD / CV and their Satterthwaite confidence intervals.

Usage

```
precision(data, form, by = NULL, NegVC = FALSE, ...)
```

Arguments

data	Data frame containing the response variable and design factor columns.
form	Formula parsed by VCA (e.g. y ~ day/run/rep).
by	Column name(s) for sample grouping, supports multiple columns.
NegVC	Allow negative variance components? Default FALSE.
...	Additional arguments passed to VCA::anovaVCA.

Value

Returns an object of class precision. Use summary, profile, etc. to view results.

Examples

```
data(VCAdata1, package = "VCA")
precision(VCAdata1, y ~ day/run, by = "sample")

# More complex: deeply nested factors with more sample groups
data(realData, package = "VCA")
precision(realData, y ~ lot/calibration/day/run, by = "PID")
```

---

predict.fit_equation *Predict fitted values or inverse-predict (from y to x)*

Description

Predict fitted values or inverse-predict (from y to x)

Usage

```
## S3 method for class 'fit_equation'
predict(
  object,
  newdata = NULL,
  x = NULL,
  y = NULL,
  inverse = FALSE,
  interval = NULL,
  level = 0.95,
  ...
)
```

Arguments

object	a fit_equation object
newdata	new data as a data.frame
x	x column name; NULL uses the original x column from the fit
y	y column name (inverse only); NULL uses the original y column
inverse	logical; if TRUE, solve for x given y values in newdata
interval	"confidence" or "prediction"
level	confidence level (default 0.95)
...	additional arguments passed to the underlying predict()

Value

data.frame with x, y, and (if requested) confidence/prediction interval columns

Examples

```
df4plc <- data.frame(
  conc = c(0.1, 0.3, 1, 3, 10, 30, 100),
  resp = c(12, 25, 48, 72, 88, 96, 99)
)
f <- fit_equation("E07", df4plc, "conc", "resp")

# Fitted values
predict(f)

# Confidence interval
predict(f, interval = "confidence")

# Prediction interval
predict(f, interval = "prediction")

# Inverse prediction: estimate x for y=50
predict(f, newdata = data.frame(resp = 50), inverse = TRUE)
```

---

predict.interference_dose_response

Predict from an interference dose-response analysis

Description

Predict from an interference dose-response analysis

Usage

```
## S3 method for class 'interference_dose_response'
predict(
  object,
  newdata = NULL,
  target = NULL,
  metric = c("result", "absolute", "percent"),
  stratum = NULL,
  ...
)
```

Arguments

object	An interference_dose_response object.
newdata	Numeric interferent concentrations for forward prediction.
target	Numeric effects for inverse prediction. Supply instead of newdata.
metric	"result", "absolute", or "percent".
stratum	Optional stratum key; by default all strata are processed.
...	Unused.

Value

A data frame of predictions or inverse interpolations.

Examples

```
d <- expand.grid(conc = c(0, 10, 20, 30, 40), rep = 1:5)
d$result <- 100 + 0.5 * d$conc
fit <- interference_dose_response(d, "result", "conc")
predict(fit, newdata = c(5, 15), metric = "percent")
predict(fit, target = 10, metric = "percent")
```

---

predict.mcr *Predict (S3 method) – predict based on regression results*

Description

Predict values based on stored regression results.

Usage

```
## S3 method for class 'mcr'
predict(
  object,
  newdata = NULL,
  reference = NULL,
```

```
  candidate = NULL,
  inverse = FALSE,
  interval = c("none", "confidence", "prediction", "both"),
  conf.level = 0.95,
  ci_method = c("auto", "analytic", "jackknife", "rank", "bootstrap_percentile",
    "bootstrap_standard"),
  bootstrap_replicates = 5000L,
  seed = NULL,
  ...
)
```

Arguments

object	mcr object (must call regression() first)
newdata	Data frame containing input values. If provided, reference or candidate names a column in it.
reference	Reference-method input for forward prediction. With newdata, a column name; otherwise a numeric vector. If NULL, uses the original reference values.
candidate	Candidate-method input for inverse prediction. With newdata, a column name; otherwise a numeric vector. If NULL, uses the original candidate values.
inverse	If FALSE, predict candidate from reference. If TRUE, algebraically predict reference from candidate.
interval	Interval type: "none" (default), "confidence", "prediction", "both". Only forward prediction (inverse=FALSE) supports intervals.
conf.level	Confidence level, default 0.95
ci_method	Parameter-uncertainty method. "auto" reuses the stored regression method except that Passing-Bablok uses percentile Bootstrap.
bootstrap_replicates	Number of paired Bootstrap resamples; at least 5000 when a Bootstrap method is used.
seed	Optional integer seed.
...	Additional arguments

Value

data.frame containing predicted values and (if requested) interval bounds

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
obj <- regression(obj, method = "ols")
predict(obj, reference = c(40, 50, 60))
predict(obj, reference = c(40, 50, 60), interval = "both")
predict(obj, newdata = data.frame(xx = c(40, 50)), reference = "xx")
predict(obj, inverse = TRUE)  # Inverse prediction on original data
```

`predict.roc` *Predict (S3 method) – predict from multivariate logistic regression*

Description

Predict positive-class probability for new data using a stored MLR model. Returns the input predictor columns together with predicted probability, standard error, confidence interval, and 0/1 classification.

Usage

```
## S3 method for class 'roc'
predict(object, mlr = NULL, newdata = NULL, column_map = NULL, ...)
```

Arguments

`object`	roc object (must run mlr() first)
`mlr`	MLR model name (character). If only one MLR exists, defaults to it; if multiple exist, must specify.
`newdata`	data.frame of new observations. If NULL (default), uses the training data (complete cases) from the roc object.
`column_map`	Named character vector mapping newdata column names to training column names, e.g. `c(x1_new = "x1", x2_new = "x2")`. If NULL, tries to match by name automatically.
`...`	Additional arguments passed to stats::predict.glm

Value

data.frame containing the predictor columns used in the model, plus:

`pred_prob`	predicted probability of the positive class
`pred_se`	standard error of the predicted probability
`ci_lower`	lower bound of the confidence interval (probability scale)
`ci_upper`	upper bound of the confidence interval (probability scale)
`pred_class`	binary prediction (1 if pred_prob >= 0.5, 0 otherwise)

Examples

```
df <- data.frame(sid = 1:100,
                 x1  = c(rnorm(50, 10, 2), rnorm(50, 12, 2)),
                 x2  = rnorm(100, 5, 1),
                 ref = rep(c(0, 1), each = 50))
obj <- roc(df, cols = c("x1", "x2"), reference = "ref")
obj <- auc(obj)
obj <- mlr(obj)
predict(obj)
```

```
predict(obj, newdata = df[1:5, ])
## column_map example: rename x1 to new_x1 in newdata
new_df <- df[1:5, c("x2", "ref")]
new_df$new_x1 <- df$x1[1:5]
predict(obj, newdata = new_df,
        column_map = c(new_x1 = "x1", x2 = "x2"))
```

`print.arrhenius` *Print Arrhenius stability-analysis results*

Description

Print Arrhenius stability-analysis results

Usage

```
## S3 method for class 'arrhenius'
print(x, ...)
```

Arguments

`x`	An `arrhenius` object.
`...`	Reserved for the S3 generic.

Value

The unchanged `x`, invisibly.

`print.bottle_anova` *S3 print method: bottle_anova*

Description

Print bottle ANOVA analysis results including descriptive statistics, ANOVA table, and assumption checks.

Usage

```
## S3 method for class 'bottle_anova'
print(x, ...)
```

Arguments

`x`	A `bottle_anova` object
`...`	Other arguments (unused, for S3 generic signature)

Value

Invisibly returns `x`.

print.c5_c95 *Print C5 and C95 estimates*

Description

Print C5 and C95 estimates

Usage

```
## S3 method for class 'c5_c95'
print(x, digits = 4L, ...)
```

Arguments

x	A c5_c95 object.
digits	Number of digits shown.
...	Reserved arguments.

Value

Invisibly returns x.

Examples

```
set.seed(12)
d <- data.frame(level = rep(1:5, each = 10),
                y = stats::rnorm(50, rep(seq(37, 43, length.out = 5), each = 10), 1))
print(c5_c95(d, "y", "level", cutoff = 40, model.no = 1))
```

print.commutability_result *Print a commutability assessment*

Description

Print a commutability assessment

Usage

```
## S3 method for class 'commutability_result'
print(x, ...)
```

Arguments

x	A commutability_result object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

`print.compare_equation` *Print a multiple-equation comparison*

Description

Print a multiple-equation comparison

Usage

```
## S3 method for class 'compare_equation'
print(x, ...)
```

Arguments

`x`	A `compare_equation` object.
`...`	Reserved for the S3 generic.

Value

The unchanged `x`, invisibly.

`print.cutpoint_split` *Print a fixed-cutpoint data split*

Description

Print a fixed-cutpoint data split

Usage

```
## S3 method for class 'cutpoint_split'
print(x, ...)
```

Arguments

`x`	A `cutpoint_split` object.
`...`	Additional arguments (currently unused).

Value

The input object, invisibly.

print.describe_table *S3 print method for describe_table*

Description

S3 print method for describe_table

Usage

```
## S3 method for class 'describe_table'
print(x, ...)
```

Arguments

x	describe_table object
...	reserved arguments

Value

Invisible describe_table object

Examples

```
df <- data.frame(
  new    = c("positive","positive","negative","negative","positive","negative"),
  gold   = c("positive","negative","positive","negative","positive","positive"),
  age    = c(45, 52, 38, 61, 47, 55),
  gender = c("M","F","F","M","M","F"),
  id     = c("S001","S002","S003","S004","S005","S006")
)
tab <- raw_to_table(df, "new", "gold", id = "id")
tab <- describe(tab)
```

print.diagnostics *S3 print method: format* diagnostics *output (diagnostic performance metrics)*

Description

Prints four-fold table frequencies, sensitivity, specificity, PPV, NPV, accuracy, likelihood ratios, etc.

Usage

```
## S3 method for class 'diagnostics'
print(x, digits = 3, ...)
```

Arguments

x	diagnostics object
digits	Number of decimal places, default 3
...	reserved arguments

Value

Invisible diagnostics object

Examples

```
df <- data.frame(
  new    = c("positive","positive","negative","negative","positive","negative"),
  gold   = c("positive","negative","positive","negative","positive","positive"),
  age    = c(45, 52, 38, 61, 47, 55),
  gender = c("M","F","F","M","M","F"),
  id     = c("S001","S002","S003","S004","S005","S006")
)
tab <- raw_to_table(df, "new", "gold", positive = "positive", id = "id")
tab <- diagnostics(tab)
```

---

print.dilution_recovery

Print dilution-recovery results

Description

Print dilution-recovery results

Usage

