Package 'auxvecLASSO'

August 28, 2025

Title LASSO Auxiliary Variable Selection and Auxiliary Vector Diagnostics

Version 0.2.0

Description Provides tools for assessing and selecting auxiliary variables using LASSO. The package includes functions for variable selection and diagnostics, facilitating survey calibration analysis with emphasis on robust auxiliary vector selection. For more details see Tibshirani (1996) <doi:10.1111/j.2517-6161.1996.tb02080.x> and Caughrey and Hartman (2017) <doi:10.2139/ssrn.3494436>.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.5.0)

Imports doParallel, parallelly, Matrix, glmnet, stats, survey, utils, pROC, crayon

Suggests testthat (>= 3.0.0), foreach, knitr, rmarkdown, withr, sampling, dplyr

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.2

URL https://github.com/gustafanderssons/auxvecLASSO-R-Package

BugReports https://github.com/gustafanderssons/auxvecLASSO-R-Package/issues

Language en

NeedsCompilation no

Author Gustaf Andersson [aut, cre, cph]

Maintainer Gustaf Andersson <gustafanderssons@gmail.com>

Repository CRAN

Date/Publication 2025-08-28 09:00:12 UTC

Contents

assess_aux_vector 2
estimate_mean_stats 6
fit_outcome 8
generate_population_totals 12
print.assess_aux_vector 14
print.select_auxiliary_variables_lasso_cv 17
select_auxiliary_variables_lasso_cv 20

Index **24**

`assess_aux_vector` *Assess Auxiliary vector Calibration and Diagnostics*

Description

This function assesses the calibration of auxiliary variables in a survey design, performs various diagnostics, and optionally calibrates weights based on a specified calibration formula. It provides diagnostics on weight variation, register data alignment, and survey data alignment. The results are returned as a list of class "assess_aux_vector".

Usage

```
assess_aux_vector(
  design,
  df,
  calibration_formula = NULL,
  calibration_pop_totals = NULL,
  register_vars = NULL,
  register_pop_means = NULL,
  survey_vars = NULL,
  domain_vars = NULL,
 diagnostics = c("weight_variation", "register_diagnostics", "survey_diagnostics"),
  already_calibrated = FALSE,
  verbose = FALSE
)
```

Arguments

`design`	A survey design object, typically of class `svydesign`, representing the survey data design.
`df`	A data frame containing the survey data to be used in the analysis.
`calibration_formula`	An optional formula object specifying the auxiliary variables used for calibration (e.g., `~age + gender`). If provided, the weights will be calibrated.

`calibration_pop_totals`
: An optional list of population totals for the auxiliary variables in `calibration_formula`. This can be used in the calibration process.

`register_vars`
: A character vector specifying the names of the auxiliary variables from the register data that should be used in the diagnostics. If `NULL`, no register diagnostics will be performed.

`register_pop_means`
: A list containing population means for the register variables. The list may include a "total" entry for the total population means and/or a "by_domain" entry for domain-specific population means.

`survey_vars`
: A character vector specifying the names of the survey variables to be used in the diagnostics. If `NULL`, no survey diagnostics will be performed.

`domain_vars`
: A character vector specifying the domain variables used to group data for domain-specific diagnostics. If `NULL`, diagnostics will be computed for the entire sample rather than for specific domains.

`diagnostics`
: A character vector specifying which diagnostics to compute. Possible values include:

  - `"weight_variation"`: Computes diagnostics related to weight variation.
  - `"register_diagnostics"`: Computes diagnostics based on the register data.
  - `"survey_diagnostics"`: Computes diagnostics based on the survey data.

  The default is all three.

`already_calibrated`
: A logical flag indicating whether the weights have already been calibrated. If `TRUE`, the calibration step will be skipped.

`verbose`
: A logical flag indicating whether to print additional messages during the execution of the function. This can be useful for debugging or monitoring progress.

Details

The function supports several diagnostic checks, including weight variation diagnostics, register diagnostics (total and by domain), and survey diagnostics (total and by domain).

The function may also calibrate survey weights based on a provided calibration formula and population totals. Calibration can be skipped if the weights are already calibrated.

The function supports several diagnostic checks, including weight variation diagnostics, register diagnostics (total and by domain), and survey diagnostics (total and by domain).

The function may also calibrate survey weights based on a provided calibration formula and population totals. Calibration can be skipped if the weights are already calibrated.

The weight diagnostics contain the following measures:

- Descriptive statistics (min, max, median, mean, standard deviation (sd), range, bottom percentile, top percentile)
- Inequality measures (coefficient of variation, Gini index, entropy)
- Skewness and (excess) kurtosis

Value

A list of class "assess_aux_vector" containing the results of the diagnostic assessments. The list includes the following components:

- weight_variation: A numeric vector or matrix containing the results of the weight variation diagnostics.
- register_diagnostics: A list containing diagnostics based on the register data. This may include the total diagnostics and/or domain-specific diagnostics.
- survey_diagnostics: A list containing diagnostics based on the survey data. This may include the total diagnostics and/or domain-specific diagnostics.

See Also

calibrate for the calibration function.

