Package 'VeraCrop'

September 28, 2026

Type Package

Title Yield Gap Analysis Using Comparative Performance Analysis

Version 0.1.0

Description Implements automated variable-type detection, preprocessing, encoding, scaling, model diagnostics, variable selection, and yield gap computation for agricultural comparative performance analysis (CPA). The comparative performance analysis approach is described in de Bie (2004) <doi:10.1016/j.scienta.2003.11.017>. For an overview of yield gap assessment methodologies, see Kamkar et al. (2025) <doi:10.1016/j.agsy.2025.104392>.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

LazyData true

Depends R (>= 4.0.0)

Imports stats, graphics, grDevices, utils, ggplot2

Suggests lmtest, car, e1071, bit64, patchwork, writexl, relaimpo, testthat (>= 3.0.0), knitr, rmarkdown, spelling

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Abolfazl Derakhshan [aut, cph],
Elham Elahifard [aut, cre],
Aghajan Bahadori [ctb],
Shaban Zarei [ctb],
Hossein Norouzi [ctb]

Maintainer Elham Elahifard <e.elahifard@asnrukh.ac.ir>

Config/roxygen2/version 8.1.0

Repository CRAN

Date/Publication 2026-09-28 07:50:07 UTC

Contents

add_missing_indicators *Add Missing Indicators*

Description

Creates binary indicator variables for missing values

Usage

```
add_missing_indicators(
  data,
  vars = NULL,
  suffix = "_missing",
  as_numeric = FALSE,
  verbose = TRUE
)
```

Arguments

data	A data frame
vars	Character vector of variable names (default: auto-detect all)
suffix	Character, suffix for indicator variables (default: "_missing")
as_numeric	Logical, return indicators as numeric (1/0) or logical (TRUE/FALSE)
verbose	Logical, print progress (default: TRUE)

Value

Data frame with missing indicators added

Examples

```
df <- data.frame(Yield = rnorm(20, 5000, 500),
                 Nitrogen = c(rnorm(17, 120, 20), NA, NA, NA))
result <- add_missing_indicators(df, verbose = FALSE)
names(result)
```

`adjust_intercept` *Adjust Intercept After Back-Transformation*

Description

When predictors are standardized, the intercept must be adjusted to make predictions in the original scale.

Usage

```
adjust_intercept(
  intercept_std,
  back_transformed_coefs,
  scaling_params,
  response_mean = NULL,
  response_sd = NULL
)
```

Arguments

`intercept_std`	The intercept from the standardized model
`back_transformed_coefs`	Back-transformed coefficients
`scaling_params`	Scaling parameters from `scale_variables`
`response_mean`	Numeric, mean of the response variable. Required if the response variable was also standardized.
`response_sd`	Numeric, standard deviation of the response variable. Required if the response variable was also standardized.

Details

For X standardized only: intercept_original = intercept_std - sum(beta_original * mean_X)

For both X and Y standardized: intercept_original = mean_Y + intercept_std * sd_Y - sum(beta_original * mean_X)

Value

Adjusted intercept for the original scale model

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
scaled <- scale_variables(df, exclude_vars = "Yield", verbose = FALSE)
model <- lm(Yield ~ N_rate_scaled, data = scaled$data)
back_coefs <- back_transform_coefficients(coef(model), scaled$scaling_params)
adjust_intercept(coef(model)["(Intercept)"], back_coefs[-1], scaled$scaling_params)
```

`back_transform_coefficients` *Back-transform Coefficients to Original Scale*

Description

Converts coefficients from standardized predictors back to the original scale of the predictors.

Usage

```
back_transform_coefficients(
  coefs,
  scaling_params,
  response_sd = NULL,
  keep_names = TRUE
)
```

Arguments

`coefs`	Named vector of standardized coefficients. Names should match the scaled variable names.
`scaling_params`	Scaling parameters from `scale_variables`.
`response_sd`	Numeric, standard deviation of the response variable. Required if the response variable was also standardized.
`keep_names`	Logical, keep scaled names or rename to original. Default: TRUE

Details

For standardized predictors only (X scaled, Y not scaled): beta_original = beta_standardized / sd(X)

For both X and Y scaled: beta_original = beta_standardized * (sd_Y / sd_X)

Value

Named vector of back-transformed coefficients

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
scaled <- scale_variables(df, exclude_vars = "Yield", verbose = FALSE)
model <- lm(Yield ~ N_rate_scaled, data = scaled$data)
back_transform_coefficients(coef(model), scaled$scaling_params)
```

back_transform_data *Back-transform Data*

Description

Transforms scaled data back to the original scale using stored parameters.

Usage

```
back_transform_data(data, scaling_params, remove_scaled = FALSE)
```

Arguments

data	A data frame containing scaled variables
scaling_params	Scaling parameters from scale_variables
remove_scaled	Logical, remove scaled variables after back-transformation. Default: FALSE

Value

Data frame with variables back-transformed to original scale

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
scaled <- scale_variables(df, exclude_vars = "Yield", verbose = FALSE)
restored <- back_transform_data(scaled$data, scaled$scaling_params)
head(restored$N_rate)
```

bootstrap_cpa *Bootstrap Confidence Intervals for the CPA Yield Gap Decomposition*

Description

Uses case resampling (nonparametric bootstrap) to estimate percentile confidence intervals for the overall yield gap and for each significant variable's contribution to it (Gap_Component and Share_Percent in the CPA table). The set of significant variables is held fixed across all resamples – by default it is determined once, from the original (non-resampled) data – so that the reported intervals describe the uncertainty of the CPA decomposition itself, not of variable selection. Optionally, variable selection is also re-run independently on each resample purely as a diagnostic, to report how often each fixed variable would be re-selected on its own (selection_stability).

Usage

```
bootstrap_cpa(
  data,
  response = "Yield",
  sig_vars = NULL,
  selection_method = c("ftest", "aic"),
  n_boot = 1000,
  conf_level = 0.95,
  top_percentile = NULL,
  min_top_samples = NULL,
  check_selection_stability = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame, already preprocessed (e.g. via prep_yield_gap) and ready for yield_gap_analysis.
response	Character. Name of the response variable. Default: "Yield".
sig_vars	Character vector of significant variable names to hold fixed across all bootstrap resamples. If NULL (default), this is determined once from the original data using selection_method (mirroring yield_gap_analysis's default of use_bonferroni = FALSE). If you already have a yield_gap_analysis result, it is recommended to pass its variables explicitly here (e.g. resultcpa_tableVariable[result$cpa_t != "Intercept"]) so the bootstrap exactly matches that analysis.
selection_method	One of "ftest" or "aic", used only when sig_vars is NULL, to select variables once from the original data. Default: "ftest".
n_boot	Integer. Number of bootstrap resamples. Default: 1000.
conf_level	Numeric. Confidence level for the percentile interval. Default: 0.95.
top_percentile	Numeric or NULL. Percentile used to define the high-yield subset, passed through to calculate_opt_values. If NULL (default), it is auto-tuned from the sample size using the same thresholds as yield_gap_analysis (0.99 for n >= 100, 0.95 for n >= 50, 0.90 for n >= 30, 0.85 otherwise), so the default reproduces that function's behavior on the same data.
min_top_samples	Integer or NULL. Minimum number of rows required in the high-yield subset before calculate_opt_values relaxes top_percentile. If NULL (default), it is auto-tuned from the sample size using the same thresholds as yield_gap_analysis (20 for n >= 100, 10 for n >= 50, 5 for n >= 30, 3 otherwise).
check_selection_stability	Logical. If TRUE, also re-run variable selection independently on each bootstrap resample and report how often each fixed variable is re-selected. Default: TRUE.
seed	Integer or NULL. Random seed for reproducibility; the caller's global random state is saved before resampling and restored afterward. Default: NULL.
verbose	Logical. Print progress and a summary table. Default: TRUE.

Value

A list with components:

`sig_vars` The fixed variable set used across all resamples.