```
## S3 method for class 'dilution_recovery'
print(x, ...)
```

Arguments

x	A dilution_recovery object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.factor_split *Print a categorical data split*

Description

Print a categorical data split

Usage

```
## S3 method for class 'factor_split'
print(x, ...)
```

Arguments

x	A factor_split object.
...	Additional arguments (currently unused).

Value

The input object, invisibly.

print.fit_equation *Print an equation fit*

Description

Print an equation fit

Usage

```
## S3 method for class 'fit_equation'
print(x, ...)
```

Arguments

x	A fit_equation object.
...	Additional arguments (currently unused).

Value

x, invisibly.

print.fourfold_table *S3 print method: format a* fourfold_table *object as a fourfold table*

Description

Prints data source, total sample size, variable names, fourfold table (with margins), and analysis status.

Usage

```
## S3 method for class 'fourfold_table'
print(x, margin = TRUE, ...)
```

Arguments

x	fourfold_table object (result from raw_to_table / counts_to_table)
margin	Whether to display margins, default TRUE
...	reserved arguments

Value

Invisible fourfold_table object

Examples

```
df <- data.frame(
  new    = c("positive","positive","negative","negative","positive","negative"),
  gold   = c("positive","negative","positive","negative","positive","positive"),
  age    = c(45, 52, 38, 61, 47, 55),
  gender = c("M","F","F","M","M","F"),
  id     = c("S001","S002","S003","S004","S005","S006")
)
tab <- raw_to_table(df, "new", "gold", id = "id")
print(tab)
```

print.hook_effect *Print high-dose hook-effect results*

Description

Print high-dose hook-effect results

Usage

```
## S3 method for class 'hook_effect'
print(x, ...)
```

Arguments

x	A hook_effect object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

---

print.interference_dose_response
Print interference dose-response results

Description

Print interference dose-response results

Usage

```
## S3 method for class 'interference_dose_response'
print(x, ...)
```

Arguments

x	An interference_dose_response object.
...	Unused.

Value

x, invisibly.

Examples

```
d <- expand.grid(conc = c(0, 10, 20), rep = 1:5)
d$result <- 100 + 0.5 * d$conc
print(interference_dose_response(d, "result", "conc"))
```

print.interference_paired

Print paired interference estimates

Description

Print paired interference estimates

Usage

```
## S3 method for class 'interference_paired'
print(x, ...)
```

Arguments

x	An interference_paired object.
...	Unused.

Value

x, invisibly.

Examples

```
d <- data.frame(group = rep(c("control", "test"), each = 5),
                result = c(10, 10.1, 9.9, 10.2, 9.8, 11, 11.1, 10.9, 11.2, 10.8))
print(interference_paired(d, "result", "ep07", condition = "group"))
```

print.interference_patient

Print patient-specimen interference results

Description

Print patient-specimen interference results

Usage

```
## S3 method for class 'interference_patient'
print(x, ...)
```

Arguments

x	An interference_patient object.
...	Unused.

Value

x, invisibly.

Examples

```
d <- expand.grid(id = paste0("S", 1:6), method = c("test", "ref"), rep = 1:2)
d$group <- ifelse(as.integer(sub("S", "", d$id)) <= 3, "control", "selected")
d$result <- 10 + as.integer(sub("S", "", d$id)) + (d$method == "test")
print(interference_patient(d, "id", "group", "method", "result",
  "selected", "control", "test", "ref"))
```

---

print.interference_replicates

Print EP07 replicate-count results

Description

Print EP07 replicate-count results

Usage

```
## S3 method for class 'interference_replicates'
print(x, ...)
```

Arguments

x	An interference_replicates object.
...	Unused.

Value

x, invisibly.

Examples

```
print(interference_replicates(5, 7))
```

print.kappa_table *S3 print method: format* kappa_table *output*

Description

Prints Cohen's Kappa / PABAK coefficient, standard error, confidence interval, and agreement rates.

Usage

```
## S3 method for class 'kappa_table'
print(x, digits = 4, ...)
```

Arguments

x	kappa_table object
digits	Number of decimal places, default 4
...	reserved arguments

Value

Invisible kappa_table object

Examples

```
tab <- counts_to_table(tp = 85, fp = 3, tn = 90, fn = 2,
                       candidate = "new method", reference = "gold standard")
tab <- kappa(tab)
```

print.linearity *Print a linearity analysis*

Description

Print a linearity analysis

Usage

```
## S3 method for class 'linearity'
print(x, ...)
```

Arguments

x	A linearity object.
...	Reserved arguments.

Value

The unchanged x, invisibly.

print.mcnemar_table *S3 print method: format* mcnemar_table *output*

Description

Prints McNemar test results, including discordant pair counts, test statistic, and p-value.

Usage

```
## S3 method for class 'mcnemar_table'
print(x, ...)
```

Arguments

x	mcnemar_table object
...	reserved arguments

Value

Invisible mcnemar_table object

Examples

```
tab <- counts_to_table(tp = 45, fp = 12, tn = 80, fn = 5,
                       candidate = "new method", reference = "gold standard")
tab <- mcnemar(tab)
```

print.mcr *S3 print method*

Description

Print data overview and analysis slot status based on $print slot.

Usage

```
## S3 method for class 'mcr'
print(x, ...)
```

Arguments

x	mcr object
...	Additional arguments

Value

Invisible `mcr` object

Examples

```
df <- data.frame(sid = 1:30,
                 test = rnorm(30, 50, 10),
                 ref  = rnorm(30, 50, 10))
obj <- mcr(df, "sid", "test", "ref")
print(obj)
```

`print.mcr_bias` *Print method for bias results*

Description

Print method for bias results

Usage

```
## S3 method for class 'mcr_bias'
print(x, ...)
```

Arguments

`x`	mcr_bias object
`...`	additional arguments

Value

The unchanged `mcr_bias` data frame, invisibly.

`print.mcr_bland_altman` *Print method for Bland-Altman results*

Description

Print method for Bland-Altman results

Usage

```
## S3 method for class 'mcr_bland_altman'
print(x, ...)
```

Arguments

x	mcr_bland_altman object
...	additional arguments

Value

The unchanged `mcr_bland_altman` object, invisibly.

---

print.mcr_correlation *Print method for correlation results*

Description

Print method for correlation results

Usage

```
## S3 method for class 'mcr_correlation'
print(x, ...)
```

Arguments

x	mcr_correlation object
...	additional arguments

Value

The unchanged `mcr_correlation` object, invisibly.

---

print.mcr_describe *Print method for describe results*

Description

Print method for describe results

Usage

```
## S3 method for class 'mcr_describe'
print(x, digits = NULL, ...)
```

Arguments

x	mcr_describe object
digits	Number of decimal places, default 4
...	additional arguments

Value

The unchanged `mcr_describe` object, invisibly.

---

print.mcr_outlier *Print method for outlier results*

Description

Print method for outlier results

Usage

```
## S3 method for class 'mcr_outlier'
print(x, ...)
```

Arguments

`x`	mcr_outlier object
`...`	additional arguments

Value

The unchanged `mcr_outlier` object, invisibly.

---

print.mcr_regression *Print method for regression results*

Description

Print method for regression results

Usage

```
## S3 method for class 'mcr_regression'
print(x, ...)
```

Arguments

`x`	mcr_regression object
`...`	additional arguments

Value

The unchanged `mcr_regression` object, invisibly.

print.mkt *Print mean kinetic temperature results*

Description

Print mean kinetic temperature results

Usage

```
## S3 method for class 'mkt'
print(x, ...)
```

Arguments

x	A mkt object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.normal_test *Print a normality test result*

Description

Print a normality test result

Usage

```
## S3 method for class 'normal_test'
print(x, ...)
```

Arguments

x	A normal_test object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.outliers_test *Print an outlier detection result*

Description

Print an outlier detection result

Usage

```
## S3 method for class 'outliers_test'
print(x, ...)
```

Arguments

x	An outliers_test object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.precision *Print a precision object overview*

Description

Print the data overview of a precision object (data class, total rows, formula, grouping variables, sample count, factor levels, balance warnings) and the analysis slot status ([x] / [ ]).

Usage

```
## S3 method for class 'precision'
print(x, ...)
```

Arguments

x	precision object
...	Reserved arguments

Value

Invisible precision object

print.precision_ci *Print confidence intervals*

Description

Prints confidence interval tables for SD and \ sample, component, estimate, lower, upper.

Usage

```
## S3 method for class 'precision_ci'
print(x, ...)
```

Arguments

x	precision_ci object (a list of data frames, keyed by "SD" and "CV")
...	Reserved arguments

Value

Invisibly returns x.

print.precision_normal *Print normality test results*

Description

Prints the selected normality test result per sample, including the actual method, sample size, statistic name, statistic, and p-value.

Usage

```
## S3 method for class 'precision_normal'
print(x, ...)
```

Arguments

x	precision_normal object
...	Reserved arguments

Value

Invisibly returns x.

print.precision_outlier

Print outlier detection results

Description

Prints the outlier detection method, count of outliers found, and the detail table (row, sample, value, statistic, critical value).

Usage

```
## S3 method for class 'precision_outlier'
print(x, ...)
```

Arguments

x	precision_outlier object
...	Reserved arguments

Value

Invisibly returns x.

print.precision_profile

Print precision profile fits

Description

Prints the best Sadler model formula, AIC, and R-squared for each variance component.

Usage

```
## S3 method for class 'precision_profile'
print(x, ...)
```

Arguments

x	precision_profile object (a list of per-component fit results)
...	Reserved arguments

Value

Invisibly returns x.

print.precision_vc *Print variance component table*

Description

Prints the variance component summary table with columns: sample, component, VC (variance component), \

Usage

```
## S3 method for class 'precision_vc'
print(x, ...)
```

Arguments

x	precision_vc object (a data frame)
...	Reserved arguments

Value

Invisibly returns x.

print.qc_chart *Print a Levey-Jennings QC chart analysis*

Description

Print a Levey-Jennings QC chart analysis

Usage

```
## S3 method for class 'qc_chart'
print(x, ...)
```

Arguments

x	A qc_chart object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.reference_bias *Print reference-material bias results*

Description

Print reference-material bias results

Usage

```
## S3 method for class 'reference_bias'
print(x, ...)
```

Arguments

x	A reference_bias object.
...	Reserved arguments.

Value

x, invisibly.

Examples

```
d <- data.frame(result = c(9.8, 10.0, 10.1, 9.9, 10.2, 10.0))
print(reference_bias(d, "result", reference = 10,
                     reference_uncertainty = 0.2))
```

print.reference_interval *Print reference interval analysis results*

Description

Print reference interval analysis results

Usage