Examples

```
## ============================================================
## Example 1: Calibrate weights, then run all diagnostics
##            (register + survey, with a by-domain breakdown)
## ============================================================
if (requireNamespace("survey", quietly = TRUE)) {
  set.seed(42)
  options(survey.lonely.psu = "adjust")

  ## --- Simulate a tiny sample
  n <- 200
  sex <- factor(sample(c("F", "M"), n, replace = TRUE))
  sex[1:2] <- c("F", "M")
  sex <- factor(sex, levels = c("F", "M"))
  region <- factor(sample(c("N", "S"), n, replace = TRUE))
  region[1:2] <- c("N", "S")
  region <- factor(region, levels = c("N", "S"))
  age <- round(rnorm(n, mean = 41, sd = 12))
  ## Register variable we have population means for:
  reg_income <- 50000 + 2000 * (region == "S") + rnorm(n, sd = 4000)
  ## A couple of survey variables to diagnose:
  y1 <- 10 + 2 * (sex == "M") + rnorm(n, sd = 2)
  y2 <- 100 + 5 * (region == "S") + rnorm(n, sd = 5)
  ## Some unequal weights (to make weight-variation meaningful)
  w <- runif(n, 0.6, 2.2) * 50

  df <- data.frame(sex, region, age, reg_income, y1, y2, w)
  design <- survey::svydesign(ids = ~1, weights = ~w, data = df)

  ## --- Calibration setup (simple main-effects formula)
  ## Model matrix columns will be: (Intercept), sexM, regionS, age
  Npop <- 5000
  pop_mean_age <- 40
  calibration_formula <- ~ sex + region + age
  calibration_pop_totals <- c(
```

```
    "(Intercept)" = Npop,
    "sexM"        = round(0.45 * Npop), # 45% of population is male
    "regionS"     = round(0.40 * Npop), # 40% in region S
    "age"         = pop_mean_age * Npop # totals (mean * N)
  )

  ## --- Register population means: total + by domain (single register var)
  register_vars <- "reg_income"
  register_pop_means <- list(
    total = c(reg_income = 51000), # overall pop mean
    by_domain = list(
      region = c(N = 50000, S = 52000) # domain-specific pop means
    )
  )

  out1 <- assess_aux_vector(
    design                  = design,
    df                      = df,
    calibration_formula     = calibration_formula,
    calibration_pop_totals  = calibration_pop_totals,
    register_vars           = register_vars,
    register_pop_means      = register_pop_means,
    survey_vars             = c("y1", "y2"),
    domain_vars             = c("region"),
   diagnostics         = c("weight_variation", "register_diagnostics", "survey_diagnostics"),
    already_calibrated      = FALSE,
    verbose                 = FALSE
  )

  ## Peek at key outputs:
  out1$weight_variation
  out1$register_diagnostics$total
  out1$register_diagnostics$by_domain$region
  out1$survey_diagnostics$total
}

## ============================================================
## Example 2: Skip calibration; survey diagnostics by domain
## ============================================================
if (requireNamespace("survey", quietly = TRUE)) {
  set.seed(99)
  options(survey.lonely.psu = "adjust")

  n <- 120
  region <- factor(sample(c("N", "S"), n, replace = TRUE))
  region[1:2] <- c("N", "S")
  region <- factor(region, levels = c("N", "S"))
  sex <- factor(sample(c("F", "M"), n, replace = TRUE))
  sex[1:2] <- c("F", "M")
  sex <- factor(sex, levels = c("F", "M"))
  age <- round(rnorm(n, 39, 11))
  yA <- rnorm(n, mean = 50 + 3 * (region == "S"))
  yB <- rnorm(n, mean = 30 + 1.5 * (sex == "M"))
```

```
  w <- runif(n, 0.7, 1.8) * 40

  toy <- data.frame(region, sex, age, yA, yB, w)
  des <- survey::svydesign(ids = ~1, weights = ~w, data = toy)

  out2 <- assess_aux_vector(
    design = des,
    df = toy,
    calibration_formula = NULL, # skip calibration
    calibration_pop_totals = NULL,
    register_vars = NULL, # no register diagnostics
    survey_vars = c("yA", "yB"),
    domain_vars = "region",
    diagnostics = c("weight_variation", "survey_diagnostics"),
    already_calibrated = TRUE, # explicitly skip calibration
    verbose = FALSE
  )

  out2$weight_variation
  out2$survey_diagnostics$by_domain$region
}
```

`estimate_mean_stats` *Estimate Survey-Weighted Means (Overall or By Domain)*

Description

Computes survey-weighted means and standard errors for one or more variables in a survey design, optionally stratified by domains (e.g., regions, groups). If population means are provided, it also calculates the bias and mean squared error (MSE) of the estimates compared to the population means.

Usage

```
estimate_mean_stats(design, vars, by = NULL, population_means = NULL)
```

Arguments

`design`	A survey design object (e.g., from the **survey** package). The object must contain survey data and weights.
`vars`	A character vector of variable names to compute means for. These variables must exist in `design$variables`.
`by`	Optional. A one-sided formula (e.g., `~region`) or a character vector of domain variables for stratified analysis. If `NULL`, computes overall means across all data.
`population_means`	Optional. A named numeric vector with population means (for `by = NULL`) or a data frame containing domain variables and population means (for `by` not `NULL`). Used to calculate bias and MSE for each variable.