`summary` A data frame with one row per quantity (overall yield mean/optimal/gap, plus each variable's Gap_Component and Share_Percent), giving the point estimate from the original data, the bootstrap mean and standard error, and the percentile confidence interval.

`boot_gap_component`
`n_boot` x `length(sig_vars)` matrix of bootstrap `Gap_Component` draws (one column per variable).

`boot_share_percent`
`n_boot` x `length(sig_vars)` matrix of bootstrap `Share_Percent` draws.

`boot_yield_mean`, `boot_yield_opt`, `boot_yield_gap`
Numeric vectors of length `n_boot` with the bootstrap draws of each overall quantity.

`selection_stability`
If requested, a data frame giving, for each fixed variable, how often it was independently re-selected across resamples (`Times_Selected`, `Selection_Rate`).

`n_boot`, `n_failed`, `conf_level`
Bookkeeping: resamples requested, resamples that failed and were skipped (e.g. due to a singular fit), and the confidence level used.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
result <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
sig_vars <- result$cpa_table$Variable[result$cpa_table$Variable != "Intercept"]
boot <- bootstrap_cpa(prep$data, response = "Yield", sig_vars = sig_vars,
                      n_boot = 200, verbose = FALSE)
boot$summary
```

`calculate_minmax_effects` *Calculate Min-Max Effects of Variables on Yield*

Description

Estimates the effect of each variable on yield by comparing predictions when the variable is set to its minimum vs. maximum.

Usage

```
calculate_minmax_effects(
  model,
  data,
  vars,
  use_quantiles = TRUE,
  quantile_range = c(0.05, 0.95)
)
```

Arguments

model	A fitted lm model object.
data	A data frame used for prediction.
vars	Character vector of variable names to evaluate.
use_quantiles	Logical. Use 5th-95th percentile range. Default: TRUE.
quantile_range	Numeric vector of length 2. Default: c(0.05, 0.95).

Value

A data frame with columns: Variable, Effect, EffectAbs, EffectPct.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
sel <- select_vars_ftest(prep$data, response = "Yield", verbose = FALSE)
mm <- calculate_minmax_effects(sel$final_model, prep$data, sel$significant_vars)
mm
```

---

calculate_opt_values *Calculate Optimal Values for Selected Variables*

Description

Determines the optimal value for each selected predictor variable.

Usage

```
calculate_opt_values(
  data,
  vars,
  coefs,
  data_top,
  response_var = "Yield",
  top_percentile = 0.99,
  min_top_samples = 20
)
```

Arguments

data	A data frame (full dataset).
vars	Character vector of selected variable names.
coefs	Named numeric vector of model coefficients.
data_top	A data frame of high-yield observations.
response_var	Character. Name of the response variable. Default: "Yield".
top_percentile	Numeric. Percentile used to define top yields. Default: 0.99.
min_top_samples	Integer. Minimum number of top-yield samples. Default: 20.

Value

A list with components: var_table, used_percentile, n_top_samples, n_total.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
sel <- select_vars_ftest(prep$data, response = "Yield", verbose = FALSE)
sig_vars <- sel$significant_vars
coefs_vars <- coef(sel$final_model)[sig_vars]
yield_thr <- quantile(prep$data$Yield, 0.90, na.rm = TRUE)
data_top <- prep$data[prep$data$Yield >= yield_thr, ]
opt <- calculate_opt_values(prep$data, sig_vars, coefs_vars, data_top,
                            response_var = "Yield", top_percentile = 0.90)
opt$var_table
```

---

calculate_relative_importance

Calculate Relative Importance of Variables

Description

Computes the relative importance of predictors using LMG or standardized beta.

Usage

```
calculate_relative_importance(
  data,
  vars,
  response = "Yield",
  method = c("lmg", "std_beta"),
  verbose = TRUE
)
```

Arguments

data	A data frame containing the variables.
vars	Character vector of variable names.
response	Character. Name of the response variable. Default: "Yield".
method	Character. One of "lmg" or "std_beta". Default: "lmg".
verbose	Logical. Print progress messages. Default: TRUE.

Value

A list with components: method and results.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
sel <- select_vars_ftest(prep$data, response = "Yield", verbose = FALSE)
ri <- calculate_relative_importance(prep$data, sel$significant_vars,
                                    response = "Yield", verbose = FALSE)
ri$results
```

---

check_assumptions *Check Linear Regression Assumptions*

Description

Automatically checks the five main assumptions of linear regression: 1. Linearity (using Ramsey RESET test or plot) 2. Independence of residuals (Durbin-Watson with p-value) 3. Homoscedasticity (Breusch-Pagan) 4. Normality of residuals (Shapiro-Wilk with skewness/kurtosis) 5. Multicollinearity (VIF/GVIF with proper scaling)

Usage

```
check_assumptions(
  model,
  data = NULL,
  plot = TRUE,
  verbose = TRUE,
  max_warnings = 2,
  seed = 42,
  alpha = 0.05,
  vif_threshold = 10,
  vif_warning_threshold = 5
)
```

Arguments

model	A fitted lm model object.
data	Optional data frame. May be either: (a) the exact complete-case data used to fit the model, or (b) the original unfiltered data - in which case row names are used to safely map influential points back to FarmID. For best results, preprocess data using handle_missing_values() before fitting.
plot	Logical. Show diagnostic plots. Default: TRUE.
verbose	Logical. Print results. Default: TRUE.
max_warnings	Integer. Maximum number of warnings allowed for data readiness. Default: 2.
seed	Integer. Random seed for reproducible sampling in large datasets. Default: 42.
alpha	Numeric. Significance level for hypothesis tests. Default: 0.05.
vif_threshold	Numeric. VIF threshold for multicollinearity. Default: 10.
vif_warning_threshold	Numeric. VIF warning threshold. Default: 5.

Value

A list with all test results, summary status, data readiness flag, and (if plot=TRUE) a recorded diagnostic plot object.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
model <- lm(Yield ~ ., data = prep$data)
result <- check_assumptions(model, data = prep$data, plot = FALSE, verbose = FALSE)
result$data_ready
```

---

check_data_ready *Quick Check for Data Readiness*

Description

Quick Check for Data Readiness

Usage

```
check_data_ready(data, response_var = NULL, verbose = TRUE, details = FALSE)
```

Arguments

data	A data frame
response_var	Name of response variable
verbose	Logical, print results
details	Logical, return detailed results (default: FALSE)

Value

If details=TRUE, returns a list with ready, issues, and warnings. Otherwise returns a logical.

Examples

```
df <- data.frame(Yield = rnorm(20, 5000, 500), Nitrogen = rnorm(20, 120, 20))
check_data_ready(df, response_var = "Yield", verbose = FALSE)
```

---

detect_variable_types *Detect variable types*

Description

Detect variable types

Usage

```
detect_variable_types(
  data,
  response_var = "Yield",
  max_categorical_levels = 10,
  treat_numeric_as_ordinal = FALSE,
  auto_detect_ordinal_from_name = TRUE,
  auto_detect_ordinal_mode = "strict",
  ordinal_level_orders = list(),
  reference_method = "mode",
  verbose = TRUE
)
```

Arguments

data	A data frame
response_var	Name of response variable
max_categorical_levels	Maximum levels for categorical (default: 10)
treat_numeric_as_ordinal	Logical, treat numeric with small levels as ordinal (default: FALSE)
auto_detect_ordinal_from_name	Logical, detect ordinal from variable names (default: TRUE)
auto_detect_ordinal_mode	Character, "strict" or "contains" (default: "strict")

ordinal_level_orders
: Named list mapping variable names to an explicit, correctly-ordered character vector of levels (low to high), e.g. `list(severity_level = c("Low", "Medium", "High"))`. Required to treat a CHARACTER or FACTOR variable as ordinal based on its name alone; without an explicit order, such variables are conservatively classified as nominal instead of guessing an order from row order in the data (which is not reproducible and can silently reverse the true ordinal relationship). Not needed for already-ordered factors, and not needed for numeric variables (numeric ordinal order is unambiguous - it is the natural numeric order).

reference_method
: Character, "mode", "first", or "alphabetical" (default: "mode")

verbose
: Logical

Value

List with variable type information

Examples

