Package: PsyML (via r-universe)

November 12, 2026

Type Package

Title Machine Learning Workflows for Supervised, Unsupervised, and Causal Analysis

Version 2.4.2

Date 2026-10-25

Description Provides single-function pipelines for applied machine learning with tabular data, as commonly found in psychological, behavioral, and health research. The supervised pipeline, supervised_ml(), performs exploratory checks, preprocessing, training of several models with cross-validation, hyperparameter tuning, test-set evaluation, threshold optimization, calibration, fairness analysis, decision curve analysis, and model interpretation with permutation importance and Shapley values. The unsupervised pipeline, unsupervised_ml(), covers k-means, hierarchical, DBSCAN, Gaussian mixture, and latent class clustering with selection of the number of clusters, dimensionality reduction (PCA, t-SNE, UMAP), and anomaly detection (isolation forest, local outlier factor, Mahalanobis distance). The causal pipeline, causal_ml(), estimates treatment effects with propensity scores, meta-learners, double machine learning, causal forests, and targeted maximum likelihood estimation. Also includes learning-curve based sample size estimation, block and diversity-based data splitting, and export of results to text and JSON.

License MIT + file LICENSE

URL https://github.com/jventural/PsyML

BugReports https://github.com/jventural/PsyML/issues

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dials, dplyr, ggplot2, grDevices, graphics, hardhat, jsonlite, nnet, parallel, parsnip, pROC, ranger, recipes, rlang, rsample, stats, tibble, tidyr, tidyselect, tune, utils, workflows, xgboost, yardstick

Suggests Boruta, bslib, cluster, cobalt, commonmark, dbscan, doParallel, DoubleML, DT, e1071, finetune, flextable, foreach, ggdendro, glmnet, grf, htmltools, httr, httr2, isotree, kernelshap, kernlab, knitr, MatchIt, mclust, mlr3, mlr3learners, officer, patchwork, poLCA, readxl, reticulate, rmarkdown, rpart, Rtsne, shapviz, shiny, shinydashboard, SuperLearner, testthat (>= 3.0.0), themis, tmle, umap, VIM

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Jose Ventura-Leon [aut, cre] (ORCID: <https://orcid.org/0000-0003-2996-4244>)

Maintainer Jose Ventura-Leon <jventuraleon@gmail.com>

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

Date/Publication 2026-11-12 15:00:02 UTC

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

RemoteRef HEAD

RemoteSha 1ec404577e481f4f2f95e817874a7263b027771f

Contents

advanced_residual_analysis 4
analyze_dca 6
analyze_fairness 7
analyze_interactions_splines 9
apply_custom_features 10
apply_threshold 11
auto_feature_engineering 12
block_split 14
calculate_shap 16
calibrate_probabilities 17
causal_ml 19
cluster_profile 22
compare_models 23
correct_fairness_threshold 24
cv_feature_selection 26
define_models 27
detect_data_leakage 28
diversity_split 29
eda_basic 32

eda_categorical . . . 33
eda_categorical_basic . . . 33
eda_correlation . . . 34
eda_correlation_basic . . . 35
eda_leakage . . . 36
eda_missing . . . 37
eda_normality . . . 37
eda_numeric . . . 38
eda_numeric_basic . . . 39
eda_outliers . . . 40
eda_structure . . . 41
eda_summary . . . 41
eda_target_classification . . . 42
eda_target_regression . . . 43
eda_vif . . . 44
estimate_n . . . 44
estimate_n_plateau . . . 45
eval_calibration . . . 47
eval_confusion . . . 48
eval_metrics . . . 49
eval_pred_vs_obs . . . 50
eval_predictions . . . 51
eval_residuals . . . 53
eval_roc . . . 54
evaluate_model . . . 55
explain_with_ai . . . 56
export_verbose_json . . . 58
export_verbose_txt . . . 59
fit_models_cv . . . 60
generate_all_plots . . . 61
generate_report_from_json . . . 63
generate_report_with_ai . . . 64
get_figures_catalog . . . 65
get_importance . . . 66
interpret_model . . . 68
launch_report_generator . . . 69
launch_supervisedml_app . . . 70
ml_sample_size . . . 71
optimize_threshold . . . 75
plot.causalml . . . 76
plot.supervisedml . . . 77
plot.unsupervisedml . . . 78
plot_all . . . 79
plot_all.ml_sample_size . . . 80
plot_calibration_curve . . . 81
plot_distribution . . . 82
plot_importance . . . 83
plot_learning_curve . . . 84

plot_metrics_profile . . . 85
plot_residual_diagnostics . . . 87
plot_shap_bar . . . 88
plot_shap_dependence . . . 89
plot_shap_summary . . . 90
plot_threshold_optimization . . . 92
plot_tuning . . . 93
predict.causalml . . . 94
predict.supervisedml . . . 95
predict.unsupervisedml . . . 96
prep_feature_selection . . . 97
prep_recipe . . . 98
prep_split . . . 100
prep_target . . . 101
preprocess_data . . . 102
print.block_split . . . 105
print.causalml . . . 105
print.diversity_split . . . 106
print.ml_sample_size . . . 107
print.supervisedml . . . 107
print.unsupervisedml . . . 108
print_figures_catalog . . . 109
resample_fairness . . . 110
run_nested_cv . . . 112
save_all_plots . . . 113
save_unsupervised_plots . . . 115
setup_cv . . . 116
summary.causalml . . . 117
summary.supervisedml . . . 118
summary.unsupervisedml . . . 119
supervised_ml . . . 120
supervised_ml_capture . . . 125
supervised_ml_export . . . 126
train_models . . . 127
tune_best_model . . . 128
unsupervised_ml . . . 130

Index **134**

`advanced_residual_analysis`

Advanced Residual Analysis

Description

Runs a residual analysis for regression predictions: descriptive statistics, Shapiro-Wilk normality test (on at most 5000 residuals), a simplified heteroscedasticity check (correlation between absolute residuals and predictions), residual outliers (3 x IQR rule), influential points (standardized residuals with absolute value > 3) and an approximate Durbin-Watson statistic.

Usage

```
advanced_residual_analysis(predictions, target, verbose = TRUE)
```

Arguments

predictions	Data frame with the observed target and a .pred column with the predictions (e.g. the predictions element of a regression supervised_ml() result).
target	Name of the target variable.
verbose	Logical. Print results (default: TRUE).

Value

An object of class c("supervisedml_residuals", "list") with elements basic_stats, normality_test (htest), homoscedasticity (correlation, p_value, heteroscedastic), outliers (n_outliers, pct_outliers, indices), influential_points (n_influential, indices), autocorrelation (durbin_watson, interpretation), residuals, predicted and observed.

See Also

Other supervised learning: analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
obs <- rnorm(150, 10, 2)
preds <- data.frame(y = obs, .pred = obs + rnorm(150))
ra <- advanced_residual_analysis(preds, target = "y", verbose = FALSE)
ra$autocorrelation
```

`analyze_dca` *Clinical Utility Analysis (Decision Curve Analysis)*

Description

Computes the net benefit of a binary classification model across probability thresholds and compares it with the "treat all" and "treat none" strategies, following Vickers and Elkin (2006).

Usage

```
analyze_dca(
  predictions,
  target,
  thresholds = seq(0.01, 0.99, by = 0.01),
  verbose = TRUE
)
```

Arguments

`predictions`	Data frame with one row per case containing the observed outcome (`target`, a factor with 2 levels) and the predicted probability of the positive class (column `.pred_<positive level>`).
`target`	Name of the outcome variable.
`thresholds`	Numeric vector of probability thresholds to evaluate (default `seq(0.01, 0.99, by = 0.01)`).
`verbose`	Logical (default TRUE). Print the results to the console.

Value

A list with:

- dca_data: Data frame with one row per threshold and columns `threshold`, `nb_model` (net benefit of the model), `nb_all` (treat all), `nb_none` (treat none, always 0), `tp_rate` and `fp_rate` (true and false positives divided by n), `nnt` (number needed to treat, `1 / nb_model`, Inf when the net benefit is not positive) and `model_useful` (TRUE when the model beats both reference strategies).
- useful_range: Lowest and highest threshold at which the model is useful (NA, NA if never).
- max_net_benefit: Maximum net benefit of the model.
- optimal_threshold_dca: Threshold with the maximum net benefit.
- prevalence: Observed prevalence of the positive class.
- n: Number of cases.

References

Vickers, A. J., & Elkin, E. B. (2006). Decision curve analysis: A novel method for evaluating prediction models. Medical Decision Making, 26(6), 565-574.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 300
prob <- plogis(rnorm(n, sd = 1.5))
preds <- data.frame(
  outcome = factor(ifelse(runif(n) < prob, "yes", "no"), levels = c("no", "yes")),
  .pred_yes = prob
)
dca <- analyze_dca(preds, target = "outcome")
dca$useful_range
head(dca$dca_data)
```

---

analyze_fairness *Fairness Analysis of a Classification Model*

Description

Evaluates the fairness of a binary classification model by comparing performance metrics across the subgroups of a protected variable. Computes demographic parity, equalized odds, predictive parity, calibration by group and the disparate impact ratio.

Usage

```
analyze_fairness(
  predictions,
  target,
  protected_var,
  fairness_threshold = 0.8,
  verbose = TRUE
)
```

Arguments

predictions	Data frame with one row per case containing the observed outcome (target, a factor with 2 levels), the predicted class (.pred_class), the protected variable and, optionally, the probability of the positive class (.pred_<positive level>).
target	Name of the outcome variable.
protected_var	Name of the protected variable (e.g., "sex", "race"). It must have at least 2 groups.
fairness_threshold	Threshold for the four-fifths rule of disparate impact (default 0.8).
verbose	Logical (default TRUE). Print the results to the console.

Value

A list with:

- group_metrics: Data frame with one row per group and columns group, n, prevalence, positive_rate, tpr, fpr, ppv, npv, accuracy, mean_prob and mean_prob_pos (mean predicted probability among observed positives).
- demographic_parity_diff: Largest difference in positive prediction rates between groups.
- equalized_odds_tpr_diff, equalized_odds_fpr_diff: Largest differences in true and false positive rates.
- predictive_parity_diff: Largest difference in PPV.
- calibration_diff: Largest difference in mean_prob_pos (NA without probabilities).
- disparate_impact_ratio: Lowest divided by highest positive prediction rate.
- fairness_threshold: Threshold used.
- passes_four_fifths: Logical, TRUE if the ratio is at least fairness_threshold.
- protected_var: Name of the protected variable.
- is_fair: Logical, TRUE if the demographic parity and TPR differences are below 0.1 and the four-fifths rule is met.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 300
sex <- sample(c("female", "male"), n, replace = TRUE)
prob <- plogis(rnorm(n) + ifelse(sex == "male", 0.5, 0))
preds <- data.frame(
  outcome = factor(ifelse(runif(n) < prob, "yes", "no"), levels = c("no", "yes")),
  .pred_yes = prob,
  sex = sex
)
preds$.pred_class <- factor(ifelse(prob >= 0.5, "yes", "no"), levels = c("no", "yes"))

fair <- analyze_fairness(preds, target = "outcome", protected_var = "sex")
fair$group_metrics
fair$disparate_impact_ratio
```

---

analyze_interactions_splines

Analyze Interactions and Nonlinear Effects

Description

Screens for potentially important interactions among the top predictors (categorical x numeric and categorical x categorical pairs) and for nonlinear effects of numeric predictors (comparison of a linear and a quadratic fit with an F test), and suggests interaction terms or splines to improve linear models.

Usage

```
analyze_interactions_splines(
  data,
  target,
  task,
  importance = NULL,
  top_n = 5,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the data.
`target`	Name of the target variable.
`task`	Type of task: "classification" or "regression".
`importance`	Optional data frame of variable importance with a `Variable` column (e.g. from `get_importance()`); its first `top_n` variables are analyzed. If NULL, the first `top_n` predictors are used.

top_n	Number of top variables to analyze (default: 5).
verbose	Logical. Print results and suggestions (default: TRUE).

Value

A list with interactions (named list; each element has var1, var2, type and strength), nonlinear (named list; each element has variable, p_value of the quadratic term, r2_gain in percentage points and a recommendation) and summary (list with n_interactions, n_nonlinear, top_interaction and top_nonlinear).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = runif(n, -2, 2), x2 = rnorm(n),
                 g = factor(sample(c("a", "b"), n, TRUE)))
df$y <- df$x1^2 + df$x2 + rnorm(n)
res <- analyze_interactions_splines(df, target = "y", task = "regression",
                                    verbose = FALSE)
res$summary
```

---

apply_custom_features *Domain Feature Engineering*

Description

Lets the user create new variables from R expressions. Each expression is evaluated using the columns of the data frame. Expressions that fail, or that do not return one value per row, are skipped and reported in failed.

Usage

```
apply_custom_features(data, custom_features, verbose = TRUE)
```

Arguments

data	Data frame with the data.
custom_features	Named list: each name is the new variable and each value an R expression given as a string (e.g. "SibSp + Parch").
verbose	Logical. Print progress (default: TRUE).

Value

A list with data (the data frame with the new variables), created (names of the variables created) and failed (names of the expressions that could not be evaluated).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
res <- apply_custom_features(mtcars,
  custom_features = list(power_weight = "hp / wt",
                         heavy = "ifelse(wt > 3, 1, 0)"),
  verbose = FALSE)
head(res$data[, c("power_weight", "heavy")])
res$created
```

---

apply_threshold *Apply a Classification Threshold*

Description

Reclassifies binary predictions with a given probability threshold (for instance, one found with optimize_threshold()).

Usage

```
apply_threshold(predictions, target, threshold)
```

Arguments

predictions	Data frame of predictions with the observed target (a factor with two levels) and a probability column named .pred_<positive class>.
target	Name of the target variable.
threshold	Probability threshold between 0 and 1; observations with probability >= threshold are assigned to the positive class.

Value

The predictions data frame with an added factor column .pred_class_optimized.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
x <- rnorm(100)
y <- factor(ifelse(x + rnorm(100) > 0.8, "yes", "no"), levels = c("no", "yes"))
preds <- data.frame(y = y, .pred_yes = plogis(1.5 * x - 1))
out <- apply_threshold(preds, target = "y", threshold = 0.3)
table(out$.pred_class_optimized)
```

---

auto_feature_engineering

Automatic Feature Engineering

Description

Applies generic transformations that may improve model performance without requiring domain knowledge: (1) suspicious zeros in numeric variables whose non-zero minimum is far from zero are converted to NA; (2) strictly positive variables with skewness > 2 get an additional log1p() version (suffix _log); (3) variables with more than 5% (and less than 90%) missing values get a missing indicator (suffix _missing); (4) pairwise products (suffix _x_) are created among the three numeric variables most correlated with the target.

Usage

```
auto_feature_engineering(data, target, verbose = TRUE)
```

Arguments

data	Data frame with the data.
target	Name of the target variable.
verbose	Logical. Print progress (default: TRUE).

Value

A list with data (the modified data frame), new_features (names of the created variables), transformations (list with elements zeros_to_na, log_transform, missing_indicators and/or interactions listing the variables involved), n_original (number of original predictors) and n_new (number of new variables).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 100
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rexp(n)^3)
df$x2[sample(n, 10)] <- NA
df$y <- df$x1 + rnorm(n)
fe <- auto_feature_engineering(df, target = "y", verbose = FALSE)
fe$new_features
```

`block_split` *Block-Based Train/Test Split for Enhanced Generalization*

Description

Splits data into train and test sets by sampling from sequential blocks, ensuring the test set represents different segments of the data distribution. This approach helps evaluate model generalization across different "contexts" or time periods, reducing the risk of overfitting to a specific data segment.

Usage

```
block_split(
  data,
  order_col = NULL,
  num_blocks = 5,
  test_prop = 0.2,
  test_n = NULL,
  stratify = NULL,
  balance_blocks = TRUE,
  seed = NULL
)
```

Arguments

`data`	Data frame to split.
`order_col`	Column name (character) or index used to order the data (e.g., date, response order, ID). If NULL, uses row order.
`num_blocks`	Number of sequential blocks to divide the data into (default 5).
`test_prop`	Proportion of data to use for test set (default 0.20). Alternatively, use `test_n` to specify exact number.
`test_n`	Exact number of observations for test set. Overrides `test_prop`.
`stratify`	Column name for stratified sampling within blocks (e.g., outcome variable). Ensures proportional class representation in both train and test sets.
`balance_blocks`	Logical. If TRUE, samples equal amounts from each block. If FALSE, samples proportionally to block size (default TRUE).
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.

Details

The function addresses the concern that traditional random splits may not adequately test generalization, since train and test come from the same homogeneous distribution. By sampling from different sequential blocks, the test set captures variability across different "periods" or segments.

When to use:

- Data collected over time (surveys, longitudinal studies)
- Data with natural ordering (response sequence, ID order)
- When concerned about temporal or contextual generalization

Stratification: When `stratify` is specified, the function ensures that within each block, samples are drawn proportionally from each class. This prevents class imbalance issues in train or test sets.

Value

A list of class `c("block_split", "list")` containing:

- train: Training data frame (rows of `data` in their original order).
- test: Test data frame (rows of `data` in their original order).
- train_idx: Sorted row indices of the training observations in `data`.
- test_idx: Sorted row indices of the test observations in `data`.
- block_info: Data frame with one row per block and columns `block`, `start_row`, `end_row` (positions after ordering), `block_size` and `test_sampled` (test cases drawn from the block).
- settings: List with the configuration used and the realized sizes (`n_total`, `n_train`, `n_test`, `num_blocks`, `test_prop_actual`, `order_col`, `stratify`, `balance_blocks`, `seed`).

References

Ventura-Leon, J. (2026). Block-based sampling for machine learning validation.

See Also

Other study design: `diversity_split()`, `estimate_n()`, `estimate_n_plateau()`, `ml_sample_size()`, `plot_all.ml_sample_size()`, `plot_distribution()`, `plot_learning_curve()`

Examples

```
# Simulated survey data collected over 100 days
set.seed(1)
survey_data <- data.frame(
  response_date = as.Date("2025-01-01") + sample(0:99, 150, replace = TRUE),
  x1 = rnorm(150),
  outcome = factor(sample(c("yes", "no"), 150, replace = TRUE))
)

# Basic usage with row order
result <- block_split(survey_data, num_blocks = 5, test_prop = 0.20, seed = 42)
nrow(result$train)
nrow(result$test)

# With date ordering and stratification
result <- block_split(
  data = survey_data,
  order_col = "response_date",
  num_blocks = 5,
```

```
  test_prop = 0.20,
  stratify = "outcome",
  seed = 42
)
result

# View block information
result$block_info
```

`calculate_shap` *SHAP Values of a Fitted Workflow*

Description

Computes Kernel SHAP values with `kernelshap::kernelshap()` for a sample of test observations, using a sample of the training data as background. For binary classification the explained output is the probability of the second class level; for multiclass outcomes the mean absolute SHAP value across classes is returned. Predictors removed by the preprocessing recipe are excluded from the output.

Usage

```
calculate_shap(
  final_fit,
  train_data,
  test_data,
  target,
  n_samples = 100,
  bg_n = 50,
  verbose = TRUE
)
```

Arguments

`final_fit`	A fitted `workflow`.
`train_data`	Training data frame (background data).
`test_data`	Test data frame (observations to explain).
`target`	Name of the outcome variable.
`n_samples`	Number of test observations to explain. In classification the sample is stratified by class.
`bg_n`	Number of background observations sampled from `train_data`.
`verbose`	Logical. If `TRUE`, progress is printed to the console.

Value

NULL if the package **kernelshap** is not installed or the computation fails; otherwise a list with elements shap_values (data frame of SHAP values, one column per predictor), importance (data frame with columns Variable and Mean_Abs_SHAP), test_sample (explained observations), removed_vars (predictors dropped by the recipe), collinear_vars (predictors with VIF > 5), and sample_design.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
if (requireNamespace("kernelshap", quietly = TRUE)) {
  set.seed(1)
  df <- data.frame(x1 = rnorm(120), x2 = rnorm(120), x3 = rnorm(120))
  df$y <- df$x1 - df$x2 + rnorm(120)
  wf <- workflows::workflow() |>
    workflows::add_formula(y ~ .) |>
    workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
  fit <- parsnip::fit(wf, data = df[1:100, ])
  sh <- calculate_shap(fit, df[1:100, ], df[101:120, ], "y",
                       n_samples = 20, bg_n = 30, verbose = FALSE)
  sh$importance
}
```

---

calibrate_probabilities

Calibrate Predicted Probabilities

Description

Calibrates the predicted probabilities of a binary classifier using Platt scaling (logistic regression of the outcome on the probabilities) or isotonic regression, and reports the Brier score and the expected calibration error (ECE, 10 bins) before and after calibration. The calibration model is fitted and evaluated on the same predictions, so the reported improvement is optimistic.

Usage

```
calibrate_probabilities(predictions, target, method = "platt", verbose = TRUE)
```

Arguments

predictions	Data frame of predictions with the observed target (a factor with two levels) and a probability column named .pred_<positive class>.
target	Name of the target variable.
method	Calibration method: "platt" (default, sigmoid) or "isotonic".
verbose	Logical. Print results (default: TRUE).

Value

A list with calibration_model (a glm or isoreg object), predictions (the input with an added .pred_calibrated column), target, method, brier_original, brier_calibrated, ece_original and ece_calibrated.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
x <- rnorm(300)
y <- factor(ifelse(x + rnorm(300) > 0, "yes", "no"), levels = c("no", "yes"))
preds <- data.frame(y = y, .pred_yes = plogis(3 * x))
cal <- calibrate_probabilities(preds, target = "y", verbose = FALSE)
c(cal$brier_original, cal$brier_calibrated)
```

causal_ml *Automated Causal Inference Pipeline*

Description

Complete causal inference pipeline for observational data. It includes a causal exploratory analysis, propensity score methods (matching, inverse probability weighting and augmented IPW), Double Machine Learning, causal forests, meta-learners (S, T and X) and, optionally, TMLE and Python methods (EconML through 'reticulate').

