Package 'collinear'

November 18, 2026

Title Automated Multicollinearity Management

Version 3.0.3

URL https://blasbenito.github.io/collinear/

BugReports https://github.com/blasbenito/collinear/issues

Description Provides a comprehensive and automated workflow for managing multicollinearity in data frames with numeric and/or categorical variables. The package integrates five robust methods into a single function: (1) target encoding of categorical variables based on response values (Micci-Barreca, 2001 (Micci-Barreca, D. 2001 <doi:10.1145/507533.507538>); (2) automated feature prioritization to preserve key predictors during filtering; (3 and 4) pairwise correlation and VIF filtering across all variable types (numeric–numeric, numeric–categorical, and categorical–categorical); (5) adaptive correlation and VIF thresholds. Together, these methods enable a reliable multicollinearity management in most use cases while maintaining model integrity. The package also supports parallel processing and progress tracking via the packages 'future' and 'progressr', and provides seamless integration with the 'tidymodels' ecosystem through a dedicated recipe step.

License MIT + file LICENSE

Encoding UTF-8

Imports progressr, future.apply, mgcv, ranger, recipes (>= 1.0.9), rlang, spatialData

Suggests future, testthat (>= 3.0.0), spelling

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

LazyDataCompression xz

Language en-US

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Blas M. Benito [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-5105-7232>)

Maintainer Blas M. Benito <blasbenito@gmail.com>

Repository CRAN

Date/Publication 2026-11-18 14:40:02 UTC

Contents

case_weights . . . 3
collinear . . . 4
collinear_select . . . 8
collinear_stats . . . 10
cor_clusters . . . 12
cor_cramer . . . 14
cor_df . . . 16
cor_matrix . . . 17
cor_select . . . 19
cor_stats . . . 22
drop_geometry_column . . . 23
experiment_adaptive_thresholds . . . 24
experiment_cor_vs_vif . . . 25
f_auto . . . 26
f_auto_rules . . . 28
f_binomial_gam . . . 28
f_binomial_glm . . . 30
f_binomial_rf . . . 31
f_categorical_rf . . . 32
f_count_gam . . . 34
f_count_glm . . . 35
f_count_rf . . . 36
f_functions . . . 37
f_numeric_gam . . . 38
f_numeric_glm . . . 39
f_numeric_rf . . . 40
gam_cor_to_vif . . . 42
identify_categorical_variables . . . 43
identify_logical_variables . . . 44
identify_numeric_variables . . . 45
identify_response_type . . . 47
identify_valid_variables . . . 48
identify_zero_variance_variables . . . 50
model_formula . . . 51
prediction_cor_to_vif . . . 54
preference_order . . . 55
print.collinear_output . . . 59
print.collinear_selection . . . 60
score_auc . . . 60
score_cramer . . . 61
score_r2 . . . 62

step_collinear 62
summary.collinear_output 65
summary.collinear_selection 66
target_encoding_lab 66
target_encoding_loo 68
toy 71
validate_arg_df 72
validate_arg_df_not_null 73
validate_arg_encoding_method 74
validate_arg_f 75
validate_arg_function_name 76
validate_arg_max_cor 77
validate_arg_max_vif 78
validate_arg_predictors 79
validate_arg_preference_order 80
validate_arg_quiet 82
validate_arg_responses 83
vif 84
vif_df 86
vif_select 87
vif_stats 90

Index **92**

`case_weights` *Generate sample weights for imbalanced responses*

Description

Computes case weights adding to one for response variables of these types:

- numeric binomial (1 and 0).
- logical (TRUE and FALSE): converted to numeric internally.
- categorical (character or factor)

Values NA, Inf, -Inf, and NaN are invalid for numeric and logical variables and will result in errors. For categorical variables, these are converted to their respective categories ("NA", "Inf", "-Inf", and "NaN") with their assigned case weights.

All returned weights sum to one.

Usage

```
case_weights(x = NULL, ...)
```

Arguments

x	(required, integer, character, or factor vector) Values of a binomial, categorical, or factor variable. Default: NULL
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

numeric vector: case weights

See Also

Other modelling_tools: model_formula(), score_auc(), score_cramer(), score_r2()

Examples

```
#numeric vector
y <- case_weights(
  x = c(0, 0, 1, 1)
  )

#logical vector
y <- case_weights(
  x = c(TRUE, TRUE, FALSE, FALSE)
  )

#character vector
y <- case_weights(
  x = c("a", "a", "b", "c")
  )
```

---

collinear *Smart multicollinearity management*

Description

Automates multicollinearity management in datasets with mixed variable types (numeric, categorical, and logical) through an integrated system of five key components:

Target Encoding Integration (opt-in) When responses is numeric, categorical predictors can be converted to numeric using response values as reference. This enables VIF and correlation analysis across mixed types. See target_encoding_lab.

Intelligent Predictor Ranking (active by default) Three prioritization strategies ensure the most relevant predictors are retained during filtering:

- **User-defined ranking** (argument preference_order): Accepts a character vector of predictor names or a dataframe from preference_order. Lower-ranked collinear predictors are removed.

- **Response-based ranking** (`f`): Uses `f_auto` to auto-select a function (see `f_numeric_glm`) to rank predictors by their univariate association with the response. Supports cross-validation via `preference_order`.
- **Multicollinearity-based ranking** (default): When both `preference_order` and `f` are `NULL`, predictors are ranked from lower to higher multicollinearity.

Unified Correlation Framework (active by default) Computes pairwise correlations between variable types using Pearson (numeric–numeric), target encoding (numeric–categorical), and Cramer's V (categorical–categorical). See `cor_df`, `cor_matrix`, and `cor_cramer`.

Adaptive Filtering Thresholds (active by default) When `max_cor` and `max_vif` are both `NULL`, thresholds are determined from the median correlation structure of the predictors.