```
df <- data.frame(
  Yield = rnorm(30, 5000, 500),
  Nitrogen = rnorm(30, 120, 20),
  Region = sample(c("North", "South", "East"), 30, replace = TRUE)
)
types <- detect_variable_types(df, response_var = "Yield", verbose = FALSE)
types$Region$type
```

---

encode_variables *Encode variables for statistical models*

Description

Encode variables for statistical models

Usage

```
encode_variables(
  data,
  var_types,
  response_var = "Yield",
  max_dummy_levels = 10,
  min_freq_for_encoding = 0.05,
  min_count_for_encoding = 5,
  grouping_rule = "or",
  verbose = TRUE
)
```

Arguments

data	A data frame
var_types	Variable types from detect_variable_types()
response_var	Name of response variable
max_dummy_levels	Maximum dummies per variable (default: 10)
min_freq_for_encoding	Minimum frequency for grouping high-cardinality variables (default: 0.05)
min_count_for_encoding	Minimum count for grouping high-cardinality variables (default: 5)
grouping_rule	Character, "or" or "and" for high-cardinality grouping (default: "or")
verbose	Logical

Value

A list with:

data Encoded data frame

encoding_info Metadata describing each encoding

dropped_vars Character vector of variables dropped during encoding

Examples

```
df <- data.frame(
  Yield = rnorm(30, 5000, 500),
  Region = sample(c("North", "South", "East"), 30, replace = TRUE)
)
types <- detect_variable_types(df, response_var = "Yield", verbose = FALSE)
result <- encode_variables(df, types, response_var = "Yield", verbose = FALSE)
names(result$data)
```

---

filter_variables *Apply All Filtering Steps*

Description

Convenience function that applies all filtering steps in the recommended order: 1. Remove non-predictors (IDs) 2. Remove constant variables 3. Remove near-zero variance variables 4. Remove highly correlated variables

Usage

```
filter_variables(
  data,
  response_var = "Yield",
  exclude_vars = NULL,
  exclude_patterns = NULL,
  freq_cut = 19,
  unique_cut = 10,
  corr_threshold = 0.9,
  keep_method = "higher_correlation",
  cor_method = "pearson",
  cor_use = "pairwise.complete.obs",
  verbose = TRUE
)
```

Arguments

data	A data frame
response_var	Name of response variable
exclude_vars	Character vector of variable names to exclude
exclude_patterns	Character vector of regex patterns for ID removal
freq_cut	Numeric, frequency ratio cutoff for NZV (default: 19)
unique_cut	Numeric, percent unique cutoff for NZV (default: 10)
corr_threshold	Numeric, correlation threshold (default: 0.90)
keep_method	Character, method for correlated variable removal
cor_method	Character, correlation method
cor_use	Character, method for handling missing values
verbose	Logical, print progress (default: TRUE)

Details

Returns a list with filtered data and detailed removal records.

Value

A list with:

data	Filtered data frame
removed	List of removed variables by category
n_removed	Total number of variables removed
n_initial	Initial number of columns
n_final	Final number of columns

Examples

```
data(wheat1)
result <- filter_variables(wheat1, response_var = "Yield", verbose = FALSE)
result$n_initial
result$n_final
```

`group_high_cardinality` *Group levels of high-cardinality categorical variables*

Description

Group levels of high-cardinality categorical variables

Usage

```
group_high_cardinality(
  x,
  min_freq = 0.05,
  min_count = 5,
  other_label = "..OTHER..",
  rule = "or"
)
```

Arguments

`x`	Vector to group
`min_freq`	Minimum frequency (proportion) for a level to be kept (default: 0.05)
`min_count`	Minimum count for a level to be kept (default: 5)
`other_label`	Label for grouped levels (default: "..OTHER..")
`rule`	Character, "or" (keep if freq OR count) or "and" (keep if freq AND count)

Value

Factor with grouped levels

Examples

```
x <- c(rep("A", 20), rep("B", 15), rep("C", 2), rep("D", 1))
grouped <- group_high_cardinality(x, min_freq = 0.05, min_count = 5)
table(grouped)
```

handle_missing_values *Handle Missing Values*

Description

Imputes missing values using specified methods. Different methods can be used for different data types.

Usage

```
handle_missing_values(
  data,
  response_var = "Yield",
  numeric_method = "median",
  categorical_method = "mode",
  logical_method = "mode",
  missing_threshold = 0.2,
  ties_method = "first",
  add_indicators = FALSE,
  indicator_suffix = "_missing",
  verbose = TRUE
)
```

Arguments

data	A data frame
response_var	Name of response variable (rows with missing response are removed)
numeric_method	Imputation method for numeric variables: "mean", "median", or "mode" (default: "median")
categorical_method	Imputation method for categorical variables: "mode" (default: "mode")
logical_method	Imputation method for logical variables: "mode" (default: "mode")
missing_threshold	Numeric, threshold for warning about high missing percentage (>= threshold triggers warning, default: 0.20)
ties_method	Character, how to handle ties in mode: "first", "random", or "na" (default: "first") When using "random", call set.seed() before the function for reproducibility.
add_indicators	Logical, add missing indicators before imputation (default: FALSE)
indicator_suffix	Character, suffix for indicator variables (default: "_missing")
verbose	Logical, print progress (default: TRUE)

Details

Percent of imputed cells is calculated relative to the original variables before indicator creation (when add_indicators = TRUE).

Note: integer64 (bit64 package) values: if bit64 is installed, values are preserved as integer64. Otherwise, they are converted to double (may lose precision for values > 2^53).

Mode appropriateness can be adjusted via options("veracrop.mode_threshold"). Default is 1.0. Increase to reduce warnings, decrease to be more strict.

Value

A list with:

`data`	Imputed data frame
`summary`	Summary statistics of imputation
`imputed_vars`	Character vector of imputed variables
`imputation_info`	Detailed information about each imputation
`initial_missing`	Initial number of missing values (before adding indicators)
`missing_after_response_removal`	Missing values after response removal
`remaining_missing`	Number of remaining missing values
`n_imputed_total`	Total number of imputed values
`pct_imputed`	Percentage of original cells imputed
`n_removed_rows`	Number of rows removed due to missing response
`all_na_vars`	Character vector of all-NA variables that could not be imputed
`unsupported_vars`	Character vector of variables with unsupported classes

Examples

```
df <- data.frame(
  Yield = rnorm(30, 5000, 500),
  Nitrogen = c(rnorm(27, 120, 20), NA, NA, NA)
)
result <- handle_missing_values(df, response_var = "Yield", verbose = FALSE)
sum(is.na(result$data))
```

missing_summary *Quick Missing Value Summary*

Description

Provides a quick summary of missing values in a data frame

Usage

```
missing_summary(data, verbose = TRUE)
```

Arguments

data	A data frame
verbose	Logical, print summary (default: TRUE)

Value

A list with missing value statistics (invisibly if verbose=TRUE)

Examples

```
df <- data.frame(Yield = rnorm(20, 5000, 500),
                 Nitrogen = c(rnorm(17, 120, 20), NA, NA, NA))
summary_result <- missing_summary(df, verbose = FALSE)
summary_result$total_missing
```

model_diagnostics *Quick Model Summary with Assumption Checks*

Description

Combines model summary and assumption checks in one function.

Usage

```
model_diagnostics(
  model,
  data = NULL,
  plot = TRUE,
  verbose = TRUE,
  max_warnings = 2,
  ...
)
```

Arguments

model	A fitted lm model object.
data	Optional data frame. May be either complete-case data or original data with NAs (row names used for mapping).
plot	Logical. Show diagnostic plots. Default: TRUE.
verbose	Logical. Print results. Default: TRUE.
max_warnings	Integer. Maximum number of warnings allowed for data readiness. Default: 2.
...	Additional arguments passed to check_assumptions.

Value

A list with model summary and assumption checks.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
model <- lm(Yield ~ ., data = prep$data)
result <- model_diagnostics(model, data = prep$data, plot = FALSE, verbose = FALSE)
result$summary$r.squared
```

---

pareto_analysis *Pareto Analysis of Yield Gap Contributions*

Description

Identifies the most important variables contributing to the yield gap.