Usage

```
causal_ml(
  data,
  treatment,
  outcome,
  covariates = NULL,
  methods = c("propensity", "dml", "causal_forest", "meta_learners"),
  treatment_type = c("auto", "binary", "continuous"),
  ps_model = c("logistic", "rf", "xgboost"),
  ps_methods = c("matching", "ipw", "doubly_robust"),
  matching_ratio = 1,
  caliper = 0.2,
  dml_model = c("rf", "xgboost", "lasso"),
  dml_n_folds = 5,
  cf_num_trees = 2000,
  cf_honesty = TRUE,
  meta_learners = c("S", "T", "X"),
  meta_base_model = c("rf", "xgboost"),
  run_tmle = FALSE,
  use_python = FALSE,
  python_methods = c("econml_dml", "econml_forest"),
  exclude_cols = NULL,
  impute = TRUE,
  impute_method = c("median", "knn", "mean"),
  normalize = TRUE,
  normalize_method = c("zscore", "minmax"),
  run_eda = TRUE,
  confidence_level = 0.95,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame with the data.
treatment	Character string with the name of the treatment variable.

outcome	Character string with the name of the outcome variable.
covariates	Character vector with the names of the covariates. If NULL (default), all remaining columns are used.
methods	Character vector with the methods to run: any of "propensity", "dml", "causal_forest" and "meta_learners" (default: all four).
treatment_type	Treatment type: "auto" (default, detected from the data), "binary" or "continuous".
ps_model	Model used to estimate the propensity score: "logistic" (default), "rf" or "xgboost".
ps_methods	Propensity score methods to use: any of "matching", "ipw" and "doubly_robust" (default: all three).
matching_ratio	Number of controls matched to each treated unit (default: 1).
caliper	Matching caliper in standard deviations of the propensity score (default: 0.2).
dml_model	Base learner for Double Machine Learning: "rf" (default), "xgboost" or "lasso".
dml_n_folds	Number of cross-fitting folds in DML (default: 5).
cf_num_trees	Number of trees of the causal forest (default: 2000).
cf_honesty	Logical; use honest splitting in the causal forest (default: TRUE).
meta_learners	Meta-learners to run: any of "S", "T" and "X" (default: all three).
meta_base_model	Base model of the meta-learners: "rf" (default) or "xgboost".
run_tmle	Logical; run TMLE (default: FALSE).
use_python	Logical; run Python methods through 'reticulate' (default: FALSE).
python_methods	Python methods to run: any of "econml_dml" and "econml_forest".
exclude_cols	Character vector of columns to exclude from the analysis (default: NULL).
impute	Logical; impute missing values (default: TRUE).
impute_method	Imputation method: "median" (default), "knn" or "mean".
normalize	Logical; normalize the covariates (default: TRUE).
normalize_method	Normalization method: "zscore" (default) or "minmax".
run_eda	Logical; run the causal exploratory analysis (default: TRUE).
confidence_level	Confidence level of the intervals (default: 0.95).
seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical; print progress to the console (default: TRUE).

Details

Propensity score methods and meta-learners are only run for binary treatments. Several methods rely on suggested packages that are used only when installed: 'MatchIt' (matching), 'cobalt' (love plot), 'grf' (causal forest), 'DoubleML', 'mlr3' and 'mlr3learners' (DML), 'tmle' and 'SuperLearner' (TMLE) and 'reticulate' (Python methods). When one of them is missing, the corresponding method is skipped with a warning.

Value

An object of class `causalml` (a list) with the components:

call The matched call of the internal pipeline.

params List of analysis parameters (treatment, outcome, covariates, methods, treatment_type, confidence_level, seed).

eda Causal exploratory analysis: distributions, pre-adjustment balance and potential confounders (only if `run_eda = TRUE`).

preprocessed List with the preprocessed data and the final covariates.

assumptions Assumption checks: overlap, positivity and balance after weighting.

propensity Propensity score results (scores, matching, IPW and AIPW estimates with their confidence intervals).

advanced DML, causal forest (with CATE and variable importance) and TMLE results, when requested.

meta_learners S-, T- and X-learner results with individual CATE estimates and the ATE.

python EconML results (only if `use_python = TRUE`).

summary_table Data frame comparing the methods, with columns `method`, `ate` (average treatment effect), `se`, `ci_lower`, `ci_upper` and `p_value`.

evalue List with the E-value of the sensitivity analysis (`evalue_point`, `evalue_ci`).

figures Named list of 'ggplot2' plots.

figures_catalog Data frame cataloguing the figures.

elapsed_time Execution time in seconds.

data_original The input data.

verbose_text, verbose_lines Captured console output (only if `verbose = TRUE`).

See Also

Other causal inference: `plot.causalml()`, `predict.causalml()`, `print.causalml()`, `summary.causalml()`

Examples

```
# Simulated confounded data: x1 drives both treatment and outcome
set.seed(1)
n <- 300
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$treat <- rbinom(n, 1, plogis(df$x1))
df$y <- 1.5 * df$treat + df$x1 + 0.5 * df$x2 + rnorm(n)

res <- causal_ml(df, treatment = "treat", outcome = "y",
                 methods = c("propensity", "meta_learners"),
                 ps_methods = c("ipw", "doubly_robust"),
                 meta_learners = "T", run_eda = FALSE,
                 seed = 1, verbose = FALSE)
res$summary_table
```

`cluster_profile` *Cluster Profiles*

Description

Computes the mean of every numeric variable within each cluster found by `unsupervised_ml()`, together with the overall (global) mean. Noise points (label 0, as produced by DBSCAN) are excluded.

Usage

```
cluster_profile(result, algorithm = NULL, original_data = NULL, verbose = TRUE)
```

Arguments

`result`	An object of class `unsupervisedml`, as returned by `unsupervised_ml()`.
`algorithm`	Name of the clustering algorithm whose labels are used (e.g. `"kmeans"`). If NULL (default), the best algorithm of the evaluation (or the first one available) is used.
`original_data`	Optional data frame with the same rows as the data used in the analysis, used to profile variables on their original scale or variables not used in the clustering. If NULL (default), the preprocessed data before normalization are used.
`verbose`	Logical. If TRUE (default), the cluster sizes and means are printed to the console.

Value

Invisibly, a data frame with one row per cluster plus a final row `"Global"`: the column `Cluster` holds the cluster label and the remaining columns hold the mean of each numeric variable.

See Also

Other unsupervised learning: `plot.unsupervisedml()`, `predict.unsupervisedml()`, `print.unsupervisedml()`, `save_unsupervised_plots()`, `summary.unsupervisedml()`, `unsupervised_ml()`

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X, methods = "clustering",
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k = 2, run_eda = FALSE, seed = 1, verbose = FALSE)
prof <- cluster_profile(res, algorithm = "kmeans", verbose = FALSE)
prof
```

compare_models *Compare Cross-Validated Models*

Description

Summarizes the cross-validated metrics of all models, adds the mean rank across metrics (rank_mean) and selects the best model according to select_metric. With verbose = TRUE it also reports the stability of the best model across folds.

Usage

```
compare_models(cv_results, task, select_metric = NULL, verbose = TRUE)
```

Arguments

cv_results	Named list of resampling results, as returned by fit_models_cv().
task	Type of task: "classification" or "regression".
select_metric	Metric used to select the best model. If NULL (default) ROC-AUC is used for classification and R-squared for regression. "rank_mean" selects the model with the best mean rank.
verbose	Logical. Print the progress and an explanation of each step in the console (default: TRUE).

Value

A list with summary (data frame with one row per model, ordered from best to worst, with column model, one column per metric and rank_mean), all_metrics (long data frame from tune::collect_metrics() with a model column), best_model, best_metric (value of the selection metric for the best model) and select_metric.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
pp <- preprocess_data(df, target = "y", task = "classification",
                      verbose = FALSE)
folds <- setup_cv(pp$train_data, target = "y", task = "classification",
                  cv_folds = 3, verbose = FALSE)
specs <- define_models(c("glm", "rf"), task = "classification",
                       verbose = FALSE)
cv_res <- fit_models_cv(specs, pp$recipe, folds, task = "classification",
                        verbose = FALSE)
cmp <- compare_models(cv_res, task = "classification", select_metric = "mcc")
cmp$best_model
```

---

correct_fairness_threshold

Fairness Correction with Group-Specific Thresholds

Description

Post-processing correction in the spirit of equality of opportunity (Hardt, Price and Srebro, 2016). For each group of the protected variable it searches (over a grid from 0.05 to 0.95 in steps of 0.01) the classification threshold whose true positive rate (TPR) is closest to a common target TPR, defined as the mean of the TPRs obtained at each group's Youden threshold. The predicted classes are then recomputed with the group-specific thresholds.

Usage

```
correct_fairness_threshold(predictions, target, protected_var, verbose = TRUE)
```

Arguments

`predictions`	Data frame with one row per case containing the observed outcome (`target`, a factor with 2 levels), the protected variable and the probability of the positive class (`.pred_<positive level>`).
`target`	Name of the outcome variable.
`protected_var`	Name of the protected variable.
`verbose`	Logical (default TRUE). Print the thresholds and the TPR before and after the correction.

Value

A list with:

- method: Character, "postprocesamiento" (post-processing).
- group_thresholds: Data frame with columns group, threshold (group-specific threshold), tpr_before (TPR at the 0.5 threshold) and tpr_after (TPR at the new threshold).
- target_tpr: Common target TPR.
- corrected_predictions: The predictions data frame with .pred_class recomputed using the group-specific thresholds.

References

Hardt, M., Price, E., & Srebro, N. (2016). Equality of opportunity in supervised learning. Advances in Neural Information Processing Systems, 29.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 300
sex <- sample(c("female", "male"), n, replace = TRUE)
prob <- plogis(rnorm(n) + ifelse(sex == "male", 0.5, 0))
preds <- data.frame(
  outcome = factor(ifelse(runif(n) < prob, "yes", "no"), levels = c("no", "yes")),
  .pred_yes = prob,
  sex = sex
)

corr <- correct_fairness_threshold(preds, target = "outcome",
                                   protected_var = "sex")
corr$group_thresholds
table(corr$corrected_predictions$.pred_class, preds$sex)
```

`cv_feature_selection` *Feature Selection with Cross-Validation*

Description

Performs feature selection inside a cross-validation loop to reduce overfitting. In each fold a random forest (ranger, 100 trees) is fitted on complete cases and the most important predictors (the top half, but at least `min_features`) are retained; predictors selected in at least half of the folds are returned.

Usage

```
cv_feature_selection(
  data,
  target,
  task,
  cv_folds = 5,
  min_features = 5,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Training data (data frame).
`target`	Name of the target variable.
`task`	Type of task: "classification" (stratified folds) or "regression".
`cv_folds`	Number of folds (default: 5).
`min_features`	Minimum number of features retained in each fold (default: 5).
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.
`verbose`	Logical. Print progress (default: TRUE).

Value

A list with `selected_features` (character vector), `selection_counts` (named numeric vector with the number of folds in which each predictor was selected, sorted), `threshold` (minimum number of folds required) and `n_folds`.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`,

optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 150
df <- as.data.frame(matrix(rnorm(n * 6), n, 6))
df$y <- df$V1 + df$V2 + rnorm(n)
fs <- cv_feature_selection(df, target = "y", task = "regression",
                           cv_folds = 3, min_features = 2, seed = 1,
                           verbose = FALSE)
fs$selected_features
```

---

define_models *Define Model Specifications*

Description

Creates the parsnip model specifications of the requested models for the given task, with default hyperparameters.

Usage

```
define_models(models, task, verbose = TRUE)
```

Arguments

models	Character vector with the models. Available: "rf" (random forest), "xgboost", "svm" (RBF kernel), "nnet" (neural network), "glm" (logistic or linear regression), "glmnet" (regularized regression), "tree" (decision tree) and "nb" (naive Bayes). Some models need suggested packages (e.g. **kernlab**, **glmnet**, **rpart**).
task	Type of task: "classification" or "regression".
verbose	Logical. Print the progress and an explanation of each step in the console (default: TRUE).

Value

A named list of parsnip model specifications (model_spec objects), one per model.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
specs <- define_models(c("glm", "rf"), task = "classification")
names(specs)
```

---

detect_data_leakage *Detect Possible Data Leakage (Post-Model Checks)*

Description

Inspects the results of a trained model for indicators of data leakage that can only be checked after training. Pre-model checks (correlation with the target, possible identifiers) are done by eda_leakage().

Checks performed:

- test AUC or R-squared > 0.99 (suspiciously high performance);
- one variable accounting for more than 80% of the total importance (dominant variable).

Usage

```
detect_data_leakage(model_results = NULL, importance = NULL, verbose = TRUE)
```

Arguments

model_results	A list with a test_metrics element (data frame with columns .metric and .estimate), such as a supervised_ml() result. Optional (default: NULL).
importance	Data frame of variable importance with columns Variable and Importance, sorted by decreasing importance (e.g. the importance element of a supervised_ml() result). Optional (default: NULL).
verbose	Logical. Print results (default: TRUE).

Value

A list with `warnings` (list with elements `high_performance` and/or `dominant_variable` describing each indicator), `n_warnings` (number of indicators) and `leakage_suspected` (logical).

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
# Inputs built by hand
metrics <- data.frame(.metric = c("roc_auc", "accuracy"),
                      .estimate = c(0.995, 0.97))
imp <- data.frame(Variable = c("a", "b", "c"), Importance = c(90, 6, 4))
detect_data_leakage(list(test_metrics = metrics), imp,
                    verbose = FALSE)$n_warnings
# From a supervised_ml() result
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
lk <- detect_data_leakage(res, importance = res$importance, verbose = FALSE)
lk$leakage_suspected
```

---

diversity_split *Diversity-Based Train/Test Split for True Generalization*

Description

Creates train/test splits where the test set is genuinely DIFFERENT from the training set, not just a random subset of the same distribution. This enables a fairer evaluation of model generalization.

Usage

```
diversity_split(
  data,
  test_prop = 0.2,
  method = "hybrid",
  n_clusters = NULL,
  exclude_cols = NULL,
  target = NULL,
  stratify = TRUE,
  diversity_strength = 0.5,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Data frame to split.
`test_prop`	Proportion for test set (default 0.20).
`method`	Method for creating diverse split: • `"cluster"`: Assigns entire clusters to test (most different) • `"dissimilarity"`: Selects most dissimilar observations for test • `"boundary"`: Selects observations at distribution boundaries • `"hybrid"`: Combines clustering with boundary selection
`n_clusters`	Number of clusters for the cluster-based methods. If NULL (default) it is set to `round(sqrt(n / 2))`, bounded between 3 and 10.
`exclude_cols`	Character vector of columns to exclude from the distance calculations (e.g., ID columns).
`target`	Name of the target variable. It is excluded from the distance calculations.
`stratify`	Logical (default TRUE). Reserved for keeping class proportions of `target`; it is passed to the cluster-based methods but class proportions are not currently enforced.
`diversity_strength`	Number between 0 and 1 (default 0.5) controlling how aggressively diversity is maximized. Higher values give a more different test set, but may be too extreme.
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.
`verbose`	Logical (default TRUE). If TRUE, prints a summary of the split.

Details

The key insight is that random sampling from a homogeneous distribution creates train and test sets that are essentially "twins" - they share the same patterns, making test performance an optimistic estimate of true generalization.

This function creates genuinely different partitions by:

- **Cluster method**: Groups similar observations, then assigns entire clusters to test. The test set contains "types" of observations not well-represented in training.
- **Dissimilarity method**: Selects observations that are most different from the data centroid, ensuring test covers edge cases.
- **Boundary method**: Selects observations at the boundaries of the multivariate distribution (extreme combinations of variables).
- **Hybrid method**: Combines clustering with dissimilarity selection within clusters.

Value

A list of class `c("diversity_split", "list")` containing:

- train: Training data frame.
- test: Test data frame.
- train_idx: Sorted row indices of the training observations.
- test_idx: Sorted row indices of the test observations.
- diversity_metrics: List with `mean_distance_ratio` (mean distance of test cases to the training centroid divided by the mean within-training distance), `ks_statistic` (average two-sample Kolmogorov-Smirnov statistic across variables), `centroid_distance` (Euclidean distance between train and test centroids), `random_baseline` (centroid distance of one random split of the same size) and `diversity_ratio` (`centroid_distance` divided by `random_baseline`). Higher values mean more different sets.
- cluster_info: Data frame with cluster sizes and test membership (methods `"cluster"` and `"hybrid"`), or NULL.
- settings: List with `n_total`, `n_train`, `n_test`, `method`, `n_clusters`, `diversity_strength` and `seed`.

If fewer than two usable numeric columns are available, a warning is issued and a plain list with `train`, `test`, `train_idx` and `test_idx` from a random split is returned.

See Also

Other study design: `block_split()`, `estimate_n()`, `estimate_n_plateau()`, `ml_sample_size()`, `plot_all.ml_sample_size()`, `plot_distribution()`, `plot_learning_curve()`

Examples

```
set.seed(1)
survey_data <- data.frame(
  x1 = rnorm(120), x2 = rnorm(120), x3 = rnorm(120),
  depression = factor(sample(c("yes", "no"), 120, replace = TRUE))
)

# Default (hybrid) method
result <- diversity_split(survey_data, target = "depression",
                          test_prop = 0.20, seed = 1)

# Cluster-based split (entire clusters go to test)
```

```
result <- diversity_split(
  data = survey_data,
  target = "depression",
  method = "cluster",
  n_clusters = 5,
  test_prop = 0.20,
  seed = 1,
  verbose = FALSE
)

# Check how different train and test are
result$diversity_metrics
```

`eda_basic` *Basic Exploratory Data Analysis*

Description

Runs a basic exploratory analysis that does not require a target variable. Useful to explore a dataset before defining the model.

Usage

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

Arguments

`data`	Data frame with the data.
`verbose`	Logical. Print progress and results (default: TRUE).

Value

An object of class `c("supervisedml_eda", "list")` with elements `structure` (`eda_structure()`), `missing` (`eda_missing()`), `numeric` (`eda_numeric_basic()`), `categorical` (`eda_categorical_basic()`), `outliers` (`eda_outliers()`) and `correlation` (`eda_correlation_basic()`).

See Also

Other exploratory data analysis: `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_correlation_basic()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_outliers()`, `eda_structure()`, `eda_summary()`, `eda_target_classification()`, `eda_target_regression()`, `eda_vif()`

Examples

```
res <- eda_basic(iris, verbose = FALSE)
res$numeric
```

eda_categorical *Categorical Variable Analysis*

Description

Lists the categorical (factor or character) predictors and their number of levels.

Usage

```
eda_categorical(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable (excluded from the analysis).
verbose	Logical. Print results (default: TRUE).

Value

A list with variables (names of the categorical predictors) and n_levels (number of distinct values of each one).

See Also

Other exploratory data analysis: eda_basic(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
df <- data.frame(g = c("a", "b", "a"), h = factor(c("x", "y", "z")), y = 1:3)
eda_categorical(df, target = "y", verbose = FALSE)
```

eda_categorical_basic *Basic Categorical Variable Analysis*

Description

Lists all categorical (factor or character) variables and their number of levels.

Usage

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

Arguments

data	Data frame.
verbose	Logical. Print results (default: TRUE).

Value

A list with variables (names of the categorical variables) and n_levels (number of distinct values of each one).

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_categorical_basic(iris, verbose = FALSE)
```

---

eda_correlation *Correlation Matrix*

Description

Pearson correlation matrix of the numeric variables (pairwise complete observations); with verbose = TRUE it reports the strongest correlations with the target and pairs with $|r| > 0.8$.