Dual Filtering Strategy (active by default) Combines two complementary methods while respecting predictor rankings:

- **Pairwise Correlation Filtering**: Removes predictors with Pearson correlation or Cramer's V above `max_cor`. See `cor_select`.
- **VIF-based Filtering**: Removes numeric predictors with VIF above `max_vif`. See `vif_select`, `vif_df`, and `vif`.

This function accepts parallelization via `future::plan()` and progress bars via `progressr::handlers()`. Parallelization benefits `target_encoding_lab`, `preference_order`, and `cor_select`.

Usage

```
collinear(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  encoding_method = NULL,
  preference_order = NULL,
  f = f_auto,
  max_cor = NULL,
  max_vif = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`responses`	(optional; character, character vector, or NULL) Name of one or several response variables in `df`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`encoding_method`	(optional; character or NULL) One of "loo", "mean", or "rank". If NULL, target encoding is disabled. Default: NULL.

`preference_order`
: (optional; character vector, dataframe from `preference_order`, or NULL) Prioritizes predictors to preserve.

`f`
: (optional; unquoted function name or NULL) Function to rank predictors by relationship with `responses`. See `f_functions`. Default: `f_auto`.

`max_cor`
: (optional; numeric or NULL) Maximum allowed pairwise correlation (0.01–0.99). Recommended between 0.5 and 0.9. If NULL and `max_vif` is NULL, it is selected automatically. Default: NULL.

`max_vif`
: (optional; numeric or NULL) Maximum allowed VIF. Recommended between 2.5 and 10. If NULL and `max_cor` is NULL, configured automatically. Default: NULL.

`quiet`
: (optional; logical) If FALSE, messages are printed. Default: FALSE.

`...`
: (optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

A list of class `collinear_output` with sublists of class `collinear_selection`. If `responses = NULL` a single sublist named "result" is returned; otherwise a sublist per response is returned.

Adaptive Multicollinearity Thresholds

When both `max_cor` and `max_vif` are `NULL`, the function determines thresholds as follows:

1. Compute the 75th percentile of pairwise correlations via `cor_stats`.
2. Map that value through a sigmoid between 0.545 (VIF~2.5) and 0.785 (VIF~7.5), centered at 0.665, to get `max_cor`.
3. Compute `max_vif` from `max_cor` using `gam_cor_to_vif`.

Variance Inflation Factors

VIF for predictor a is computed as $1/(1 - R^2)$, where R^2 is the multiple R-squared from regressing a on the other predictors. Recommended maximums commonly used are 2.5, 5, and 10.

VIF-based Filtering

`vif_select` ranks numeric predictors (user `preference_order` if provided, otherwise from lower to higher VIF) and sequentially adds predictors whose VIF against the current selection is below `max_vif`.

Pairwise Correlation Filtering

`cor_select` computes the global correlation matrix, orders predictors by `preference_order` or by lower-to-higher summed correlations, and sequentially selects predictors with pairwise correlations below `max_cor`.

References

- David A. Belsley, D.A., Kuh, E., Welsch, R.E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons. DOI: 10.1002/0471725153.
- Micci-Barreca, D. (2001) A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems. SIGKDD Explor. Newsl. 3, 1, 27-32. DOI: 10.1145/507533.507538

See Also

Other multicollinearity_filtering: `collinear_select()`, `cor_select()`, `step_collinear()`, `vif_select()`

Examples

```
#load example data
data(
  vi_smol,
  vi_predictors,
  package = "spatialData"
  )

#select numeric predictors only
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

#OPTIONAL: parallelization setup
#worth it for large data
# future::plan(
#   strategy = future::multisession,
#   workers = future::availableCores() - 2
# )

#OPTIONAL
#progress bar (doesn't work when running an R example)
#progressr::handlers(global = TRUE)

#filter numeric predictors automatically
x <- collinear(
  df = vi_smol,
  response = "vi_numeric",
  predictors = vi_predictors_numeric
  )

#selected variable names
x$vi_numeric$selection

#preference order dataframe
x$vi_numeric$preference_order

#model formulas
```

```
x$vi_numeric$formulas

#using a formula
m <- lm(
  formula = x$vi_numeric$formulas$linear,
  data = x$vi_numeric$df
)

summary(m)

#disable parallelization
# future::plan(
#   strategy = future::sequential
# )
```

`collinear_select` *Dual multicollinearity filtering algorithm*

Description

Automatizes multicollinearity filtering via pairwise correlation and/or variance inflation factors in dataframes with numeric and categorical predictors.

The argument `max_cor` determines the maximum pairwise correlation allowed in the resulting selection of predictors, while `max_vif` does the same for variance inflation factors.

The argument `preference_order` accepts a character vector of predictor names ranked from first to last index, or a dataframe resulting from `preference_order()`. When two predictors in this vector or dataframe are highly collinear, the one with a lower ranking is removed. This option helps protect predictors of interest. If not provided, predictors are ranked from lower to higher multicollinearity. Please check the sections **Variance Inflation Factors**, **VIF-based Filtering**, and **Pairwise Correlation Filtering** at the end of this help file for further details.

Usage

```
collinear_select(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  max_cor = 0.61,
  max_vif = 5,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional; character or NULL) Name of one response variable in df. Used to exclude columns when predictors is NULL, and to filter preference_order when it is a dataframe and contains several responses. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
preference_order	(optional; character vector, dataframe from preference_order, or NULL) Prioritizes predictors to preserve.
max_cor	(optional; numeric or NULL) Maximum correlation allowed between pairs of predictors. Valid values are between 0.01 and 0.99, and recommended values are between 0.5 (strict) and 0.9 (permissive). Default: 0.7
max_vif	(optional, numeric or NULL) Maximum Variance Inflation Factor allowed for predictors during multicollinearity filtering. Recommended values are between 2.5 (strict) and 10 (permissive). Default: 5
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

character vector: names of selected predictors

Pairwise Correlation Filtering

cor_select computes the global correlation matrix, orders predictors by preference_order or by lower-to-higher summed correlations, and sequentially selects predictors with pairwise correlations below max_cor.