Usage

```
pareto_analysis(yg_results, top_n = NULL, verbose = TRUE)
```

Arguments

yg_results	List. Output from yield_gap_components.
top_n	Integer or NULL. Number of top variables to report.
verbose	Logical. Print the Pareto table to the console. Default: TRUE.

Value

A list with components: pareto_table, top_n, top_variables.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
cpa_input <- results$cpa_table[, c("Variable", "Beta", "Mean", "Opt")]
yg <- yield_gap_components(cpa_input)
pareto <- pareto_analysis(yg, verbose = FALSE)
pareto$pareto_table
```

---

plot_all_graphs *Generate All Yield Gap Plots*

Description

Convenience function that calls all five plotting functions and optionally combines them into a single figure using **patchwork**.

Usage

```
plot_all_graphs(
  results,
  top_percentile = 0.99,
  save_plots = FALSE,
  output_dir = tempdir(),
  verbose = TRUE
)
```

Arguments

results	List. Output from yield_gap_analysis.
top_percentile	Numeric. Passed to plot_contribution_shares. Default: 0.99.
save_plots	Logical. Save individual and combined plots. Default: FALSE.
output_dir	Character. Directory for saved plots. Default: tempdir().
verbose	Logical. Print progress messages while generating plots. Default: TRUE.

Value

Invisibly returns a named list of all five ggplot2 objects.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_all_graphs(results, top_percentile = 0.95,
                save_plots = TRUE, output_dir = tempdir(), verbose = FALSE)
```

plot_contribution_shares *Plot Variable Contributions to Yield Gap*

Description

Horizontal bar chart showing the percentage contribution of each selected variable to the total yield gap.

Usage

```
plot_contribution_shares(
  results,
  top_percentile = 0.99,
  save_plot = FALSE,
  filename = "contribution_shares.png"
)
```

Arguments

results	List. Output from yield_gap_analysis.
top_percentile	Numeric. Used in subtitle label. Default: 0.99.
save_plot	Logical. Save plot to file. Default: FALSE.
filename	Character. Output file name. Default: "contribution_shares.png".

Value

A ggplot2 object, or NULL if no data available.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_contribution_shares(results)
```

plot_minmax_effects *Plot Min-Max Effects of Variables on Yield*

Description

Diverging horizontal bar chart showing how much yield changes when each variable moves from its minimum to maximum value.

Usage

```
plot_minmax_effects(
  results,
  save_plot = FALSE,
  filename = "minmax_effects.png"
)
```

Arguments

results	List. Output from yield_gap_analysis.
save_plot	Logical. Save plot to file. Default: FALSE.
filename	Character. Output file name. Default: "minmax_effects.png".

Value

A ggplot2 object, or NULL if no data available.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_minmax_effects(results)
```

plot_observed_vs_predicted *Plot Observed vs. Predicted Yield*

Description

Scatter plot comparing observed and model-predicted yields, including a 1:1 line, regression line, and R-squared/RMSE annotations.

Usage

```
plot_observed_vs_predicted(
  results,
  save_plot = FALSE,
  filename = "observed_vs_predicted.png"
)
```

Arguments

results	List. Output from yield_gap_analysis.
save_plot	Logical. Save plot to file. Default: FALSE.
filename	Character. Output file name. Default: "observed_vs_predicted.png".

Value

A ggplot2 object, or NULL if no data available.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_observed_vs_predicted(results)
```

---

plot_relative_importance

Plot Relative Importance of Variables

Description

Horizontal bar chart of relative importance percentages, using either the LMG or standardized beta method.

Usage

```
plot_relative_importance(
  results,
  save_plot = FALSE,
  filename = "relative_importance.png"
)
```

Arguments

results	List. Output from yield_gap_analysis.
save_plot	Logical. Save plot to file. Default: FALSE.
filename	Character. Output file name. Default: "relative_importance.png".

Value

A ggplot2 object, or NULL if no data available.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_relative_importance(results)
```

---

plot_yield_gap_distribution

Plot Distribution of Yield Gap Across Fields

Description

Histogram with density curve showing the distribution of per-field yield gaps, with mean and median reference lines.

Usage

```
plot_yield_gap_distribution(
  results,
  save_plot = FALSE,
  filename = "yield_gap_distribution.png"
)
```

Arguments

results	List. Output from yield_gap_analysis.
save_plot	Logical. Save plot to file. Default: FALSE.
filename	Character. Output file name. Default: "yield_gap_distribution.png".

Value

A ggplot2 object, or NULL if no data available.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
plot_yield_gap_distribution(results)
```

preprocessing_report *Create a Preprocessing Report*

Description

Create a Preprocessing Report

Usage

```
preprocessing_report(
  preprocessed_result,
  validation_result = NULL,
  output_file = NULL,
  verbose = TRUE
)
```

Arguments

preprocessed_result	Output from prep_yield_gap()
validation_result	Optional output from validate_preprocessing()
output_file	Path to output file (optional)
verbose	Logical, print report to console

Value

Character string with report (invisibly if verbose=TRUE)

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
report <- preprocessing_report(prep, verbose = FALSE)
cat(report[1:5], sep = "\n")
```

prep_yield_gap *Main Preprocessing Pipeline for Yield Gap Analysis*

Description

Runs the full preprocessing pipeline end to end: removes non-predictor (ID-like) columns, imputes missing values, detects variable types, encodes variables for linear regression, removes constant variables introduced by encoding, optionally scales continuous variables, and optionally removes near-zero-variance and highly correlated variables.

Usage

```
prep_yield_gap(data, response_var = "Yield", ...)
```

Arguments

`data`	A data frame.
`response_var`	Name of the response variable. Default: `"Yield"`.
`...`	Additional named parameters overriding the defaults below.

trim_whitespace Logical, default TRUE. Trims leading/trailing whitespace from all character columns before any other processing (see `trim_whitespace`), so that e.g. "Sirvan" and "Sirvan " are correctly treated as the same category rather than two distinct ones.

max_categorical_levels Integer or NULL (auto-tuned). Passed to `detect_variable_types()`.

max_dummy_levels Integer or NULL (auto-tuned). Passed to `encode_variables()`.

treat_numeric_as_ordinal Logical, default FALSE. Passed to `detect_variable_types()`.

auto_detect_ordinal_from_name Logical, default TRUE. Passed to `detect_variable_types()`.

auto_detect_ordinal_mode Character, "strict" or "contains", default "strict". Passed to `detect_variable_types()`.

ordinal_level_orders Named list, default `list()`. Passed to `detect_variable_types()`; required to treat a character/factor variable as ordinal based on its name alone.

reference_method Character, "mode", "first", or "alphabetical", default "mode". Passed to `detect_variable_types()` and honored during encoding.

impute_missing Logical, default TRUE. If FALSE, missing-value imputation is skipped entirely (rows/columns with NA are left as is, which may cause errors downstream).

impute_method Character, "mean", "median", or "mode", default "median". Passed as `numeric_method` to `handle_missing_values()`.

missing_threshold Numeric in [0, 1], default 0.20. Passed to `handle_missing_values()`.

ties_method Character, "first", "random", or "na", default "first". Passed to `handle_missing_values()`.

add_missing_indicators Logical, default FALSE. Passed as `add_indicators` to `handle_missing_values()`.

indicator_suffix Character, default "_missing". Passed to `handle_missing_values()`.

exclude_vars Character vector or NULL. Passed to `remove_non_predictors()`.

exclude_patterns Character vector or NULL. Passed to `remove_non_predictors()`.

min_freq_for_encoding Numeric in (0, 1) or NULL (auto-tuned). Passed to `encode_variables()`.

min_count_for_encoding Integer or NULL (auto-tuned). Passed to `encode_variables()`.

scale_vars Logical, default TRUE. Whether to scale continuous variables via `scale_variables()`.

scale_method Character, "standard" or "robust", default "standard". Passed to `scale_variables()`.

remove_nzv Logical, default TRUE. Whether to remove near-zero- variance variables via `remove_near_zero_variance()`.

nzv_freq_cut Numeric > 1 or NULL (auto-tuned). Passed to `remove_near_zero_variance()`.

nzv_unique_cut Numeric in (0, 100) or NULL (auto-tuned). Passed to `remove_near_zero_variance()`

remove_correlated Logical, default FALSE. Whether to remove highly correlated variables via `remove_highly_correlated()`.

corr_threshold Numeric in (0, 1), default 0.90. Passed to `remove_highly_correlated()`.

corr_keep_method Character, "higher_correlation" or "first", default "higher_correlation". Passed to `remove_highly_correlated()`.

verbose Logical, default TRUE.