Usage

```
eda_correlation(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable.
verbose	Logical. Print results (default: TRUE).

Value

A numeric correlation matrix, or NULL if there are fewer than two numeric variables.

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
round(eda_correlation(mtcars[, 1:4], target = "mpg", verbose = FALSE), 2)
```

`eda_correlation_basic` *Basic Correlation Matrix*

Description

Pearson correlation matrix of all numeric variables (pairwise complete observations); with `verbose = TRUE` it reports pairs with $|r| > 0.7$.

Usage

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

Arguments

`data`	Data frame.
`verbose`	Logical. Print results (default: TRUE).

Value

A numeric correlation matrix, or NULL if there are fewer than two numeric variables.

See Also

Other exploratory data analysis: `eda_basic()`, `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_outliers()`, `eda_structure()`, `eda_summary()`, `eda_target_classification()`, `eda_target_regression()`, `eda_vif()`

Examples

```
round(eda_correlation_basic(iris, verbose = FALSE), 2)
```

eda_leakage *Basic Data Leakage Check*

Description

Detects possible indicators of data leakage BEFORE training the model: numeric predictors with a very high correlation with the target ($|r| > 0.95$) and possible identifier variables (one unique value per row).

Usage

```
eda_leakage(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable (factors are converted to their integer codes for the correlation).
verbose	Logical. Print results (default: TRUE).

Value

A list with warnings (list with elements high_correlation and/or possible_ids, each with the flagged variables), n_warnings (number of indicators found) and has_leakage (logical).

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
set.seed(1)
df <- data.frame(id = 1:50, x = rnorm(50))
df$y <- df$x + rnorm(50)
df$leak <- df$y + rnorm(50, sd = 0.01)
eda_leakage(df, target = "y", verbose = FALSE)$n_warnings
```

eda_missing *Missing Value Analysis*

Description

Counts missing values by variable and in the whole dataset.

Usage

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

Arguments

data	Data frame.
verbose	Logical. Print results (default: TRUE).

Value

A list with by_variable (data frame with columns variable, n_missing and pct_missing, sorted by decreasing missingness), total (total number of missing cells) and total_pct (percentage of missing cells).

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_missing(airquality, verbose = FALSE)$by_variable
```

eda_normality *Normality Test of the Target*

Description

Shapiro-Wilk test of a numeric target (a random sample of 5000 observations is used when there are more). This is a diagnostic: most machine learning models do not assume normality.

Usage

```
eda_normality(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the numeric target variable.
verbose	Logical. Print results (default: TRUE).

Value

An object of class htest (see stats::shapiro.test()), or invisibly NULL if the target is not numeric.

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_normality(mtcars, "mpg", verbose = FALSE)
```

---

eda_numeric *Numeric Variable Analysis*

Description

Descriptive statistics of the numeric predictors.

Usage

```
eda_numeric(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable (excluded from the analysis).
verbose	Logical. Print results (default: TRUE).

Value

A list with variables (names of the numeric predictors) and stats (data frame with columns variable, mean, sd, min and max). If there are no numeric predictors only variables (empty) is returned.

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_numeric(mtcars, target = "mpg", verbose = FALSE)$stats
```

---

eda_numeric_basic *Basic Numeric Variable Analysis*

Description

Descriptive statistics of all numeric variables.

Usage

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

Arguments

data	Data frame.
verbose	Logical. Print results (default: TRUE).

Value

A data frame with columns variable, n, missing, mean, sd, min, median and max (rounded to 2 decimals); an empty data frame if there are no numeric variables.

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_numeric_basic(iris, verbose = FALSE)
```

`eda_outliers` *Outlier Detection*

Description

Counts univariate outliers in every numeric variable using a robust z-score based on the median absolute deviation (MAD) or the interquartile range (IQR) rule.

Usage

```
eda_outliers(data, verbose = TRUE, method = c("mad", "iqr"), threshold = NULL)
```

Arguments

`data`	Data frame.
`verbose`	Logical. Print results (default: TRUE).
`method`	Detection method: "mad" (default, robust) or "iqr" (classical).
`threshold`	Detection threshold. For MAD, the robust z-score cutoff (used when NULL: 3.29). For IQR, the multiplier of the interquartile range (used when NULL: 3).

Value

A data frame with columns `variable`, `n_outliers` and `pct` (percentage of non-missing values flagged), sorted by decreasing number of outliers.

See Also

Other exploratory data analysis: `eda_basic()`, `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_correlation_basic()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_structure()`, `eda_summary()`, `eda_target_classification()`, `eda_target_regression()`, `eda_vif()`

Examples

```
eda_outliers(mtcars, verbose = FALSE)
eda_outliers(mtcars, method = "iqr", threshold = 1.5, verbose = FALSE)
```

eda_structure *Data Structure*

Description

Counts observations and variables and classifies variables as numeric or categorical.

Usage

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

Arguments

data	Data frame.
verbose	Logical. Print results (default: TRUE).

Value

A list with n_obs, n_vars, n_numeric, n_categorical and var_types (named character vector with the type of each variable).

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_summary(), eda_target_classification(), eda_target_regression(), eda_vif()

Examples

```
eda_structure(iris, verbose = FALSE)
```

eda_summary *Exploratory Data Analysis*

Description

Runs a complete exploratory analysis of the data: target distribution, missing values, categorical and numeric variables, outliers, correlations, multicollinearity (VIF) and a basic data leakage check. If target is NULL, a basic EDA without target is run (see eda_basic()).

Usage

```
eda_summary(data, target = NULL, task = "auto", verbose = TRUE)
```

Arguments

data	Data frame with the data.
target	Name of the target variable (optional; if NULL a basic EDA is performed).
task	Type of task: "classification", "regression" or "auto" (detected from the target).
verbose	Logical. Print progress and results (default: TRUE).

Value

An object of class `c("supervisedml_eda", "list")`. With a target it contains `structure` (see `eda_structure()`), `target_dist`, `normality` (regression only), `missing`, `categorical`, `numeric`, `outliers`, `correlation`, `vif`, `leakage_check`, `task` and `target`. Without a target, the elements returned by `eda_basic()`.

See Also

Other exploratory data analysis: `eda_basic()`, `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_correlation_basic()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_outliers()`, `eda_structure()`, `eda_target_classification()`, `eda_target_regression()`, `eda_vif()`

Examples

```
res <- eda_summary(iris, target = "Species", verbose = FALSE)
names(res)
res_reg <- eda_summary(mtcars, target = "mpg", verbose = FALSE)
res_reg$vif
```

---

eda_target_classification

Target Distribution for Classification

Description

Frequencies and proportions of the classes of a categorical target, with the imbalance ratio.

Usage

```
eda_target_classification(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable (coerced to factor).
verbose	Logical. Print results (default: TRUE).

Value

A list with `frequencies` (table), `proportions` (table, in percent), `n_classes` and `imbalance_ratio` (largest over smallest class frequency).

See Also

Other exploratory data analysis: `eda_basic()`, `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_correlation_basic()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_outliers()`, `eda_structure()`, `eda_summary()`, `eda_target_regression()`, `eda_vif()`

Examples

```
eda_target_classification(iris, "Species", verbose = FALSE)
```

---

eda_target_regression *Target Distribution for Regression*

Description

Descriptive statistics of a numeric target.

Usage

```
eda_target_regression(data, target, verbose = TRUE)
```

Arguments

`data`	Data frame.
`target`	Name of the numeric target variable.
`verbose`	Logical. Print results (default: TRUE).

Value

A list with `mean`, `sd`, `median`, `min`, `max` and `skewness` (a simple skewness index: (mean - median) / sd).

See Also

Other exploratory data analysis: `eda_basic()`, `eda_categorical()`, `eda_categorical_basic()`, `eda_correlation()`, `eda_correlation_basic()`, `eda_leakage()`, `eda_missing()`, `eda_normality()`, `eda_numeric()`, `eda_numeric_basic()`, `eda_outliers()`, `eda_structure()`, `eda_summary()`, `eda_target_classification()`, `eda_vif()`

Examples

```
eda_target_regression(mtcars, "mpg", verbose = FALSE)
```

eda_vif *Variance Inflation Factor Analysis*

Description

Computes the variance inflation factor (VIF) of each numeric predictor by regressing it on the remaining numeric predictors.

Usage

```
eda_vif(data, target, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable (excluded from the analysis).
verbose	Logical. Print results (default: TRUE).

Value

A data frame with columns variable and VIF, sorted by decreasing VIF, or NULL if there are fewer than two numeric predictors, too few complete observations, or the computation fails.

See Also

Other exploratory data analysis: eda_basic(), eda_categorical(), eda_categorical_basic(), eda_correlation(), eda_correlation_basic(), eda_leakage(), eda_missing(), eda_normality(), eda_numeric(), eda_numeric_basic(), eda_outliers(), eda_structure(), eda_summary(), eda_target_classification(), eda_target_regression()

Examples

```
eda_vif(mtcars, target = "mpg", verbose = FALSE)
```

estimate_n *Estimate N for a Target Metric Using the Curve Fit*

Description

Uses the fitted power-law model to estimate the sample size required to achieve a specified target metric value.

Usage

```
estimate_n(object, target_metric, model = NULL, warn_extrapolation = TRUE)
```

Arguments

object	An object of class ml_sample_size returned by ml_sample_size().
target_metric	Target metric value.
model	For multi-model results, the model to use. If NULL (default), the first model is used.
warn_extrapolation	Logical (default TRUE). Warn if the fit is poor or the estimate extrapolates beyond the observed sample sizes.

Details

This function inverts the power-law equation to find the required n. Warnings are issued when:

- The target exceeds the estimated asymptote.
- The required n exceeds 2 times the largest observed sample size.
- The curve fit quality is poor (pseudo-R2 < 0.80).

Value

A single number: the estimated sample size (rounded up to an integer value), or Inf if the target is beyond the estimated asymptote.

See Also

Other study design: block_split(), diversity_split(), estimate_n_plateau(), ml_sample_size(), plot_all.ml_sample_size(), plot_distribution(), plot_learning_curve()

Examples

```
res <- ml_sample_size(model = "glm", n_grid = c(50, 100, 200, 400), reps = 5,
                      n_test = 500, sim_params = list(N_pop = 2000, p = 5),
                      seed = 1, verbose = FALSE)
a <- coef(res$curve_fit)["a"]
estimate_n(res, target_metric = a - 0.02)
```

---

estimate_n_plateau *Estimate N to Reach the Plateau (Near Maximum Performance)*

Description

Computes the sample size at which performance approaches the asymptote within a specified distance (delta).

Usage

```
estimate_n_plateau(
  object,
  delta = 0.01,
  relative = FALSE,
  model = NULL,
  warn_extrapolation = TRUE
)
```

Arguments

object	An object of class ml_sample_size returned by ml_sample_size().
delta	Distance to the asymptote that defines the plateau (default 0.01).
relative	Logical (default FALSE). If TRUE, delta is interpreted as a proportion of the target metric (e.g., delta = 0.01 means 1% of the target). Requires a result computed with a target.
model	For multi-model results, the model to use. If NULL (default), the first model is used.
warn_extrapolation	Logical (default TRUE). Warn if the fit is poor or the estimate extrapolates beyond the observed sample sizes.

Details

The plateau is defined operationally as n_plateau = (b / delta)^(1 / c).

When relative = TRUE and target = 0.80, delta = 0.01 becomes 0.008 (1% of 0.80), yielding a more stringent plateau definition.

Value

A single number: the estimated sample size for the plateau (rounded up to an integer value).

See Also

Other study design: block_split(), diversity_split(), estimate_n(), ml_sample_size(), plot_all.ml_sample_size(), plot_distribution(), plot_learning_curve()

Examples

```
result <- ml_sample_size(model = "glm", n_grid = c(50, 100, 200, 400),
                         reps = 5, target = 0.75, n_test = 500,
                         sim_params = list(N_pop = 2000, p = 5),
                         seed = 1, verbose = FALSE)

# Absolute delta (default)
estimate_n_plateau(result, delta = 0.01, warn_extrapolation = FALSE)

# Relative delta (1% of target)
estimate_n_plateau(result, delta = 0.01, relative = TRUE,
```

```
                    warn_extrapolation = FALSE)
```

`eval_calibration` *Calibration Analysis by Probability Bins*

Description

Compares the mean predicted probability of the positive class with the observed proportion of positives within ten probability bins of width 0.1. Binary classification only.

Usage

```
eval_calibration(predictions, target, verbose = TRUE)
```

Arguments

`predictions`	A data frame of predictions with the observed outcome column and, for classification, the `.pred_class` column and one `.pred_<level>` probability column per class (as returned by `eval_predictions()`).
`target`	Character string with the name of the outcome variable.
`verbose`	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A tibble with columns `prob_bin` (probability bin), `n` (number of cases), `mean_pred` (mean predicted probability) and `mean_obs` (observed proportion of positives); NULL for multiclass outcomes.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
y <- factor(sample(c("no", "yes"), 100, replace = TRUE))
p <- stats::plogis(ifelse(y == "yes", 1, -1) + rnorm(100))
preds <- data.frame(y = y,
                    .pred_class = factor(ifelse(p >= 0.5, "yes", "no"),
                                         levels = levels(y)),
                    .pred_no = 1 - p, .pred_yes = p)
eval_calibration(preds, target = "y", verbose = FALSE)
```

`eval_confusion` *Compute the Confusion Matrix*

Description

Cross-tabulates predicted and observed classes and, when `verbose = TRUE`, explains the true/false positives and negatives.

Usage

```
eval_confusion(predictions, target, verbose = TRUE)
```

Arguments

`predictions`	A data frame of predictions with the observed outcome column and, for classification, the `.pred_class` column and one `.pred_<level>` probability column per class (as returned by `eval_predictions()`).
`target`	Character string with the name of the outcome variable.
`verbose`	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

An object of class `conf_mat` (see `yardstick::conf_mat()`), whose `table` element holds the counts (predictions in rows, truth in columns).

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`,

predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
y <- factor(sample(c("no", "yes"), 100, replace = TRUE))
p <- stats::plogis(ifelse(y == "yes", 1, -1) + rnorm(100))
preds <- data.frame(y = y,
                    .pred_class = factor(ifelse(p >= 0.5, "yes", "no"),
                                         levels = levels(y)),
                    .pred_no = 1 - p, .pred_yes = p)
eval_confusion(preds, target = "y")
```

---

eval_metrics *Compute Performance Metrics*

Description

Computes performance metrics from a data frame of predictions. For binary classification the positive class is detected from the level names (e.g. "yes", "1", "positive"; otherwise the second level). Multiclass metrics use macro averaging and the Hand-Till ROC-AUC.

Usage

```
eval_metrics(predictions, target, task, verbose = TRUE)
```

Arguments

predictions	A data frame of predictions with the observed outcome column and, for classification, the .pred_class column and one .pred_<level> probability column per class (as returned by eval_predictions()).
target	Character string with the name of the outcome variable.
task	Type of task: "classification" or "regression".
verbose	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A tibble with columns .metric, .estimator and .estimate. Classification: roc_auc, accuracy, sensitivity, specificity, f_meas, mcc, kap and f2_meas. Regression: rmse, rsq and mae.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
y <- factor(sample(c("no", "yes"), 100, replace = TRUE))
p <- stats::plogis(ifelse(y == "yes", 1, -1) + rnorm(100))
preds <- data.frame(y = y,
                    .pred_class = factor(ifelse(p >= 0.5, "yes", "no"),
                                         levels = levels(y)),
                    .pred_no = 1 - p, .pred_yes = p)
eval_metrics(preds, target = "y", task = "classification")

# Regression
yr <- rnorm(100, mean = 10)
preds_r <- data.frame(yr = yr, .pred = yr + rnorm(100, sd = 0.5))
eval_metrics(preds_r, target = "yr", task = "regression", verbose = FALSE)
```

---

eval_pred_vs_obs *Compare Predicted and Observed Values*

Description

Computes the correlation between observed and predicted values and the mean bias of the predictions (regression).

Usage

```
eval_pred_vs_obs(predictions, target, verbose = TRUE)
```

Arguments

predictions	A data frame of regression predictions with the observed outcome column and the .pred column (as returned by eval_predictions()).
target	Character string with the name of the outcome variable.
verbose	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A list with observed and predicted (numeric vectors), correlation (Pearson correlation) and bias (mean of predicted minus observed).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
y <- rnorm(100, mean = 10)
preds <- data.frame(y = y, .pred = y + rnorm(100, sd = 0.5))
pvo <- eval_pred_vs_obs(preds, target = "y")
pvo$correlation
```

---

eval_predictions *Generate Predictions on a Data Set*

Description

Predicts the outcome of a data set with a fitted workflow and binds the predictions (and class probabilities for classification) to the data.

Usage

```
eval_predictions(final_fit, test_data, target, task, verbose = TRUE)
```

Arguments

final_fit	A fitted tidymodels workflow, e.g. the final_fit element of a supervised_ml() result.
test_data	Data frame with the observations to predict.
target	Character string with the name of the outcome variable.
task	Type of task: "classification" or "regression".
verbose	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A tibble with the columns of test_data plus .pred_class and one .pred_<level> probability column per class (classification), or .pred (regression).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
preds <- eval_predictions(res$final_fit, res$test_data, target = "y",
                          task = "classification")
head(preds)
```

`eval_residuals` *Residual Analysis*

Description

Computes the residuals (observed minus predicted), their descriptive statistics and a Shapiro-Wilk normality test (on a random subsample of 5000 residuals when there are more).

Usage

```
eval_residuals(predictions, target, verbose = TRUE)
```

Arguments

`predictions`	A data frame of regression predictions with the observed outcome column and the `.pred` column (as returned by `eval_predictions()`).
`target`	Character string with the name of the outcome variable.
`verbose`	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A list with `residuals` (numeric vector), `stats` (list with `mean`, `sd`, `min`, `max` and `median` of the residuals) and `normality_test` (an `htest` object from `stats::shapiro.test()`).

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
y <- rnorm(100, mean = 10)
preds <- data.frame(y = y, .pred = y + rnorm(100, sd = 0.5))
rs <- eval_residuals(preds, target = "y")
rs$stats
```

eval_roc *Compute the ROC Curve*

Description

Computes the ROC curve and the area under the curve from classification predictions (one-vs-all curves and Hand-Till AUC for multiclass outcomes).

Usage

```
eval_roc(predictions, target, verbose = TRUE)
```

Arguments

predictions	A data frame of predictions with the observed outcome column and, for classification, the .pred_class column and one .pred_<level> probability column per class (as returned by eval_predictions()).
target	Character string with the name of the outcome variable.
verbose	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

A list with curve (tibble from yardstick::roc_curve() with .threshold, specificity, sensitivity and, for multiclass, .level), auc (numeric area under the curve) and event_info (list describing the positive class and event level).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
y <- factor(sample(c("no", "yes"), 100, replace = TRUE))
p <- stats::plogis(ifelse(y == "yes", 1, -1) + rnorm(100))
```

```
preds <- data.frame(y = y,
                    .pred_class = factor(ifelse(p >= 0.5, "yes", "no"),
                                         levels = levels(y)),
                    .pred_no = 1 - p, .pred_yes = p)
roc <- eval_roc(preds, target = "y")
roc$auc
```

`evaluate_model` *Evaluate the Final Model on the Test Set*

Description

Fits the final workflow on the whole training set and evaluates it on the test set: predictions, performance metrics, overfitting diagnosis (cross-validation vs test, out-of-bag vs test for random forests, train vs test for other models) and, depending on the task, ROC curve, confusion matrix and calibration (classification) or predicted vs observed values and residual analysis (regression).

Usage

```
evaluate_model(
  tuning_result,
  train_data,
  test_data,
  target,
  task,
  modeling_result = NULL,
  best_model = NULL,
  verbose = TRUE
)
```

Arguments

`tuning_result`	A list with a `final_workflow` element (an unfitted tidymodels workflow), such as the result of `tune_best_model()` or the `tuning` element of a `supervised_ml()` result.
`train_data`	Data frame with the (preprocessed) training set.
`test_data`	Data frame with the (preprocessed) test set.
`target`	Character string with the name of the outcome variable.
`task`	Type of task: `"classification"` or `"regression"`.
`modeling_result`	Optional result of `train_models()`; its cross-validation summary is used in the overfitting diagnosis (default: NULL).
`best_model`	Optional name of the best model, e.g. `"rf"` or `"xgboost"` (default: NULL).
`verbose`	Logical. Print the results and their interpretation in the console (default: TRUE).

Value

An object of class supervisedml_evaluation (a list) with: final_fit (fitted workflow), predictions (test predictions, see eval_predictions()), metrics (test metrics, see eval_metrics()), train_metrics (training metrics, NULL for random forests), overfitting (list with the overfitting diagnosis: cv_auc, test_auc, cv_gap, train vs test comparison, and oob_auc, oob_gap for random forests), task and target. Classification adds roc_data (eval_roc()), confusion (eval_confusion()) and, for binary outcomes, calibration (eval_calibration()). Regression adds pred_obs (eval_pred_vs_obs()) and residuals (eval_residuals()).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
ev <- evaluate_model(res$tuning, res$train_data, res$test_data,
                     target = "y", task = "classification",
                     verbose = FALSE)
ev$metrics
```

---

explain_with_ai *Interpret Results of supervised_ml() with a Large Language Model*

Description

Sends a structured summary of a `supervised_ml()` result to a large language model (Groq or OpenAI chat completions API) and returns a plain-language interpretation of the analysis. A built-in diagnostic engine first checks for common problems (possible data leakage, overfitting, class imbalance, untuned models, etc.), and its findings are included in the request. An API key of the chosen provider is required; no key is bundled with the package.