Variance Inflation Factors

VIF for predictor a is computed as $1/(1-R^2)$, where R^2 is the multiple R-squared from regressing a on the other predictors. Recommended maximums commonly used are 2.5, 5, and 10.

VIF-based Filtering

vif_select ranks numeric predictors (user preference_order if provided, otherwise from lower to higher VIF) and sequentially adds predictors whose VIF against the current selection is below max_vif.

Author(s)

Blas M. Benito, PhD

See Also

Other multicollinearity_filtering: collinear(), cor_select(), step_collinear(), vif_select()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

x <- collinear_select(
  df = vi_smol,
  predictors = c(
  "koppen_zone",           #character
  "soil_type",             #factor
  "topo_elevation",        #numeric
  "soil_temperature_mean"  #numeric
    ),
  max_cor = 0.7,
  max_vif = 5
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

---

collinear_stats *Compute summary statistics for correlation and VIF*

Description

Computes the the minimum, mean, maximum, and quantiles 0.05, 0.25, median (0.5), 0.75, and 0.95 of the correlations and variance inflation factors in a given dataframe. Wraps the functions cor_stats() and vif_stats()

Usage

```
collinear_stats(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with predictors or the output of cor_df(). Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

dataframe with columns method (with values "correlation" and "vif"), statistic and value

See Also

Other multicollinearity_assessment: cor_clusters(), cor_cramer(), cor_df(), cor_matrix(), cor_stats(), vif(), vif_df(), vif_stats()

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

x <- collinear_stats(
  df = vi_smol,
  predictors = vi_predictors_numeric
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`cor_clusters` *Group predictors by hierarchical correlation clustering*

Description

Hierarchical clustering of predictors from their correlation matrix. Computes the correlation matrix with `cor_df()` and `cor_matrix()`, transforms it to a distance matrix using `stats::dist()`, computes a clustering solution with `stats::hclust()`, and applies `stats::cutree()` to separate groups based on the value of the argument `max_cor`.

Returns a dataframe with predictor names and their clusters, and optionally, prints a dendrogram of the clustering solution.

Accepts a parallelization setup via `future::plan()` and a progress bar via `progressr::handlers()` (see examples).

Usage

```
cor_clusters(
  df = NULL,
  predictors = NULL,
  max_cor = 0.7,
  method = "complete",
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with predictors or the output of `cor_df()`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`max_cor`	(optional; numeric or NULL) Correlation value used to separate clustering groups. Valid values are between 0.01 and 0.99. Default: 0.7
`method`	(optional, character string) Argument of `stats::hclust()` defining the agglomerative method. One of: "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Unambiguous abbreviations are accepted as well. Default: "complete".
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

list:

- df: dataframe with predictor names and their cluster IDs.
- hclust: clustering object

See Also

Other multicollinearity_assessment: collinear_stats(), cor_cramer(), cor_df(), cor_matrix(), cor_stats(), vif(), vif_df(), vif_stats()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

#group predictors using max_cor as clustering threshold
clusters <- cor_clusters(
  df = vi_smol,
  predictors = c(
    "koppen_zone", #character
    "soil_type", #factor
    "topo_elevation", #numeric
    "soil_temperature_mean" #numeric
  ),
  max_cor = 0.75
)

#clusters dataframe
clusters$df

##plot hclust object
# graphics::plot(clusters$hclust)

##plot max_cor threshold
# graphics::abline(
#   h = 1 - 0.75,
#   col = "red4",
#   lty = 3,
#   lwd = 2
# )
```

```
## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`cor_cramer` *Quantify association between categorical variables*

Description

Cramer's V extends the chi-squared test to quantify how strongly the categories of two variables co-occur. The value ranges from 0 to 1, where 0 indicates no association and 1 indicates perfect association.

This function implements a bias-corrected version of Cramer's V, which adjusts for sample size and is more accurate for small samples. However, this bias correction means that even for binary variables, Cramer's V will no equal the Pearson correlation (the standard, uncorrected Cramer's V does match Pearson for binary data).

As the number of categories increases, Cramer's V and Pearson correlation measure increasingly different aspects of association and should not be directly compared.

If you intend to combine these measures in a multicollinearity analysis, interpret them with care. It is often preferable to convert non-numeric variables to numeric form (for example, via target encoding) before assessing multicollinearity.

Usage

```
cor_cramer(x = NULL, y = NULL, check_input = TRUE, ...)
```

Arguments

`x`	(required; vector) Values of a categorical variable (character or vector). Converted to character if numeric or logical. Default: NULL
`y`	(required; vector) Values of a categorical variable (character or vector). Converted to character if numeric or logical. Default: NULL
`check_input`	(required; logical) If FALSE, disables data checking for a slightly faster execution. Default: TRUE
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

numeric: Cramer's V

Author(s)

Blas M. Benito, PhD

References

- Cramér, H. (1946). Mathematical Methods of Statistics. Princeton: Princeton University Press, page 282 (Chapter 21. The two-dimensional case). ISBN 0-691-08004-6

See Also

Other multicollinearity_assessment: collinear_stats(), cor_clusters(), cor_df(), cor_matrix(), cor_stats(), vif(), vif_df(), vif_stats()