Details

This function is a thin orchestrator: all actual logic lives in the dedicated modules (`remove_non_predictors`, `handle_missing_values`, `detect_variable_types`, `encode_variables`, `remove_constant_vars`, `scale_variables`, `remove_near_zero_variance`, `remove_highly_correlated`).

Value

A list with:

data Final preprocessed data frame.

var_types Variable type metadata from `detect_variable_types()`.

encoding_info Encoding metadata from `encode_variables()`.

scaling_params Scaling parameters from `scale_variables()`, or NULL if `scale_vars = FALSE`.

removed_vars Named list of character vectors: variables removed at each filtering stage (`non_predictors`, `constant`, `nzv`, `correlated`).

data_size One of "small", "medium", "large", based on input row count.

response_var The response variable name used throughout.

preprocessing_params The fully-resolved settings list actually used.

Examples

```
df <- data.frame(
  Yield = rnorm(30, 5000, 500),
  Nitrogen = rnorm(30, 120, 20),
  Region = sample(c("North", "South", "East"), 30, replace = TRUE)
)
prep <- prep_yield_gap(df, response_var = "Yield", verbose = FALSE)
dim(prep$data)

# Using the bundled real dataset:
data(wheat1)
prep2 <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
dim(prep2$data)
```

remove_constant_vars *Remove Constant Variables*

Description

Removes variables that have only one unique non-NA value (including all-NA columns). Works for ALL data types (numeric, factor, character).

Usage

```
remove_constant_vars(data, response_var = "Yield", verbose = TRUE)
```

Arguments

data	A data frame
response_var	Name of response variable (excluded from removal)
verbose	Logical, print progress (default: TRUE)

Value

A list with:

data	Filtered data frame
removed	Character vector of removed variables

Examples

```
df <- data.frame(Yield = rnorm(20, 5000, 500), AlwaysFive = 5)
result <- remove_constant_vars(df, response_var = "Yield", verbose = FALSE)
names(result$data)
```

remove_highly_correlated *Remove Highly Correlated Variables*

Description

Identifies and removes highly correlated numeric variables using an iterative greedy algorithm similar to caret::findCorrelation().

Usage

```
remove_highly_correlated(
  data,
  response_var = "Yield",
  threshold = 0.9,
  keep_method = "higher_correlation",
  cor_method = "pearson",
  cor_use = "pairwise.complete.obs",
  verbose = TRUE
)
```

Arguments

data	A data frame
response_var	Name of response variable (used to decide which variable to keep)
threshold	Numeric, correlation threshold (default: 0.90)
keep_method	Character, "higher_correlation" (keep variable with higher cor to response), or "first" (keep the first variable encountered by position)
cor_method	Character, correlation method: "pearson", "spearman", "kendall"
cor_use	Character, method for handling missing values (default: "pairwise.complete.obs")
verbose	Logical, print progress (default: TRUE)

Details

The algorithm: 1. Finds the pair with the highest absolute correlation above threshold 2. Removes one variable from the pair (based on keep_method) 3. Recalculates correlations and repeats until no pair exceeds threshold

Value

A list with:

data	Filtered data frame
removed	Character vector of removed variables

Examples

```
set.seed(1)
x <- rnorm(30, 100, 10)
df <- data.frame(Yield = rnorm(30, 5000, 500),
                 Nitrogen = x,
                 Nitrogen_copy = x + rnorm(30, 0, 0.01))
result <- remove_highly_correlated(df, response_var = "Yield", verbose = FALSE)
names(result$data)
```

`remove_near_zero_variance` *Remove Near Zero Variance Variables*

Description

Identifies and removes variables with near zero variance using the same logic as caret::nearZeroVar: - freqRatio: ratio of most common to second most common value - percentUnique: percentage of unique values

Usage

```
remove_near_zero_variance(
  data,
  response_var = "Yield",
  freq_cut = 19,
  unique_cut = 10,
  verbose = TRUE
)
```

Arguments

`data`	A data frame
`response_var`	Name of response variable (excluded from removal)
`freq_cut`	Numeric, frequency ratio cutoff (default: 19)
`unique_cut`	Numeric, percent unique cutoff (default: 10)
`verbose`	Logical, print progress (default: TRUE)

Details

A variable is considered near-zero variance if: freqRatio > freq_cut AND percentUnique < unique_cut

Value

A list with:

`data`	Filtered data frame
`removed`	Character vector of removed variables

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500),
                 AlmostConstant = c(rep(1, 28), 2, 3))
result <- remove_near_zero_variance(df, response_var = "Yield", verbose = FALSE)
names(result$data)
```

`remove_non_predictors` *Remove Non-Predictor Variables*

Description

Removes variables that should not be used as predictors, such as ID columns. Date and location variables are NOT removed by default.

Usage

```
remove_non_predictors(
  data,
  exclude_vars = NULL,
  exclude_patterns = NULL,
  verbose = TRUE
)
```

Arguments

`data`	A data frame
`exclude_vars`	Character vector of variable names to exclude
`exclude_patterns`	Character vector of regex patterns (default: ID-related)
`verbose`	Logical, print progress (default: TRUE)

Value

A list with:

`data`	Filtered data frame
`removed`	Character vector of removed variables

Examples

```
df <- data.frame(Field_ID = 1:10, Yield = rnorm(10, 5000, 500),
                 Nitrogen = rnorm(10, 120, 20))
result <- remove_non_predictors(df, verbose = FALSE)
names(result$data)
```

save_field_level_excel *Save Field-Level Yield Gap Analysis to Excel*

Description

Creates a detailed field-level Excel report showing per-field predictions, yield gaps, factor contributions, limiting factors, and rankings.

Usage

```
save_field_level_excel(
  results,
  df_final,
  output_path = file.path(tempdir(), "field_level_analysis.xlsx"),
  verbose = TRUE
)
```

Arguments

results	List. Output from yield_gap_analysis.
df_final	Data frame. The preprocessed dataset (before analysis).
output_path	Character. Full path for the output Excel file. Default: file.path(tempdir(), "field_level_analysis.xlsx").
verbose	Logical. Print a confirmation message with the saved file path and sheet names. Default: TRUE.

Details

Requires the **writexl** package. Please install it with: install.packages("writexl")

Value

Invisibly returns the output path.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
save_field_level_excel(results, prep$data,
                       output_path = file.path(tempdir(), "field_analysis.xlsx"))
```

`save_results_excel` *Save Yield Gap Analysis Results to Excel*

Description

Exports all analysis results (CPA table, predictions, variable statistics, relative importance, Pareto analysis, cross-validation, and model summary) to a multi-sheet Excel file.