```
## S3 method for class 'reference_interval'
print(x, ...)
```

Arguments

x	A "reference_interval" object.
...	Additional arguments (reserved for generic).

Value

The unchanged reference_interval object, invisibly.

print.roc *Print a ROC analysis object*

Description

Print a ROC analysis object

Usage

```
## S3 method for class 'roc'
print(x, ...)
```

Arguments

x	A roc object.
...	Additional arguments (currently unused).

Value

x, invisibly.

print.roc_auc_comparison *Print a paired AUC comparison*

Description

Print a paired AUC comparison

Usage

```
## S3 method for class 'roc_auc_comparison'
print(x, ...)
```

Arguments

x	A roc_auc_comparison object.
...	Additional arguments (currently unused).

Value

The input object, invisibly.

print.roc_auc_table *Print method for roc AUC table*

Description

Print method for roc AUC table

Usage

```
## S3 method for class 'roc_auc_table'
print(x, ...)
```

Arguments

x	roc_auc_table object (data.frame)
...	additional arguments

Value

The unchanged `roc_auc_table` data frame, invisibly.

print.roc_cutoff_table *Print method for roc cutoff table*

Description

Print method for roc cutoff table

Usage

```
## S3 method for class 'roc_cutoff_table'
print(x, ...)
```

Arguments

x	roc_cutoff_table object (data.frame)
...	additional arguments

Value

The unchanged `roc_cutoff_table` data frame, invisibly.

print.roc_describe *Print method for roc describe results*

Description

Print method for roc describe results

Usage

```
## S3 method for class 'roc_describe'
print(x, digits = NULL, ...)
```

Arguments

x	roc_describe object
digits	number of decimal places, default 4
...	additional arguments

Value

The unchanged roc_describe object, invisibly.

print.roc_mlr *Print a multivariate ROC model*

Description

Print a multivariate ROC model

Usage

```
## S3 method for class 'roc_mlr'
print(x, ...)
```

Arguments

x	A roc_mlr object.
...	Additional arguments (currently unused).

Value

x, invisibly.

print.sample_size_bland_altman

Print Bland-Altman sample-size results

Description

Print Bland-Altman sample-size results

Usage

```
## S3 method for class 'sample_size_bland_altman'
print(x, ...)
```

Arguments

x	A sample_size_bland_altman result.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.sample_size_proportion

Print single-proportion target-value sample-size results

Description

Print single-proportion target-value sample-size results

Usage

```
## S3 method for class 'sample_size_proportion'
print(x, ...)
```

Arguments

x	A sample_size_proportion result.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.sample_size_proportion_ci *Print single-proportion confidence-interval sample-size results*

Description

Print single-proportion confidence-interval sample-size results

Usage

```
## S3 method for class 'sample_size_proportion_ci'
print(x, ...)
```

Arguments

x	A sample_size_proportion_ci result.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.sensitivity *Print an analytical-sensitivity analysis result*

Description

Print an analytical-sensitivity analysis result

Usage

```
## S3 method for class 'sensitivity'
print(x, ...)
```

Arguments

x	A sensitivity_lob, sensitivity_lod, sensitivity_loq, or sensitivity_verification object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.sensitivity_design

Print an analytical-sensitivity study design

Description

Print an analytical-sensitivity study design

Usage

```
## S3 method for class 'sensitivity_design'
print(x, ...)
```

Arguments

x	A sensitivity_design object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.spike_concentration

Print spike-concentration calculations

Description

Print spike-concentration calculations

Usage

```
## S3 method for class 'spike_concentration'
print(x, ...)
```

Arguments

x	A spike_concentration data frame.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.spike_recovery *Print spike-recovery results*

Description

Print spike-recovery results

Usage

```
## S3 method for class 'spike_recovery'
print(x, ...)
```

Arguments

x	A spike_recovery object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.stability_bias *Print stability-bias results*

Description

Print stability-bias results

Usage

```
## S3 method for class 'stability_bias'
print(x, ...)
```

Arguments

x	A stability_bias object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.stability_plan *Print a stability study time-point plan*

Description

Print a stability study time-point plan

Usage

```
## S3 method for class 'stability_plan'
print(x, ...)
```

Arguments

x	A stability_plan object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.stability_regression *Print stability-regression results*

Description

Print stability-regression results

Usage

```
## S3 method for class 'stability_regression'
print(x, ...)
```

Arguments

x	A stability_regression object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.stability_time *Print stability limit-time predictions*

Description

Print stability limit-time predictions

Usage

```
## S3 method for class 'stability_time'
print(x, ...)
```

Arguments

x	A stability_time object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.tukey *S3 print method: tukey*

Description

Print Tukey HSD post-hoc test results including pairwise comparison table, significance markers, and compact letter display.

Usage

```
## S3 method for class 'tukey'
print(x, ...)
```

Arguments

x	A tukey object
...	Other arguments (unused, for S3 generic signature)

Value

Invisibly returns x.

print.uncertainty_budget

Print a combined uncertainty budget

Description

Print a combined uncertainty budget

Usage

```
## S3 method for class 'uncertainty_budget'
print(x, ...)
```

Arguments

x	An uncertainty_budget object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.uncertainty_characterization

Print characterization uncertainty results

Description

Print characterization uncertainty results

Usage

```
## S3 method for class 'uncertainty_characterization'
print(x, ...)
```

Arguments

x	An uncertainty_characterization object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

`print.uncertainty_component` *Print an uncertainty component*

Description

Print an uncertainty component

Usage

```
## S3 method for class 'uncertainty_component'
print(x, ...)
```

Arguments

`x`	An uncertainty_component object.
`...`	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

`print.uncertainty_homogeneity` *Print between-unit homogeneity uncertainty results*

Description

Print between-unit homogeneity uncertainty results

Usage

```
## S3 method for class 'uncertainty_homogeneity'
print(x, ...)
```

Arguments

`x`	An uncertainty_homogeneity object.
`...`	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.uncertainty_iqc *Print IQC uncertainty results*

Description

Print IQC uncertainty results

Usage

```
## S3 method for class 'uncertainty_iqc'
print(x, ...)
```

Arguments

x	An uncertainty_iqc object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.uncertainty_propagation *Print propagated uncertainty results*

Description

Print propagated uncertainty results

Usage

```
## S3 method for class 'uncertainty_propagation'
print(x, ...)
```

Arguments

x	An uncertainty_propagation object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.uncertainty_stability *Print stability uncertainty results*

Description

Print stability uncertainty results

Usage

```
## S3 method for class 'uncertainty_stability'
print(x, ...)
```

Arguments

x	An uncertainty_stability object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.uncertainty_traceability *Print traceability-chain uncertainty results*

Description

Print traceability-chain uncertainty results

Usage

```
## S3 method for class 'uncertainty_traceability'
print(x, ...)
```

Arguments

x	An uncertainty_traceability object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

print.youden_plot *Print a Youden plot analysis*

Description

Print a Youden plot analysis

Usage

```
## S3 method for class 'youden_plot'
print(x, ...)
```

Arguments

x	A youden_plot object.
...	Reserved for the S3 generic.

Value

The unchanged x, invisibly.

profile.precision *S3 precision profile method*

Description

Fit precision (SD) vs. mean across samples using the Sadler model selection algorithm (10 candidate models) based on VFP::fit_vfp().

Usage

```
## S3 method for class 'precision'
profile(object, model.no = 1:10, ...)
```

Arguments

object	precision object (requires variance() to be run first)
model.no	Vector of candidate model numbers, default 1:10
...	Additional arguments passed to .sadler() (e.g. K)

Value

Updated precision object with results stored in $profile

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- variance(obj)
obj <- profile(obj, model.no = 1)
```

`qc_chart` *Levey-Jennings QC chart + Westgard rule evaluation*

Description

Draws a Levey-Jennings QC chart and detects out-of-control points based on selected Westgard rules. If target mean and SD are not provided, they are estimated from the data.

Usage

```
qc_chart(
  data,
  value,
  rules = "1-2s,1-3s,2-2s,R-4s,4-1s,10x",
  mean = NULL,
  sd = NULL,
  group = NULL,
  run = NULL
)
```

Arguments

`data`	A data frame.
`value`	Column name (numeric) containing measured values.
`rules`	Comma-separated rule names, e.g. "1-2s,1-3s,10x". Rule names must match those printed by list_westgard().
`mean`	Target mean; NULL to estimate from the selected data column.
`sd`	Target SD; NULL to estimate from the selected data column.
`group`	Optional grouping column name (e.g. QC lot), used for faceted plots.
`run`	Run-order column name (optional, default 1:n).

Value

An object of class `"qc_chart"` with violation results and plot data. Use `print()` to show summary and `plot()` to draw the Levey-Jennings chart.

Examples

```
set.seed(42)
df <- data.frame(
  run   = 1:30,
  value = c(rnorm(27, 100, 5), 108, 115, 85)
)
res <- qc_chart(df, "value", rules = "1-2s,1-3s,10x")
print(res)
plot(res)
```

`raw_to_table` *Core function: raw data -> frequency fourfold table (silent construction)*

Description

Core function: raw data -> frequency fourfold table (silent construction)

Usage

```
raw_to_table(
  data,
  candidate,
  reference,
  id = NULL,
  na.rm = TRUE,
  positive = NULL
)
```

Arguments

`data`	Data frame, one observation per row
`candidate`	Candidate method variable name (character), e.g. "method_new"
`reference`	Reference method variable name (character), e.g. "method_standard"
`id`	ID variable name (character), optional, used for duplicate detection in describe
`na.rm`	Whether to remove NA, default TRUE
`positive`	Specify the positive label (character); if non-NULL, levels will be uniformly mapped to positive/negative. If NULL, the last sorted level is treated as positive.

Value

A fourfold_table object (list with S3 class "fourfold_table"), containing: - $freq_raw Raw frequencies (long format) - $table Fourfold matrix (with margins) - $n Total sample size - $print Print slot (descriptive metadata) - $candidate_levels Levels of the candidate method - $reference_levels Levels of the reference method Use print() directly to display the fourfold table.

Examples

```
df <- data.frame(
  new    = c("positive","positive","negative","negative","positive","negative"),
  gold   = c("positive","negative","positive","negative","positive","positive"),
  age    = c(45, 52, 38, 61, 47, 55),
  gender = c("M","F","F","M","M","F"),
  id     = c("S001","S002","S003","S004","S005","S006")
)
result <- raw_to_table(df, "new", "gold", id = "id")
```

`reference_bias` *Estimate bias using reference materials*

Description

Calculates measurement bias and its interval according to Section 5.3 of YY/T 1789.2-2021. For each reference-material level, the uncertainty of the measured mean is calculated as `s / sqrt(n)` and expanded by `k`. This expanded uncertainty is then combined with the reference-material uncertainty. The function reports results only and applies no acceptance criterion.