Details

- Observations with non-finite **weights** are excluded from the analysis globally.
- For each variable, observations with non-finite values are also dropped (in addition to the global weight filter). An error will occur if no valid data remains.
- For domain-specific means, `population_means` must include all domain columns and the variable being estimated. The rows are merged by domain key.

Value

A named list where each component is a data frame with the following columns:

domain columns Columns for the domain variables (if `by` is specified).

variable The name of the variable.

mean The survey-weighted mean of the variable.

se The standard error of the survey-weighted mean.

bias The difference between the survey-weighted mean and the population mean. `NA` if `population_means` is not provided.

mse The mean squared error (MSE) of the survey-weighted mean. `NA` if `population_means` is not provided.

Examples

```
## ============================================================
## Example 1: Overall means with population means (bias/MSE)
## ============================================================
if (requireNamespace("survey", quietly = TRUE)) {
  set.seed(123)
  options(survey.lonely.psu = "adjust")

  n <- 200
  region <- factor(sample(c("N", "S"), n, replace = TRUE))
  sex <- factor(sample(c("F", "M"), n, replace = TRUE))
  y <- 10 + 2 * (sex == "M") + rnorm(n, sd = 1.5)
  z <- 100 + 5 * (region == "S") + rnorm(n, sd = 3)
  w <- runif(n, 0.8, 1.8) * 50
  df <- data.frame(region, sex, y, z, w)

  des <- survey::svydesign(ids = ~1, weights = ~w, data = df)

  ## Named vector of population means for overall case
  pop_means <- c(y = 11.2, z = 103.0)

  res_overall <- estimate_mean_stats(
    design = des,
    vars = c("y", "z"),
    by = NULL,
    population_means = pop_means
  )
```

```
  ## Each element is a one-row data frame
  res_overall$y
  res_overall$z
}

## ============================================================
## Example 2: Domain means by region with population means table
## ============================================================
if (requireNamespace("survey", quietly = TRUE)) {
  set.seed(456)
  options(survey.lonely.psu = "adjust")

  n <- 150
  region <- factor(sample(c("N", "S"), n, replace = TRUE), levels = c("N", "S"))
  sex <- factor(sample(c("F", "M"), n, replace = TRUE))
  y <- 12 + 1.5 * (region == "S") + rnorm(n, sd = 1.2)
  z <- 95 + 6 * (region == "S") + rnorm(n, sd = 2.5)
  w <- runif(n, 0.7, 2.0) * 40
  toy <- data.frame(region, sex, y, z, w)

  des2 <- survey::svydesign(ids = ~1, weights = ~w, data = toy)

  ## Population means by domain must include the domain column(s) + vars
  pop_by_region <- data.frame(
    region = c("N", "S"),
    y = c(12.2, 13.8),
    z = c(95.5, 101.0),
    stringsAsFactors = FALSE
  )

  res_by <- estimate_mean_stats(
    design = des2,
    vars = c("y", "z"),
    by = ~region, # or equivalently: by = "region"
    population_means = pop_by_region # merged by the 'region' key
  )

  ## Each element is a data frame with domain rows
  res_by$y
  res_by$z
}
```

`fit_outcome` *Fit LASSO model for a single outcome with cross-validation*

Description

Fits a LASSO regression model (logistic regression for binary outcomes or linear regression for continuous outcomes) for a single outcome variable using cross-validation. The function drops

rows where the outcome is missing, and ensures that the predictors do not have missing values. The model is fitted using the `glmnet` package, with the option to apply cross-validation for selecting the optimal regularization parameter (`lambda`).

Usage

```
fit_outcome(
  yvar,
  df,
  X,
  penalty_factors,
  nfolds = 5,
  standardize = TRUE,
  parallel = FALSE,
  return_models = FALSE,
  verbose = FALSE
)
```

Arguments

`yvar`	Character scalar. The name of the outcome variable in `df`. This can be either a binary or continuous outcome variable.
`df`	Data frame containing the outcome and predictors. The outcome variable (`yvar`) and predictor variables must be included in `df`.
`X`	Model matrix (rows must align with `df`). The matrix must not contain missing values, and its rows must match those in `df`.
`penalty_factors`	Named numeric vector of penalty factors for each predictor variable. The names should match the column names of `X`.
`nfolds`	Number of folds for cross-validation. Default is 5.
`standardize`	Logical; should the predictors be standardized before fitting the model? Default is `TRUE`.
`parallel`	Logical; should the cross-validation be performed in parallel? Default is `FALSE`.
`return_models`	Logical; should the fitted `cv.glmnet` object be returned? Default is `FALSE`.
`verbose`	Logical; if `TRUE`, prints progress messages. Default is `FALSE`.