Usage

```
save_results_excel(
  results,
  output_path = file.path(tempdir(), "yield_gap_analysis_full.xlsx"),
  verbose = TRUE
)
```

Arguments

`results`	List. Output from `yield_gap_analysis`.
`output_path`	Character. Full path for the output Excel file. Default: `file.path(tempdir(), "yield_gap_analysis_full.xlsx")`.
`verbose`	Logical. Print a confirmation message with the saved file path and sheet names. Default: TRUE.

Details

Requires the **writexl** package. Please install it with: `install.packages("writexl")`

Value

Invisibly returns the output path.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
save_results_excel(results,
                   output_path = file.path(tempdir(), "yield_gap_results.xlsx"))
```

scale_new_data *Scale New Data Using Existing Parameters*

Description

Applies the same scaling parameters to new data (e.g., test set).

Usage

```
scale_new_data(data, scaling_params, keep_original = FALSE, verbose = TRUE)
```

Arguments

data	A data frame with variables to scale
scaling_params	Scaling parameters from scale_variables
keep_original	Logical, keep original variables in the data frame. Default: FALSE
verbose	Logical, print progress messages. Default: TRUE

Value

Data frame with scaled variables

Examples

```
train <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
scaled <- scale_variables(train, exclude_vars = "Yield", verbose = FALSE)
new_obs <- data.frame(N_rate = c(90, 110, 130))
scale_new_data(new_obs, scaled$scaling_params, verbose = FALSE)
```

scale_variables *Scale Continuous Variables*

Description

Standardizes (or robustly scales) numeric variables in a data frame.

Usage

```
scale_variables(
  data,
  vars = NULL,
  exclude_vars = NULL,
  method = "standard",
  keep_original = FALSE,
  verbose = TRUE
)
```

Arguments

data	A data frame.
vars	Character vector of variable names to scale, or NULL to auto-detect all numeric columns (excluding exclude_vars). Default: NULL.
exclude_vars	Character vector of variable names to NEVER scale, even if they are numeric and would otherwise be auto-detected. This should always include the response variable - scaling the response changes its name (e.g. "Yield" -> "Yield_scaled"), which silently breaks any downstream code (model fitting, NZV/correlation filtering, reporting) that still refers to it by its original name. Default: NULL.
method	Character, "standard" (mean/sd) or "robust" (median/mad). Default: "standard".
keep_original	Logical, keep the original (unscaled) column alongside the new scaled column. Default: FALSE.
verbose	Logical, print progress messages. Default: TRUE.

Value

A list with:

data	Data frame with scaled variables
scaling_params	List of scaling parameters for back-transformation
scaled_vars	Character vector of scaled variable names
method	Scaling method used

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
result <- scale_variables(df, exclude_vars = "Yield", method = "standard", verbose = FALSE)
result$scaling_params
```

---

select_vars_aic *Variable Selection via AIC-Based Stepwise Regression*

Description

Selects variables by minimizing AIC (Akaike Information Criterion) using step, as an alternative to F-test-based selection (see select_vars_ftest).

Usage

```
select_vars_aic(data, response = "Yield", direction = "both", verbose = TRUE)
```

Arguments

data	A data frame.
response	Character. Name of the response variable. Default: "Yield".
direction	Character, one of "both", "forward", "backward". Passed to step. Default: "both".
verbose	Logical. Print progress and the AIC trace. Default: TRUE.

Details

AIC-based and F-test-based selection answer different questions and will often disagree on how many variables to keep: AIC-based stepwise selection optimizes overall model fit (penalized by model complexity) and has no fixed significance threshold, so it will often retain MORE variables than a strict F-test-based approach with typical F-to-Enter/F-to-Remove thresholds (e.g. around 4 and 3.9, as used by default in software such as SigmaPlot) - variables that improve AIC slightly may not individually clear a fixed F/p-value bar. Neither method is universally "correct"; they encode different criteria for what counts as a useful predictor.

Value

A list with:

final_model	The final fitted lm model, or NULL if selection failed or no predictors were available.
significant_vars	Character vector of selected variable names.
step_history	The step-by-step ANOVA/AIC trace from stats::step(), or NULL.
method	Always "aic".

Examples

```
df <- data.frame(Yield = rnorm(40, 5000, 500),
                 Nitrogen = rnorm(40, 120, 20),
                 Phosphorus = rnorm(40, 40, 10))
sel <- select_vars_aic(df, response = "Yield", verbose = FALSE)
sel$significant_vars
```

---

select_vars_ftest *Forward Stepwise Variable Selection Using F-test*

Description

Selects significant predictor variables using a forward stepwise algorithm based on partial F-tests. Supports both P-value threshold and F-to-Enter/F-to-Remove criteria (like SigmaPlot).

Usage

```
select_vars_ftest(
  data,
  response = "Yield",
  p_threshold = 0.05,
  use_bonferroni = TRUE,
  max_vars = NULL,
  f_to_enter = NULL,
  f_to_remove = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame containing predictor and response variables.
response	Character. Name of the response variable. Default: "Yield".
p_threshold	Numeric. Significance threshold (alpha). Default: 0.05.
use_bonferroni	Logical. Apply Bonferroni correction. Default: TRUE.
max_vars	Integer or NULL. Maximum variables to select. Default: NULL (no limit).
f_to_enter	Numeric or NULL. F-to-Enter threshold (like SigmaPlot). Default: NULL (uses p_threshold instead).
f_to_remove	Numeric or NULL. F-to-Remove threshold (like SigmaPlot). Default: NULL (uses p_threshold instead).
verbose	Logical. Print progress. Default: TRUE.

Value

A list with components:

final_model	The selected linear model (lm object).
significant_vars	Character vector of selected variable names.
step_history	Data frame recording each step.
method	Character, method used ("p_value" or "f_to_enter").

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
sel <- select_vars_ftest(prep$data, response = "Yield", verbose = FALSE)
sel$significant_vars
```

sugarcane1 *Sugarcane Yield Trial Data*

Description

Field-level sugarcane yield data from 371 fields, recording yield and a broad range of crop-cycle, fertilization, pest/disease, and remote-sensing/soil variables. No missing values. A larger, cleaner third example dataset (compared to `wheat2`), useful for illustrating the package on a different crop and a dataset with several genuinely ordinal and perfectly collinear (aliased) variables - see Examples.

Usage

```
sugarcane1
```

Format

A data frame with 371 rows and 30 variables:

field_id Character. Field identifier (non-predictor; removed automatically by `remove_non_predictors`).

cultivar Character. Sugarcane cultivar name (nominal).

crop_type Character. Crop cycle stage: `"PC"` (plant cane) followed by `"R1"` through `"R9"` (successive ratoon/regrowth cycles). This is a genuinely ORDINAL variable stored as text (PC < R1 < R2 < ... < R9) - see Examples for how to encode its order explicitly via `ordinal_level_orders`.

texture_class Character. Soil texture class (nominal, e.g. "Loam", "Silty clay").

drainage_issue Character. "Yes"/"No": whether the field has a drainage issue.

smut_incidence Character. "Yes"/"No": presence of sugarcane smut disease.

other_weeds_presence Character. "Yes"/"No": presence of weeds other than the main tracked species.

harvestable_area Numeric. Harvestable area (hectares).

harvest_days_from_September23_2023 Integer. Harvest date, as days after September 23, 2023.

harvest_duration Numeric. Duration of the harvest operation (days).

crop_cycle_days Integer. Total length of the crop cycle (days).

first_topdressing_days_from_February20_2023 Integer. First nitrogen topdressing date, as days after February 20, 2023.

second_topdressing_days_from_February20_2023 Integer. Second nitrogen topdressing date, as days after February 20, 2023.

third_topdressing_days_from_February20_2023 Integer. Third nitrogen topdressing date, as days after February 20, 2023.

starter_urea_kg_ha Numeric. Starter urea fertilizer rate (kg/ha).

first_topdressing_urea_kg_ha Numeric. First topdressing urea rate (kg/ha).

second_topdressing_urea_kg_ha Numeric. Second topdressing urea rate (kg/ha).