Usage

```
reference_bias(
  data,
  result,
  reference,
  reference_uncertainty,
  level = NULL,
  k = 2
)
```

Arguments

`data`	A data frame containing replicate measurement results.
`result`	Name of the numeric measurement-result column.
`reference`	Reference-material assigned value. Supply a finite scalar, a numeric vector with one value per row or level, or the name of a numeric column in `data`.
`reference_uncertainty`	Expanded uncertainty of the reference material, at the same coverage level as the uncertainty calculated with `k`. Supply a non-negative scalar, a numeric vector with one value per row or level, or the name of a numeric column in `data`.
`level`	Optional name of the reference-material level column. When omitted, all rows are treated as one level.
`k`	Positive coverage factor used to expand the standard uncertainty of the measured mean. The standard's example uses `k = 2`.

Value

An object of class "reference_bias". Its result component contains the sample size, mean, standard deviation, uncertainty, bias, relative bias, and interval for each reference-material level.

References

YY/T 1789.2-2021, Section 5.3 and Appendix A.

Examples

```
reference_data <- data.frame(
  level = rep(c("low", "high"), each = 6),
  result = c(9.8, 10.1, 10.0, 9.9, 10.2, 10.0,
             19.7, 20.1, 20.0, 19.9, 20.2, 20.1)
)
bias_result <- reference_bias(
  reference_data, result = "result",
  reference = c(10, 20),
  reference_uncertainty = c(0.2, 0.3),
  level = "level"
)
bias_result$result
```

---

reference_interval *Reference interval analysis*

Description

Calculates reference intervals using the CLSI EP28-A3c simple nonparametric or biweight procedures, a normal/log-normal parametric procedure, or the retained R type-6 and Huber procedures.

Usage

```
reference_interval(
  data,
  col,
  interval = 0.95,
  ci = 0.95,
  method = "nonparametric",
  id = NULL,
  bootstrap_replicates = 1999L,
  seed = NULL
)
```

Arguments

data	A data frame.
col	A single character string naming the numeric result column.
interval	Central reference interval coverage in (0, 1); the default is 0.95.
ci	Confidence level for each reference-limit interval in (0, 1); the default is 0.95.
method	One or more of "nonparametric" (default), "biweight", "type6", "parametric", or "huber"; "all" selects all five.
id	Optional single character string naming an ID column used only to report duplicate IDs.
bootstrap_replicates	Number of percentile-bootstrap resamples for "biweight" and "huber". Must be an integer of at least 100.
seed	Optional integer seed for biweight and Huber bootstrap resampling. The caller's random-number state is preserved.

Details

method = "nonparametric" uses ranks p * (n + 1), linear interpolation, and exact binomial order-statistic confidence intervals. method = "biweight" implements the EP28 Appendix B biweight location and scale calculation and percentile bootstrap confidence intervals. method = "type6" retains the previous R type-6 quantiles with normal-approximation rank intervals. method = "huber" retains the previous Huber M-estimator.

Value

An object of class reference_interval. Selected results are stored in correspondingly named elements: nonparametric, biweight, type6, parametric, and huber. Data-quality counts and cleaned values are also retained.

References

Clinical and Laboratory Standards Institute. *Defining, Establishing, and Verifying Reference Intervals in the Clinical Laboratory; Approved Guideline—Third Edition*. CLSI document EP28-A3c. Wayne, PA: CLSI; 2010.

Examples

```
set.seed(17)
d <- data.frame(id = seq_len(150), value = rnorm(150, 100, 15))
reference_interval(d, "value", id = "id")
reference_interval(d, "value", method = c("nonparametric", "parametric"))
reference_interval(d, "value", method = "all",
                   bootstrap_replicates = 199, seed = 28)
```

`regression.mcr` *Regression analysis (S3 method)*

Description

Perform regression analysis with the reference/comparative method on the X-axis and the candidate/test method on the Y-axis. Supports five methods: Ordinary Least Squares (OLS), Weighted Least Squares (WLS), Deming regression, Weighted Deming regression, and Passing-Bablok regression.

Usage

```
## S3 method for class 'mcr'
regression(
  x,
  method = c("ols", "wls", "deming", "wdeming", "pb"),
  conf.level = 0.95,
  lambda = 1,
  weights = NULL,
  variance_models = NULL,
  profile_components = c(reference = "total", candidate = "total"),
  ci_method = c("auto", "analytic", "jackknife", "rank", "bootstrap_percentile",
    "bootstrap_standard"),
  bootstrap_replicates = 5000L,
  seed = NULL,
  mdl = NULL,
  weight_iterations = 4L,
  epsilon = NULL,
  ...
)
```

Arguments

`x`	mcr object
`method`	Regression method: "ols" (default), "wls", "deming", "wdeming", "pb"
`conf.level`	Confidence level, default 0.95
`lambda`	Variance ratio for Deming regression (candidate/Y variance divided by reference/X variance), or `"replicates"` to estimate it from the summarized SD/n columns supplied to `mcr()`. Default 1.
`weights`	Weights specification, default NULL (unweighted). Character string for built-in modes: "equal", "1/y", "1/y^2", "1/x", "1/x^2", "constant_cv", "precision_profile", "residual", or a column name in data containing non-negative weights. Constant-CV means exactly `1/x^2` for WLS and Linnet's iterative fit for weighted Deming. The residual model follows EP09c Appendix H: fit absolute residuals on reference concentration and use the inverse squared fitted SD. For OLS, non-residual weights are passed to lm(). WLS requires a weight specification and is

the only method supporting "residual". Weighted Deming requires non-residual weights. Deming does not support weights; Passing-Bablok ignores them with a warning.

`variance_models`
: For precision-profile Deming, a two-element list identifying `reference` and `candidate` variance models. Each is either a `precision` object after `profile()` or a function of concentration that returns positive finite variances.

`profile_components`
: Precision-profile components for reference and candidate methods; both default to `"total"`.

`ci_method`
: Regression-parameter interval method. `"auto"` uses analytic intervals for OLS/WLS, full-fit jackknife intervals for Deming methods, and the Appendix I rank interval for Passing-Bablok. Explicit Appendix K2 choices are `"bootstrap_percentile"` and `"bootstrap_standard"`.

`bootstrap_replicates`
: Number of paired Bootstrap resamples; at least 5000 when a Bootstrap interval is requested.

`seed`
: Optional integer seed. The caller's random-number state is restored after a seeded analysis.

`mdl`
: Optional medical decision levels for which bias Bootstrap draws are retained with the regression result.

`weight_iterations`
: Number of residual-weight updates when `weights = "residual"`; default 4, as recommended by EP09c.

`epsilon`
: Convergence tolerance for constant-CV or precision-profile Deming. NULL uses the underlying algorithm's default.

`...`
: Additional arguments passed to lm (OLS/WLS only)

Value

Updated mcr object (invisible), results stored in $regression

Examples

```
obj <- mcr(ivd_mcr_example, "sid", "test", "ref")
regression(obj, method = "ols")
regression(obj, method = "deming")
```

`replicate_to_mean` *Summarize replicate measurements*

Description

Groups data by x, computes mean, SD, and sample size for each group. When `weights` is specified, returns an additional `weight` column. Summary metadata (call, weighting method, column names) is stored as attributes for downstream reference.

Usage

```
replicate_to_mean(data, x, y, weights = NULL, na.rm = TRUE)
```

Arguments

data	a data frame
x	x column name (grouping variable)
y	One or more y column names (response variables). A single response retains the historical y_mean, y_sd, and y_n column names; multiple responses use <name>_mean, <name>_sd, and <name>_n.
weights	weighting method: NULL / "n" / "1/sd" / "1/sd^2" NULL – summary only, no weight column "n" – w = number of replicates "1/sd" – w = 1/sd(y) "1/sd^2" – w = 1/var(y) (inverse-variance weighting)
na.rm	remove NA? (default TRUE)

Value

A data.frame with columns:

x	grouping variable
y_mean, y_sd, y_n	Historical names used for one response.
\if{html}{\out{<name>}}_mean, \if{html}{\out{<name>}}_sd, \if{html}{\out{<name>}}_n	Names used for each response when two or more response columns are supplied.
weight	(only when weights is specified) weight values

Attributes: call, weights (method name), x_col, y_col

Examples

```
# Basic summary
df <- data.frame(
  conc = rep(c(1, 2, 5, 10), each = 3),
  resp = c(3.1, 3.2, 2.9, 5.8, 6.1, 5.9,
           14.2, 14.5, 14.0, 28.1, 27.9, 28.3)
)
rep <- replicate_to_mean(df, "conc", "resp")
rep

# Inverse-variance weighting
rep_w <- replicate_to_mean(df, "conc", "resp", weights = "1/sd^2")
rep_w
```

`report.precision` *Generate an EP05 precision summary table*

Description

Generate an EP05 precision summary table

Usage

```
## S3 method for class 'precision'
report(
  x,
  day,
  run = NULL,
  site = NULL,
  table = c("simple", "detailed", "CI"),
  digits = 4L,
  ...
)
```

Arguments

`x`	A precision object after `variance()` has been run.
`day`	Name of the day (or equivalent) design variable.
`run`	Name of the run variable, when present.
`site`	Name of the site variable for a multisite design.
`table`	One of `"simple"`, `"detailed"`, or `"CI"`.
`digits`	Number of decimal places.
`...`	Reserved for future extensions.

Details

Interaction terms are matched independently of the textual order of their factors. For a multi-site design, within-laboratory precision is formed from the within-site components (for example, `site:day` + `error`), whereas reproducibility includes the between-site component. With `table = "CI"`, confidence intervals for combined components use the variance-component covariance matrix and a Satterthwaite approximation.

Value

A data.frame containing the requested report table.

Examples

```
if (requireNamespace("VCA", quietly = TRUE)) {
  data(VCAdata1, package = "VCA")
  p <- precision(VCAdata1, y ~ day/run, by = "sample")
  p <- variance(p)
  report(p, day = "day", run = "run", table = "simple")
}
```

`residuals.fit_equation` *Extract residuals from an equation fit*

Description

Extract residuals from an equation fit

Usage

```
## S3 method for class 'fit_equation'
residuals(object, ...)
```

Arguments

`object`	A `fit_equation` object.
`...`	Additional arguments passed to the fitted model.

Value

A numeric vector of residuals.