Details

- The outcome variable (`yvar`) can be either **binary** or **continuous**:
  - **Binary outcomes**: LASSO logistic regression is used. The outcome variable must have exactly two levels after missing values are removed.
  - **Continuous outcomes**: LASSO linear regression is used. The outcome variable should be numeric.
- Rows with missing values for the outcome (`yvar`) or predictors (`X`) will be dropped.
- The function uses the `glmnet` package to fit the LASSO model.
- Cross-validation is used to select the optimal regularization parameter (`lambda`).

- Model performance metrics, including **AUC**, **accuracy**, **Brier score** (for binary outcomes) or **RSS**, **MSE**, **RMSE**, **MAE**, **R-squared** (for continuous outcomes), are computed on the non-missing rows.

Value

A list containing:

selected A character vector of the names of the selected variables (non-zero coefficients).

lambda_min The value of `lambda` that minimizes the cross-validation error.

goodness A list containing performance metrics for the model:

- `cross_validated`: A list with `cv_error` (cross-validation error) and `cv_error_sd` (standard deviation of CV error).
- `full_data`: A list with:
- `deviance_explained`: Proportion of deviance explained by the model (only for binary outcomes).
- `auc`: Area under the ROC curve (only for binary outcomes).
- `accuracy`: Classification accuracy (only for binary outcomes).
- `brier_score`: Brier score (mean squared error for probabilities) (only for binary outcomes).
- `rss`: Residual sum of squares (only for continuous outcomes).
- `mse`: Mean squared error (only for continuous outcomes).
- `rmse`: Root mean squared error (only for continuous outcomes).
- `mae`: Mean absolute error (only for continuous outcomes).
- `r_squared`: R-squared (only for continuous outcomes).
- `raw_coefs`: Raw coefficients from the LASSO model.
- `abs_coefs`: Absolute values of the coefficients.

model The fitted `cv.glmnet` model, if `return_models = TRUE`. Otherwise, `NULL`.

Examples

```
## ============================================================
## Example 1: Binary outcome (binomial LASSO with CV)
## ============================================================
if (requireNamespace("glmnet", quietly = TRUE) &&
  requireNamespace("pROC", quietly = TRUE)) {
  set.seed(101)

  n <- 180
  x1 <- rnorm(n)
  x2 <- rnorm(n)
  f <- factor(sample(c("A", "B", "C"), n, replace = TRUE))

  ## Construct a binary outcome with signal in x2 and x1:x2
  lin <- -0.3 + 1.0 * x2 + 0.6 * (x1 * x2) - 0.7 * (f == "C")
  p <- 1 / (1 + exp(-lin))
  yfac <- factor(rbinom(n, 1, p), labels = c("No", "Yes")) # 2-level factor
```

```
  df <- data.frame(y = yfac, x1 = x1, x2 = x2, f = f)

  ## Model matrix with main effects + one interaction, no intercept
  X <- model.matrix(~ x1 + x2 + f + x1:x2 - 1, data = df)

  ## Penalty factors must match X's columns (names + length).
  penalty_factors <- rep(1, ncol(X))
  names(penalty_factors) <- colnames(X)
  ## (Optional) keep x1 unpenalized:
  if ("x1" %in% names(penalty_factors)) penalty_factors["x1"] <- 0

  fit_bin <- fit_outcome(
    yvar            = "y",
    df              = df,
    X               = X,
    penalty_factors = penalty_factors,
    nfolds          = 3,
    standardize     = TRUE,
    parallel        = FALSE,
    return_models   = FALSE,
    verbose         = FALSE
  )

  ## Peek at the results
  fit_bin$selected
  fit_bin$lambda_min
  fit_bin$goodness$full_data$auc
  fit_bin$goodness$full_data$accuracy
}

## ============================================================
## Example 2: Continuous outcome (gaussian LASSO with CV)
## ============================================================
if (requireNamespace("glmnet", quietly = TRUE)) {
  set.seed(202)

  n <- 160
  x1 <- rnorm(n)
  x2 <- rnorm(n)
  f <- factor(sample(c("L", "H"), n, replace = TRUE))

  y <- 1.5 * x1 + 0.8 * x2 - 1.0 * (f == "H") + 0.6 * (x1 * x2) + rnorm(n, sd = 0.7)
  df <- data.frame(y = y, x1 = x1, x2 = x2, f = f)

  ## Main effects only, no intercept
  X <- model.matrix(~ x1 + x2 + f - 1, data = df)

  penalty_factors <- rep(1, ncol(X))
  names(penalty_factors) <- colnames(X)

  fit_cont <- fit_outcome(
    yvar            = "y",
    df              = df,
```

```
  X              = X,
  penalty_factors = penalty_factors,
  nfolds         = 3,
  standardize    = TRUE,
  parallel       = FALSE,
  return_models  = FALSE,
  verbose        = FALSE
)

## Key metrics
fit_cont$selected
fit_cont$lambda_min
fit_cont$goodness$full_data$mse
fit_cont$goodness$full_data$r_squared
}
```

`generate_population_totals` *Generate population totals for a calibration design matrix*

Description

Build a fixed model matrix on a population frame and return the column totals needed for calibration (optionally weighted). The function freezes dummy/interaction structure on the population by constructing a `terms` object, so downstream use on respondent data can reuse the exact same encoding.