third_topdressing_urea_kg_ha Numeric. Third topdressing urea rate (kg/ha).

total_urea_kg_ha Numeric. Total urea applied (kg/ha). NOTE: this is the exact sum of `starter_urea_kg_ha`, `first_topdressing_urea_kg_ha`, `second_topdressing_urea_kg_ha`, and `third_topdressing_urea_kg_ha` - i.e. perfectly collinear (aliased) with them when all four are included in the same model. Both `select_vars_aic` and `select_vars_ftest` handle this gracefully (see Examples), but including this column alongside all four of its components in a manually-fit `lm()` will produce an `NA` coefficient for whichever is entered last.

leaf_N_at_mid_stem_elongation Numeric. Leaf nitrogen concentration at mid stem elongation (percent).

leaf_N_at_grand_growth Numeric. Leaf nitrogen concentration at grand growth stage (percent).

leaf_N_before_irrigation_cutoff Numeric. Leaf nitrogen concentration before irrigation cutoff (percent).

irrigation_cutoff_days_from_September23_2023 Integer. Irrigation cutoff date, as days after September 23, 2023.

sesamia_bored_internodes_percent Numeric. Percentage of internodes bored by *Sesamia* stem borer.

cogongrass_infested_m2 Integer. Area infested by cogongrass (square meters).

irrigation_interval Numeric. Average interval between irrigation events (days).

max_LAI Numeric. Maximum Leaf Area Index observed during the season.

MCARI Numeric. Modified Chlorophyll Absorption in Reflectance Index (a remote-sensing vegetation index).

soil_EC Numeric. Soil electrical conductivity.

Yield Numeric. Cane yield. Response variable.

Source

Field trial data used during package development and testing.

Examples

```
data(sugarcane1)

# crop_type is genuinely ordinal (PC < R1 < ... < R9); without an
# explicit order it would conservatively be treated as nominal.

prep <- prep_yield_gap(
  sugarcane1,
  response_var = "Yield",
  ordinal_level_orders = list(
    crop_type = c("PC", paste0("R", 1:9))
  ),
  verbose = FALSE
)
dim(prep$data)

# total_urea_kg_ha is a perfect linear combination of the four
# individual urea application columns; both selection methods drop it
```

```
# automatically (with a warning) rather than erroring.
result <- yield_gap_analysis(prep$data, response = "Yield",
                             selection_method = "aic", verbose = FALSE)
result$yield_gap
```

`summary_scaling` *Get Scaling Summary*

Description

Provides a human-readable summary of scaling parameters.

Usage

```
summary_scaling(scaling_params, verbose = TRUE)
```

Arguments

`scaling_params`	Scaling parameters from `scale_variables`
`verbose`	Logical, print summary. Default: TRUE

Value

A data frame with scaling parameters (invisibly if verbose=TRUE)

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), N_rate = rnorm(30, 100, 20))
scaled <- scale_variables(df, exclude_vars = "Yield", verbose = FALSE)
summary_scaling(scaled$scaling_params, verbose = FALSE)
```

`trim_whitespace` *Trim leading/trailing whitespace from character columns*

Description

Real-world CSV/Excel exports very commonly introduce accidental leading or trailing whitespace on text values (from copy-paste, cell formatting, trailing tabs, etc.). Left untrimmed, this silently splits what should be a single category into two distinct levels - e.g. "Sirvan" (22 rows) and "Sirvan " (4 rows) being treated as two different cultivars - with no error or warning anywhere downstream, since both are perfectly valid (if different) character strings from R's point of view. This function is called automatically and by default at the start of `prep_yield_gap`.

Usage

```
trim_whitespace(data, verbose = TRUE)
```

Arguments

data	A data frame
verbose	Logical. Report which columns/values were affected. Default: TRUE.

Value

The data frame with whitespace trimmed from all character columns. Factor and non-character columns are left unchanged.

Examples

```
df <- data.frame(Cultivar = c("Sirvan", "Sirvan ", " Sirvan"),
                 Yield = c(5000, 4900, 5100),
                 stringsAsFactors = FALSE)
trimmed <- trim_whitespace(df, verbose = FALSE)
length(unique(trimmed$Cultivar))
```

---

validate_model_with_cv

k-Fold Cross-Validation for Yield Gap Model

Description

Evaluates model predictive performance using k-fold cross-validation. A NEW model is refit on each training fold (using `sig_vars` and `response`); the `model` argument is used only to validate that `sig_vars`/`response` are consistent with an already- fitted whole-data model, not refit directly, since cross-validation by definition requires refitting on each fold's training subset.

Usage

```
validate_model_with_cv(
  model,
  data,
  sig_vars,
  response = "Yield",
  k_folds = 5,
  seed = NULL
)
```

Arguments

model	A fitted lm model, used only for consistency validation (its response variable must match response). Pass NULL to skip this check.
data	A data frame.
sig_vars	Character vector of selected predictor variable names.
response	Character. Name of the response variable. Default: "Yield".
k_folds	Integer. Number of folds. Default: 5.
seed	Integer or NULL. If supplied, used to make the fold assignment reproducible; the caller's global random state is saved before, and restored after, this function runs, so calling this function never permanently alters the user's RNG stream. Default: NULL (fold assignment uses whatever RNG state is currently active, exactly like any other call to sample() would).

Value

A list with components: by_fold and summary.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
sel <- select_vars_ftest(prep$data, response = "Yield", verbose = FALSE)
cv <- validate_model_with_cv(sel$final_model, prep$data, sel$significant_vars,
                             response = "Yield", k_folds = 5, seed = 42)
cv$summary
```

---

validate_preprocessing

Validate Preprocessing Results

Description

Performs comprehensive validation of preprocessing output.

Usage

```
validate_preprocessing(preprocessed_result, verbose = TRUE)
```

Arguments

preprocessed_result	Output from prep_yield_gap()
verbose	Logical, print validation results

Value

List with validation results, always including: valid, issues, warnings, n_errors, n_warnings, summary, response_var, has_encoding, has_scaling.

Examples

```
df <- data.frame(Yield = rnorm(30, 5000, 500), Nitrogen = rnorm(30, 120, 20))
prep <- prep_yield_gap(df, response_var = "Yield", verbose = FALSE)
val <- validate_preprocessing(prep, verbose = FALSE)
val$valid
```

---

wheat1 *Wheat Yield Trial Data (Clean)*

Description

Field-level wheat yield data from 200 fields, recording yield and agronomic management/environmental variables. Used throughout the package documentation and vignette as the primary worked example: a relatively large, clean dataset (no missing values) suitable for demonstrating the full `prep_yield_gap` / `yield_gap_analysis` workflow end to end.

Usage

```
wheat1
```

Format

A data frame with 200 rows and 13 variables:

Yield Numeric. Grain yield (kg/ha). Response variable.

Irrigation Numeric. Irrigation water applied.

Nitrogen Numeric. Nitrogen fertilizer applied (kg/ha).

Phosphorus Numeric. Phosphorus fertilizer applied (kg/ha).

Potassium Numeric. Potassium fertilizer applied (kg/ha).

Soil_pH Numeric. Soil pH.

Soil_Organic_Matter Numeric. Soil organic matter (percent).

Plant_density Numeric. Plant density (plants per unit area).

Weed_Infestation Numeric. Weed infestation score/count.

Drought Integer (0/1). Whether the field experienced drought stress.

Pest Integer (0/1). Whether the field experienced pest pressure.

Cultivar Character. Wheat cultivar name (nominal, several levels).

Sowing_Date Numeric. Sowing date, encoded as day offset.

Source

Field trial data used during package development and testing.

Examples

```
data(wheat1)
str(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
dim(prep$data)
```

---

wheat2 *Wheat Yield Trial Data (Small, High-Dimensional, Real-World Messy)*

Description

Field-level wheat yield data from 40 fields with many recorded management, socioeconomic, and pest/disease variables (29 columns). Deliberately kept in its original, unedited form (including a non-predictor ID column and a stray leading/trailing whitespace inconsistency in `Cultivar`, where "Sirvan" appears both as `"Sirvan"` and `"Sirvan "`) to serve as a realistic example of common real-world data-quality issues and of a small, high predictor-to-sample-ratio dataset - exactly the profile that exercises `trim_whitespace`, near-zero-variance filtering, and the auto-tuning logic in `prep_yield_gap`.