Usage

```
explain_with_ai(
  result,
  api_key = NULL,
  language = c("es", "en"),
  model = NULL,
  provider = c("groq", "openai"),
  max_tokens = 2000L,
  verbose = TRUE
)
```

Arguments

`result`	An object of class `supervisedml` returned by `supervised_ml()`.
`api_key`	API key of the provider. If `NULL` (default), the environment variable `PSYML_API_KEY` is used; an error is raised if it is not set.
`language`	Language of the interpretation: `"es"` (Spanish, default) or `"en"` (English).
`model`	Model identifier. If `NULL` (default), `"llama-3.3-70b-versatile"` is used for Groq and `"gpt-4.1-mini"` for OpenAI.
`provider`	API provider: `"groq"` (default) or `"openai"`.
`max_tokens`	Maximum number of tokens of the response (default 2000).
`verbose`	Logical. If `TRUE` (default), the diagnostics and the interpretation are printed to the console.

Value

Invisibly, a list with components `success` (logical), `text` (character, the interpretation or the error message), and `diagnostics` (list of findings of the diagnostic engine; each finding has elements `id`, `issue`, `severity`, `details`, and `suggested_code`).

See Also

Other reporting and export: `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
## Not run:
# Requires an API key and an internet connection
set.seed(1)
df <- data.frame(x1 = rnorm(150), x2 = rnorm(150))
df$y <- factor(ifelse(df$x1 + rnorm(150) > 0, "yes", "no"))
res <- supervised_ml(df, "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
Sys.setenv(PSYML_API_KEY = "your-groq-key")
explain_with_ai(res, language = "en")
explain_with_ai(res, provider = "openai", api_key = "your-openai-key")

## End(Not run)
```

`export_verbose_json` *Export the Results of an Analysis as a JSON File*

Description

Writes the result of a PsyML analysis to a structured JSON file (metadata, metrics, model information, variable importance, etc.). When the analysis was run with `verbose = TRUE`, the JSON also includes the complete console output (`verbose_text` and `verbose_lines`). The file can be used by `generate_report_from_json()`, `generate_report_with_ai()` and `launch_report_generator()`.

Usage

```
export_verbose_json(result, file_path)
```

Arguments

`result`	Result of `supervised_ml()` or `supervised_ml_capture()`.
`file_path`	Path of the output file (required, no default).

Value

Invisibly returns `file_path`.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
out <- file.path(tempdir(), "psyml_report.json")
export_verbose_json(res, out)
names(jsonlite::fromJSON(out))
unlink(out)
```

---

export_verbose_txt *Export the Console Output of an Analysis as a TXT File*

Description

Writes the result of a PsyML analysis to a plain text file. When the analysis was run with `verbose = TRUE`, `supervised_ml()` stores the console output, so the file contains exactly the same output shown in the console. Otherwise a summary is built from the result object.

Usage

```
export_verbose_txt(result, file_path)
```

Arguments

`result`	Result of `supervised_ml()` or `supervised_ml_capture()`.
`file_path`	Path of the output file (required, no default).

Value

Invisibly returns `file_path`.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
out <- file.path(tempdir(), "psyml_report.txt")
export_verbose_txt(res, out)
head(readLines(out))
unlink(out)
```

---

fit_models_cv *Fit Models with Cross-Validation*

Description

Fits each model specification with the preprocessing recipe on every cross-validation fold and computes the performance metrics (classification: ROC-AUC, accuracy, sensitivity, specificity, F1, balanced accuracy, PR-AUC (binary only), MCC, kappa and F2; regression: RMSE, R-squared and MAE).

Usage

```
fit_models_cv(
  model_specs,
  recipe,
  cv_folds,
  task,
  select_metric = NULL,
  verbose = TRUE
)
```

Arguments

model_specs	Named list of model specifications, as returned by define_models().
recipe	A recipe object (e.g. the recipe element of preprocess_data()).
cv_folds	An rset of folds, as returned by setup_cv().
task	Type of task: "classification" or "regression".
select_metric	Metric shown in the console for each model. If NULL (default) ROC-AUC is shown for classification and R-squared for regression.
verbose	Logical. Print the progress and an explanation of each step in the console (default: TRUE).

Value

A named list with one resampling result (`resample_results` object from `tune::fit_resamples()`, with saved predictions) per model.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
pp <- preprocess_data(df, target = "y", task = "classification",
                      verbose = FALSE)
folds <- setup_cv(pp$train_data, target = "y", task = "classification",
                  cv_folds = 3, verbose = FALSE)
specs <- define_models("glm", task = "classification", verbose = FALSE)
cv_res <- fit_models_cv(specs, pp$recipe, folds, task = "classification",
                        verbose = FALSE)
tune::collect_metrics(cv_res$glm)
```

---

generate_all_plots *Generate All Plots of a supervisedml Object*

Description

Generates all relevant plots of a `supervised_ml()` result together with metadata for scientific reports. Figures are numbered (Figura 1, Figura 2, ...) and given descriptive names and file names (plot titles and catalog texts are in Spanish). Plots depend on the task and on the analyses run: variable importance, model comparison, ROC curve, confusion matrix, calibration, threshold optimization, predicted vs observed values, residuals, tuning, SHAP, fairness and decision curve analysis.

Usage

```
generate_all_plots(x, verbose = TRUE)
```

Arguments

x	An object of class supervisedml returned by supervised_ml().
verbose	Logical. Print the list of generated figures (default: TRUE).

Value

A list with plots (named list of ggplot objects), figures_catalog (named list with, for each plot, number, type, id, title, description and filename) and total_figures (number of figures).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
gp <- generate_all_plots(res, verbose = FALSE)
names(gp$plots)
gp$plots$roc_curve
```

generate_report_from_json *Generate a Report from an Exported JSON File*

Description

Builds an R Markdown report from a JSON file written by `export_verbose_json()` and renders it to Word, HTML or PDF. Requires the rmarkdown package and pandoc (and a LaTeX installation for PDF).

Usage

```
generate_report_from_json(
  json_path,
  output_path,
  format = c("docx", "html", "pdf"),
  template = c("scientific", "brief", "full")
)
```

Arguments

`json_path`	Path to the JSON file.
`output_path`	Path of the output file (required, no default).
`format`	Output format: `"docx"` (default), `"html"` or `"pdf"`.
`template`	Report template: `"scientific"` (default), `"brief"` or `"full"`.

Value

Invisibly returns `output_path`.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
if (requireNamespace("rmarkdown", quietly = TRUE) &&
    rmarkdown::pandoc_available()) {
  set.seed(1)
  n <- 200
  df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
  df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
  res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                       tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                       verbose = FALSE)
```

```
  json_file <- file.path(tempdir(), "psyml_report.json")
  html_file <- file.path(tempdir(), "psyml_report.html")
  export_verbose_json(res, json_file)
  generate_report_from_json(json_file, html_file, format = "html")
  unlink(c(json_file, html_file))
}
```

`generate_report_with_ai` *Generate a Scientific Report with AI*

Description

Generates a scientific report (Word document) with the OpenAI chat completions API directly from R, without opening the Shiny application. The report text is written by the language model from the JSON file and the tables of metrics and variable importance are inserted from the data. Requires an OpenAI API key, an internet connection and the packages httr2 and officer.

Usage

```
generate_report_with_ai(
  json_path,
  api_key = Sys.getenv("PSYML_API_KEY"),
  output_path = NULL,
  title = "Analisis de Machine Learning",
  author = NULL,
  language = c("es", "en"),
  model = "gpt-4.1-mini",
  custom_instructions = NULL
)
```

Arguments

`json_path`	Path to the JSON file exported with `export_verbose_json()`.
`api_key`	OpenAI API key. Defaults to the environment variable `PSYML_API_KEY`; an error is raised if it is empty.
`output_path`	Path of the output .docx file (required; an error is raised if NULL, the default).
`title`	Report title (default `"Analisis de Machine Learning"`).
`author`	Author name (optional, default NULL).
`language`	Language of the report: `"es"` (Spanish, default) or `"en"` (English).
`model`	OpenAI model to use (default `"gpt-4.1-mini"`).
`custom_instructions`	Optional character string with additional instructions for the language model (e.g., journal style or sections to emphasize). They are appended to the request with high priority. NULL (default) adds nothing.

Value

Invisibly returns `output_path`.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
## Not run:
# Requires an OpenAI API key and an internet connection
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml_capture(df, target = "y", models = "glm", cv_folds = 3,
                             tune_best = FALSE, run_shap = FALSE,
                             n_cores = 1)
json_file <- file.path(tempdir(), "my_analysis.json")
export_verbose_json(res, json_file)

Sys.setenv(PSYML_API_KEY = "your-openai-key")
generate_report_with_ai(
  json_path = json_file,
  output_path = file.path(tempdir(), "my_report.docx"),
  title = "Machine Learning Analysis",
  author = "Jane Doe",
  language = "en"
)

## End(Not run)
```

`get_figures_catalog` *Get the Catalog of Figures*

Description

Returns the catalog of the figures generated by `supervised_ml()`, with their numbers, titles, descriptions and suggested file names (texts in Spanish).

Usage

```
get_figures_catalog(x)
```

Arguments

`x`	An object of class `supervisedml` returned by `supervised_ml()`.

Value

A data frame with one row per figure, ordered by number, and columns `key` (name of the plot in `x$figures`), `number`, `id`, `title`, `description` and `filename`; NULL (with a message) when `x` has no plots or no catalog.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
get_figures_catalog(res)
```

---

get_importance *Variable Importance of a Fitted Workflow*

Description

Extracts variable importance from a fitted `workflow`. With `method = "auto"`, a model-specific measure is used when available: impurity importance for `ranger` and `rpart`, gain for `xgboost`, absolute coefficients for `glmnet`, and absolute z/t statistics for `glm`/`lm`. For other engines (for example `nnet` or `kernlab`), or with `method = "permutation"`, permutation importance is computed on `data`: the drop in performance (AUC for binary outcomes, accuracy for multiclass outcomes, RMSE for numeric outcomes) after shuffling each predictor.

Usage

```
get_importance(
  final_fit,
  data = NULL,
  target = NULL,
  method = c("auto", "native", "permutation"),
  nsim = 5,
  verbose = TRUE
)
```

Arguments

final_fit	A fitted workflow (for example result$final_fit from supervised_ml()).
data	Optional data frame with the predictors and the outcome, used for permutation importance. If NULL, only model-specific importance can be returned.
target	Name of the outcome variable in data.
method	One of "auto" (model-specific if available, otherwise permutation), "native", or "permutation".
nsim	Number of permutations per predictor.
verbose	Logical. If TRUE, the top 10 variables are printed.

Value

A data frame with columns Variable, Importance, and Rank, sorted by decreasing importance. The attribute "method" records the importance measure used. An empty data frame is returned when importance cannot be computed.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
df <- data.frame(x1 = rnorm(100), x2 = rnorm(100))
df$y <- 2 * df$x1 + rnorm(100)
```

```
wf <- workflows::workflow() |>
  workflows::add_formula(y ~ .) |>
  workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
fit <- parsnip::fit(wf, data = df)
get_importance(fit, verbose = FALSE)
get_importance(fit, data = df, target = "y", method = "permutation",
               verbose = FALSE)
```

`interpret_model` *Model Interpretability Analysis*

Description

Computes variable importance and Shapley (SHAP) values for a fitted workflow returned by `evaluate_model()`.

Usage

```
interpret_model(
  evaluation_result,
  train_data,
  test_data,
  target,
  n_shap = 100,
  verbose = TRUE
)
```

Arguments

`evaluation_result`	Result of `evaluate_model()`; it must contain the element `final_fit` (a fitted `workflow`).
`train_data`	Training data frame (used as background data for SHAP).
`test_data`	Test data frame (used to compute importance and SHAP).
`target`	Name of the outcome variable (character).
`n_shap`	Number of test observations used to compute SHAP values.
`verbose`	Logical. If `TRUE`, progress is printed to the console.

Value

A list of class `supervisedml_interpret` with elements `importance` (data frame returned by `get_importance()`) and `shap` (list returned by `calculate_shap()`, or `NULL` if SHAP values could not be computed).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
idx <- sample(n, 150)
rec <- recipes::recipe(y ~ ., data = df[idx, ])
wf <- workflows::workflow() |>
  workflows::add_recipe(rec) |>
  workflows::add_model(parsnip::logistic_reg() |> parsnip::set_engine("glm"))
fit <- parsnip::fit(wf, data = df[idx, ])
res <- interpret_model(list(final_fit = fit), df[idx, ], df[-idx, ], "y",
                       n_shap = 20, verbose = FALSE)
res$importance
```

---

launch_report_generator

Launch the Shiny Report Generator

Description

Opens the Shiny application that generates scientific reports from JSON files exported with export_verbose_json(). Requires the shiny package.

Usage

```
launch_report_generator(launch.browser = TRUE)
```

Arguments

launch.browser Logical (default TRUE). Open the app in the web browser.

Value

Called for its side effect (runs the Shiny app); returns the value returned by `shiny::runApp()` when the app is closed.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_supervisedml_app()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
if (interactive()) {
  launch_report_generator()
}
```

`launch_supervisedml_app`

Launch the Integrated PsyML Shiny App

Description

Opens the integrated Shiny application that combines the machine learning pipeline, the sample size estimation and the AI report generator. Requires the shiny package.

Usage

```
launch_supervisedml_app(launch.browser = TRUE)
```

Arguments

`launch.browser` Logical (default TRUE). Open the app in the web browser.

Value

Called for its side effect (runs the Shiny app); returns the value returned by `shiny::runApp()` when the app is closed.

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `supervised_ml_capture()`, `supervised_ml_export()`

Examples

```
if (interactive()) {
  launch_supervisedml_app()
}
```

`ml_sample_size` *Sample Size Estimation for Machine Learning via Monte Carlo Simulation*

Description

Estimates the minimum sample size needed to achieve a target performance in machine learning models using Monte Carlo simulation and learning curves. It plays a role similar to G*Power, but for machine learning.

Usage

```
ml_sample_size(
  data = NULL,
  formula = NULL,
  task = c("classification", "regression"),
  model = c("rf"),
  metric = NULL,
  n_grid = c(50, 100, 150, 200, 300, 400, 600, 800, 1000, 1500, 2000),
  reps = 50,
  target = NULL,
  target_dir = c("higher_better", "lower_better"),
  prob_min = 0.8,
  sd_max = NULL,
  n_test = 5000,
  n_outer_splits = 1,
  bootstrap_ci = FALSE,
  n_boot = 1000,
  sampling = c("stratified", "none", "up", "down"),
  positive = "1",
  threshold = 0.5,
  sim_params = list(),
  model_params = list(),
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with real data. If NULL (default), a synthetic population is simulated using `sim_params`.
`formula`	Model formula (e.g., `y ~ .`). Required when `data` is supplied.
`task`	Task type: `"classification"` (default) or `"regression"`.
`model`	Model(s) to use: `"rf"` (random forest, default), `"xgboost"`, `"svm"` or `"glm"`. Can be a vector to compare several models.

`metric` Evaluation metric (yardstick naming convention). If NULL (default), `"roc_auc"` is used for classification and `"rsq"` for regression. Classification: `"roc_auc"`, `"accuracy"`, `"f_meas"`, `"mcc"`, `"kap"`, `"bal_accuracy"`. Regression: `"rsq"`, `"rmse"`, `"mae"`. The aliases `auc`, `f1`, `kappa`, `balanced_accuracy` and `r2` are accepted.

`n_grid` Numeric vector with the sample sizes to evaluate (default `c(50, 100, 150, 200, 300, 400, 600, 800, 1000, 1500, 2000)`). Values larger than the available training pool are dropped.

`reps` Number of Monte Carlo repetitions per sample size (default 50). Recommendations: R = 50 for exploratory analysis (SE about 0.057), R = 100 for standard use (SE about 0.040), R >= 200 for publication (SE about 0.028).

`target` Target metric value (e.g., 0.80 for AUC >= 0.80). If NULL (default), no sample size is recommended and only the learning curve is estimated.

`target_dir` Direction of the target: `"higher_better"` or `"lower_better"`. If not supplied, it is set automatically (`"lower_better"` for `rmse` and `mae`, `"higher_better"` otherwise).

`prob_min` Minimum proportion of repetitions that must meet the target (default 0.80). This is the probability-of-success criterion p_k.

`sd_max` Optional maximum SD of the metric (stability criterion). NULL (default) disables the criterion.

`n_test` Size of the fixed test set (default 5000). If it is at least 50% of the available data, it is reduced to 20% of the data with a warning.

`n_outer_splits` Number of independent test partitions used to assess the robustness of n* (default 1). It is set to 5 automatically when the data have fewer than 1000 rows and it is left at 1.

`bootstrap_ci` Logical (default FALSE). If TRUE and `target` is given, a bootstrap 95% CI for n* is computed (single partition only).

`n_boot` Number of bootstrap resamples (default 1000).

`sampling` Sampling type used to draw training samples in classification: `"stratified"` (default), `"none"`, `"up"` or `"down"`.

`positive` Label of the positive class (default `"1"`).

`threshold` Classification threshold (default 0.5).

`sim_params` List with simulation parameters, used when `data` is NULL:

- N_pop: Population size to simulate (default 30000).
- p: Number of predictors (default 20).
- prevalence: Positive class prevalence for binary outcomes (default 0.20).
- signal: Signal strength (default 2.5). It determines the maximum achievable AUC: signal = 1.5 gives AUC about 0.70-0.75, signal = 2.5 about 0.85-0.90 and signal = 3.5 about 0.92-0.95.
- noise_sd: Noise SD for regression (default 1.0).
- n_classes: Number of classes for classification (default 2). With 3 or more classes a multinomial logit (softmax) simulation is used.

- class_probs: Numeric vector of class probabilities for multiclass outcomes (default: equal probabilities 1/K). Ignored for binary outcomes.

`model_params` List with model hyperparameters:

- num.trees: Number of trees for random forest (default 500).
- mtry: Number of variables considered at each split.
- min.node.size: Minimum node size (default 5).
- max.depth: Maximum depth (default 0, no limit).

`seed` Optional integer seed. If NULL (default) the random number generator state is not modified.

`verbose` Logical (default TRUE). If TRUE, progress and results are reported with `message()`.

Details

The function implements the methodology described in Ventura-Leon (2026) for sample size estimation in machine learning. Key features:

Decision criteria: n* is the smallest n where: (1) mean performance >= target, (2) P(metric >= target) >= prob_min, (3) SD <= sd_max (optional).

SE for p_k: The standard error of the success proportion is SE(p_k) = sqrt(p_k * (1 - p_k) / R). Use R >= 100 for reliable estimates.

Multiple partitions: When N < 1000, using n_outer_splits > 1 is recommended to assess variability in n* estimates.

The power-law learning curve is $metric(n) = a - bn^{-c}$ (or $a + bn^{-c}$ when lower values are better), where a is the asymptote. See `estimate_n()` and `estimate_n_plateau()` to invert the fitted curve, and `plot_learning_curve()` and `plot_distribution()` to display the results.

Value

An object of class `c("ml_sample_size", "list")` with:

- raw: Data frame with one row per repetition (`model`, `n`, `rep`, `metric_value` and, if `target` is given, `meets_target`).
- summary: Data frame by model and sample size with `mean`, `sd`, `se`, `p_meet` (proportion of repetitions meeting the target) and `se_p_meet` (its binomial standard error).
- recommend_n: Recommended sample size n* (a number, or a named list with one value per model); NA if no evaluated size meets the criteria. With several partitions it is the median n* across partitions.
- recommend_n_ci: Bootstrap 95% CI for n* (list with `ci_lower`, `ci_upper`, `ci_median`, `n_valid`), or NULL.
- curve_fit: Fitted power-law `nls` model (or a list of them per model), NULL if the fit failed.
- curve_diagnostics: Fit diagnostics (`pseudo_r2`, `rmse_fit`, `residuals`, `fit_quality`).
- outer_splits_results: Summary of n* across partitions (`median`, `mean`, `sd`, `cv`, `range`, `n_valid`, `all_values`) when `n_outer_splits` > 1, otherwise NULL.
- settings: List with the configuration used.