Usage

```
generate_population_totals(
  population_df,
  calibration_formula,
  weights = NULL,
  contrasts = NULL,
  include_intercept = TRUE,
  sparse = FALSE,
  na_action = stats::na.pass,
  drop_zero_cols = FALSE
)
```

Arguments

`population_df`	A data frame containing the calibration population.
`calibration_formula`	A one-sided formula specifying main effects and interactions (e.g., `~ stype + api00_bin:stype`). The intercept is handled by `include_intercept`.

weights	Optional numeric vector of population weights (length nrow(population_df)). If NULL (default), unweighted totals are computed.
contrasts	Optional named list of contrasts to pass to model.matrix() (e.g., list(stype = contr.treatment)). If NULL, the current global options(contrasts=...) are used.
include_intercept	Logical; if TRUE (default) keep the (Intercept) column in the totals (it will sum to sum(weights) or nrow(population_df) if unweighted).
sparse	Logical; if TRUE, return the population model matrix internally as a sparse Matrix while computing totals. (Totals are always returned as a base numeric vector.)
na_action	NA handling passed to model.frame(); defaults to stats::na.pass. Consider stats::na.omit for stricter behavior.
drop_zero_cols	Logical; if TRUE, drop columns whose population total is exactly zero. Default FALSE. A message is emitted if any zero-total columns are found.

Value

An object of class "calib_totals": a list with

- population_totals: named numeric vector of column totals
- levels: list of factor levels observed in the population (for reproducibility)
- terms: the terms object built on population_df
- contrasts: the contrasts actually used (from the model matrix)

Examples

```
# Example using the API data from the survey package
library(survey)
data(api) # loads apipop, apisrs, apistrat, etc.

# Build a population frame and create some binary fields used in a formula
pop <- apipop
pop$api00_bin <- as.factor(ifelse(pop$api00 >= 700, "700plus", "lt700"))
pop$growth_bin <- as.factor(ifelse(pop$growth >= 0, "nonneg", "neg"))
pop$ell_bin <- as.factor(ifelse(pop$ell >= 10, "highELL", "lowELL"))
pop$comp.imp_bin <- as.factor(ifelse(pop$comp.imp >= 50, "highComp", "lowComp"))
pop$hsg_bin <- as.factor(ifelse(pop$hsg >= 60, "highHSG", "lowHSG"))

# A calibration formula with main effects + a few interactions
cal_formula <- ~ stype + growth_bin + api00_bin + ell_bin + comp.imp_bin + hsg_bin +
  api00_bin:stype + hsg_bin:stype + comp.imp_bin:stype + api00_bin:growth_bin

# (Optional) frame weights if available; here we use unweighted totals
gp <- generate_population_totals(
  population_df        = pop,
  calibration_formula  = cal_formula,
  include_intercept    = TRUE
)
```

```
# Named totals ready for calibration:
head(gp$population_totals)

# If you later build a respondent model matrix, reuse gp$terms to ensure alignment:
# X_resp <- model.matrix(gp$terms, data = apisrs)
# stopifnot(identical(colnames(X_resp), names(gp$population_totals)))
```

`print.assess_aux_vector` *Print Summary of Auxiliary Vector Assessment*

Description

S3 print method for objects of class `assess_aux_vector`. Displays a formatted, colorized summary of weight variation metrics, register diagnostics (overall and by domain), and survey diagnostics (overall and by domain).

Usage

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

Arguments

`x`	An object of class `assess_aux_vector` containing diagnostic results. Expected to have components: **weight_variation** Named numeric vector or list of weight variation metrics. **register_diagnostics** List with `total` and `by_domain` components. Each inner data frame typically includes columns `variable`, `mean`, `se`, `rse`, `bias`, `mse`, and (when population means are available) `p_bias`. **survey_diagnostics** List with `total` and `by_domain` components. Each inner data frame typically includes columns `variable`, `mean`, `se`, `rse`; `bias`, `mse`, and `p_bias` are `NA` unless population means were provided.
`...`	Additional arguments (currently ignored).

Details

In addition to means and standard errors, the printer shows the **relative standard error** (**RSE** = SE / |mean|) and—when population means are supplied to `estimate_mean_stats()`—two-sided p-values for the bias testing H_0 : mean = population mean.

Requires the **crayon** package for colored output.

The print method outputs sections with colored headers for easier readability:

- Weight Variation Metrics

- Register Diagnostics summarized for all units and by domain
- Survey Diagnostics summarized for all units and by domain

For each variable shown, the following metrics are printed when present:

- **Mean** — survey-weighted mean.
- **SE** — design-based standard error from **survey**.
- **RSE** — relative standard error, $\mathrm{SE}/|\mathrm{Mean}|$.
- **Bias** — difference between estimate and population mean (if supplied).
- **MSE** — $\mathrm{Bias}^2 + \mathrm{SE}^2$ (if population means supplied).
- **p(Bias)** — two-sided p-value testing $H_0 : \mathrm{Bias} = 0$, computed as $2\Phi(-|z|)$ with $z = \mathrm{Bias}/\mathrm{SE}$ (shown when population means are available).