Examples

```
# perfect one-to-one association
cor_cramer(
  x = c("a", "a", "b", "c"),
  y = c("a", "a", "b", "c")
)

# still perfect: labels differ but mapping is unique
cor_cramer(
  x = c("a", "a", "b", "c"),
  y = c("a", "a", "b", "d")
)

# high but < 1: mostly aligned, one category of y repeats
cor_cramer(
  x = c("a", "a", "b", "c"),
  y = c("a", "a", "b", "b")
)

# appears similar by position, but no association by distribution
# (x = "a" mixes with y = "a" and "b")
cor_cramer(
  x = c("a", "a", "a", "c"),
  y = c("a", "a", "b", "b")
)

# numeric inputs are coerced to character internally
cor_cramer(
  x = c(1, 1, 2, 3),
  y = c(1, 1, 2, 2)
)

# logical inputs are also coerced to character
cor_cramer(
  x = c(TRUE, TRUE, FALSE, FALSE),
  y = c(TRUE, TRUE, FALSE, FALSE)
)
```

cor_df *Compute signed pairwise correlations dataframe*

Description

Computes pairwise correlations between predictors using appropriate methods for different variable types:

- **Numeric vs. Numeric**: Pearson correlation via `stats::cor()`.
- **Numeric vs. Categorical**: Target-encodes the categorical variable using the numeric variable as reference via `target_encoding_lab()` with leave-one-out method, then computes Pearson correlation.
- **Categorical vs. Categorical**: Cramer's V via `cor_cramer()` as a measure of association. See `cor_cramer()` for important notes on mixing Pearson correlation and Cramer's V in multicollinearity analysis.

Parallelization via `future::plan()` and progress bars via `progressr::handlers()` are supported but only beneficial for large datasets with categorical predictors. Numeric-only correlations do not use parallelization or progress bars. Example: With 16 workers, 30k rows (dataframe spatialData::vi), 49 numeric and 12 categorical predictors (see spatialData::vi_predictors), parallelization achieves a 5.4x speedup (147s → 27s).

Usage

```
cor_df(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

dataframe with columns:

- `x`: character, first predictor name.
- `y`: character, second predictor name.
- `correlation`: numeric, Pearson correlation (numeric vs. numeric and numeric vs. categorical) or Cramer's V (categorical vs. categorical).

See Also

Other multicollinearity_assessment: collinear_stats(), cor_clusters(), cor_cramer(), cor_matrix(), cor_stats(), vif(), vif_df(), vif_stats()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

#predictors
predictors = c(
  "koppen_zone", #character
  "soil_type", #factor
  "topo_elevation", #numeric
  "soil_temperature_mean" #numeric
)

x <- cor_df(
  df = vi_smol,
  predictors = predictors
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

---

cor_matrix *Signed pairwise correlation matrix*

Description

Computes a square matrix of pairwise correlations for a set of numeric and/or categorical predictors.

If df is already a correlation dataframe generated by cor_df()), the function transforms it into a correlation matrix. Otherwise, cor_df() is used internally to compute pairwise correlations before generating the matrix.

Supports parallel computation via future::plan() and optional progress reporting via progressr::handlers().

Usage

```
cor_matrix(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with predictors or the output of cor_df(). Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

correlation matrix

Author(s)

Blas M. Benito, PhD

See Also

Other multicollinearity_assessment: collinear_stats(), cor_clusters(), cor_cramer(), cor_df(), cor_stats(), vif(), vif_df(), vif_stats()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

predictors <- c(
  "koppen_zone", #character
  "soil_type", #factor
  "topo_elevation", #numeric
  "soil_temperature_mean" #numeric
)

#from dataframe with predictors
```

```
x <- cor_matrix(
  df = vi_smol,
  predictors = predictors
)

x

#from correlation dataframe
x <- cor_df(
  df = vi_smol,
  predictors = predictors
) |>
  cor_matrix()

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`cor_select` *Multicollinearity filtering by pairwise correlation threshold*

Description

Wraps `collinear_select()` to automatize multicollinearity filtering via pairwise correlation in dataframes with numeric and categorical predictors.

The argument `max_cor` determines the maximum variance inflation factor allowed in the resulting selection of predictors.

The argument `preference_order` accepts a character vector of predictor names ranked from first to last index, or a dataframe resulting from `preference_order()`. When two predictors in this vector or dataframe are highly collinear, the one with a lower ranking is removed. This option helps protect predictors of interest. If not provided, predictors are ranked from lower to higher multicollinearity.

Please check the section **Pairwise Correlation Filtering** at the end of this help file for further details.

Usage

```
cor_select(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  max_cor = 0.7,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional; character or NULL) Name of one response variable in df. Used to exclude columns when predictors is NULL, and to filter preference_order when it is a dataframe and contains several responses. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
preference_order	(optional; character vector, dataframe from preference_order, or NULL) Prioritizes predictors to preserve.
max_cor	(optional; numeric or NULL) Maximum correlation allowed between pairs of predictors. Valid values are between 0.01 and 0.99, and recommended values are between 0.5 (strict) and 0.9 (permissive). Default: 0.7
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

character vector of selected predictors

Pairwise Correlation Filtering

cor_select computes the global correlation matrix, orders predictors by preference_order or by lower-to-higher summed correlations, and sequentially selects predictors with pairwise correlations below max_cor.

Author(s)

Blas M. Benito, PhD

See Also

Other multicollinearity_filtering: collinear(), collinear_select(), step_collinear(), vif_select()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
```

```
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

#predictors
predictors = c(
  "koppen_zone", #character
  "soil_type", #factor
  "topo_elevation", #numeric
  "soil_temperature_mean" #numeric
)

#predictors ordered from lower to higher multicollinearity
x <- cor_select(
  df = vi_smol,
  predictors = predictors,
  max_cor = 0.7
)

x


#with custom preference order
x <- cor_select(
  df = vi_smol,
  predictors = predictors,
  preference_order = c(
    "koppen_zone",
    "soil_type"
  ),
  max_cor = 0.7
)

x

#with automated preference order
df_preference <- preference_order(
  df = vi_smol,
  response = "vi_numeric",
  predictors = predictors
)

df_preference

x <- cor_select(
  df = vi_smol,
  predictors = predictors,
  preference_order = df_preference,
  max_cor = 0.7
)

x
```

```
#OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`cor_stats` *Compute summary statistics for absolute pairwise correlations*

Description

Computes the the minimum, mean, maximum, and quantiles 0.05, 0.25, median (0.5), 0.75, and 0.95 on the absolute values of the column "correlation" in the output of `cor_df()`.

Usage