References

Ventura-Leon, J. (2026). Sample Size Estimation for Machine Learning via Monte Carlo Simulation: Learning Curves and Power Laws.

See Also

Other study design: block_split(), diversity_split(), estimate_n(), estimate_n_plateau(), plot_all.ml_sample_size(), plot_distribution(), plot_learning_curve()

Examples

```
# Simulated population (classification), small settings for speed
result <- ml_sample_size(
  task = "classification",
  model = "glm",
  metric = "roc_auc",
  n_grid = c(50, 100, 200, 400),
  reps = 5,
  target = 0.75,
  n_test = 500,
  sim_params = list(N_pop = 2000, p = 5, prevalence = 0.30),
  seed = 1,
  verbose = FALSE
)
result

# View recommendation
result$recommend_n

# Plot learning curve
plot_learning_curve(result)

# Small real dataset (N < 1000): several partitions and stability check
set.seed(2)
df <- data.frame(x1 = rnorm(400), x2 = rnorm(400))
df$y <- factor(ifelse(df$x1 + rnorm(400) > 0, "1", "0"))
res_real <- ml_sample_size(
  data = df,
  formula = y ~ .,
  task = "classification",
  model = "glm",
  n_grid = c(40, 80),
  reps = 3,
  target = 0.75,
  n_test = 100,
  n_outer_splits = 2,
  seed = 1,
  verbose = FALSE
)
res_real$outer_splits_results
```

`optimize_threshold` *Optimize the Classification Threshold*

Description

Finds the optimal probability threshold for binary classification by evaluating thresholds from 0.01 to 0.99 (step 0.01) under one of several criteria: Youden's J, F1, a user-defined misclassification cost, or balanced accuracy. The positive class is detected from the levels of the target (recognized labels such as "yes", "1" or "positive"; otherwise the second level).

Usage

```
optimize_threshold(
  predictions,
  target,
  method = "youden",
  cost_fp = 1,
  cost_fn = 1,
  verbose = TRUE
)
```

Arguments

`predictions`	Data frame of predictions with the observed target (a factor with two levels) and a probability column named `.pred_<positive class>`, such as the `predictions` element of a `supervised_ml()` result.
`target`	Name of the target variable.
`method`	Optimization criterion: "youden" (default; maximizes sensitivity + specificity - 1), "f1" (maximizes F1), "cost" (minimizes `cost_fp * FP + cost_fn * FN`) or "balanced" (maximizes balanced accuracy). Unknown values fall back to "youden".
`cost_fp`	Cost of a false positive (used when `method = "cost"`; default: 1).
`cost_fn`	Cost of a false negative (used when `method = "cost"`; default: 1).
`verbose`	Logical. Print results (default: TRUE).

Value

A list with `optimal_threshold` (numeric), `optimal_metrics` (one-row data frame with the metrics at the optimal threshold), `all_results` (data frame with columns `threshold`, `sensitivity`, `specificity`, `precision`, `f1`, `youden_j`, `balanced_acc`, `total_cost`, `tp`, `tn`, `fp` and `fn` for every threshold) and `method`.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
x <- rnorm(n)
y <- factor(ifelse(x + rnorm(n) > 0.8, "yes", "no"), levels = c("no", "yes"))
preds <- data.frame(y = y, .pred_yes = plogis(1.5 * x - 1))
thr <- optimize_threshold(preds, target = "y", method = "f1", verbose = FALSE)
thr$optimal_threshold
```

---

plot.causalml *Plot a causalml Object*

Description

Returns the figures stored in a causalml object.

Usage

```
## S3 method for class 'causalml'
plot(x, type = "all", ...)
```

Arguments

x	An object of class causalml, as returned by causal_ml().
type	Plot type: "all" (default) or one of "treatment", "outcome", "ps", "overlap", "love", "cate", "heterogeneity", "importance", "ate_forest", "meta", "evalue" and "balance". The internal name of a figure (see names(x$figures)) is also accepted.
...	Further arguments (not used).

Value

For `type = "all"`, the named list of 'ggplot2' objects in `x$figures`; otherwise the requested 'ggplot2' object (printed when called at top level). If no figure is available or the type is not found, a message is issued and `NULL` is returned invisibly.

See Also

Other causal inference: `causal_ml()`, `predict.causalml()`, `print.causalml()`, `summary.causalml()`

Examples

```
set.seed(1)
n <- 300
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$treat <- rbinom(n, 1, plogis(df$x1))
df$y <- 1.5 * df$treat + df$x1 + 0.5 * df$x2 + rnorm(n)
res <- causal_ml(df, treatment = "treat", outcome = "y",
                 methods = c("propensity", "meta_learners"),
                 ps_methods = c("ipw", "doubly_robust"),
                 meta_learners = "T", run_eda = FALSE,
                 seed = 1, verbose = FALSE)
plot(res, type = "ate_forest")
```

`plot.supervisedml` *Plot a supervisedml Object*

Description

Returns the plots of a `supervised_ml()` result: a combined summary panel (default), the list of all plots, or a single plot.

Usage

```
## S3 method for class 'supervisedml'
plot(x, type = "panel", top_n = 15, ...)
```

Arguments

`x`	An object of class `supervisedml` returned by `supervised_ml()`.
`type`	Type of plot: `"panel"` (default; four-plot summary panel, requires **patchwork**), `"all"` (list with all plots) or a single plot: `"importance"`, `"metrics"` (or `"comparison"`), `"roc"`, `"confusion"`, `"calibration"`, `"calibration_curve"`, `"threshold"`, `"pred_vs_obs"`, `"residuals"`, `"residual_diagnostics"`, `"tuning"`, `"shap"` (or `"shap_beeswarm"`), `"shap_bar"`, `"fairness"` or `"dca"`. Availability depends on the task and on the analyses run.
`top_n`	Number of variables shown in the importance plot when it has to be regenerated (default: 15).
`...`	Additional arguments (not used).

Value

A ggplot (or patchwork) object, a named list of ggplot objects when type = "all", or NULL (with a message) when the requested plot is not available. The plot is drawn when the returned object is printed.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
plot(res, type = "roc")
plot(res, type = "confusion")
names(plot(res, type = "all"))
if (requireNamespace("patchwork", quietly = TRUE)) plot(res)
```

---

plot.unsupervisedml *Plot an unsupervisedml Object*

Description

Displays the figures stored in an unsupervisedml object.

Usage

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

Arguments

x	An object of class unsupervisedml, as returned by unsupervised_ml().
which	Figures to display: a numeric vector of indices, a character vector of figure names (see names(x$figures)), or "all". If NULL (default), the catalogue of available figures is listed with message().
...	Further arguments (not used).

Value

Invisibly returns x.

See Also

Other unsupervised learning: cluster_profile(), predict.unsupervisedml(), print.unsupervisedml(), save_unsupervised_plots(), summary.unsupervisedml(), unsupervised_ml()

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X,
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k_range = 2:4, reduction_methods = "pca",
                       anomaly_methods = "mahalanobis",
                       run_eda = FALSE, seed = 1, verbose = FALSE)
plot(res)
plot(res, which = 1)
```

---

plot_all *Plot a Summary Panel of All Results*

Description

Generic function that combines the main plots of a result into a single panel. The supervisedml method arranges four plots with **patchwork**: variable importance, model comparison and, for classification, confusion matrix and ROC curve or, for regression, predicted vs observed values and residuals.

Usage

```
plot_all(x, ...)

## S3 method for class 'supervisedml'
plot_all(x, ...)
```

Arguments

x	An object with a plot_all() method, such as a supervisedml object returned by supervised_ml().
...	Additional arguments passed to methods (not used by the supervisedml method).

Value

A patchwork object (drawn when printed).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
if (requireNamespace("patchwork", quietly = TRUE)) plot_all(res)
```

---

plot_all.ml_sample_size

Plot All Graphs for Sample Size Results

Description

Displays the learning curve (plot_learning_curve()) and the metric distribution (plot_distribution()) side by side (using patchwork when it is installed; otherwise the plots are printed one after the other).

Usage

```
## S3 method for class 'ml_sample_size'
plot_all(x, ...)
```

Arguments

x	An object of class ml_sample_size (single model).
...	Additional arguments (not used).

Value

Invisibly returns a list with the ggplot objects learning_curve and distribution.

See Also

Other study design: block_split(), diversity_split(), estimate_n(), estimate_n_plateau(), ml_sample_size(), plot_distribution(), plot_learning_curve()

Examples

```
result <- ml_sample_size(model = "glm", n_grid = c(50, 100, 200), reps = 5,
                         target = 0.75, n_test = 500,
                         sim_params = list(N_pop = 2000, p = 5),
                         seed = 1, verbose = FALSE)
plot_all(result)
```

---

plot_calibration_curve

Plot the Calibration Curve

Description

Reliability diagram comparing the mean predicted probability with the observed proportion of the positive class per probability bin, before (red) and after (blue) calibration.

Usage

```
plot_calibration_curve(calibration_result, n_bins = 10)
```

Arguments

calibration_result	A list returned by calibrate_probabilities().
n_bins	Number of probability bins (default: 10).

Value

A ggplot object.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
x <- rnorm(300)
y <- factor(ifelse(x + rnorm(300) > 0, "yes", "no"), levels = c("no", "yes"))
preds <- data.frame(y = y, .pred_yes = plogis(3 * x))
cal <- calibrate_probabilities(preds, target = "y", verbose = FALSE)
plot_calibration_curve(cal)
```

---

plot_distribution *Plot the Metric Distribution by Sample Size*

Description

Shows boxplots of the Monte Carlo distribution of the metric for each sample size of an ml_sample_size() result, with the target as a dashed line.

Usage

```
plot_distribution(x, ...)
```

Arguments

x	An object of class ml_sample_size.
...	Additional arguments (not used).

Value

If ggplot2 is available (it is a dependency), the plot is printed and the ggplot object is returned invisibly. Otherwise a base R boxplot is drawn and x is returned invisibly.

See Also

Other study design: block_split(), diversity_split(), estimate_n(), estimate_n_plateau(), ml_sample_size(), plot_all.ml_sample_size(), plot_learning_curve()

Examples

```
result <- ml_sample_size(model = "glm", n_grid = c(50, 100, 200), reps = 5,
                         target = 0.75, n_test = 500,
                         sim_params = list(N_pop = 2000, p = 5),
                         seed = 1, verbose = FALSE)
plot_distribution(result)
```

---

plot_importance *Plot Variable Importance*

Description

Horizontal bar chart of the most important variables.

Usage

```
plot_importance(interpret_result, top_n = 15)
```

Arguments

interpret_result	A list with an importance element (data frame with columns Variable and Importance, sorted by decreasing importance), such as the result of interpret_model().
top_n	Maximum number of variables shown (default: 15).

Value

A ggplot object, or NULL (with a message) when no importance data are available.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(),

get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
df <- data.frame(x1 = rnorm(100), x2 = rnorm(100), x3 = rnorm(100))
df$y <- 2 * df$x1 - df$x2 + rnorm(100)
wf <- workflows::workflow() |>
  workflows::add_formula(y ~ .) |>
  workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
fit <- parsnip::fit(wf, data = df)
interp <- list(importance = get_importance(fit, verbose = FALSE))
plot_importance(interp)
```

---

plot_learning_curve *Plot the Learning Curve of a Sample Size Analysis*

Description

Plots the learning curve of an ml_sample_size() result, showing how model performance changes with sample size, together with the fitted power-law curve, its confidence band, the target and the recommended n*.

Usage

```
plot_learning_curve(
  x,
  show_target = TRUE,
  show_fit = TRUE,
  show_ci = TRUE,
  ci_level = 0.95,
  title = NULL,
  ...
)
```

Arguments

x	An object of class ml_sample_size.
show_target	Logical (default TRUE). Draw a horizontal line at the target.
show_fit	Logical (default TRUE). Draw the fitted power-law curve.

show_ci	Logical (default TRUE). Draw the confidence band of the mean.
ci_level	Confidence level of the band (default 0.95).
title	Plot title. If NULL (default), it is built automatically.
...	Additional arguments (not used).

Value

If ggplot2 is available (it is a dependency), the plot is printed and the ggplot object is returned invisibly. Otherwise a base R plot is drawn and x is returned invisibly.

See Also

Other study design: block_split(), diversity_split(), estimate_n(), estimate_n_plateau(), ml_sample_size(), plot_all.ml_sample_size(), plot_distribution()

Examples

```
result <- ml_sample_size(
  task = "classification",
  model = "glm",
  metric = "accuracy",
  n_grid = c(50, 100, 200, 400),
  reps = 5,
  target = 0.75,
  n_test = 500,
  sim_params = list(N_pop = 2000, p = 5),
  seed = 1,
  verbose = FALSE
)
plot_learning_curve(result)
```

---

plot_metrics_profile *Plot the Profile of Performance Metrics*

Description

Draws a point-and-line plot with the profile of all cross-validated performance metrics of each model. Each model is a line connecting its values across metrics, which makes it easy to compare strengths and weaknesses.

Usage

```
plot_metrics_profile(
  x,
  metrics = NULL,
  highlight = NULL,
  size_point = 3.5,
```

```
  size_line = 0.9,
  show_values = TRUE
)
```

Arguments

x	A supervisedml object (its cv_summary element is used) or a data frame with a model column and numeric metric columns.
metrics	Optional character vector with the metrics to include. If NULL (default), all numeric columns are used.
highlight	Optional name of the model to highlight. If NULL (default) and x is a supervisedml object, the best model is highlighted.
size_point	Numeric. Size of the points (default: 3.5).
size_line	Numeric. Width of the lines (default: 0.9).
show_values	Logical. Show the numeric values above the points (default: TRUE).

Value

A ggplot object.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
cv <- data.frame(model = c("glm", "rf", "xgboost"),
                 roc_auc = c(0.81, 0.86, 0.84),
                 accuracy = c(0.74, 0.79, 0.78),
                 mcc = c(0.48, 0.57, 0.55))
plot_metrics_profile(cv, highlight = "rf")
plot_metrics_profile(cv, metrics = c("roc_auc", "mcc"), show_values = FALSE)

set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
```

```
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
plot_metrics_profile(res)
```

`plot_residual_diagnostics` *Residual Diagnostic Plots*

Description

Residuals versus predictions, normal Q-Q plot, histogram of residuals and scale-location plot.

Usage

```
plot_residual_diagnostics(residual_analysis)
```

Arguments

`residual_analysis`	A list returned by `advanced_residual_analysis()`.

Value

If the patchwork package is installed, a patchwork object combining the four plots; otherwise a list of four ggplot objects (`residuals_vs_fitted`, `qq_plot`, `histogram`, `scale_location`).

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
obs <- rnorm(150, 10, 2)
preds <- data.frame(y = obs, .pred = obs + rnorm(150))
ra <- advanced_residual_analysis(preds, target = "y", verbose = FALSE)
p <- plot_residual_diagnostics(ra)
```

`plot_shap_bar` *Plot SHAP Feature Importance*

Description

Bar chart of the mean absolute SHAP value of the most important variables, drawn with **shapviz** when it is installed (otherwise with **ggplot2**).

Usage

```
plot_shap_bar(interpret_result, top_n = 15)
```

Arguments

`interpret_result`	A list with a `shap` element as returned by `calculate_shap()`, such as the result of `interpret_model()` or the `interpretation` element of a `supervised_ml()` result.
`top_n`	Maximum number of variables shown (default: 15).

Value

A `ggplot` object, or NULL (with a message) when no SHAP values are available.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
if (requireNamespace("kernelshap", quietly = TRUE)) {
  set.seed(1)
  df <- data.frame(x1 = rnorm(120), x2 = rnorm(120), x3 = rnorm(120))
  df$y <- df$x1 - df$x2 + rnorm(120)
  wf <- workflows::workflow() |>
    workflows::add_formula(y ~ .) |>
    workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
  fit <- parsnip::fit(wf, data = df[1:100, ])
  sh <- calculate_shap(fit, df[1:100, ], df[101:120, ], "y",
                       n_samples = 20, bg_n = 30, verbose = FALSE)
  interp <- list(shap = sh)
  plot_shap_bar(interp)
}
```

---

plot_shap_dependence *Plot SHAP Dependence for One Variable*

Description

Scatter plot of the SHAP values of one variable against its observed values in the explained sample, with a loess smoother.

Usage

```
plot_shap_dependence(interpret_result, variable, test_data)
```

Arguments

`interpret_result`	A list with a `shap` element as returned by `calculate_shap()`, such as the result of `interpret_model()` or the `interpretation` element of a `supervised_ml()` result.
`variable`	Character string with the name of the variable to plot.
`test_data`	Not used; the explained sample stored in `interpret_result$shap$test_sample` is plotted. Kept for backward compatibility.

Value

A `ggplot` object, or NULL (with a message) when no SHAP values are available or `variable` is not among them.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
if (requireNamespace("kernelshap", quietly = TRUE)) {
  set.seed(1)
  df <- data.frame(x1 = rnorm(120), x2 = rnorm(120), x3 = rnorm(120))
  df$y <- df$x1 - df$x2 + rnorm(120)
  wf <- workflows::workflow() |>
    workflows::add_formula(y ~ .) |>
    workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
  fit <- parsnip::fit(wf, data = df[1:100, ])
  sh <- calculate_shap(fit, df[1:100, ], df[101:120, ], "y",
                       n_samples = 20, bg_n = 30, verbose = FALSE)
  interp <- list(shap = sh)
  plot_shap_dependence(interp, variable = "x1", test_data = df[101:120, ])
}
```

---

plot_shap_summary *Plot a SHAP Summary (Beeswarm)*

Description

Beeswarm plot of the SHAP values of the most important variables, colored by the feature value, drawn with **shapviz** when it is installed. Otherwise a boxplot of the SHAP values per variable is drawn.

Usage

```
plot_shap_summary(interpret_result, top_n = 15)
```

Arguments

interpret_result	A list with a shap element as returned by calculate_shap(), such as the result of interpret_model() or the interpretation element of a supervised_ml() result.
top_n	Maximum number of variables shown (default: 15).

Value

A ggplot object, or NULL (with a message) when no SHAP values are available.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
if (requireNamespace("kernelshap", quietly = TRUE)) {
  set.seed(1)
  df <- data.frame(x1 = rnorm(120), x2 = rnorm(120), x3 = rnorm(120))
  df$y <- df$x1 - df$x2 + rnorm(120)
  wf <- workflows::workflow() |>
    workflows::add_formula(y ~ .) |>
    workflows::add_model(parsnip::linear_reg() |> parsnip::set_engine("lm"))
  fit <- parsnip::fit(wf, data = df[1:100, ])
  sh <- calculate_shap(fit, df[1:100, ], df[101:120, ], "y",
                       n_samples = 20, bg_n = 30, verbose = FALSE)
  interp <- list(shap = sh)
  plot_shap_summary(interp)
}
```

plot_threshold_optimization *Plot the Threshold Optimization*

Description

Plots sensitivity, specificity, F1 and precision as a function of the threshold, marking the optimal threshold and 0.5.

Usage

```
plot_threshold_optimization(threshold_result)
```

Arguments

threshold_result
: A list returned by optimize_threshold().

Value

A ggplot object.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
x <- rnorm(200)
y <- factor(ifelse(x + rnorm(200) > 0.8, "yes", "no"), levels = c("no", "yes"))
preds <- data.frame(y = y, .pred_yes = plogis(1.5 * x - 1))
thr <- optimize_threshold(preds, target = "y", verbose = FALSE)
plot_threshold_optimization(thr)
```

plot_tuning *Plot the Hyperparameter Tuning Results*

Description

Plots the cross-validated performance of the evaluated hyperparameter combinations with tune::autoplot().

Usage

```
plot_tuning(tune_result, metric = NULL)
```

Arguments

tune_result	Result of tune_best_model() (or the tuning element of a supervised_ml() result).
metric	Optional character vector with the metrics to plot. If NULL (default) all computed metrics are shown.