Edge cases: if `mean == 0` the RSE is reported as `NA`; if `SE == 0`, `p(Bias)` is `1` when `|Bias|` is numerically zero and `0` otherwise. Objects created with earlier versions that lack `rse` or `p_bias` columns are handled gracefully (those fields are simply not printed).

If the **crayon** package is not installed, the function will stop with an error.

Value

Invisibly returns the input object `x`.

Examples

```
## ============================================================
## Example 1: Print with register + survey diagnostics
##            (includes population means -> prints p(Bias))
## ============================================================
if (requireNamespace("survey", quietly = TRUE) &&
  requireNamespace("crayon", quietly = TRUE)) {
  set.seed(7)
  options(survey.lonely.psu = "adjust")

  ## --- Simulate a small survey
  n <- 180
  sex <- factor(sample(c("F", "M"), n, replace = TRUE), levels = c("F", "M"))
  region <- factor(sample(c("N", "S"), n, replace = TRUE), levels = c("N", "S"))
  age <- round(rnorm(n, mean = 42, sd = 12))
  reg_income <- 52000 + 1500 * (region == "S") + rnorm(n, sd = 3500) # register var
  y1 <- 10 + 1.8 * (sex == "M") + rnorm(n, sd = 2) # survey vars
  y2 <- 95 + 4.5 * (region == "S") + rnorm(n, sd = 3.5)
  w <- runif(n, 0.7, 2.1) * 40
  df <- data.frame(sex, region, age, reg_income, y1, y2, w)
  des <- survey::svydesign(ids = ~1, weights = ~w, data = df)

  ## --- Optional calibration inputs (simple main effects)
  ## Model matrix columns: (Intercept), sexM, regionS, age
  Npop <- 4000
  calibration_formula <- ~ sex + region + age
  calibration_pop_totals <- c(
```

```
    "(Intercept)" = Npop,
    "sexM"        = round(0.48 * Npop),
    "regionS"     = round(0.52 * Npop),
    "age"         = 41 * Npop
  )

  ## --- Population means for the register var: total + by domain
  register_vars <- "reg_income"
  register_pop_means <- list(
    total = c(reg_income = 52500),
    by_domain = list(
      region = c(N = 51500, S = 53500)
    )
  )

  ## --- Build assessment object
  aux1 <- assess_aux_vector(
    design                = des,
    df                    = df,
    calibration_formula   = calibration_formula,
    calibration_pop_totals = calibration_pop_totals,
    register_vars         = register_vars,
    register_pop_means    = register_pop_means,
    survey_vars           = c("y1", "y2"),
    domain_vars           = "region",
   diagnostics        = c("weight_variation", "register_diagnostics", "survey_diagnostics"),
    already_calibrated    = FALSE,
    verbose               = FALSE
  )

  ## Colorized, formatted summary:
  print(aux1)
}

## ============================================================
## Example 2: Print with survey diagnostics only (by domain)
##            (no population means -> p(Bias) omitted)
## ============================================================
if (requireNamespace("survey", quietly = TRUE) &&
  requireNamespace("crayon", quietly = TRUE)) {
  set.seed(11)
  options(survey.lonely.psu = "adjust")

  n <- 120
  region <- factor(sample(c("N", "S"), n, replace = TRUE), levels = c("N", "S"))
  sex <- factor(sample(c("F", "M"), n, replace = TRUE), levels = c("F", "M"))
  yA <- 50 + 2.5 * (region == "S") + rnorm(n, sd = 2)
  yB <- 30 + 1.5 * (sex == "M") + rnorm(n, sd = 1.5)
  w <- runif(n, 0.8, 1.9) * 35
  toy <- data.frame(region, sex, yA, yB, w)

  des2 <- survey::svydesign(ids = ~1, weights = ~w, data = toy)
```

```
  aux2 <- assess_aux_vector(
    design                = des2,
    df                    = toy,
    calibration_formula   = NULL, # skip calibration
    calibration_pop_totals = NULL,
    register_vars         = NULL, # no register diagnostics
    survey_vars           = c("yA", "yB"),
    domain_vars           = "region",
    diagnostics           = c("weight_variation", "survey_diagnostics"),
    already_calibrated    = TRUE,
    verbose               = FALSE
  )

  print(aux2)
}
```

`print.select_auxiliary_variables_lasso_cv` *Print Summary of LASSO Auxiliary Variable Selection Object*

Description

S3 print method for objects of class `select_auxiliary_variables_lasso_cv`. Displays a formatted and colorized summary of the selected auxiliary variables, their grouping by outcome, selected penalty parameters (lambdas), penalty factors, stored models, goodness-of-fit metrics, coefficient estimates, and interaction metadata.

Usage

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

Arguments

`x`	An object of class `select_auxiliary_variables_lasso_cv` containing results from LASSO auxiliary variable selection with cross-validation. Expected to have components: **selected_variables** Character vector of variables selected across outcomes. **by_outcome** Named list, with each element a character vector of selected variables for that outcome. **selected_lambdas** Named numeric vector or list of selected lambda values by outcome. **penalty_factors** Numeric vector of penalty factors (0 = must-keep, 1 = regular penalty). **models** List of fitted models stored for each outcome.

goodness_of_fit Named list of goodness-of-fit results by outcome, each containing `cross_validated` (with `cv_error`, `cv_error_sd`) and `full_data` (with `deviance_explained`, `auc`, `accuracy`, `brier_score`, `raw_coefs`).

interaction_metadata List with `interaction_terms`, `main_effects_in_interactions`, and `full_formula`.