```
cor_stats(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with predictors or the output of `cor_df()`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

dataframe with columns `method` (with value "correlation"), `statistic` and `value`

See Also

Other multicollinearity_assessment: `collinear_stats()`, `cor_clusters()`, `cor_cramer()`, `cor_df()`, `cor_matrix()`, `vif()`, `vif_df()`, `vif_stats()`

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
```

```
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

x <- cor_stats(
  df = vi_smol,
  predictors = vi_predictors_numeric
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`drop_geometry_column` *Removes* `geometry` *Column From* `sf` *Dataframes*

Description

Remove geometry column from sf objects

Usage

```
drop_geometry_column(df = NULL, quiet = FALSE, ...)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

dataframe

Author(s)

Blas M. Benito, PhD

See Also

Other argument_validation: validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet(), validate_arg_responses()

Examples

```
data(vi_smol, package = "spatialData")

#creating fake geometry column without sf loaded
vi_smol$geometry <- NA
attr(
  x = vi_smol,
  which = "sf_column"
  ) <- "geometry"

#check new attribute
attributes(vi_smol)$sf_column

#drop geometry column
df <- drop_geometry_column(
  df = vi_smol
  )

#checking that the geometry was droppped
"geometry" %in% colnames(df)
attributes(df)$sf_column
```

---

experiment_adaptive_thresholds

Dataframe resulting from experiment to test the automatic selection of multicollinearity thresholds

---

Description

A dataframe summarizing 10,000 experiments validating the adaptive multicollinearity threshold system in collinear(). Each row records input data characteristics and the resulting multicollinearity metrics after filtering.

Usage

```
data(experiment_adaptive_thresholds)
```

Format

A dataframe with 10,000 rows and 9 variables:

input_rows Number of rows in the input data subset.

input_predictors Number of predictors in the input data subset.

output_predictors Number of predictors retained after filtering.

input_cor_q75 75th percentile of pairwise correlations in the input data.

output_cor_q75 75th percentile of pairwise correlations in the selected predictors.

input_cor_max Maximum pairwise correlation in the input data.

output_cor_max Maximum pairwise correlation in the selected predictors.

input_vif_max Maximum VIF in the input data.

output_vif_max Maximum VIF in the selected predictors.

Details

The source data is a synthetic dataframe with 500 columns and 10,000 rows generated using `distantia::zoo_simulate()` with correlated time series (`independent = FALSE`, `seasons = 0`).

Each iteration randomly subsets 10-100 predictors and 30-100 rows per predictor, then applies `collinear()` with automatic threshold configuration to assess:

- Whether output VIF stays bounded between ~2.5 and ~7.5
- How the system adapts to different correlation structures
- How predictor retention scales with input size

See Also

Other experiments: `experiment_cor_vs_vif`, `gam_cor_to_vif`, `prediction_cor_to_vif`

Examples

```
data(experiment_adaptive_thresholds)
str(experiment_adaptive_thresholds)
```

---

experiment_cor_vs_vif *Dataframe with results of experiment comparing correlation and VIF thresholds*

Description

A dataframe summarizing 10,000 experiments comparing the output of `cor_select()` and `vif_select()`. Each row records the input sampling parameters and the resulting feature-selection metrics.

Usage

```
data(experiment_cor_vs_vif)
```

Format

A dataframe with 10,000 rows and 6 variables:

input_rows Number of rows in the input data subset.

input_predictors Number of predictors in the input data subset.

output_predictors Number of predictors selected by `vif_select()` at the best-matching `max_vif`.

max_cor Maximum allowed pairwise correlation supplied to `cor_select()`.

max_vif VIF threshold at which `vif_select()` produced the highest Jaccard similarity with `cor_select()` for the given `max_cor`.

out_selection_jaccard Jaccard similarity between the predictors selected by `cor_select()` and `vif_select()`.

Details

The source data is a synthetic dataframe with 500 columns and 10,000 rows generated using `distantia::zoo_simulate()` with correlated time series (`independent = FALSE`).

Each iteration randomly subsets 10-50 predictors and 30-100 rows per predictor, applies `cor_select()` with a random `max_cor` threshold, then finds the `max_vif` value that maximizes Jaccard similarity between the two selections.

See Also

Other experiments: `experiment_adaptive_thresholds`, `gam_cor_to_vif`, `prediction_cor_to_vif`

Examples

```
data(experiment_cor_vs_vif)
str(experiment_cor_vs_vif)
```

---

f_auto *Automatic selection of predictor scoring method*

Description

Internal function to select a proper `f_...()` function to compute preference order depending on the types of the response variable and the predictors. The selection criteria is available as a dataframe generated by `f_auto_rules()`.

Usage

```
f_auto(df = NULL, response = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional, character string) Name of a response variable in df. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

function name

See Also

Other preference_order_tools: f_auto_rules(), f_functions()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid
vi_predictors_categorical <- identify_categorical_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

f_auto(
  df = vi_smol,
  response = "vi_numeric",
  predictors = vi_predictors_numeric
  )

f_auto(
  df = vi_smol,
  response = "vi_binomial",
  predictors = vi_predictors_numeric
  )

f_auto(
  df = vi_smol,
  response = "vi_categorical",
  predictors = vi_predictors_categorical
  )
```

f_auto_rules *Decision rules for* f_auto()

Description

Dataframe with rules used by f_auto() to select the function f in f_functions() to compute preference order in preference_order(). In most cases, random forest is selected as base model to provide homogeneous results across case types.

Usage