Examples

```
df <- data.frame(x = 1:5, y = c(2.1, 4.0, 6.2, 7.9, 10.1))
f <- fit_equation("E01", df, "x", "y")
residuals(f)
```

roc *roc constructor*

Description

Create an ROC analysis object.

Usage

```
roc(data, cols, reference, id = NULL, positive = NULL)
```

Arguments

data	data frame
cols	evaluation column names (character vector, quantitative, can be multiple)
reference	reference column name (character, single column). Must be binary 0/1, a factor with 2 levels, or a character vector with 2 unique values. If the original reference is quantitative, use `continuous_to_binary()` first to convert it to 0/1 and pass the new column name.
id	ID column name (optional, for duplicate detection)
positive	specify the positive level of reference (for factor/character)

Details

Supports multiple evaluation columns (quantitative) and a single reference column (binary 0/1, factor, or character with two levels). Automatically detects direction, reports missing values and ID duplicates.

Value

Returns an S3 object of class "roc"

Examples

```
df <- data.frame(
  sid = 1:100,
  x1  = c(rnorm(50, 10, 2), rnorm(50, 12, 2)),
  ref = rep(c(0, 1), each = 50),
  age = 1:100,
  sex = rep(c("F", "M"), each = 50)
)
obj <- roc(df, cols = "x1", reference = "ref", id = "sid")
print(obj)
```

`sample_size_bland_altman` *Sample size for Bland-Altman agreement assessment*

Description

Calculates the minimum number of measurement pairs required to attain a target power using the noncentral-t approximation of Lu et al. (2016).

Usage

```
sample_size_bland_altman(
  power = 0.8,
  mu,
  sd,
  delta,
  conf.level = 0.95,
  agree.level = 0.95,
  method = "lu",
  n.min = 3L,
  n.max = 1e+05
)
```

Arguments

`power`	Target power in `(0, 1)`.
`mu`	Anticipated mean difference between the two measurement methods.
`sd`	Anticipated standard deviation of the differences.
`delta`	Positive symmetric clinical agreement limit.
`conf.level`	Confidence level for the confidence intervals around the limits of agreement. Default 0.95.
`agree.level`	Central proportion defining the limits of agreement. Default 0.95.
`method`	Calculation method. Currently only `"lu"` is available.
`n.min`	Minimum sample size considered. Default 3.
`n.max`	Maximum sample size considered. Default 100000.

Value

An object of class `sample_size_bland_altman`. The required sample size is available as `x$n`; the requested and attained powers are available as `x$target_power` and `x$achieved_power`.

References

Lu MJ, Zhong WH, Liu YX, Miao HZ, Li YC, Ji MH (2016). Sample size for assessing agreement between two methods of measurement by Bland-Altman method. *International Journal of Biostatistics*, 12(2). doi:10.1515/ijb20150039

Examples

```
sample_size_bland_altman(
  power = 0.80, mu = 0.2, sd = 1, delta = 2.5
)
```

`sample_size_proportion` *Sample size for a single-proportion target-value test*

Description

Calculates the minimum sample size required to test a single proportion against a target value. The test statistic and the power-calculation method are selected independently. Regardless of the planning method, the returned result includes the exact binomial Type I error and power of the resulting discrete rejection region.

Usage

```
sample_size_proportion(
  p0,
  p1,
  alpha = 0.05,
  power = 0.8,
  alternative = c("greater", "less", "two.sided"),
  test = c("exact", "score", "score-cc", "wald", "wald-cc", "agresti-coull", "jeffreys"),
  power_method = c("binomial", "normal"),
  n.max = 1e+06
)
```

Arguments

`p0`	Null or target proportion in `(0, 1)`.
`p1`	Anticipated proportion under the alternative in `(0, 1)`.
`alpha`	Target significance level in `(0, 1)`.
`power`	Target power in `(0, 1)`.
`alternative`	One of `"greater"`, `"less"`, or `"two.sided"`.
`test`	Rejection-region method. One of `"exact"`, `"score"`, `"score-cc"`, `"wald"`, `"wald-cc"`, `"agresti-coull"`, or `"jeffreys"`.
`power_method`	Power calculation used to select the sample size: `"binomial"` enumerates the discrete binomial distribution and requires both actual alpha and actual power to meet their targets; `"normal"` uses the continuous normal approximation and is available for exact, Score, and Wald tests, with or without continuity correction.
`n.max`	Maximum sample size considered. Default 1000000.

Details

`test = "score", power_method = "normal"` uses the null variance for the alpha term and the alternative variance for the beta term. This is the normal-approximation procedure used by the CMDE target-value formula. With an infinite population, `test = "exact"` and `test = "score"` have the same normal-approximation formula, matching PASS. The test choice still determines the discrete rejection region used for the reported actual alpha and achieved power.

Value

An object of class `sample_size_proportion`. The required sample size is available as `x$n`. The object also contains target and planning power, exact achieved power, actual alpha, and critical success counts.

Examples

```
# Exact binomial design
sample_size_proportion(
  p0 = 0.2, p1 = 0.35, alpha = 0.05, power = 0.8,
  alternative = "greater"
)

# CMDE continuous normal approximation
sample_size_proportion(
  p0 = 0.85, p1 = 0.90, alpha = 0.05, power = 0.8,
  alternative = "two.sided", test = "score", power_method = "normal"
)
```

---

sample_size_proportion_ci

Sample size for a single-proportion confidence interval

Description

Calculates the minimum sample size whose two-sided confidence interval has total width no greater than twice `half_width`, or whose one-sided limit is no farther than `half_width` from the anticipated sample proportion.

Usage

```
sample_size_proportion_ci(
  p,
  half_width,
  conf.level = 0.95,
  interval = c("two.sided", "lower", "upper"),
 method = c("wilson", "wald", "wald-cc", "agresti-coull", "jeffreys", "wilson-cc",
    "clopper-pearson"),
  n.max = 1e+06
)
```

Arguments

p	Anticipated proportion in [0, 1]. Boundary values are useful for one-sided exact intervals.
half_width	For a two-sided interval, one half of the total interval width. For a one-sided interval, the maximum distance from the anticipated proportion to the confidence limit.
conf.level	Confidence level. Default 0.95.
interval	One of "two.sided", "lower", or "upper".
method	Confidence interval method. One of "wald", "wald-cc", "wilson", "wilson-cc", "agresti-coull", "jeffreys", or "clopper-pearson".
n.max	Maximum sample size considered. Default 1000000.

Value

An object of class sample_size_proportion_ci. The required sample size is available as x$n. The returned object also contains the achieved half-width and the exact binomial coverage at the assumed value of p.

Examples

```
sample_size_proportion_ci(p = 0.3, half_width = 0.05)
sample_size_proportion_ci(
  p = 0.3, half_width = 0.03, method = "clopper-pearson"
)
```

---

sensitivity_design *Summarize an analytical-sensitivity study design*

Description

Returns minimum CLSI EP17-A2 design requirements and, optionally, counts observed in a supplied data set. Differences are descriptive only; the function does not issue a pass/fail conclusion.

Usage

```
sensitivity_design(
  method = c("classical", "profile", "probit", "loq", "verification"),
  data = NULL,
  result = NULL,
  sample = NULL,
  lot = NULL,
  day = NULL,
  concentration = NULL,
  limit = c("lob", "lod", "loq")
)
```

Arguments

`method`	One or more of `"classical"`, `"profile"`, `"probit"`, `"loq"`, `"verification"`, or `"all"`.
`data`	Optional study data.
`result`, `sample`, `lot`, `day`, `concentration`	Optional column names used to summarize actual study counts.
`limit`	Detection limit for a verification design.

Value

An object of class `sensitivity_design`.

References

CLSI. Evaluation of Detection Capability for Clinical Laboratory Measurement Procedures. EP17-A2, 2nd ed. 2012.

Examples

```
sensitivity_design(c("classical", "loq"))

design_data <- data.frame(
  result = 1:8, sample = rep(c("S1", "S2"), each = 4),
  lot = rep(c("A", "B"), 4), day = rep(1:2, each = 4)
)
sensitivity_design("verification", data = design_data,
                   result = "result", sample = "sample",
                   lot = "lot", day = "day", limit = "lod")
```

`sensitivity_lob` *Establish a limit of blank*

Description

Establishes a LoB from replicate-level blank results using the CLSI EP17-A2 nonparametric, parametric, or zero-LoB procedure. With two or three reagent lots, lots are analyzed separately and the maximum estimate is reported; four or more lots are combined by default.

Usage

```
sensitivity_lob(
  data,
  result,
  sample,
  lot = NULL,
  method = c("nonparametric", "parametric", "zero"),
  alpha = 0.05,
```

```
  lot_rule = c("ep17", "separate", "combined"),
  exclude = NULL
)
```

Arguments

`data`	A data frame containing replicate-level results.
`result, sample`	Column names for the numeric result and blank sample ID.
`lot`	Optional reagent-lot column.
`method`	One of `"nonparametric"`, `"parametric"`, or `"zero"`.
`alpha`	Type I error risk.
`lot_rule`	EP17 lot handling, forced separate analysis, or forced pooling.
`exclude`	Optional row indices or logical exclusion indicator. Exclusions are recorded in the returned object.

Value

An object of class `sensitivity_lob`.

References

CLSI. Evaluation of Detection Capability for Clinical Laboratory Measurement Procedures. EP17-A2, 2nd ed. 2012.

Examples

```
blank <- data.frame(
  sample = rep(c("blank_1", "blank_2"), each = 5),
  result = c(0.02, 0.04, 0.03, 0.05, 0.01,
             0.03, 0.06, 0.04, 0.02, 0.05)
)
lob <- sensitivity_lob(blank, result = "result", sample = "sample")
lob$estimate
```

`sensitivity_lod` *Establish a limit of detection*

Description

Establishes LoD using the classical, nonparametric trial, precision-profile, or probit approach described in CLSI EP17-A2.

Usage

```
sensitivity_lod(
  data,
  result = NULL,
  sample = NULL,
  lob,
  lot = NULL,
  concentration = NULL,
  detected = NULL,
  method = c("classical", "nonparametric", "profile", "probit"),
  beta = 0.05,
  profile_model = c("linear", "quadratic", "sadler", "vfp"),
  concentration_scale = c("log10", "identity"),
  total = NULL,
  strata = NULL,
  model_no = 1:9,
  lot_rule = c("ep17", "separate", "combined"),
  exclude = NULL
)
```

Arguments

`data`	A replicate-level data frame, or a completed `precision` object for `method = "profile"`.
`result, sample`	Column names for result and sample ID. `result` may be `NULL` only when a `precision` object is supplied.
`lob`	Numeric LoB or a `sensitivity_lob` object.
`lot`	Optional reagent-lot column.
`concentration`	Concentration column for probit analysis.
`detected`	Optional detection indicator or hit-count column for probit.
`method`	LoD method.
`beta`	Type II error risk.
`profile_model`	Precision-profile model.
`concentration_scale`	Probit concentration scale.
`total`	Optional total-count column for summarized probit data.
`strata`	Optional additional probit stratification columns.
`model_no`	VFP model numbers when `profile_model = "vfp"`.
`lot_rule`	EP17, separate, or combined lot handling.
`exclude`	Optional excluded rows.