Value

A ggplot object, or NULL (with a message) when no tuning was performed.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
plot_tuning(res$tuning)  # NULL: no tuning was performed
```

```
res_rf <- supervised_ml(df, target = "y", models = "rf", cv_folds = 3,
                        tune_grid = 3, run_shap = FALSE, n_cores = 1,
                        verbose = FALSE)
plot_tuning(res_rf$tuning, metric = "roc_auc")
```

`predict.causalml` *Predict Conditional Average Treatment Effects from a causalml Object*

Description

Returns conditional average treatment effect (CATE) estimates, either for the original data or for new observations.

Usage

```
## S3 method for class 'causalml'
predict(object, new_data = NULL, method = "best", ...)
```

Arguments

`object`	An object of class `causalml`, as returned by `causal_ml()`.
`new_data`	A data frame with new observations containing the covariates used in the analysis. If NULL (default), the CATE estimates of the original data are returned (from the causal forest or, if not available, the X-, T- or S-learner).
`method`	Method used for prediction on new data: `"best"` (default; causal forest if available, otherwise T-learner), `"causal_forest"` or `"T"`.
`...`	Further arguments (not used).

Value

A numeric vector of CATE estimates, one per observation.

See Also

Other causal inference: `causal_ml()`, `plot.causalml()`, `print.causalml()`, `summary.causalml()`

Examples

```
set.seed(1)
n <- 300
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$treat <- rbinom(n, 1, plogis(df$x1))
df$y <- 1.5 * df$treat + df$x1 + 0.5 * df$x2 + rnorm(n)
res <- causal_ml(df, treatment = "treat", outcome = "y",
                 methods = c("propensity", "meta_learners"),
```

```
                ps_methods = c("ipw", "doubly_robust"),
                meta_learners = "T", run_eda = FALSE,
                seed = 1, verbose = FALSE)
head(predict(res))
# New data must be on the same (normalized) scale as the covariates
predict(res, new_data = res$preprocessed$data[1:5, ], method = "T")
```

`predict.supervisedml` *Predict with a supervisedml Model*

Description

Generates predictions for new data with the final model of a `supervised_ml()` result. When the model was fitted with `fairness_correction = "postprocesamiento"` and `new_data` contains the protected variable, class predictions use the group-specific thresholds (observations of unknown groups use 0.5).

Usage

```
## S3 method for class 'supervisedml'
predict(object, new_data, type = "class", ...)
```

Arguments

`object`	An object of class `supervisedml`.
`new_data`	A data frame with the same predictors used for training.
`type`	Type of prediction: `"class"` (default; predicted class for classification, numeric prediction for regression) or `"prob"` (class probabilities, classification only).
`...`	Additional arguments (not used).

Value

A tibble with one row per row of `new_data`: column `.pred_class` (factor) for class predictions, `.pred` (numeric) for regression, or one `.pred_<level>` column per class with the probabilities when `type = "prob"`.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`,

plot_tuning(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
new_df <- data.frame(x1 = rnorm(5), x2 = rnorm(5), x3 = rnorm(5))
predict(res, new_df)
predict(res, new_df, type = "prob")
```

---

predict.unsupervisedml

Assign New Observations to Clusters

Description

Assigns new observations to the clusters of an unsupervisedml object. The new data are normalized with the parameters of the original analysis and assigned to the nearest centroid (k-means and hierarchical), with the fitted mixture model (GMM, requires 'mclust') or with the posterior class probabilities (LCA, requires 'poLCA').

Usage

```
## S3 method for class 'unsupervisedml'
predict(object, newdata, algorithm = NULL, ...)
```

Arguments

object	An object of class unsupervisedml, as returned by unsupervised_ml().
newdata	A data frame with the new observations; it must contain the variables used in the analysis (names(object$preprocessed$data_clean)), on their original scale.
algorithm	Algorithm used for the assignment: "kmeans", "hierarchical", "gmm" or "lca". If NULL (default), the best algorithm of the evaluation (or the first one available) is used.
...	Further arguments (not used).

Value

An integer vector with the cluster label of each row of `newdata`.

See Also

Other unsupervised learning: `cluster_profile()`, `plot.unsupervisedml()`, `print.unsupervisedml()`, `save_unsupervised_plots()`, `summary.unsupervisedml()`, `unsupervised_ml()`

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X, methods = "clustering",
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k = 2, run_eda = FALSE, seed = 1, verbose = FALSE)
new_obs <- data.frame(a = c(0, 3), b = c(0, 3), c = c(0, 0))
predict(res, new_obs, algorithm = "kmeans")
```

---

prep_feature_selection

Feature Selection with Boruta

Description

Runs the Boruta all-relevant feature selection algorithm (maximum 100 runs) on complete cases and keeps the confirmed and tentative variables. If the Boruta package is not installed, the step is skipped.

Usage

```
prep_feature_selection(data, target, seed = NULL, verbose = TRUE)
```

Arguments

`data`	Data frame (usually the training data).
`target`	Name of the target variable.
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.
`verbose`	Logical. Print results (default: TRUE).

Value

A list with `selected` (names of the confirmed and tentative variables), `boruta` (the `Boruta` object) and `importance` (data frame from `Boruta::attStats()` with a `Variable` column). If Boruta is not available, `selected` and `boruta` are NULL.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
if (requireNamespace("Boruta", quietly = TRUE)) {
  fs <- prep_feature_selection(iris, "Species", seed = 1, verbose = FALSE)
  fs$selected
}
```

---

prep_recipe *Create the Preprocessing Recipe*

Description

Defines a recipes recipe with, in this order: imputation, winsorization of outliers, removal of highly correlated and high-VIF numeric predictors, pooling of rare categories, dummy encoding, removal of zero-variance predictors, normalization, PCA and class balancing (each step optional). Thresholds and winsorization limits are computed on data.

Usage

```
prep_recipe(
  data,
  target,
  task,
  impute = TRUE,
  impute_method = c("knn", "median", "mean"),
  normalize = FALSE,
  normalize_method = c("zscore", "minmax"),
  use_pca = FALSE,
  pca_threshold = 0.95,
  balance_classes = FALSE,
  balance_method = c("smote", "adasyn", "rose", "up", "down"),
  treat_outliers = TRUE,
```

```
  outlier_percentile = 0.05,
  remove_high_cor = TRUE,
  cor_threshold = 0.9,
  remove_high_vif = TRUE,
  vif_threshold = 5,
  group_rare = TRUE,
  other_threshold = 0.05,
  remove_zv = TRUE,
  verbose = TRUE
)
```

Arguments

`data`	Data frame (usually the training data).
`target`	Name of the target variable.
`task`	Type of task: "classification" or "regression".
`impute`	Logical. Impute missing values (default: TRUE).
`impute_method`	Imputation method for numeric predictors: "knn" (default), "median" or "mean". Nominal predictors are imputed with the mode.
`normalize`	Logical. Normalize numeric predictors (default: FALSE).
`normalize_method`	Normalization method: "zscore" (default) or "minmax" (rescaling between 0 and 1).
`use_pca`	Logical. Add a PCA step (default: FALSE).
`pca_threshold`	Proportion of variance retained by the PCA step (default: 0.95).
`balance_classes`	Logical. Balance classes for classification (default: FALSE; requires the themis package).
`balance_method`	Balancing method: "smote" (default), "adasyn", "rose", "up" or "down".
`treat_outliers`	Logical. Winsorize numeric predictors with outliers (1.5 x IQR rule) (default: TRUE).
`outlier_percentile`	Winsorization percentile (default: 0.05, i.e. 5th and 95th percentiles).
`remove_high_cor`	Logical. Remove highly correlated predictors (default: TRUE).
`cor_threshold`	Absolute correlation threshold (default: 0.90).
`remove_high_vif`	Logical. Iteratively remove predictors with high VIF (default: TRUE).
`vif_threshold`	VIF threshold (default: 5).
`group_rare`	Logical. Pool rare categories (default: TRUE).
`other_threshold`	Minimum category proportion for pooling (default: 0.05).
`remove_zv`	Logical. Remove zero-variance predictors (default: TRUE).
`verbose`	Logical. Print the configured steps and treatments (default: TRUE).

Value

An untrained `recipe` object (class `recipe`) with an attribute `"treatments_applied"`: a list with elements `outliers`, `correlation` and/or `vif` describing the variables treated or removed.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
df <- data.frame(x1 = rnorm(80), x2 = rnorm(80))
df$x3 <- df$x1 + rnorm(80, sd = 0.05)
df$y <- df$x1 + df$x2 + rnorm(80)
rec <- prep_recipe(df, target = "y", task = "regression", verbose = FALSE)
attr(rec, "treatments_applied")$correlation$vars_removed
```

---

prep_split *Train/Test Split*

Description

Stratified train/test split with `rsample::initial_split()` (strata by class for classification and by quartiles for regression).

Usage

```
prep_split(data, target, task, test_split = 0.2, seed = NULL, verbose = TRUE)
```

Arguments

`data`	Data frame.
`target`	Name of the target variable (used for stratification).
`task`	Type of task (kept for compatibility; stratification is always applied).
`test_split`	Proportion of observations assigned to the test set (default: 0.20).

seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical. Print results (default: TRUE).

Value

A list with `train` and `test` (data frames) and `split` (the `rsample` split object).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
sp <- prep_split(iris, "Species", task = "classification", seed = 1,
                 verbose = FALSE)
table(sp$test$Species)
```

---

prep_target *Prepare the Target Variable*

Description

Converts character columns to factors and coerces the target to a factor (classification) or to numeric (regression).

Usage

```
prep_target(data, target, task, verbose = TRUE)
```

Arguments

data	Data frame.
target	Name of the target variable.
task	Type of task: "classification" or "regression".
verbose	Logical. Print results (default: TRUE).

Value

The data frame with character columns converted to factors and the target coerced to the type required by `task`.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
df <- data.frame(x = 1:6, y = c("a", "b", "a", "b", "a", "b"))
str(prep_target(df, "y", task = "classification", verbose = FALSE))
```

`preprocess_data` *Preprocessing Pipeline*

Description

Runs the complete preprocessing stage: preparation of the target, stratified train/test split, optional feature selection with Boruta and definition of a `recipes` preprocessing recipe. The recipe is only defined here (not trained); it is estimated later inside the resampling to avoid data leakage.

Usage

```
preprocess_data(
  data,
  target,
  task = "auto",
  test_split = 0.2,
  feature_selection = FALSE,
  balance_classes = FALSE,
  balance_method = c("smote", "adasyn", "rose", "up", "down"),
  impute = TRUE,
  impute_method = c("knn", "median", "mean"),
  normalize = FALSE,
```

```
  normalize_method = c("zscore", "minmax"),
  use_pca = FALSE,
  pca_threshold = 0.95,
  treat_outliers = TRUE,
  outlier_percentile = 0.05,
  remove_high_cor = TRUE,
  cor_threshold = 0.9,
  remove_high_vif = TRUE,
  vif_threshold = 5,
  group_rare = TRUE,
  other_threshold = 0.05,
  remove_zv = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the data.
`target`	Name of the target variable.
`task`	Type of task: "classification", "regression" or "auto" (default; detected from the target).
`test_split`	Proportion of observations assigned to the test set (default: 0.20).
`feature_selection`	Logical. Use Boruta for feature selection (default: FALSE; requires the Boruta package).
`balance_classes`	Logical. Balance the classes of a classification target (default: FALSE; requires the themis package).
`balance_method`	Balancing method: "smote" (default), "adasyn", "rose", "up" (random upsampling) or "down" (random downsampling).
`impute`	Logical. Impute missing values (default: TRUE).
`impute_method`	Imputation method for numeric predictors: "knn" (default), "median" or "mean". Nominal predictors are imputed with the mode.
`normalize`	Logical. Normalize numeric predictors (default: FALSE).
`normalize_method`	Normalization method: "zscore" (default) or "minmax" (rescaling between 0 and 1).
`use_pca`	Logical. Add a principal component analysis step (default: FALSE).
`pca_threshold`	Proportion of variance retained by the PCA step (default: 0.95).
`treat_outliers`	Logical. Treat outliers by winsorization (default: TRUE).
`outlier_percentile`	Percentile used for winsorization (default: 0.05, i.e. values are limited to the 5th and 95th percentiles).

remove_high_cor
: Logical. Remove predictors with high correlation (default: TRUE).

cor_threshold
: Absolute correlation threshold for removal (default: 0.90).

remove_high_vif
: Logical. Remove predictors with high VIF (default: TRUE).

vif_threshold
: VIF threshold for removal (default: 5).

group_rare
: Logical. Pool rare categories with `recipes::step_other()` (default: TRUE).

other_threshold
: Minimum proportion below which a category is pooled into "other" (default: 0.05).

remove_zv
: Logical. Remove zero-variance predictors (default: TRUE).

seed
: Optional integer seed. If NULL (default) the random number generator state is not modified.

verbose
: Logical. Print progress (default: TRUE).

Value

An object of class `c("supervisedml_preprocess", "list")` with elements `data` (data with the prepared target), `train_data` and `test_data` (data frames), `split` (the `rsample` split object), `selected_features` (character vector, or NULL without feature selection), `boruta_result` (only with feature selection), `recipe` (an untrained `recipes::recipe` object, see `prep_recipe()`), `task` and `target`.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 120
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n),
                 g = sample(c("a", "b", "c"), n, TRUE))
df$y <- factor(ifelse(df$x1 + rnorm(n) > 0, "yes", "no"))
pp <- preprocess_data(df, target = "y", seed = 1, verbose = FALSE)
dim(pp$train_data)
pp$recipe
```

print.block_split *Print a Block-Based Split*

Description

Prints the sizes of the training and test sets and the distribution of test cases across blocks.

Usage

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

Arguments

x	A block_split object returned by block_split().
...	Additional arguments (ignored).

Value

Invisibly returns x.

Examples

```
res <- block_split(iris, num_blocks = 3, test_prop = 0.2, seed = 1)
print(res)
```

print.causalml *Print a causalml Object*

Description

Prints the configuration of the analysis, the average treatment effect estimated by each method and the E-value.

Usage

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

Arguments

x	An object of class causalml, as returned by causal_ml().
...	Further arguments (not used).

Value

Invisibly returns `x`.

See Also

Other causal inference: `causal_ml()`, `plot.causalml()`, `predict.causalml()`, `summary.causalml()`

Examples

```
set.seed(1)
n <- 300
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$treat <- rbinom(n, 1, plogis(df$x1))
df$y <- 1.5 * df$treat + df$x1 + 0.5 * df$x2 + rnorm(n)
res <- causal_ml(df, treatment = "treat", outcome = "y",
                 methods = c("propensity", "meta_learners"),
                 ps_methods = c("ipw", "doubly_robust"),
                 meta_learners = "T", run_eda = FALSE,
                 seed = 1, verbose = FALSE)
print(res)
```

---

print.diversity_split *Print a Diversity-Based Split*

Description

Prints the sizes of the training and test sets and the diversity metrics.

Usage

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

Arguments

`x`	A `diversity_split` object returned by `diversity_split()`.
`...`	Additional arguments (ignored).

Value

Invisibly returns `x`.

Examples

```
res <- diversity_split(iris, target = "Species", method = "dissimilarity",
                       seed = 1, verbose = FALSE)
print(res)
```

print.ml_sample_size *Print ML Sample Size Results*

Description

Prints the settings, the recommended sample size n* (with its bootstrap CI and curve-fit quality when available) and the summary by sample size of an `ml_sample_size()` result.

Usage

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

Arguments

`x`	An object of class `ml_sample_size`.
`...`	Additional arguments (ignored).

Value

Invisibly returns `x`.

Examples

```
res <- ml_sample_size(model = "glm", n_grid = c(50, 100, 200), reps = 3,
                      target = 0.75, n_test = 300,
                      sim_params = list(N_pop = 1500, p = 5),
                      seed = 1, verbose = FALSE)
print(res)
```

print.supervisedml *Print a supervisedml Object*

Description

Prints a console report of a `supervised_ml()` result: configuration, best model and tuned parameters, test metrics, top five variables and, when available, fairness and decision curve results (console text is in Spanish).

Usage

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

Arguments

x	An object of class supervisedml.
...	Additional arguments (not used).

Value

Invisibly returns x. Called for its side effect of printing.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
print(res)
```

---

print.unsupervisedml *Print an unsupervisedml Object*

Description

Prints a short overview of the analysis: optimal k, clustering algorithms and best algorithm, dimensionality reduction and anomaly counts.

Usage

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

Arguments

x	An object of class unsupervisedml, as returned by unsupervised_ml().
...	Further arguments (not used).

Value

Invisibly returns x.

See Also

Other unsupervised learning: cluster_profile(), plot.unsupervisedml(), predict.unsupervisedml(), save_unsupervised_plots(), summary.unsupervisedml(), unsupervised_ml()

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X,
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k_range = 2:4, reduction_methods = "pca",
                       anomaly_methods = "mahalanobis",
                       run_eda = FALSE, seed = 1, verbose = FALSE)
print(res)
```

---

print_figures_catalog *Print the Catalog of Figures*

Description

Prints the catalog of figures returned by get_figures_catalog(): id, title, description, file name and how to access each figure (console text in Spanish).

Usage

```
print_figures_catalog(x)
```

Arguments

x	An object of class supervisedml returned by supervised_ml().

Value

Invisibly returns NULL. Called for its side effect of printing.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
print_figures_catalog(res)
```

---

resample_fairness *Fairness-Oriented Stratified Resampling*

Description

Pre-processing correction for fairness. Every combination of outcome class and protected group is upsampled with replacement until it has as many rows as the largest combination, so that all outcome-by-group cells are balanced in the training data.

Usage

```
resample_fairness(
  train_data,
  target,
  protected_var,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

train_data	Training data frame.
target	Name of the outcome variable.
protected_var	Name of the protected variable.
seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical (default TRUE). Print the size of each combination before and after resampling.

Value

A data frame with the same columns as train_data, containing the original rows plus the upsampled rows (grouped by outcome-by-group combination, with reset row names).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
train <- data.frame(
  x = rnorm(100),
  outcome = factor(sample(c("no", "yes"), 100, replace = TRUE, prob = c(0.7, 0.3))),
  sex = sample(c("female", "male"), 100, replace = TRUE, prob = c(0.8, 0.2))
)
balanced <- resample_fairness(train, target = "outcome",
                              protected_var = "sex", seed = 1)
table(balanced$outcome, balanced$sex)
```

run_nested_cv *Nested Cross-Validation*

Description

Runs nested cross-validation to obtain a less biased estimate of the performance of a model whose hyperparameters are tuned: an inner cross-validation tunes the model by grid search and an outer cross-validation evaluates the tuned model on data not used for tuning.

Usage

```
run_nested_cv(
  data,
  target,
  task,
  recipe,
  model_name = "rf",
  outer_folds = 5,
  inner_folds = 5,
  grid_size = 10,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	Training data (data frame).
target	Name of the target variable.
task	Type of task: "classification" or "regression".
recipe	A recipes recipe for the preprocessing (e.g. from prep_recipe() or recipes::recipe()).
model_name	Model to tune: "rf" (default, random forest with 500 trees) or "xgboost".
outer_folds	Number of outer folds (default: 5).
inner_folds	Number of inner folds (default: 5).
grid_size	Size of the tuning grid (default: 10).
seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical. Print progress (default: TRUE).

Value

A list with results (data frame with one row per outer fold: fold and roc_auc and accuracy for classification, or rmse and rsq for regression), summary (list with the mean and SD of those metrics), task, model and config (numbers of outer and inner folds).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 100
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- df$x1 + 0.5 * df$x2 + rnorm(n)
rec <- recipes::recipe(y ~ ., data = df)
ncv <- run_nested_cv(df, target = "y", task = "regression", recipe = rec,
                     outer_folds = 2, inner_folds = 2, grid_size = 2,
                     seed = 1, verbose = FALSE)
ncv$results
```

---

save_all_plots *Save All Plots to PNG Files*

Description

Saves every plot of a supervised_ml() result as a PNG file with a descriptive name (e.g. Figura_1_Importancia_Variable plus the combined summary panel (Figura_0_Panel_Resumen.png, only when **patchwork** is installed). The directory is created if needed.

Usage

```
save_all_plots(
  x,
  path,
  prefix = NULL,
  width = 8,
  height = 6,
  dpi = 300,
  verbose = TRUE
)
```

Arguments

x	An object of class supervisedml returned by supervised_ml().
path	Directory where the files are written. Required; there is no default.
prefix	Optional character prefix added to every file name (default: NULL).
width	Width of each figure in inches (default: 8).
height	Height of each figure in inches (default: 6).
dpi	Resolution in dots per inch (default: 300).
verbose	Logical. Print the catalog of saved figures (default: TRUE).

Value

Invisibly, a named list with one element per saved file, each a list with filename, full_path and catalog (the figure metadata, see generate_all_plots()).

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
out_dir <- file.path(tempdir(), "psyml_figures")
saved <- save_all_plots(res, path = out_dir, dpi = 72, verbose = FALSE)
list.files(out_dir)
unlink(out_dir, recursive = TRUE)
```

save_unsupervised_plots *Save All Figures of an Unsupervised Analysis*

Description

Saves every ggplot2 figure stored in an unsupervisedml object as a PNG file in output_dir. Nothing is written unless a directory is supplied.

Usage

```
save_unsupervised_plots(
  result,
  output_dir,
  width = 10,
  height = 7,
  dpi = 300,
  verbose = TRUE
)
```

Arguments

result	Object of class unsupervisedml.
output_dir	Directory where the PNG files are written (required; for example tempdir()). It is created if it does not exist.
width	Width in inches (default: 10).
height	Height in inches (default: 7).
dpi	Resolution in dots per inch (default: 300).
verbose	Logical. If TRUE, the output directory and the number of saved files are reported with message().

Value

Invisibly, a character vector with the paths of the saved files.

See Also

Other unsupervised learning: cluster_profile(), plot.unsupervisedml(), predict.unsupervisedml(), print.unsupervisedml(), summary.unsupervisedml(), unsupervised_ml()

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)), b = c(rnorm(40), rnorm(40, 3)))
res <- unsupervised_ml(X, methods = "clustering", clustering_algorithms = "kmeans",
                       k = 2, run_eda = FALSE,
```

```
                        verbose = FALSE)
out <- file.path(tempdir(), "psyml_unsup_figs")
files <- save_unsupervised_plots(res, output_dir = out, dpi = 72)
unlink(out, recursive = TRUE)
```

`setup_cv` *Set Up Cross-Validation Folds*

Description

Creates k-fold cross-validation folds of the training data, stratified by the outcome in classification.