```
f_auto_rules()
```

Value

dataframe

See Also

Other preference_order_tools: f_auto(), f_functions()

Examples

```
f_auto_rules()
```

f_binomial_gam *Area under the curve of binomial GAM predictions vs. observations*

Description

Fits a Quasibinomial GAM model y ~ s(x) (y ~ x if x is non-numeric) with the binomial response y (values 0 and 1) and the numeric, character or factor predictor x using mgcv::gam() and returns the area under the ROC curve between the observed responses and the model predictions (see score_auc()).

Cases are weighted with case_weights() to prevent issues arising from class imbalance.

Supports cross-validation via the arguments arguments cv_training_fraction (numeric between 0 and 1) and cv_iterations (integer between 1 and n) introduced via ellipsis (...). See preference_order() for further details.

Usage

```
f_binomial_gam(df, ...)
```

Arguments

df	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) binomial response with unique values 0 and 1.
...	(optional) Accepts the arguments cv_training_fraction (numeric between 0 and 1) and cv_iterations (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: AUC

See Also

Other preference_order_functions: f_binomial_glm(), f_binomial_rf(), f_categorical_rf(), f_count_gam(), f_count_glm(), f_count_rf(), f_numeric_gam(), f_numeric_glm(), f_numeric_rf(), preference_order()

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_binomial_gam(df = df)

#cross-validation
f_binomial_gam(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["koppen_zone"]]
)

f_binomial_gam(df = df)
```

f_binomial_glm *Area Under the Curve of Binomial GLM predictions vs. observations*

Description

Fits a Quasibinomial GLM model `y ~ x` with the binomial response `y` (values 0 and 1) and the numeric, character, or factor predictor `x` using `stats::glm()` and returns the area under the ROC curve of the observations against the predictions (see `score_auc()`).

Cases are weighted with `case_weights()` to prevent issues arising from class imbalance.

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and n) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_binomial_glm(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) binomial response with unique values 0 and 1.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: AUC

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_gam()`, `f_count_glm()`, `f_count_rf()`, `f_numeric_gam()`, `f_numeric_glm()`, `f_numeric_rf()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_binomial_glm(df = df)
```

```
#cross-validation
f_binomial_glm(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["koppen_zone"]]
)

f_binomial_glm(df = df)
```

`f_binomial_rf` *Area Under the Curve of Binomial Random Forest predictions vs. observations*

Description

Fits a univariate random forest model `y ~ x` with the binomial (values 0 and 1) response `y` and the numeric, character or factor predictor `x` using `ranger::ranger()` and returns the area under the ROC curve between the observed responses and the model predictions (see `score_auc()`).

Cases are weighted with `case_weights()` to prevent issues arising from class imbalance.

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_binomial_rf(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) binomial response with unique values 0 and 1.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: AUC

See Also

Other preference_order_functions: f_binomial_gam(), f_binomial_glm(), f_categorical_rf(), f_count_gam(), f_count_glm(), f_count_rf(), f_numeric_gam(), f_numeric_glm(), f_numeric_rf(), preference_order()

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_binomial_rf(df = df)

#cross-validation
f_binomial_rf(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_binomial"]],
  x = vi_smol[["koppen_zone"]]
)

f_binomial_rf(df = df)
```

---

f_categorical_rf *Cramer's V of Categorical Random Forest predictions vs. observations*

Description

Fits a univariate random forest model `y ~ x` with the character or factor response `y` and the numeric, character or factor predictor `x` using `ranger::ranger()` and returns the Cramer's V (see cor_cramer()) between the observed responses and the model predictions. Cases are weighted with case_weights() to prevent issues arising from class imbalance.

Cases are weighted with case_weights() to prevent issues arising from class imbalance.

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See preference_order() for further details.

Usage

```
f_categorical_rf(df, ...)
```

Arguments

df	(required, dataframe) with columns: • x: (numeric) numeric, character, or factor predictor. • y (numeric) character or factor response.
...	(optional) Accepts the arguments cv_training_fraction (numeric between 0 and 1) and cv_iterations (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: Cramer's V

See Also

Other preference_order_functions: f_binomial_gam(), f_binomial_glm(), f_binomial_rf(), f_count_gam(), f_count_glm(), f_count_rf(), f_numeric_gam(), f_numeric_glm(), f_numeric_rf(), preference_order()

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_factor"]],
  x = vi_smol[["soil_type"]]
)

#no cross-validation
f_categorical_rf(df = df)

#cross-validation
f_categorical_rf(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#numeric predictor
df <- data.frame(
  y = vi_smol[["vi_categorical"]],
  x = vi_smol[["swi_max"]]
)

f_categorical_rf(df = df)
```

f_count_gam *R-squared of Poisson GAM predictions vs. observations*

Description

Fits a Poisson GAM model `y ~ s(x)` (`y ~ x` if `x` is non-numeric) with the numeric response `y` and the numeric, character or factor predictor `x` using `mgcv::gam()` and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_count_gam(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) counts response.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_glm()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_glm()`, `f_count_rf()`, `f_numeric_gam()`, `f_numeric_glm()`, `f_numeric_rf()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_counts"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_count_gam(df = df)

#cross-validation
```

```
f_count_gam(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )
```

`f_count_glm` *R-squared of Poisson GLM predictions vs. observations*

Description

Fits a Poisson GLM model `y ~ x` with the numeric response `y` and the numeric predictor `x` using `stats::glm()` and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_count_glm(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) counts response.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_glm()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_gam()`, `f_count_rf()`, `f_numeric_gam()`, `f_numeric_glm()`, `f_numeric_rf()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_counts"]],
  x = vi_smol[["swi_max"]]
)
```

```
#no cross-validation
f_count_glm(df = df)