Value

An object of class `sensitivity_lod`.

References

CLSI. Evaluation of Detection Capability for Clinical Laboratory Measurement Procedures. EP17-A2, 2nd ed. 2012.

Examples

```
low <- data.frame(
  sample = rep(c("low_1", "low_2"), each = 5),
  result = c(0.22, 0.26, 0.24, 0.25, 0.23,
             0.31, 0.29, 0.33, 0.30, 0.32)
)
lod <- sensitivity_lod(low, "result", "sample", lob = 0.10,
                       method = "classical")
lod$estimate
```

---

sensitivity_loq *Establish a limit of quantitation*

Description

Establishes LoQ from Westgard total error, RMS error, a precision goal, or a user-supplied accuracy function. The original candidate and the LoD- constrained reported LoQ are retained separately.

Usage

```
sensitivity_loq(
  data,
  result,
  sample,
  assigned,
  lot = NULL,
  method = c("westgard", "rms", "precision", "custom"),
  goal,
  goal_scale = c("relative", "absolute"),
  lod = NULL,
  k = 2,
  accuracy_function = NULL,
  profile = FALSE,
  profile_model = c("linear", "quadratic"),
  lot_rule = c("ep17", "separate", "combined"),
  exclude = NULL,
  ...
)
```

Arguments

data	Replicate-level data frame.
result, sample	Column names for result and sample ID.
assigned	Assigned-value column or numeric value/vector.
lot	Optional reagent-lot column.
method	Accuracy definition.
goal	Accuracy goal; proportions are used for relative goals.
goal_scale	Relative or absolute scale.
lod	Optional numeric LoD or sensitivity_lod object.
k	SD multiplier for the Westgard definition.
accuracy_function	Function used by method = "custom".
profile	Fit an error profile and calculate its goal crossing?
profile_model	Linear or quadratic error profile.
lot_rule	EP17, separate, or combined lot handling.
exclude	Optional excluded rows.
...	Additional arguments passed to accuracy_function.

Value

An object of class sensitivity_loq.

References

CLSI. Evaluation of Detection Capability for Clinical Laboratory Measurement Procedures. EP17-A2, 2nd ed. 2012.

Examples

```
loq_data <- data.frame(
  sample = rep(paste0("L", 1:3), each = 4),
  assigned = rep(c(0.5, 1, 2), each = 4),
  result = c(0.42, 0.48, 0.51, 0.46,
             0.91, 1.02, 0.96, 1.05,
             1.91, 2.04, 1.98, 2.08)
)
loq <- sensitivity_loq(
  loq_data, "result", "sample", assigned = "assigned",
  method = "precision", goal = 0.15, goal_scale = "relative"
)
loq$summary
```

`sensitivity_verify` *Verify an LoB, LoD, or LoQ claim*

Description

Calculates the proportion of verification results consistent with a stated claim, its one-sided Wilson score limits, and the CLSI EP17-A2 Table 1 boundary. No pass/fail field or regulatory conclusion is produced.

Usage

```
sensitivity_verify(
  data,
  result,
  sample,
  limit = c("lob", "lod", "loq"),
  claim,
  lob = NULL,
  assigned = NULL,
  allowable_error = NULL,
  error_scale = c("relative", "absolute"),
  alpha = 0.05,
  exclude = NULL
)
```

Arguments

`data`	Replicate-level verification data.
`result`, `sample`	Column names for result and sample ID.
`limit`	One of `"lob"`, `"lod"`, or `"loq"`.
`claim`	Stated limit being verified.
`lob`	LoB threshold required for LoD verification.
`assigned`	Assigned-value column or vector for LoQ verification.
`allowable_error`	Accuracy window half-width for LoQ verification.
`error_scale`	Relative or absolute LoQ error window.
`alpha`	One-sided confidence error probability.
`exclude`	Optional excluded rows.

Value

An object of class `sensitivity_verification`.

References

CLSI. Evaluation of Detection Capability for Clinical Laboratory Measurement Procedures. EP17-A2, 2nd ed. 2012.

Examples

```
verification <- data.frame(
  sample = rep(c("blank_1", "blank_2"), each = 10),
  result = c(rep(0.03, 9), 0.07, rep(0.04, 9), 0.08)
)
verified <- sensitivity_verify(
  verification, "result", "sample", limit = "lob", claim = 0.05
)
verified$result
```

---

spike_concentration *Calculate the concentration of a spiked sample*

Description

Applies the mass-balance equation for mixing a sample and a spiking material. Arguments are vectorized using ordinary R recycling.

Usage

```
spike_concentration(
  sample_concentration,
  sample_volume,
  stock_concentration,
  spike_volume
)
```

Arguments

`sample_concentration`	Concentration in the sample before spiking.
`sample_volume`	Volume of sample used.
`stock_concentration`	Concentration of the spiking material.
`spike_volume`	Volume of spiking material added.

Value

A data frame containing total volume, spike fraction, concentration added by the spike, and final calculated concentration.

References

CLSI. Establishing and Verifying an Extended Measuring Interval Through Specimen Dilution and Spiking. EP34, 1st ed. 2018.

Examples

```
spike_concentration(
  sample_concentration = 10, sample_volume = 0.95,
  stock_concentration = 210, spike_volume = 0.05
)
```

---

spike_recovery *Calculate spike recovery against a solvent control*

Description

Calculates the difference between analyte-spiked and equal-volume solvent-control means and expresses that difference as a percentage of the known added concentration. No acceptance criterion is applied.

Usage

```
spike_recovery(
  data,
  sample,
  result,
  condition,
  analyte_level,
  solvent_level,
  added_concentration = NULL,
  sample_volume = NULL,
  stock_concentration = NULL,
  spike_volume = NULL,
  conf.level = 0.95
)
```

Arguments

data	Data frame in long format.
sample	Sample-identification column.
result	Numeric measurement-result column.
condition	Column distinguishing analyte-spiked and solvent-control preparations.
analyte_level	Value identifying the analyte-spiked preparation.
solvent_level	Value identifying the solvent-control preparation.

`added_concentration`
: A finite scalar or numeric column containing the theoretical concentration added to the final mixture. When omitted, `sample_volume`, `stock_concentration`, and `spike_volume` are required.

`sample_volume`, `stock_concentration`, `spike_volume`
: Finite scalars or numeric columns used to calculate the added concentration.

`conf.level`
: Confidence level for the across-sample mean recovery.

Value

An object of class `spike_recovery`.

References

CLSI. Establishing and Verifying an Extended Measuring Interval Through Specimen Dilution and Spiking. EP34, 1st ed. 2018.

Examples

```
spike_data <- data.frame(
  sample = rep(c("S1", "S2"), each = 4),
  condition = rep(rep(c("spiked", "control"), each = 2), 2),
  result = c(20.1, 19.9, 10.0, 10.2, 30.2, 29.8, 20.1, 19.9)
)
recovery <- spike_recovery(
  spike_data, "sample", "result", "condition",
  analyte_level = "spiked", solvent_level = "control",
  added_concentration = 10
)
recovery$results
```

---

split_by_cutpoints *Split a data frame at fixed numeric cutpoints*

Description

Split an input data frame into a named list of data frames before analysis. Cutpoints are applied to one numeric variable using left-closed, right-open intervals. For example, cutpoints `c(100, 500)` create `[-Inf, 100)`, `[100, 500)`, and `[500, Inf)`.

Usage

```
split_by_cutpoints(
  data,
  variable,
  cutpoints,
  labels = NULL,
  na_action = c("error", "drop"),
  drop_empty = FALSE
)
```

Arguments

data	A data frame.
variable	A single character string naming the numeric variable used to split the rows.
cutpoints	A non-empty, finite, strictly increasing numeric vector.
labels	Optional unique names for the returned data frames. Its length must be length(cutpoints) + 1. By default the names are segment_1, segment_2, and so on.
na_action	How non-finite splitting values are handled: "error" (default) stops; "drop" excludes them and records the excluded count.
drop_empty	Whether empty intervals are removed. The default is FALSE, so the returned list preserves all prespecified intervals.

Value

A named list of data frames with class cutpoint_split. The segment_info attribute records interval bounds, labels, and row counts.

Examples

```
parts <- split_by_cutpoints(
  ivd_mcr_example, variable = "ref", cutpoints = c(18, 26)
)
parts
parts$segment_1
fits <- lapply(parts, function(d) mcr(d, "sid", "test", "ref"))
```

---

split_by_factors *Split a data frame by the levels of one categorical variable*

Description

Split an input data frame into a named list of data frames before analysis. Factor levels retain their declared order; character and logical values use first-appearance order. The splitting column is retained in every returned data frame.

Usage

```
split_by_factors(
  data,
  variable,
  na_action = c("drop", "error"),
  drop_empty = TRUE
)
```

Arguments

data	A data frame.
variable	A single character string naming the factor, character, or logical variable used to split the rows.
na_action	How missing splitting values are handled: "drop" (default) excludes them and records the excluded count; "error" stops.
drop_empty	Whether unused factor levels are removed. The default is TRUE. This argument has no effect for character or logical variables, whose groups are determined from observed non-missing values.

Value

A named list of data frames with class factor_split. List names are the original factor levels or categorical values. The group_info attribute records their order and row counts.

Examples

```
d <- data.frame(
  batch = factor(c("Batch-A", "Batch-A", "Batch-B")),
  result = c(10.1, 9.9, 10.4)
)
parts <- split_by_factors(d, variable = "batch")
parts
parts[["Batch-A"]]
means <- lapply(parts, function(z) mean(z$result))
```

---

stability_bias *stability_bias — Node-wise bias analysis across storage conditions*

Description

Each storage condition is baseline-corrected to its own first time point. Absolute and relative biases are computed and compared across conditions at matched time points.

Usage

```
stability_bias(
  data,
  condition,
  time,
  value,
  reference_condition = NULL,
  limit = NULL,
  bias_type = c("relative", "absolute"),
  time_unit = "d"
)
```

Arguments

data	A data frame.
condition	Column name for storage condition (e.g. temperature).
time	Column name for time point.
value	Column name for measurement value.
reference_condition	Reference condition for between-condition comparison; NULL uses the first occurring condition.
limit	A positive number for symmetric allowable bias; NULL to skip.
bias_type	"relative" (percent) or "absolute".
time_unit	Time unit: m, min, h, d, month, or y.