... Additional arguments (currently ignored).

Details

Requires the **crayon** package for colored output.

The print method outputs information using colored text (via **crayon**), making it easier to visually parse the summary. It organizes output into sections:

- Selected variables and their counts
- Variables selected by each outcome
- Selected lambda tuning parameters
- Summary of penalty factors
- Number of stored models
- Goodness-of-fit metrics for each outcome, including cross-validation error statistics and metrics on full data fit
- Coefficients at the lambda minimizing error, ordered by magnitude
- Interaction terms and main effects metadata

If the **crayon** package is not installed, the function will stop with an error.

Value

Invisibly returns the input object `x`.

Examples

```
## ============================================================
## Example 1: Binary + continuous outcomes, with interactions
##            (prints selected vars, lambdas, GOF, coef table, interactions)
## ============================================================
if (requireNamespace("glmnet", quietly = TRUE) &&
  requireNamespace("pROC", quietly = TRUE) &&
  requireNamespace("crayon", quietly = TRUE)) {
  set.seed(123)

  n <- 180
  x1 <- rnorm(n)
  x2 <- rnorm(n)
  grp <- factor(sample(c("A", "B", "C"), n, replace = TRUE))

  ## Binary outcome with signal in x2, grp, and x1:x2 (make it a 2-level factor)
  eta <- -0.4 + 1.0 * x2 - 0.8 * (grp == "C") + 0.6 * (x1 * x2)
  p <- 1 / (1 + exp(-eta))
```

```
  y_bin <- factor(rbinom(n, 1, p), labels = c("No", "Yes"))

  ## Continuous outcome with some interaction
  y_cont <- 1.4 * x1 + 0.9 * x2 - 1.1 * (grp == "B") + 0.5 * (x1 * x2) + rnorm(n, sd = 0.7)

  df <- data.frame(y_bin = y_bin, y_cont = y_cont, x1 = x1, x2 = x2, grp = grp)

  lasso_obj1 <- select_auxiliary_variables_lasso_cv(
    df               = df,
    outcome_vars     = c("y_bin", "y_cont"),
    auxiliary_vars   = c("x1", "x2", "grp"),
    must_have_vars   = c("x1", "grp"), # 'grp' expands to its dummy columns
    check_twoway_int = TRUE, # include all two-way interactions
    nfolds           = 3,
    verbose          = FALSE,
    standardize      = TRUE,
    return_models    = FALSE, # models not stored (printer still shows GOF & coefs)
    parallel         = FALSE
  )

  ## Colorized, formatted summary:
  print(lasso_obj1)
}

## ============================================================
## Example 2: Single continuous outcome, main effects only
##            (stores model so the printer reports it)
## ============================================================
if (requireNamespace("glmnet", quietly = TRUE) &&
  requireNamespace("crayon", quietly = TRUE)) {
  set.seed(456)

  n <- 140
  a <- rnorm(n)
  b <- rnorm(n)
  f <- factor(sample(c("L", "H"), n, replace = TRUE))
  y <- 2 * a + 0.8 * b - 1.2 * (f == "H") + rnorm(n, sd = 0.8)

  toy <- data.frame(y = y, a = a, b = b, f = f)

  lasso_obj2 <- select_auxiliary_variables_lasso_cv(
    df               = toy,
    outcome_vars     = "y",
    auxiliary_vars   = c("a", "b", "f"),
    must_have_vars   = "f", # keep factor (its dummies get zero penalty)
    check_twoway_int = FALSE, # main effects only
    nfolds           = 3,
    verbose          = FALSE,
    standardize      = TRUE,
    return_models    = TRUE, # store cv.glmnet model
    parallel         = FALSE
  )
```

```
  print(lasso_obj2)
}
```

`select_auxiliary_variables_lasso_cv` *Select Auxiliary Variables via LASSO with Cross-Validation (Binary and Continuous Outcomes)*

Description

This function performs LASSO-penalized regression (logistic regression for binary outcomes or linear regression for continuous outcomes) with cross-validation to select auxiliary variables for modeling one or more outcome variables. It allows for the inclusion of all two-way interactions among the auxiliary variables and the option to force certain variables to remain in the model through the use of zero penalty factors.