#cross-validation
f_count_glm(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_counts"]],
  x = vi_smol[["koppen_zone"]]
)

f_count_glm(df = df)
```

`f_count_rf` *R-squared of Random Forest predictions vs. observations*

Description

Fits a univariate random forest model `y ~ x` with the integer response `y` and the numeric, character or factor predictor `x` using `ranger::ranger()` and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_count_rf(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • "x": (numeric, character, factor) predictor. • "y" (integer) counts response.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: f_binomial_gam(), f_binomial_glm(), f_binomial_rf(), f_categorical_rf(), f_count_gam(), f_count_glm(), f_numeric_gam(), f_numeric_glm(), f_numeric_rf(), preference_order()

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_counts"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_count_rf(df = df)

#cross-validation
f_count_rf(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_counts"]],
  x = vi_smol[["koppen_zone"]]
)

f_count_rf(df = df)
```

---

f_functions *List predictor scoring functions*

Description

List predictor scoring functions

Usage

```
f_functions()
```

Value

dataframe

See Also

Other preference_order_tools: `f_auto()`, `f_auto_rules()`

Examples

```
f_functions()
```

`f_numeric_gam` *R-squared of Gaussian GAM predictions vs. observations*

Description

Fits a Gaussian GAM model `y ~ s(x)` (`y ~ x` if `x` is non-numeric) with the numeric response `y` and the numeric, character or factor predictor `x` using `mgcv::gam()` and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and n) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_numeric_gam(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • `x`: (numeric, character, factor) predictor. • `y` (numeric) continuous response.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_glm()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_gam()`, `f_count_glm()`, `f_count_rf()`, `f_numeric_glm()`, `f_numeric_rf()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_numeric_gam(df = df)

#cross-validation
f_numeric_gam(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["koppen_zone"]]
)

f_numeric_gam(df = df)
```

`f_numeric_glm` *R-squared of Gaussian GLM predictions vs. observations*

Description

Fits a Gaussian GLM model `y ~ x` with the numeric response `y` and the numeric, character, or factor predictor `x` using `stats::glm()` and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and n) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_numeric_glm(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • `x`: (numeric, character, factor) predictor.

- y (numeric) continuous response.

... (optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_glm()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_gam()`, `f_count_glm()`, `f_count_rf()`, `f_numeric_gam()`, `f_numeric_rf()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_numeric_glm(df = df)

#cross-validation
f_numeric_glm(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )

#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["koppen_zone"]]
)

f_numeric_glm(df = df)
```

`f_numeric_rf` *R-squared of Random Forest predictions vs. observations*

Description

Fits a univariate random forest model `y ~ x` with the numeric response `y` and the numeric, character or factor predictor `x` using `ranger::ranger()` and returns the R-squared between the observed responses and returns the R-squared of the observations against the predictions (see `score_r2()`).

Supports cross-validation via the arguments arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and `n`) introduced via ellipsis (`...`). See `preference_order()` for further details.

Usage

```
f_numeric_rf(df, ...)
```

Arguments

`df`	(required, dataframe) with columns: • `x`: (numeric, character, factor) predictor. • `y` (numeric) continuous response.
`...`	(optional) Accepts the arguments `cv_training_fraction` (numeric between 0 and 1) and `cv_iterations` (integer between 1 and Inf) for cross validation.

Value

numeric or numeric vector: R-squared

See Also

Other preference_order_functions: `f_binomial_gam()`, `f_binomial_glm()`, `f_binomial_rf()`, `f_categorical_rf()`, `f_count_gam()`, `f_count_glm()`, `f_count_rf()`, `f_numeric_gam()`, `f_numeric_glm()`, `preference_order()`

Examples

```
data(vi_smol, package = "spatialData")

df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["swi_max"]]
)

#no cross-validation
f_numeric_rf(df = df)

#cross-validation
f_numeric_rf(
  df = df,
  cv_training_fraction = 0.5,
  cv_iterations = 10
  )
```

```
#categorical predictor
df <- data.frame(
  y = vi_smol[["vi_numeric"]],
  x = vi_smol[["koppen_zone"]]
)

f_numeric_rf(df = df)
```

`gam_cor_to_vif` *GAM describing the relationship between correlation and VIF thresholds*

Description

A fitted generalized additive model describing `max_vif` as a function of `max_cor` in experiment_cor_vs_vif.

Usage

```
data(gam_cor_to_vif)
```

Format

A `gam` object.

Details

The model parameters (basis dimension `k` and weight exponent) were selected via optimization, filtering for models in the top 90\

The final model uses squared Jaccard similarity as weights to emphasize cases with high agreement between `cor_select()` and `vif_select()`.

Model performance:

- Adjusted R-squared: 0.834
- Deviance explained: 83.4\
- Effective degrees of freedom for smooth: ~6

Source

Generated internally from experiment_cor_vs_vif.

See Also

Other experiments: `experiment_adaptive_thresholds`, `experiment_cor_vs_vif`, `prediction_cor_to_vif`

Examples

```
data(gam_cor_to_vif)
plot(gam_cor_to_vif, shade = TRUE)
```

identify_categorical_variables *Find valid categorical variables in a dataframe*

Description

Identifies valid and invalid character or factor variables. Invalid categorical predictors are those with a single category, or as many categories as cases (full-cardinality).

Usage

```
identify_categorical_variables(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
predictors	(required, character vector) Names of the predictors to identify. Default: NULL
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

list:

- valid: character vector with valid categorical predictor names.
- invalid: character vector with invalid categorical predictor names due to degenerate cardinality (1 or nrow(df) categories).

Author(s)

Blas M. Benito, PhD

See Also

Other data_types: identify_logical_variables(), identify_numeric_variables(), identify_response_type(), identify_valid_variables(), identify_zero_variance_variables()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

#create an invalid categorical
vi_smol$invalid_categorical <- "a"

x <- identify_categorical_variables(
  df = vi_smol,
  responses = "vi_categorical",
  predictors = vi_predictors
)

x$valid
x$invalid
```

`identify_logical_variables`

Find logical variables in a dataframe

Description

Identifies logical predictors and excludes those with constant values.