Usage

```
setup_cv(train_data, target, task, cv_folds = 10, seed = NULL, verbose = TRUE)
```

Arguments

`train_data`	Data frame with the training data.
`target`	Character string with the name of the outcome variable.
`task`	Type of task: `"classification"` or `"regression"`.
`cv_folds`	Number of folds (default: 10).
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.
`verbose`	Logical. Print the progress and an explanation of each step in the console (default: TRUE).

Value

An `rset` object (tibble with columns `splits` and `id`) created by `rsample::vfold_cv()`.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `summary.supervisedml()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
folds <- setup_cv(df, target = "y", task = "classification",
                  cv_folds = 3, seed = 1)
folds
```

---

summary.causalml *Summarize a causalml Object*

Description

Prints the comparison table of the methods and the sensitivity analysis (E-value).

Usage

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

Arguments

`object`	An object of class `causalml`, as returned by `causal_ml()`.
`...`	Further arguments (not used).

Value

Invisibly returns `object`.

See Also

Other causal inference: `causal_ml()`, `plot.causalml()`, `predict.causalml()`, `print.causalml()`

Examples

```
set.seed(1)
n <- 300
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$treat <- rbinom(n, 1, plogis(df$x1))
df$y <- 1.5 * df$treat + df$x1 + 0.5 * df$x2 + rnorm(n)
res <- causal_ml(df, treatment = "treat", outcome = "y",
                 methods = c("propensity", "meta_learners"),
                 ps_methods = c("ipw", "doubly_robust"),
                 meta_learners = "T", run_eda = FALSE,
                 seed = 1, verbose = FALSE)
summary(res)
```

`summary.supervisedml` *Summarize a supervisedml Object*

Description

Prints the cross-validation comparison of all trained models, the test metrics and the tuned hyperparameters of a `supervised_ml()` result (console text is in Spanish).

Usage

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

Arguments

`object`	An object of class `supervisedml`.
`...`	Additional arguments (not used).

Value

Invisibly returns `object`. Called for its side effect of printing.

See Also

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `supervised_ml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
summary(res)
```

summary.unsupervisedml *Summarize an unsupervisedml Object*

Description

Prints a detailed summary: data dimensions, comparison of the clustering algorithms, best algorithm and PCA explained variance.

Usage

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

Arguments

object	An object of class unsupervisedml, as returned by unsupervised_ml().
...	Further arguments (not used).

Value

Invisibly returns object.

See Also

Other unsupervised learning: cluster_profile(), plot.unsupervisedml(), predict.unsupervisedml(), print.unsupervisedml(), save_unsupervised_plots(), unsupervised_ml()

Examples

```
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X,
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k_range = 2:4, reduction_methods = "pca",
                       anomaly_methods = "mahalanobis",
                       run_eda = FALSE, seed = 1, verbose = FALSE)
summary(res)
```

supervised_ml *Supervised Machine Learning for Classification and Regression*

Description

Main function of the package. It runs a complete supervised machine learning pipeline: exploratory data analysis, preprocessing, training of several models with cross-validation, hyperparameter tuning of the best model, evaluation on a held-out test set, interpretability (permutation importance and SHAP values) and optional advanced diagnostics (data leakage, threshold optimization, interactions, calibration, nested cross-validation, fairness and decision curve analysis).

Usage

```
supervised_ml(
  data,
  target,
  task = c("auto", "classification", "regression"),
  models = c("rf", "xgboost", "svm", "nnet", "glm", "tree"),
  test_split = 0.2,
  cv_folds = 10,
  select_metric = NULL,
  tune_best = TRUE,
  tune_method = c("random", "grid", "bayes", "racing"),
  tune_grid = 20,
  tune_iter = 30,
  feature_engineering = FALSE,
  custom_features = NULL,
  feature_selection = FALSE,
  balance_classes = FALSE,
  balance_method = c("smote", "adasyn", "rose", "up", "down"),
  impute = TRUE,
  impute_method = c("knn", "median", "mean"),
  normalize = TRUE,
  normalize_method = c("zscore", "minmax"),
  use_pca = FALSE,
  pca_threshold = 0.95,
  treat_outliers = TRUE,
  outlier_percentile = 0.05,
  remove_high_cor = TRUE,
  cor_threshold = 0.9,
  remove_high_vif = TRUE,
  vif_threshold = 5,
  run_eda = TRUE,
  run_shap = TRUE,
  n_shap = 100,
  optimize_threshold = TRUE,
  threshold_method = "youden",
```

```
  calibrate_probs = FALSE,
  calibration_method = "platt",
  check_leakage = TRUE,
  nested_cv = FALSE,
  analyze_interactions = TRUE,
  protected_var = NULL,
  run_fairness = FALSE,
  fairness_threshold = 0.8,
  fairness_correction = c("none", "postprocesamiento", "preprocesamiento"),
  run_dca = FALSE,
  dca_thresholds = seq(0.01, 0.99, by = 0.01),
  n_cores = 2,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame with the predictors and the outcome.
target	Character string with the name of the outcome variable.
task	Type of task: "auto" (default; detected from the outcome), "classification" or "regression".
models	Character vector with the models to train (default: c("rf", "xgboost", "svm", "nnet", "glm", "tree")). Available: "rf" (random forest), "xgboost", "svm" (RBF kernel), "nnet" (neural network), "glm" (logistic or linear regression), "glmnet" (regularized regression), "tree" (decision tree) and "nb" (naive Bayes). Some models need suggested packages (e.g. **kernlab**, **glmnet**, **rpart**).
test_split	Proportion of the data held out as test set (default: 0.20).
cv_folds	Number of cross-validation folds (default: 10).
select_metric	Metric used to select the best model. If NULL (default) it is chosen automatically: ROC-AUC for classification and R-squared for regression. Classification: "roc_auc", "mcc", "f_meas" (F1), "f2_meas" (F2), "accuracy", "sensitivity", "specificity", "bal_accuracy", "pr_auc", "kap", "rank_mean". Regression: "rsq", "rmse", "mae", "rank_mean".
tune_best	Logical. Tune the hyperparameters of the best model (default: TRUE).
tune_method	Hyperparameter search method: "random" (random search, default), "grid" (grid search), "bayes" (Bayesian optimization) or "racing" (racing with ANOVA).
tune_grid	Number of hyperparameter combinations evaluated during tuning (default: 20).
tune_iter	Number of iterations for Bayesian optimization (default: 30).
feature_engineering	Logical. Apply automatic feature engineering: detection of suspicious zeros, log-transformation of skewed variables, missing-value indicators and interactions between the top variables (default: FALSE).
custom_features	Optional named list to create domain variables. Each element name is the new variable and its value is an R expression given as a character string, e.g.

list(family = "SibSp + Parch", fare_pp = "Fare / (SibSp + Parch + 1)") (default: NULL).

feature_selection
: Logical. Perform variable selection with cross-validation (default: FALSE).

balance_classes
: Logical. For classification, balance the classes of the training set (default: FALSE).

balance_method
: Balancing method: "smote" (default), "adasyn", "rose", "up" or "down".

impute
: Logical. Impute missing values (default: TRUE).

impute_method
: Imputation method: "knn" (default), "median" or "mean".

normalize
: Logical. Normalize numeric predictors (default: TRUE).

normalize_method
: Normalization method: "zscore" (default, standardization) or "minmax" (rescaling between 0 and 1).

use_pca
: Logical. Apply principal component analysis after normalization (default: FALSE).

pca_threshold
: Proportion of variance retained by PCA (default: 0.95).

treat_outliers
: Logical. Treat outliers by winsorization (default: TRUE).

outlier_percentile
: Percentile used for winsorization (default: 0.05).

remove_high_cor
: Logical. Remove highly correlated predictors (default: TRUE).

cor_threshold
: Correlation threshold for removal (default: 0.90).

remove_high_vif
: Logical. Remove predictors with a high variance inflation factor (default: TRUE).

vif_threshold
: VIF threshold for removal (default: 5).

run_eda
: Logical. Run the exploratory data analysis (default: TRUE).

run_shap
: Logical. Compute variable importance and SHAP values (default: TRUE).

n_shap
: Number of observations used to compute SHAP values (default: 100).

optimize_threshold
: Logical. Optimize the decision threshold in binary classification (default: TRUE).

threshold_method
: Threshold optimization criterion: "youden" (default), "f1" or "balanced".

calibrate_probs
: Logical. Calibrate predicted probabilities in binary classification (default: FALSE).

calibration_method
: Calibration method: "platt" (default) or "isotonic".

check_leakage
: Logical. Check for possible data leakage after fitting (default: TRUE).

nested_cv
: Logical. Run nested cross-validation for a less biased performance estimate; only used when the best model is "rf" or "xgboost" (default: FALSE).

analyze_interactions
: Logical. Analyze interactions and non-linear effects with splines (default: TRUE).

protected_var
: Optional name of a protected variable used in the fairness analysis, e.g. "sex", "race" or "age_group" (default: NULL).

`run_fairness` Logical. Run the fairness analysis (default: FALSE). Requires `protected_var`. Computes demographic parity, equalized odds, predictive parity and the disparate impact ratio (four-fifths rule). Classification only.

`fairness_threshold` Threshold of the four-fifths rule for the disparate impact ratio (default: 0.8).

`fairness_correction` Bias mitigation strategy for classification: `"none"` (default, no correction), `"postprocesamiento"` (post-processing: group-specific decision thresholds, stored in `group_thresholds` and used by `predict.supervisedml()`) or `"preprocesamiento"` (pre-processing: the training set is resampled by protected group and outcome before fitting). Any value other than `"none"` requires `protected_var` and switches `run_fairness` on; it is ignored for regression.

`run_dca` Logical. Run decision curve analysis of clinical utility, classification only: net benefit compared with treating all or none (default: FALSE).

`dca_thresholds` Numeric vector of probability thresholds for the decision curve analysis (default: `seq(0.01, 0.99, by = 0.01)`).

`n_cores` Number of cores for parallel processing (default: 2). Use 1 for sequential processing. If NULL, all available cores minus one are used. Parallel processing requires the packages doParallel and foreach; if they are not installed, the analysis runs sequentially.

`seed` Optional integer seed. If NULL (default) the random number generator state is not modified.

`verbose` Logical. Print the progress and the interpretation of each step in the console (default: TRUE).

Details

The data are split into a training and a test set (`test_split`). All models listed in `models` are fitted with `cv_folds`-fold cross-validation on the training set and compared with `select_metric`. The best model is optionally tuned (`tune_best`), refitted on the whole training set and evaluated on the test set. For binary classification the decision threshold is chosen with out-of-fold training predictions and evaluated on the test set. When `verbose = TRUE` the console output (in Spanish) is shown in real time and also stored in the returned object (`verbose_text`, `verbose_lines`).

Value

An object of class `supervisedml` (a list). Main elements:

task Type of task: `"classification"` or `"regression"`.

target Name of the outcome variable.

best_model Name of the selected model.

best_params Tuned hyperparameters of the best model (NULL if not tuned).

final_fit Final fitted model (a tidymodels workflow).

test_metrics Tibble with the metrics on the test set (`.metric`, `.estimator`, `.estimate`).

predictions Tibble with the predictions on the test set.

cv_summary Cross-validation summary of all trained models.

importance Data frame with variable importance (`Variable`, `Importance`).

train_data, test_data Preprocessed training and test sets.

eda, preprocessing, modeling, tuning, evaluation Detailed results of each step.

interpretation Importance and SHAP results (if `run_shap = TRUE`).

threshold_optimization, optimal_threshold Threshold optimization (binary classification).

leakage_check, interactions_analysis, calibration, residual_analysis, nested_cv Advanced analyses, when run.

fairness_analysis, fairness_correction, group_thresholds Fairness results (if requested).

dca Decision curve analysis (if `run_dca = TRUE`).

plots, figures Lists of ggplot2 figures.

figures_catalog Catalog of figures with metadata for reports.

verbose_text, verbose_lines Captured console output (only when `verbose = TRUE`).

metadata List with analysis information (sample sizes, models, folds, metric, seed, elapsed time, date).

See Also

`predict.supervisedml()`, `plot.supervisedml()`, `save_all_plots()`

Other supervised learning: `advanced_residual_analysis()`, `analyze_dca()`, `analyze_fairness()`, `analyze_interactions_splines()`, `apply_custom_features()`, `apply_threshold()`, `auto_feature_engineering()`, `calculate_shap()`, `calibrate_probabilities()`, `compare_models()`, `correct_fairness_threshold()`, `cv_feature_selection()`, `define_models()`, `detect_data_leakage()`, `eval_calibration()`, `eval_confusion()`, `eval_metrics()`, `eval_pred_vs_obs()`, `eval_predictions()`, `eval_residuals()`, `eval_roc()`, `evaluate_model()`, `fit_models_cv()`, `generate_all_plots()`, `get_figures_catalog()`, `get_importance()`, `interpret_model()`, `optimize_threshold()`, `plot.supervisedml()`, `plot_all()`, `plot_calibration_curve()`, `plot_importance()`, `plot_metrics_profile()`, `plot_residual_diagnostics()`, `plot_shap_bar()`, `plot_shap_dependence()`, `plot_shap_summary()`, `plot_threshold_optimization()`, `plot_tuning()`, `predict.supervisedml()`, `prep_feature_selection()`, `prep_recipe()`, `prep_split()`, `prep_target()`, `preprocess_data()`, `print.supervisedml()`, `print_figures_catalog()`, `resample_fairness()`, `run_nested_cv()`, `save_all_plots()`, `setup_cv()`, `summary.supervisedml()`, `train_models()`, `tune_best_model()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "glm", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
print(res)
summary(res)

# Predictions for new observations
```

```
new_df <- data.frame(x1 = rnorm(5), x2 = rnorm(5), x3 = rnorm(5))
predict(res, new_df)
predict(res, new_df, type = "prob")

plot(res, type = "roc")

# Two models, tuning and SHAP values (slower)
res2 <- supervised_ml(df, target = "y", models = c("glm", "rf"),
                      cv_folds = 3, tune_grid = 3, n_shap = 20,
                      n_cores = 1, seed = 123, verbose = FALSE)
res2$best_model
plot(res2, type = "shap_bar")
if (requireNamespace("patchwork", quietly = TRUE)) plot(res2)
```

`supervised_ml_capture` *Run supervised_ml() and Capture Its Console Output*

Description

Runs `supervised_ml()` with `verbose = TRUE`, capturing all the console output so that it can be exported with `export_verbose_txt()` or `export_verbose_json()`. The captured output is not shown in the console.

Usage

```
supervised_ml_capture(...)
```

Arguments

`...`	Arguments passed to `supervised_ml()` (do not pass `verbose`, it is always set to TRUE).

Value

The `supervisedml` object returned by `supervised_ml()` with three additional elements: `verbose_text` (the captured output as a single string), `verbose_lines` (character vector, one element per line) and `verbose_json` (list with the output parsed into sections).

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_export()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml_capture(df, target = "y", models = "glm", cv_folds = 3,
                             tune_best = FALSE, run_shap = FALSE,
                             n_cores = 1)
cat(head(res$verbose_lines, 10), sep = "\n")
```

---

supervised_ml_export *Run and Export a Complete Analysis*

Description

Convenience function that runs `supervised_ml()` through `supervised_ml_capture()` and exports both a TXT and a JSON file to `output_dir`. The file names are `<prefix>_<timestamp>.txt` and `<prefix>_<timestamp>.json`.

Usage

```
supervised_ml_export(..., output_dir, prefix = "supervisedml")
```

Arguments

`...`	Arguments passed to `supervised_ml()` (do not pass `verbose`).
`output_dir`	Output directory (required, no default). It is created if it does not exist.
`prefix`	Prefix of the file names (default `"supervisedml"`).

Value

Invisibly returns the result of `supervised_ml_capture()` (a `supervisedml` object with the captured console output).

See Also

Other reporting and export: `explain_with_ai()`, `export_verbose_json()`, `export_verbose_txt()`, `generate_report_from_json()`, `generate_report_with_ai()`, `launch_report_generator()`, `launch_supervisedml_app()`, `supervised_ml_capture()`

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
out_dir <- file.path(tempdir(), "psyml_export")
res <- supervised_ml_export(df, target = "y", models = "glm", cv_folds = 3,
                            tune_best = FALSE, run_shap = FALSE,
                            n_cores = 1, output_dir = out_dir)
list.files(out_dir)
unlink(out_dir, recursive = TRUE)
```

---

train_models *Train and Compare Models with Cross-Validation*

Description

Modeling step of the pipeline: creates the cross-validation folds (`setup_cv()`), defines the model specifications (`define_models()`), fits every model with cross-validation (`fit_models_cv()`) and compares them to select the best one (`compare_models()`).

Usage

```
train_models(
  preprocess_result,
  models = c("rf", "xgboost", "glm"),
  cv_folds = 10,
  select_metric = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`preprocess_result`	Result of `preprocess_data()`. It must contain `task`, `target`, `train_data` and `recipe`.
`models`	Character vector with the models to train (default: `c("rf", "xgboost", "glm")`). Available: `"rf"` (random forest), `"xgboost"`, `"svm"` (RBF kernel), `"nnet"` (neural network), `"glm"` (logistic or linear regression), `"glmnet"` (regularized regression), `"tree"` (decision tree) and `"nb"` (naive Bayes). Some models need suggested packages (e.g. **kernlab**, **glmnet**, **rpart**).
`cv_folds`	Number of cross-validation folds (default: 10).
`select_metric`	Metric used to select the best model. If NULL (default) ROC-AUC is used for classification and R-squared for regression. See `supervised_ml()` for the available metrics.

seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical. Print the progress and an explanation of each step in the console (default: TRUE).

Value

An object of class supervisedml_modeling (a list) with cv_folds (the rset of folds), model_specs (named list of parsnip specifications), cv_results (named list of resampling results), comparison (result of compare_models()), best_model (name of the best model), task, target, recipe and train_data.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), tune_best_model()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
pp <- preprocess_data(df, target = "y", task = "classification",
                      verbose = FALSE)
mod <- train_models(pp, models = c("glm", "rf"), cv_folds = 3,
                    verbose = FALSE)
mod$best_model
mod$comparison$summary
```

---

tune_best_model *Tune the Hyperparameters of the Best Model*

Description

Tunes the hyperparameters of the best model selected by `train_models()` with cross-validation on the training folds, and returns the finalized workflow with the best combination. Tunable models are `"rf"`, `"xgboost"`, `"svm"`, `"nnet"`, `"glmnet"` and `"tree"`; for other models (e.g. `"glm"`) tuning is skipped and the untuned workflow is returned.