Usage

```
wheat2
```

Format

A data frame with 40 rows and 29 variables:

Field_id Integer. Field identifier (non-predictor; removed automatically by `remove_non_predictors`).

Farmers_age Integer. Age of the farmer (years).

Litracy_Level Character. Farmer's education level (a true ordinal variable stored as text - see Examples for how to encode its order explicitly via `ordinal_level_orders`).

Experience_Years Integer. Farmer's years of experience.

Cultivated_Area Numeric. Cultivated area (hectares).

Rotation Character. Crop rotation sequence preceding wheat.

Crop_Residue_Mgmt Character. How previous crop residue was managed (e.g. "Burned", "Incorporated", "Retained").

Pre_Plow_Irrigation Character. "Yes"/"No": whether a pre-plowing irrigation was applied.

Disk_Passes Integer. Number of disk harrow passes.

Land_Leveled Character. "Yes"/"No": whether the land was laser-leveled.

Seeder_Type Character. Type of seeder used (e.g. "Seed Drill", "Centrifugal").

Planting_Day_From_Sep23 Integer. Planting date, as days after September 23.

Seed_Rate Integer. Seeding rate.

Cultivar Character. Wheat cultivar name. Contains a real leading/trailing whitespace inconsistency ("Sirvan" vs. "Sirvan ") used to illustrate `trim_whitespace`.

TSP_Rate Integer. Triple superphosphate fertilizer rate.

Good_Drainage Character. "Yes"/"No": whether the field has good drainage.

Irrigation_Count Integer. Number of irrigation events.

Soil_texture Character. Soil texture class (e.g. "Clay", "Clay-sand").

Weed_species Character. Multi-value field: weed species present, concatenated with "-" separators - treated as a high-cardinality nominal variable; see `group_high_cardinality`.

Spraying_DAP Integer. Herbicide spraying date, in days after planting.

Herbicide_Type Character. Multi-value field: herbicide active ingredient(s) applied, concatenated with "_" separators.

Urea_as_adjuvant Character. "Yes"/"No": whether urea was used as a spray adjuvant.

Nitrogen_topdressing_Number Integer. Number of nitrogen topdressing applications.

Topdressing_N Integer. Total topdressing nitrogen applied.

Puccinia_disease Character. "Yes"/"No": presence of *Puccinia* (rust) disease.

Days_To_Harvest Integer. Days from planting to harvest.

Selection_Pressure Character. "Yes"/"No": indicator of herbicide selection pressure.

Herbicide_Resistance_Suspected Character. "Yes"/"No": whether herbicide resistance was suspected in the field.

Yield Integer. Grain yield (kg/ha). Response variable.

Source

Field trial data used during package development and testing.

Examples

```
data(wheat2)

# Demonstrates automatic whitespace trimming: "Sirvan " and "Sirvan"
# are correctly merged into a single category.
length(unique(wheat2$Cultivar))
trimmed <- trim_whitespace(wheat2, verbose = FALSE)
length(unique(trimmed$Cultivar))


# Full pipeline, including an explicit ordinal level order for
# Litracy_Level since it cannot be safely inferred automatically.
prep <- prep_yield_gap(
  wheat2,
  response_var = "Yield",
  ordinal_level_orders = list(
    Litracy_Level = c("Elementary School", "Middle School", "Diploma",
```

```
                        "Bachelor's Degree", "Master's Degree")
  ),
  verbose = FALSE
)
dim(prep$data)
```

`yield_gap_analysis` *Complete Yield Gap Analysis Using CPA Method*

Description

Runs the full Comparative Performance Analysis (CPA) pipeline: variable selection, optimal value calculation, min-max effects, yield gap decomposition, cross-validation, relative importance, and Pareto analysis.

Usage

```
yield_gap_analysis(
  data,
  response = "Yield",
  p_threshold = NULL,
  top_percentile = NULL,
  max_vars = NULL,
  calculate_ri = TRUE,
  ri_method = c("lmg", "std_beta"),
  selection_method = c("ftest", "aic"),
  aic_direction = c("both", "forward", "backward"),
  perform_cv = TRUE,
  cv_seed = 42,
  use_bonferroni = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A preprocessed data frame (numeric columns only; see `prep_yield_gap`).
`response`	Character. Name of the response (yield) variable. Default: "Yield".
`p_threshold`	Numeric. Significance level for variable selection. Auto-tuned if NULL.
`top_percentile`	Numeric. Top percentile threshold for defining optimal yield. Auto-tuned if NULL.
`max_vars`	Integer. Maximum variables to select. Auto-tuned if NULL.
`calculate_ri`	Logical. Calculate relative importance of variables. Default: TRUE.
`ri_method`	Character. Method for relative importance: "lmg" or "std_beta". Default: "lmg".
`selection_method`	Character, "ftest" or "aic". Variable selection method.

"ftest" Stepwise selection based on a fixed F-to-Enter / F-to-Remove (or p-value) threshold, similar to software such as SigmaPlot. See `select_vars_ftest` for the `p_threshold` / `use_bonferroni` controls. Tends to select FEWER variables, since each one must individually clear a fixed significance bar.

"aic" Stepwise selection minimizing AIC via `select_vars_aic`. Tends to select MORE variables than "ftest", since it optimizes overall predictive fit rather than requiring each variable to be individually significant at a fixed threshold.

Neither method is universally "correct" - they encode different criteria for what counts as a useful predictor, and it is normal for them to select a different number and/or set of variables on the same data. Default: "ftest".

`aic_direction` Character, "both", "forward", or "backward". Only used when `selection_method = "aic"`; passed to `step`. Default: "both".

`perform_cv` Logical. Perform cross-validation. Auto-disabled for small data. Default: TRUE.

`cv_seed` Integer or NULL. Random seed used only for cross-validation fold assignment, for reproducibility. The caller's global random state is saved and restored afterward, so this has no persistent side effect on the user's session. Set to NULL for non-reproducible fold assignment. Default: 42.

`use_bonferroni` Logical. Apply Bonferroni correction. Default: FALSE.

`verbose` Logical. Print step-by-step progress. Default: TRUE.

Value

A list with the following components:

`model` The final fitted `lm` model.

`significant_vars` Character vector of selected variables.

`var_table` Data frame of variable statistics and optimal values.

`minmax_results` Data frame of min-max effects.

`cpa_table` Full CPA table with gap contributions and shares.

`yield_mean` Predicted mean yield.

`yield_opt` Predicted optimal yield.

`yield_gap` Total yield gap.

`predictions` Data frame: Observed, Predicted, Yield_Gap_Obs per row.

`metrics` List: R2, RMSE, MAE for the fitted model.

`cross_validation` Cross-validation results (if performed).

`relative_importance` Relative importance results.

`pareto_analysis` Pareto analysis results.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
results$yield_gap
results$cpa_table
```

`yield_gap_components` *Calculate Yield Gap Components from CPA Table*

Description

Computes contributions of each variable to the yield gap.

Usage

```
yield_gap_components(cpa_table)
```

Arguments

`cpa_table`	A data frame with columns: Variable, Beta, Mean, Opt.

Value

A list with components: cpa_table, yield_mean, yield_opt, yield_gap, gap_positive, gap_negative.

Examples

```
data(wheat1)
prep <- prep_yield_gap(wheat1, response_var = "Yield", verbose = FALSE)
results <- yield_gap_analysis(prep$data, response = "Yield", verbose = FALSE)
cpa_input <- results$cpa_table[, c("Variable", "Beta", "Mean", "Opt")]
yg <- yield_gap_components(cpa_input)
yg$yield_gap
```

Index