Value

An S3 object of class "stability_bias".

Examples

```
set.seed(42)
df <- data.frame(cond = rep(c("2-8C","25C"), each = 8),
                 t = rep(c(0,1,3,7), 4),
                 val = c(rnorm(4,100,2), rnorm(4,98,2),
                         rnorm(4,100,2), rnorm(4,92,3)))
stability_bias(df, "cond", "t", "val", limit = 5)
```

---

stability_plan *stability_plan — CLSI EP25 Appendix A time-point planning*

Description

Based on the ratio of expected drift to allowable drift and the reproducibility variability, looks up the minimum number of time points required per regression series in the EP25 Appendix A tables.

Usage

```
stability_plan(
  allowable_drift,
  variability,
  replicates = 1L,
  expected_drift = NULL,
  power = c(0.8, 0.9),
  mode = c("simple", "components"),
  bias_type = c("relative", "absolute")
)
```

Arguments

allowable_drift	Allowable drift (positive number).
variability	Repeatability CV/SD (simple mode) or a data frame of variance components.
replicates	Number of replicate measurements per time point.
expected_drift	Expected drift; NULL defaults to half of allowable_drift.
power	Statistical power: 0.8 or 0.9.
mode	"simple" or "components".
bias_type	"relative" or "absolute".

Value

An S3 object of class "stability_plan".

Examples

```
stability_plan(4, 1.5, replicates = c(1,2,3))

comp <- data.frame(source = c("repeatability","lot","operator"),
                   variability = c(1.5,1.0,0.8),
                   levels = c(1,3,2))
stability_plan(4, comp, replicates = c(1,2,3), mode = "components")
```

---

stability_regression *stability_regression — CLSI EP25 regression-based stability assessment*

Description

Performs simple linear regression on a single condition's measurements (self mode) or on paired biases between test and reference conditions (compare mode), with one-sided confidence limits to evaluate whether drift remains within allowable limits.

Usage

```
stability_regression(
  data,
  time,
  value,
  condition = NULL,
  mode = c("self", "compare"),
  test_condition = NULL,
  reference_condition = NULL,
  bias_type = c("relative", "absolute"),
  direction = c("auto", "decrease", "increase"),
  conf.level = 0.95,
```

```
  limit = NULL,
  time_unit = "d"
)
```

Arguments

data	A data frame.
time	Column name for time point.
value	Column name for measurement value.
condition	Column name for storage condition; required for compare mode.
mode	"self" (single-condition self-reference) or "compare" (test vs reference condition).
test_condition	Test condition; auto-selected if NULL.
reference_condition	Reference condition; NULL uses the first condition.
bias_type	"relative" (percent) or "absolute".
direction	"auto", "decrease", or "increase".
conf.level	One-sided confidence level, default 0.95.
limit	A positive number for symmetric allowable bias; NULL to skip.
time_unit	Time unit: m, min, h, d, month, or y.

Value

An S3 object of class "stability_regression".

Examples

```
set.seed(42)
df <- data.frame(cond = rep(c("2-8C","25C"), each = 8),
                 t = rep(c(0,1,3,7), 4),
                 val = c(rnorm(4,100,2), rnorm(4,98,2),
                         rnorm(4,100,2), rnorm(4,92,3)))
stability_regression(df, "t", "val", "cond", test_condition = "25C", limit = 5)

set.seed(42)
df <- data.frame(cond = rep(c("Ref","Test"), each = 8),
                 t = rep(c(0,1,3,7), 4),
                 val = c(rnorm(4,100,2), rnorm(4,99,2),
                         rnorm(4,100,2), rnorm(4,94,3)))
stability_regression(df, "t", "val", "cond", mode = "compare",
                     test_condition = "Test", reference_condition = "Ref", limit = 5)
```

stability_time *stability_time — Predict time-to-limit from a stability regression*

Description

Given a fitted regression model, computes the time at which the point estimate or one-sided confidence limit first reaches the allowable bias. May extrapolate beyond the observed time range.

Usage

```
stability_time(object, limit = NULL, max_time = NULL)
```

Arguments

object	A stability_regression object.
limit	Symmetric allowable bias.
max_time	Maximum search time; NULL defaults to 100x the maximum observed time.

Value

An S3 object of class "stability_time".

Examples

```
set.seed(42)
df <- data.frame(cond = rep(c("2-8C","25C"), each = 8),
                 t = rep(c(0,1,3,7), 4),
                 val = c(rnorm(4,100,2), rnorm(4,98,2),
                         rnorm(4,100,2), rnorm(4,92,3)))
r <- stability_regression(df, "t", "val", "cond", test_condition = "25C", limit = 5)
stability_time(r)

set.seed(42)
df <- data.frame(cond = rep(c("Ref","Test"), each = 8),
                 t = rep(c(0,1,3,7), 4),
                 val = c(rnorm(4,100,2), rnorm(4,99,2),
                         rnorm(4,100,2), rnorm(4,94,3)))
r2 <- stability_regression(df, "t", "val", "cond", mode = "compare",
                           test_condition = "Test", reference_condition = "Ref", limit = 5)
stability_time(r2, limit = 5)
```

summary.commutability_result

Summarize a commutability assessment

Description

Summarize a commutability assessment

Usage

```
## S3 method for class 'commutability_result'
summary(object, ...)
```

Arguments

object	A commutability_result object.
...	Reserved for the S3 generic.

Value

The unchanged object, invisibly.

summary.fourfold_table

S3 summary method

Description

Summarizes all key analysis information already executed in a fourfold_table object, formatted following summary.mcr() pattern (mcr.R).

Usage

```
## S3 method for class 'fourfold_table'
summary(object, digits = 3, ...)
```

Arguments

object	a fourfold_table object
digits	Number of decimal places, default 3
...	Reserved arguments

Value

Invisible fourfold_table object

Examples

```
tab <- counts_to_table(tp = 85, fp = 3, tn = 90, fn = 2,
                       candidate = "new method", reference = "gold standard")
summary(tab)
tab <- diagnostics(tab); tab <- kappa(tab); tab <- mcnemar(tab)
summary(tab)
```

`summary.mcr` *S3 summary method*

Description

Summarize key information from all executed analyses in the `mcr` object (descriptive statistics, correlation analysis, regression analysis, outlier detection, Bland-Altman analysis, and bias analysis).

Usage

```
## S3 method for class 'mcr'
summary(object, ...)
```

Arguments

`object`	`mcr` object
`...`	Additional arguments

Value

Invisible `mcr` object

Examples

```
df <- data.frame(sid = 1:30,
                 test = rnorm(30, 50, 10),
                 ref  = rnorm(30, 50, 10))
obj <- mcr(df, "sid", "test", "ref")
summary(obj)
obj <- correlation(obj)
obj <- regression(obj, method = "ols")
obj <- bland_altman(obj)
summary(obj)
```

summary.precision *S3 summary method*

Description

Summarize all executed analyses in a `precision` object, including data description, analysis status checklist, outlier detection, normality tests, variance components, confidence intervals, and precision profile.

Usage

```
## S3 method for class 'precision'
summary(object, ...)
```

Arguments

`object`	`precision` object
`...`	Reserved arguments

Value

Invisible `precision` object

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
summary(obj)
```

summary.roc *Summary (S3 method)*

Description

Output key information from all analyses performed on the roc object.

Usage

```
## S3 method for class 'roc'
summary(object, ...)
```

Arguments

`object`	roc object
`...`	reserved arguments

Value

invisible roc object

Examples

```
obj <- roc(ivd_roc_example, cols = c("x1", "x2"), reference = "ref")
summary(obj)
obj <- auc(obj)
obj <- cutoff(obj)
obj <- mlr(obj)
summary(obj)
```

---

tukey.bottle_anova *Tukey HSD post-hoc test (S3 method)*

Description

Perform Tukey HSD pairwise comparisons on an ANOVA result, with compact letter display.

Usage

```
## S3 method for class 'bottle_anova'
tukey(x, term = NULL, conf.level = NULL, ...)
```

Arguments

x	A bottle_anova object
term	Effect term to analyse (character vector); NULL for all non-residual terms
conf.level	Confidence level; NULL uses the value stored in the object
...	Other arguments (unused)

Value

An S3 object of class "tukey" with components:

response_name	Response variable name
conf.level	Confidence level
tukey_list	List, one element per term, each with matrix, means, cld

Note: The original bottle_anova object's $tukey_results is also updated.

Examples

```
obj <- bottle_anova(ivd_bottle_example, value ~ bottle)
res <- tukey(obj)
res
```

`uncertainty_characterization` *Estimate uncertainty from interlaboratory characterization*

Description

Calculates the unweighted mean of laboratory means and its standard error. Common calibration or group-correlated components should be supplied separately to `uncertainty_combine()` so they are not counted repeatedly.

Usage

```
uncertainty_characterization(
  data,
  laboratory,
  result,
  name = "characterization",
  relative = FALSE
)
```

Arguments

`data`	Data frame.
`laboratory`	Laboratory or measurement-procedure column.
`result`	Numeric result column.
`name`	Component name.
`relative`	Return the component relative to the assigned value.

Value

An `uncertainty_characterization` object.

References

CLSI EP30-A; CLSI EP32-R.

Examples

```
characterization_data <- data.frame(
  laboratory = rep(paste0("Lab", 1:4), each = 3),
  result = c(99, 100, 101, 100, 101, 100,
             98, 99, 100, 101, 102, 101)
)
characterization <- uncertainty_characterization(
  characterization_data, "laboratory", "result"
)
characterization$laboratory_summary
```

`uncertainty_combine` *Combine standard uncertainty components*

Description

Applies the GUM first-order covariance propagation formula `c' Sigma c`. Components from any estimator in this module can be supplied directly.

Usage

```
uncertainty_combine(
  ...,
  sensitivity = NULL,
  correlation = NULL,
  value = NULL,
  coverage = 0.95,
  k = NULL
)
```

Arguments

`...`	Uncertainty components, analysis objects containing `$component` or `$components`, or lists of such objects.
`sensitivity`	Optional sensitivity coefficient per component.
`correlation`	Optional correlation matrix.
`value`	Optional output quantity value, needed to mix relative and absolute components.
`coverage`	Coverage probability.
`k`	Optional fixed coverage factor. Otherwise a normal or effective-df Student t factor is used.

Value

An `uncertainty_budget` object.

References

CLSI EP29-A.

Examples