Usage

```
select_auxiliary_variables_lasso_cv(
  df,
  outcome_vars,
  auxiliary_vars,
  must_have_vars = NULL,
  check_twoway_int = TRUE,
  nfolds = 5,
  verbose = TRUE,
  standardize = TRUE,
  return_models = FALSE,
  parallel = FALSE
)
```

Arguments

`df`	A data frame containing the data for modeling.
`outcome_vars`	Character vector of outcome variable names to model. These can be either binary or continuous outcomes. Each must exist in `df` and have at least two unique values (after factor conversion for binary outcomes).
`auxiliary_vars`	Character vector of auxiliary variable names to be used as predictors.
`must_have_vars`	Optional character vector of variable names that must be included in the model (penalty factor 0). If interactions are included, any interaction containing a must-have variable is also assigned zero penalty. The variables in `must_have_vars` should refer to either individual variables or the main effect part of interaction terms.
`check_twoway_int`	Logical; include all two-way interactions among auxiliary variables. Defaults to `TRUE`.

nfolds	Number of folds for cross-validation. Defaults to 5.
verbose	Logical; print progress messages. Defaults to TRUE.
standardize	Logical; standardize predictors before fitting. Defaults to TRUE.
return_models	Logical; return fitted cv.glmnet objects. Defaults to FALSE.
parallel	Logical; run cross-validation in parallel (requires **doParallel**). Defaults to FALSE.

Details

The function supports both binary and continuous outcomes. For binary outcomes, logistic regression is used, and for continuous outcomes, linear regression is used. The function outputs a list with the selected variables across outcomes, the associated lambda values, the goodness-of-fit statistics, and optionally the fitted models and interaction terms.

The function supports two types of outcome variables:

- **Binary outcomes**: LASSO logistic regression is used. The outcome variable must have exactly two levels after missing values are removed.
- **Continuous outcomes**: LASSO linear regression is used. The outcome variable should be numeric.

For factor variables in auxiliary_vars, dummy variables are created to represent each level of the factor. If a factor variable is specified in must_have_vars, its dummy variables will be included in the model, ensuring that any interactions containing those variables are also forced into the model.

Value

An object of class "select_auxiliary_variables_lasso_cv" with the following components:

selected_variables Character vector of variables selected across all outcome models. This includes the main effect variables and any interaction terms.

by_outcome Named list of character vectors, each containing the selected variables for each outcome.

selected_lambdas Named numeric vector of lambda values (specifically, lambda.min) for each outcome.

penalty_factors Named numeric vector with penalty factors (0 for must-keep, 1 otherwise).

models List of cv.glmnet objects per outcome if return_models = TRUE, otherwise an empty list.

goodness_of_fit Named list per outcome with cross-validation metrics (cv_error, cv_error_sd) and full data metrics (deviance_explained for binary outcomes, auc, accuracy, brier_score, rss, mse, r_squared, raw_coefs).

interaction_metadata List containing metadata on interaction terms, main effects in interactions, and the full formula used.

Examples

```
## ------------------------------------------------------------
## Example 1: Binary + continuous outcomes, with interactions
##             and must-have variables (factor expanded to dummies)
## ------------------------------------------------------------
```

```
set.seed(123)
n <- 150
x1 <- rnorm(n)
x2 <- rnorm(n)
group <- factor(sample(c("A", "B", "C"), n, replace = TRUE))

## Generate outcomes with some signal in x1, x2 and group, plus an interaction
eta_bin <- -0.5 + 1.2 * x2 - 0.8 * (group == "C") + 0.5 * x1 * x2
p <- 1 / (1 + exp(-eta_bin))
y_bin <- rbinom(n, 1, p)
y_cont <- 1.5 * x1 - 2 * (group == "B") + 0.7 * x1 * x2 + rnorm(n, sd = 0.7)

df <- data.frame(y_bin = y_bin, y_cont = y_cont, x1 = x1, x2 = x2, group = group)

res1 <- select_auxiliary_variables_lasso_cv(
  df = df,
  outcome_vars = c("y_bin", "y_cont"),
  auxiliary_vars = c("x1", "x2", "group"),
  must_have_vars = c("x1", "group"), # 'group' (factor) expands to its dummies
  check_twoway_int = TRUE,
  nfolds = 3,
  verbose = FALSE,
  standardize = TRUE,
  return_models = FALSE
)

## Inspect selections and metadata
res1$selected_variables
res1$by_outcome
res1$selected_lambdas
names(which(res1$penalty_factors == 0)) # must-keep terms (incl. factor dummies & interactions)
res1$interaction_metadata$full_formula

## ------------------------------------------------------------
## Example 2: Single continuous outcome, main effects only
## ------------------------------------------------------------
set.seed(456)
n2 <- 120
a <- rnorm(n2)
b <- rnorm(n2)
f <- factor(sample(c("a", "b"), n2, replace = TRUE))
y <- 2 * a - 1 * (f == "b") + rnorm(n2, sd = 1)

toy <- data.frame(y = y, a = a, b = b, f = f)

res2 <- select_auxiliary_variables_lasso_cv(
  df = toy,
  outcome_vars = "y",
  auxiliary_vars = c("a", "b", "f"),
  check_twoway_int = FALSE, # main effects only
  nfolds = 3,
  verbose = FALSE
)
```

```
res2$selected_variables
res2$selected_lambdas
res2$goodness_of_fit$y
```

Index

assess_aux_vector, 2

calibrate, *4*

estimate_mean_stats, 6

fit_outcome, 8

generate_population_totals, 12

print.assess_aux_vector, 14
print.select_auxiliary_variables_lasso_cv, 17

select_auxiliary_variables_lasso_cv, 20