Usage

```
tune_best_model(
  modeling_result,
  method = "random",
  grid_size = 20,
  iter = 30,
  select_metric = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`modeling_result`	Result of `train_models()` (or the `modeling` element of a `supervised_ml()` result). It must contain `best_model`, `task`, `recipe` and `cv_folds`.
`method`	Search method: `"random"` (random search, default), `"grid"` (grid search), `"bayes"` (Bayesian optimization) or `"racing"` (racing with ANOVA; requires **finetune**, otherwise random search is used with a warning).
`grid_size`	Number of hyperparameter combinations evaluated with the grid, random and racing methods (default: 20).
`iter`	Maximum number of iterations of Bayesian optimization (default: 30).
`select_metric`	Metric used to select the best combination. If NULL (default) ROC-AUC is used for classification and R-squared for regression (also when `"rank_mean"` is given).
`seed`	Optional integer seed. If NULL (default) the random number generator state is not modified.
`verbose`	Logical. Print the progress and the interpretation of the best hyperparameters (default: TRUE).

Value

A list (class `supervisedml_tuning` when tuning was performed) with `original_model` (name of the model), `tuned` (logical), `best_params` (named list with the best hyperparameter values, NULL if not tuned), `final_workflow` (finalized, unfitted tidymodels workflow) and, when tuned, `tune_results` (the `tune_results` object), `task` and `method`.

See Also

Other supervised learning: advanced_residual_analysis(), analyze_dca(), analyze_fairness(), analyze_interactions_splines(), apply_custom_features(), apply_threshold(), auto_feature_engineering(), calculate_shap(), calibrate_probabilities(), compare_models(), correct_fairness_threshold(), cv_feature_selection(), define_models(), detect_data_leakage(), eval_calibration(), eval_confusion(), eval_metrics(), eval_pred_vs_obs(), eval_predictions(), eval_residuals(), eval_roc(), evaluate_model(), fit_models_cv(), generate_all_plots(), get_figures_catalog(), get_importance(), interpret_model(), optimize_threshold(), plot.supervisedml(), plot_all(), plot_calibration_curve(), plot_importance(), plot_metrics_profile(), plot_residual_diagnostics(), plot_shap_bar(), plot_shap_dependence(), plot_shap_summary(), plot_threshold_optimization(), plot_tuning(), predict.supervisedml(), prep_feature_selection(), prep_recipe(), prep_split(), prep_target(), preprocess_data(), print.supervisedml(), print_figures_catalog(), resample_fairness(), run_nested_cv(), save_all_plots(), setup_cv(), summary.supervisedml(), supervised_ml(), train_models()

Examples

```
set.seed(1)
n <- 200
df <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))
df$y <- factor(ifelse(df$x1 + 0.5 * df$x2 + rnorm(n) > 0, "yes", "no"))
res <- supervised_ml(df, target = "y", models = "rf", cv_folds = 3,
                     tune_best = FALSE, run_shap = FALSE, n_cores = 1,
                     verbose = FALSE)
tuned <- tune_best_model(res$modeling, method = "random", grid_size = 3,
                         select_metric = "roc_auc", seed = 1,
                         verbose = FALSE)
tuned$best_params
```

---

unsupervised_ml *Automated Unsupervised Learning Pipeline*

Description

Runs a complete unsupervised learning pipeline: exploratory data analysis, preprocessing, dimensionality reduction, clustering, cluster evaluation, anomaly detection and visualization.

Usage

```
unsupervised_ml(
  data,
  methods = c("clustering", "reduction", "anomaly"),
  clustering_algorithms = c("kmeans", "hierarchical", "dbscan", "gmm"),
  k = NULL,
  k_range = 2:10,
  distance = c("euclidean", "manhattan", "gower"),
```

```
  hclust_method = c("ward.D2", "complete", "average", "single"),
  dbscan_eps = NULL,
  dbscan_minPts = 5,
  gmm_type = "VVV",
  lca_nclass = NULL,
  lca_nrep = 10,
  reduction_methods = c("pca", "tsne", "umap"),
  n_components = 2,
  pca_threshold = 0.95,
  tsne_perplexity = 30,
  umap_n_neighbors = 15,
  umap_min_dist = 0.1,
  anomaly_methods = c("isolation_forest", "lof", "mahalanobis"),
  contamination = 0.05,
  lof_k = 20,
  exclude_cols = NULL,
  impute = TRUE,
  impute_method = c("median", "knn", "mean"),
  normalize = TRUE,
  normalize_method = c("zscore", "minmax"),
  treat_outliers = FALSE,
  outlier_percentile = 0.05,
  remove_high_cor = TRUE,
  cor_threshold = 0.9,
  run_eda = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame with the data.
methods	Character vector with the analyses to run: any of "clustering", "reduction" and "anomaly" (default: all three).
clustering_algorithms	Clustering algorithms: any of "kmeans", "hierarchical", "dbscan", "gmm" and "lca" (default: all except "lca"). LCA (latent class analysis) requires categorical or binary data.
k	Number of clusters. If NULL (default), the optimal number is searched automatically.
k_range	Range of k values evaluated when k = NULL (default: 2:10).
distance	Distance metric: "euclidean" (default), "manhattan" or "gower".
hclust_method	Linkage method for hierarchical clustering: "ward.D2" (default), "complete", "average" or "single".
dbscan_eps	Epsilon radius for DBSCAN. If NULL (default), it is computed automatically.
dbscan_minPts	Minimum number of points for DBSCAN (default: 5).

gmm_type	Covariance model for GMM, passed to 'mclust' (default: "VVV").
lca_nclass	Number of latent classes for LCA. If NULL (default), the number of classes is searched automatically.
lca_nrep	Number of LCA repetitions to avoid local maxima (default: 10).
reduction_methods	Dimensionality reduction methods: any of "pca", "tsne" and "umap" (default: all three).
n_components	Number of target dimensions for t-SNE and UMAP (default: 2).
pca_threshold	Cumulative proportion of variance retained by PCA (default: 0.95).
tsne_perplexity	Perplexity for t-SNE (default: 30).
umap_n_neighbors	Number of neighbours for UMAP (default: 15).
umap_min_dist	Minimum distance for UMAP (default: 0.1).
anomaly_methods	Anomaly detection methods: any of "isolation_forest", "lof" and "mahalanobis" (default: all three).
contamination	Expected proportion of anomalies (default: 0.05).
lof_k	Number of neighbours for LOF (default: 20).
exclude_cols	Character vector of columns to exclude from the analysis (IDs, text, etc.).
impute	Logical; impute missing values (default: TRUE).
impute_method	Imputation method: "median" (default), "knn" or "mean".
normalize	Logical; normalize the numeric variables (default: TRUE).
normalize_method	Normalization method: "zscore" (default) or "minmax".
treat_outliers	Logical; winsorize outliers before the analysis (default: FALSE).
outlier_percentile	Percentile used for winsorization (default: 0.05).
remove_high_cor	Logical; remove highly correlated variables (default: TRUE).
cor_threshold	Correlation threshold used by remove_high_cor (default: 0.90).
run_eda	Logical; run the exploratory data analysis (default: TRUE).
seed	Optional integer seed. If NULL (default) the random number generator state is not modified.
verbose	Logical; print progress to the console (default: TRUE).

Details

Only numeric columns are analysed. Several algorithms rely on suggested packages and are skipped (with a note in the verbose output) when they are not installed: 'dbscan' (DBSCAN and LOF), 'mclust' (GMM), 'poLCA' (LCA), 'Rtsne' (t-SNE), 'umap' (UMAP), 'isotree' (isolation forest), 'cluster' (silhouette, gap statistic and Gower distance), 'ggdendro' (dendrogram) and 'VIM' (kNN imputation).

Value

An object of class `unsupervisedml` (a list) with the components:

call The matched call of the internal pipeline.

params List with the main analysis parameters.

eda Exploratory analysis: structure, missing values, descriptive statistics, outliers, correlations and normality (only if `run_eda = TRUE`).

preprocessed List with the preprocessed data (`data_clean`), the data before normalization (`data_before_norm`), the normalization parameters (`norm_params`), the removed columns (`removed_cols`) and the number of variables before and after preprocessing.

reduction Dimensionality reduction results (`pca`, `tsne`, `umap`).

clustering Clustering results: `optimal_k`, `k_analysis` and `results`, a list with one element per algorithm containing the cluster `labels` and the fitted model.

evaluation Cluster evaluation: `comparison` (data frame with the silhouette, Calinski-Harabasz and Davies-Bouldin indices of each algorithm) and `best_algorithm`.

anomalies Anomaly detection results per method (`results`) and the majority `consensus`.

figures Named list of 'ggplot2' plots.

elapsed_time Execution time in seconds.

data_original The input data.

verbose_text, verbose_lines Captured console output (only if `verbose = TRUE`).

See Also

Other unsupervised learning: `cluster_profile()`, `plot.unsupervisedml()`, `predict.unsupervisedml()`, `print.unsupervisedml()`, `save_unsupervised_plots()`, `summary.unsupervisedml()`

Examples

```
# Two well separated groups
set.seed(1)
X <- data.frame(a = c(rnorm(40), rnorm(40, 3)),
                b = c(rnorm(40), rnorm(40, 3)),
                c = rnorm(80))
res <- unsupervised_ml(X,
                       clustering_algorithms = c("kmeans", "hierarchical"),
                       k_range = 2:4, reduction_methods = "pca",
                       anomaly_methods = "mahalanobis",
                       run_eda = FALSE, seed = 1, verbose = FALSE)
res$evaluation$comparison
```

Index

∗ **causal inference**
causal_ml, 19
plot.causalml, 76
predict.causalml, 94
print.causalml, 105
summary.causalml, 117
∗ **exploratory data analysis**
eda_basic, 32
eda_categorical, 33
eda_categorical_basic, 33
eda_correlation, 34
eda_correlation_basic, 35
eda_leakage, 36
eda_missing, 37
eda_normality, 37
eda_numeric, 38
eda_numeric_basic, 39
eda_outliers, 40
eda_structure, 41
eda_summary, 41
eda_target_classification, 42
eda_target_regression, 43
eda_vif, 44
∗ **reporting and export**
explain_with_ai, 56
export_verbose_json, 58
export_verbose_txt, 59
generate_report_from_json, 63
generate_report_with_ai, 64
launch_report_generator, 69
launch_supervisedml_app, 70
supervised_ml_capture, 125
supervised_ml_export, 126
∗ **study design**
block_split, 14
diversity_split, 29
estimate_n, 44
estimate_n_plateau, 45
ml_sample_size, 71
plot_all.ml_sample_size, 80
plot_distribution, 82
plot_learning_curve, 84
∗ **supervised learning**
advanced_residual_analysis, 4
analyze_dca, 6
analyze_fairness, 7
analyze_interactions_splines, 9
apply_custom_features, 10
apply_threshold, 11
auto_feature_engineering, 12
calculate_shap, 16
calibrate_probabilities, 17
compare_models, 23
correct_fairness_threshold, 24
cv_feature_selection, 26
define_models, 27
detect_data_leakage, 28
eval_calibration, 47
eval_confusion, 48
eval_metrics, 49
eval_pred_vs_obs, 50
eval_predictions, 51
eval_residuals, 53
eval_roc, 54
evaluate_model, 55
fit_models_cv, 60
generate_all_plots, 61
get_figures_catalog, 65
get_importance, 66
interpret_model, 68
optimize_threshold, 75
plot.supervisedml, 77
plot_all, 79
plot_calibration_curve, 81
plot_importance, 83
plot_metrics_profile, 85
plot_residual_diagnostics, 87
plot_shap_bar, 88

plot_shap_dependence, 89
plot_shap_summary, 90
plot_threshold_optimization, 92
plot_tuning, 93
predict.supervisedml, 95
prep_feature_selection, 97
prep_recipe, 98
prep_split, 100
prep_target, 101
preprocess_data, 102
print.supervisedml, 107
print_figures_catalog, 109
resample_fairness, 110
run_nested_cv, 112
save_all_plots, 113
setup_cv, 116
summary.supervisedml, 118
supervised_ml, 120
train_models, 127
tune_best_model, 128
∗ **unsupervised learning**
cluster_profile, 22
plot.unsupervisedml, 78
predict.unsupervisedml, 96
print.unsupervisedml, 108
save_unsupervised_plots, 115
summary.unsupervisedml, 119
unsupervised_ml, 130

advanced_residual_analysis, 4, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
advanced_residual_analysis(), *87*
analyze_dca, *5*, 6, *8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
analyze_fairness, *5*, 7, *7*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
analyze_interactions_splines, *5*, *7, 8*, 9, *11–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
apply_custom_features, *5*, *7, 8*, 10, *10*, *12, 13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
apply_threshold, *5*, *7, 8*, *10*, 11, *11*, *13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
auto_feature_engineering, *5*, *7, 8*, *10, 11*, 12, *12*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*

block_split, 14, *31*, *45, 46*, *74*, *81*, *83*, *85*
block_split(), *105*
Boruta::attStats(), *97*

calculate_shap, *5*, *7, 8*, *10–13*, 16, *18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
calculate_shap(), *68*, *88, 89*, *91*
calibrate_probabilities, *5*, *7, 8*, *10–13*, 17, *17*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
calibrate_probabilities(), *81*
causal_ml, 19, *77*, *94*, *106*, *117*
causal_ml(), *76*, *94*, *105*, *117*
cluster_profile, 22, *79*, *97*, *109*, *115*, *119*, *133*
compare_models, *5*, *7, 8*, *10–13*, *17, 18*, 23, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61*,

62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
compare_models(), 127, 128
correct_fairness_threshold, 5, 7, 8, 10–13, 17, 18, 23, 24, 26, 28, 29, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
cv_feature_selection, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130

define_models, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 27, 29, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
define_models(), 60, 127
detect_data_leakage, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 28, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
diversity_split, 15, 29, 45, 46, 74, 81, 83, 85
diversity_split(), 106

eda_basic, 32, 33–44
eda_basic(), 41, 42
eda_categorical, 32, 33, 34–44
eda_categorical_basic, 32, 33, 33–44
eda_categorical_basic(), 32
eda_correlation, 32, 33, 34, 34–44
eda_correlation_basic, 32–34, 35, 36–44
eda_correlation_basic(), 32
eda_leakage, 32–35, 36, 37–44
eda_leakage(), 28
eda_missing, 32–36, 37, 38–44
eda_missing(), 32
eda_normality, 32–36, 37, 37, 39–44
eda_numeric, 32–37, 38, 38–44
eda_numeric_basic, 32–38, 39, 39–44
eda_numeric_basic(), 32
eda_outliers, 32–39, 40, 41–44
eda_outliers(), 32
eda_structure, 32–40, 41, 42–44
eda_structure(), 32, 42
eda_summary, 32–40, 41, 41, 43, 44
eda_target_classification, 32–41, 42, 42–44
eda_target_regression, 32–42, 43, 43, 44
eda_vif, 32–43, 44
estimate_n, 15, 31, 44, 46, 74, 81, 83, 85
estimate_n(), 73
estimate_n_plateau, 15, 31, 45, 45, 74, 81, 83, 85
estimate_n_plateau(), 73
eval_calibration, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
eval_calibration(), 56
eval_confusion, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 50–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
eval_confusion(), 56
eval_metrics, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 49, 51–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
eval_metrics(), 56
eval_pred_vs_obs, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 50, 50, 52–54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98, 100–102, 104, 108, 110, 111, 113, 114, 116, 118, 124, 128, 130
eval_pred_vs_obs(), 56
eval_predictions, 5, 7, 8, 10–13, 17, 18, 23, 25, 26, 28, 29, 47, 48, 50, 51, 51, 53, 54, 56, 61, 62, 66, 67, 69, 76, 78, 80, 82, 83, 86–88, 90–93, 95, 98,

100–102, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
eval_predictions(), *47–49*, *51*, *53, 54*, *56*
eval_residuals, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–52*, 53, *54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
eval_residuals(), *56*
eval_roc, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–53*, 54, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
eval_roc(), *56*
evaluate_model, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, 55, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
evaluate_model(), *68*
explain_with_ai, 56, *58, 59*, *63*, *65*, *70*, *125, 126*
export_verbose_json, *57*, 58, *59*, *63*, *65*, *70*, *125, 126*
export_verbose_json(), *63, 64*, *69*, *125*
export_verbose_txt, *57, 58*, 59, *63*, *65*, *70*, *125, 126*
export_verbose_txt(), *125*

fit_models_cv, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, 60, *62*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
fit_models_cv(), *23*, *127*

generate_all_plots, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, 61, *61*, *66, 67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
generate_all_plots(), *114*
generate_report_from_json, *57–59*, 63, *65*, *70*, *125, 126*
generate_report_from_json(), *58*
generate_report_with_ai, *57–59*, *63*, 64, *70*, *125, 126*
generate_report_with_ai(), *58*
get_figures_catalog, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, 65, *67*, *69*, *76*, *78*, *80*, *82, 83*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
get_figures_catalog(), *109*
get_importance, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, 66, *66*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
get_importance(), *9*, *68*

interpret_model, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25, 26*, *28, 29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, 68, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
interpret_model(), *83*, *88, 89*, *91*

kernelshap::kernelshap(), *16*

launch_report_generator, *57–59*, *63*, *65*, 69, *70*, *125, 126*
launch_report_generator(), *58*
launch_supervisedml_app, *57–59*, *63*, *65*, 70, *70*, *125, 126*

message(), *79*, *115*
ml_sample_size, *15*, *31*, *45, 46*, 71, *81*, *83*, *85*
ml_sample_size(), *45, 46*, *82*, *84*, *107*

optimize_threshold, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, 75, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
optimize_threshold(), *11*, *92*

plot.causalml, *21*, 76, *94*, *106*, *117*

plot.supervisedml, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, 77, *80*, *82*, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot.supervisedml(), *124*
plot.unsupervisedml, *22*, 78, *97*, *109*, *115*, *119*, *133*
plot_all, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, 79, *82*, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_all.ml_sample_size, *15*, *31*, *45, 46*, *74*, 80, *83*, *85*
plot_calibration_curve, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, 81, *84*, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_distribution, *15*, *31*, *45, 46*, *74*, *81*, 82, *85*
plot_distribution(), *73*, *80*
plot_importance, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, 83, *86–88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_learning_curve, *15*, *31*, *45, 46*, *74*, *81*, *83*, 84
plot_learning_curve(), *73*, *80*
plot_metrics_profile, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, 85, *87, 88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_residual_diagnostics, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86*, 87, *88*, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_shap_bar, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86, 87*, 88, *90–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_shap_dependence, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, 89, *91–93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_shap_summary, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, 90, *90*, *92, 93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_threshold_optimization, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90, 91*, 92, *93*, *95*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
plot_tuning, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47, 48*, *50–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–92*, 93, *96*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
predict.causalml, *21*, *77*, 94, *106*, *117*
predict.supervisedml, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, 95, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
predict.supervisedml(), *123, 124*
predict.unsupervisedml, *22*, *79*, 96, *109*, *115*, *119*, *133*
prep_feature_selection, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, 97, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, *128*, *130*
prep_recipe, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*,

96, 98, *98*, *101*, *102*, *104*, *108*, *110*, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
prep_recipe(), *104*, *112*
prep_split, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, 100, *100*, *102*, *104*, *108*, *110*, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
prep_target, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100*, 101, *101*, *104*, *108*, *110*, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
preprocess_data, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100*, *101*, 102, *102*, *108*, *110*, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
preprocess_data(), *60*, *127*
print.block_split, 105
print.causalml, *21*, *77*, *94*, 105, *117*
print.diversity_split, 106
print.ml_sample_size, 107
print.supervisedml, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, 107, *110*, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
print.unsupervisedml, *22*, *79*, *97*, 108, *115*, *119*, *133*
print_figures_catalog, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, 109, *111*, *113*, *114*, *116*, *118*, *124*, *128*, *130*

recipes::recipe(), *112*
recipes::step_other(), *104*
resample_fairness, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, 110, *110*, *113*, *114*, *116*, *118*, *124*, *128*, *130*
rsample::initial_split(), *100*
rsample::vfold_cv(), *116*
run_nested_cv, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110*, *111*, 112, *114*, *116*, *118*, *124*, *128*, *130*

save_all_plots, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110*, *111*, 113, *113*, *116*, *118*, *124*, *128*, *130*
save_all_plots(), *124*
save_unsupervised_plots, *22*, *79*, *97*, *109*, 115, *119*, *133*
setup_cv, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110*, *111*, *113*, *114*, 116, *118*, *124*, *128*, *130*
setup_cv(), *60*, *127*
stats::shapiro.test(), *38*, *53*
summary.causalml, *21*, *77*, *94*, *106*, 117
summary.supervisedml, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110*, *111*, *113*, *114*, *116*, 118, *124*, *128*, *130*
summary.unsupervisedml, *22*, *79*, *97*, *109*, *115*, 119, *133*
supervised_ml, *5*, *7*, *8*, *10–13*, *17*, *18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61*, *62*, *66*, *67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110*, *111*, *113*, *114*, *116*, *118*, 120, *128*, *130*
supervised_ml(), *5*, *28*, *52*, *55*, *58*, *59*, *61*, *62*, *65*, *67*, *75*, *77*, *80*, *88*, *89*, *91*, *93*, *95*, *107*, *109*, *113*, *114*, *118*, *125–127*, *129*
supervised_ml_capture, *57–59*, *63*, *65*, *70*, 125, *126*
supervised_ml_capture(), *58*, *59*, *126*
supervised_ml_export, *57–59*, *63*, *65*, *70*, *125*, 126

train_models, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, 127, *130*
train_models(), *55*, *129*
tune::autoplot(), *93*
tune::collect_metrics(), *23*
tune::fit_resamples(), *61*
tune_best_model, *5*, *7, 8*, *10–13*, *17, 18*, *23*, *25*, *27–29*, *47*, *49–54*, *56*, *61, 62*, *66, 67*, *69*, *76*, *78*, *80*, *82*, *84*, *86–88*, *90–93*, *96*, *98*, *100–102*, *104*, *108*, *110, 111*, *113, 114*, *116*, *118*, *124*, 128, *128*
tune_best_model(), *55*, *93*

unsupervised_ml, *22*, *79*, *97*, *109*, *115*, *119*, 130
unsupervised_ml(), *22*, *79*, *96*, *109*, *119*

yardstick::conf_mat(), *48*
yardstick::roc_curve(), *54*