```
repeatability <- uncertainty_type_a(
  c(9.9, 10.0, 10.1, 10.0), "repeatability", quantity = "mean"
)
calibrator <- uncertainty_type_b(
  "calibrator", estimate = 10, uncertainty = 0.2, k = 2,
  distribution = "normal"
)
```

```
budget <- uncertainty_combine(repeatability, calibrator, value = 10)
budget$components
```

`uncertainty_homogeneity` *Estimate the between-unit homogeneity uncertainty of a reference material*

Description

Uses the EP30 one-way ANOVA calculations, including the effective replicate count for an unbalanced design and the inhomogeneity that can be hidden by measurement repeatability. The reported component is the larger of the observed between-unit SD and the hidden-inhomogeneity estimate.

Usage

```
uncertainty_homogeneity(
  data,
  unit,
  result,
  name = "between-unit homogeneity",
  relative = FALSE
)
```

Arguments

`data`	Data frame in long format.
`unit`	Bottle, vial, or material-unit column.
`result`	Numeric measurement-result column.
`name`	Component name.
`relative`	Return the component on a relative scale.

Value

An `uncertainty_homogeneity` object.

References

CLSI EP30-A.

Examples

```
homogeneity_data <- data.frame(
  unit = rep(paste0("V", 1:4), each = 2),
  result = c(99, 100, 101, 100, 98, 99, 102, 101)
)
homogeneity <- uncertainty_homogeneity(
  homogeneity_data, "unit", "result"
)
homogeneity$component
```

`uncertainty_iqc` *Estimate top-down uncertainty from internal quality-control data*

Description

Uses a one-way random-effects ANOVA when runs contain replicates. The object reports within-run, between-run, single-result, and overall-mean estimates.

Usage

```
uncertainty_iqc(
  data,
  result,
  run = NULL,
  quantity = c("single", "mean"),
  name = "IQC",
  relative = FALSE
)
```

Arguments

`data`	Data frame.
`result`	Numeric result column.
`run`	Optional run column. If omitted, the SD of all results is used.
`quantity`	Component returned in `$component`: `"single"` or `"mean"`.
`name`	Component name.
`relative`	Return the selected component on a relative scale.

Value

An `uncertainty_iqc` object.

References

CLSI EP29-A.

Examples

```
iqc_data <- data.frame(
  run = rep(1:5, each = 2),
  result = c(99, 101, 100, 102, 98, 100, 101, 103, 99, 100)
)
iqc <- uncertainty_iqc(iqc_data, "result", run = "run")
iqc$variance_components
```

---

uncertainty_propagate *Propagate uncertainty through a general measurement model*

Description

The delta method numerically estimates sensitivity coefficients and applies GUM first-order propagation. Monte Carlo propagation samples the stored component distributions and returns a central quantile interval.

Usage

```
uncertainty_propagate(
  model,
  components,
  method = c("delta", "monte_carlo"),
  correlation = NULL,
  simulations = 100000L,
  coverage = 0.95,
  seed = NULL
)
```

Arguments

model	Function whose named arguments match component names.
components	Components or analysis objects containing components.
method	"delta" or "monte_carlo".
correlation	Optional correlation matrix. Nonidentity correlation in Monte Carlo mode currently requires normal or standardized components.
simulations	Number of Monte Carlo draws.
coverage	Coverage probability.
seed	Optional random seed.

Value

An uncertainty_propagation object.

References

CLSI EP29-A.

Examples

```
numerator <- uncertainty_type_b(
  "numerator", estimate = 20, uncertainty = 0.4,
  distribution = "standard"
)
denominator <- uncertainty_type_b(
  "denominator", estimate = 10, uncertainty = 0.1,
  distribution = "standard"
)
propagated <- uncertainty_propagate(
  function(numerator, denominator) numerator / denominator,
  list(numerator, denominator), method = "delta"
)
propagated$estimate
```

---

uncertainty_stability *Estimate long-term stability uncertainty*

Description

Fits result on time and calculates the standard uncertainty at the requested shelf life as the slope standard error multiplied by shelf life. Separate components are returned for each optional storage condition.

Usage

```
uncertainty_stability(
  data,
  time,
  result,
  shelf_life,
  condition = NULL,
  name = "long-term stability",
  relative = FALSE
)
```

Arguments

data	Data frame.
time	Numeric storage-time column.
result	Numeric result column.
shelf_life	Positive time on the same scale as time.
condition	Optional storage-condition column.
name	Base component name.
relative	Return components relative to their fitted intercept-level mean.

Value

An uncertainty_stability object.

References

CLSI EP30-A.

Examples

```
stability_data <- data.frame(
  time = rep(0:4, each = 2),
  result = c(100.1, 99.9, 100.0, 99.8, 99.7, 99.9,
             99.6, 99.8, 99.5, 99.7)
)
stability_u <- uncertainty_stability(
  stability_data, "time", "result", shelf_life = 6
)
stability_u$summary
```

---

uncertainty_traceability

Accumulate uncertainty along a metrological traceability chain

---

Description

Accumulate uncertainty along a metrological traceability chain

Usage

```
uncertainty_traceability(stages, value = NULL, coverage = 0.95, k = NULL)
```

Arguments

stages	Named list. Each element contains the new uncertainty components introduced at that traceability stage.
value	Optional assigned value for absolute/relative conversion.
coverage	Coverage probability.
k	Optional fixed coverage factor.

Value

An uncertainty_traceability object.

References

CLSI EP29-A; CLSI EP32-R.

Examples

```
calibration <- uncertainty_type_b(
  "calibration", estimate = 100, uncertainty = 1, k = 2,
  distribution = "normal"
)
transport <- uncertainty_type_b(
  "transport", estimate = 100, lower = 99.5, upper = 100.5,
  distribution = "rectangular"
)
chain <- uncertainty_traceability(
  list(calibration = list(calibration), transport = list(transport)),
  value = 100
)
chain$summary
```

---

uncertainty_type_a *Estimate a Type A uncertainty component*

Description

Uses the sample standard deviation for a future single result and the standard error for a mean. No normality or fitness-for-purpose decision is made.

Usage

```
uncertainty_type_a(x, name, quantity = c("single", "mean"), relative = FALSE)
```

Arguments

x	Numeric repeated measurements.
name	Component name.
quantity	"single" or "mean".
relative	Store the standard uncertainty relative to the absolute mean.

Value

An uncertainty_component object.

References

CLSI EP29-A.

Examples

```
type_a <- uncertainty_type_a(
  c(10.1, 9.9, 10.0, 10.2, 9.8),
  name = "repeatability", quantity = "mean"
)
type_a
```

`uncertainty_type_b` *Estimate a Type B uncertainty component*

Description

Converts limits, an expanded uncertainty, or an already standardized value to standard uncertainty. For bounded distributions `lower` and `upper` define the interval. For normal and Student t distributions they define a central interval with probability `coverage`.

Usage

```
uncertainty_type_b(
  name,
  estimate,
  lower = NULL,
  upper = NULL,
  uncertainty = NULL,
 distribution = c("rectangular", "triangular", "normal", "student_t", "u_shaped",
    "standard"),
  coverage = 0.95,
  k = NULL,
  df = Inf,
  relative = FALSE
)
```

Arguments

`name`	Component name.
`estimate`	Best estimate of the input quantity.
`lower, upper`	Optional interval limits.
`uncertainty`	Optional standard or expanded uncertainty.
`distribution`	Distribution model.
`coverage`	Central probability associated with normal or t limits.
`k`	Coverage factor when `uncertainty` is expanded.
`df`	Degrees of freedom for Student t or the component.
`relative`	Store uncertainty relative to the absolute estimate.

Value

An `uncertainty_component` object.

References

CLSI EP29-A.

Examples

```
rectangular <- uncertainty_type_b(
  name = "calibrator", estimate = 100, lower = 98, upper = 102,
  distribution = "rectangular"
)
rectangular

uncertainty_type_b(
  "certificate", estimate = 100, uncertainty = 2, k = 2,
  distribution = "normal"
)
```

---

variance.precision *S3 variance component estimation method*

Description

For each sample, run VCA (`VCA::anovaVCA` for balanced designs / ANOVA, or `VCA::remlVCA` for REML) to estimate variance components, compute SD, CV%, and their percentage of total variance. Results are stored in the `$results` and `$vc` slots.

Usage

```
## S3 method for class 'precision'
variance(x, digits = 4, method = c("auto", "anova", "reml"), ...)
```

Arguments

`x`	`precision` object
`digits`	Number of decimal places, default 4
`method`	Estimation method: `"auto"` (default) chooses per sample group `anovaVCA` when the design is balanced and `remlVCA` otherwise, using `VCA::isBalanced()` (VCA's own balancedness criterion); `"anova"` forces ANOVA, `"reml"` forces REML.
`...`	Additional arguments passed to `VCA::anovaVCA()` / `VCA::remlVCA()` (e.g. `NegVC`, `conf.level`)

Value

Updated `precision` object, `$results` contains per-sample VCA results, `$vc` is the summary variance component table

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- variance(obj)
```

vc.precision *Backward-compatible alias for variance.precision*

Description

Backward-compatible alias for variance.precision

Usage

```
## S3 method for class 'precision'
vc(x, digits = 4, method = c("auto", "anova", "reml"), ...)
```

Arguments

x	A precision object.
digits	Number of decimal places, default 4.
method	Estimation method: "auto", "anova", or "reml".
...	Additional arguments passed to the underlying VCA method.

Value

The same result as variance.precision: an updated precision object, invisibly.

Examples

```
data(VCAdata1, package = "VCA")
obj <- precision(VCAdata1, y ~ day/run, by = "sample")
obj <- vc(obj)
```

youden_plot *Youden plot*

Description

Draws a Youden plot comparing two measurement systems. If target means and SDs are not provided, they are estimated from the data.

Usage

```
youden_plot(
  data,
  sample1,
  sample2,
  mean1 = NULL,
  mean2 = NULL,
  sd1 = NULL,
  sd2 = NULL
)
```

Arguments

data	A data frame.
sample1	Column name for sample 1.
sample2	Column name for sample 2.
mean1	Target mean for sample 1; NULL to estimate.
mean2	Target mean for sample 2; NULL to estimate.
sd1	Target SD for sample 1; NULL to estimate.
sd2	Target SD for sample 2; NULL to estimate.

Value

An object of class "youden_plot" with summary statistics and plot data. Use print() to show summary and plot() to draw the Youden plot.

Examples

```
set.seed(42)
df <- data.frame(
  m1 = rnorm(30, 100, 5),
  m2 = rnorm(30,  98, 5)
)
res <- youden_plot(df, "m1", "m2", mean1 = 100, mean2 = 100, sd1 = 5, sd2 = 5)
print(res)
plot(res)
```

Index