Usage

```
identify_logical_variables(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`responses`	(optional; character, character vector, or NULL) Name of one or several response variables in `df`. Default: NULL.
`predictors`	(required, character vector) Names of the predictors to identify. Default: NULL
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

list:

- `valid`: character vector with valid logical predictor names.
- `invalid`: character vector with invalid logical predictor names.

Author(s)

Blas M. Benito, PhD

See Also

Other data_types: `identify_categorical_variables()`, `identify_numeric_variables()`, `identify_response_type()`, `identify_valid_variables()`, `identify_zero_variance_variables()`

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

#invalid logical
vi_smol$logical_invalid <- TRUE

#valid logical
vi_smol$logical_valid <- sample(
  x = c(TRUE, FALSE),
  size = nrow(vi_smol),
  replace = TRUE
)

x <- identify_logical_variables(
  df = vi_smol,
  predictors = c(
    vi_predictors,
    "logical_invalid",
    "logical_valid"
  )
)

x$valid
x$invalid
```

---

identify_numeric_variables

Find valid numeric variables in a dataframe

Description

Identifies valid numeric variables and ignores those with constant values.

Usage

```
identify_numeric_variables(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  decimals = 4,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
predictors	(required, character vector) Names of the predictors to identify. Default: NULL
decimals	(required, integer) Number of decimal places for the zero variance test. Smaller numbers will increase the number of variables detected as near-zero variance. Recommended values will depend on the range of the numeric variables in 'df'. Default: 4
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

list:

- valid: character vector with valid numeric predictor names.
- invalid: character vector with invalid numeric predictor names due to near-zero variance.

Author(s)

Blas M. Benito, PhD

See Also

Other data_types: identify_categorical_variables(), identify_logical_variables(), identify_response_type(
identify_valid_variables(), identify_zero_variance_variables()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

x <- identify_numeric_variables(
  df = vi_smol,
```

```
  responses = "vi_numeric",
  predictors = vi_predictors
)

#valid numeric predictors
x$valid

#invalid due to zero variance (none here)
x$invalid
```

---

identify_response_type

Detect response variable type for model selection

Description

Used by f_auto() to identify the type of a response variable and select a proper modelling method to compute preference order. Supported types are:

- "continuous-binary": decimal numbers and two unique values; results in a warning, as this type is difficult to model.
- "continuous-low": decimal numbers and 3 to 5 unique values; results in a message, as this type is difficult to model.
- "continuous-high": decimal numbers and more than 5 unique values.
- "integer-binomial": integer with 0s and 1s, suitable for binomial models.
- "integer-binary": integer with 2 unique values other than 0 and 1; returns a warning, as this type is difficult to model.
- "integer-low": integer with 3 to 5 unique values or meets specified thresholds.
- "integer-high": integer with more than 5 unique values suitable for count modelling.
- "categorical": character or factor with 2 or more levels.
- "unknown": when the response type cannot be determined.

Usage

```
identify_response_type(df = NULL, response = NULL, quiet = FALSE, ...)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional, character string) Name of a response variable in df. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

character string: response type

See Also

Other data_types: identify_categorical_variables(), identify_logical_variables(), identify_numeric_variab
identify_valid_variables(), identify_zero_variance_variables()

Examples

```
data(vi_smol, package = "spatialData")

identify_response_type(
  df = vi_smol,
  response = "vi_numeric"
)

identify_response_type(
  df = vi_smol,
  response = "vi_counts"
)

identify_response_type(
  df = vi_smol,
  response = "vi_binomial"
)

identify_response_type(
  df = vi_smol,
  response = "vi_categorical"
)

identify_response_type(
  df = vi_smol,
  response = "vi_factor"
)
```

---

identify_valid_variables

Find valid numeric, categorical, and logical variables in a dataframe

---

Description

Returns a list with the names of the valid numeric, categorical, and logical variables in a modelling dataframe.

Usage

```
identify_valid_variables(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  decimals = 4,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
predictors	(required, character vector) Names of the predictors to identify. Default: NULL
decimals	(required, integer) Number of decimal places for the zero variance test. Smaller numbers will increase the number of variables detected as near-zero variance. Recommended values will depend on the range of the numeric variables in 'df'. Default: 4
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

list

- numeric: character vector of numeric predictors.
- categorical: character vector of categorical (character and factor) predictors.
- logical: character vector of logical predictors.

Author(s)

Blas M. Benito, PhD

See Also

Other data_types: identify_categorical_variables(), identify_logical_variables(), identify_numeric_variab
identify_response_type(), identify_zero_variance_variables()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

x <- identify_valid_variables(
  df = vi_smol,
  predictors = vi_predictors
)

x
```

`identify_zero_variance_variables` *Find near-zero variance variables in a dataframe*

Description

Returns the names of near-zero variance variables in a modelling dataframe.

Usage

```
identify_zero_variance_variables(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  decimals = 4,
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`responses`	(optional; character, character vector, or NULL) Name of one or several response variables in `df`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`decimals`	(required, integer) Number of decimal places for the zero variance test. Smaller numbers will increase the number of variables detected as near-zero variance. Recommended values will depend on the range of the numeric variables in 'df'. Default: 4
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

character vector: names of near-zero variance columns.

Author(s)

Blas M. Benito, PhD

See Also

Other data_types: identify_categorical_variables(), identify_logical_variables(), identify_numeric_variab
identify_response_type(), identify_valid_variables()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

#create zero and near variance predictors
vi_smol$zero_variance <- 1
vi_smol$near_zero_variance <- runif(
  n = nrow(vi_smol),
  min = 0,
  max = 0.0001
  )


#add to vi predictors
vi_predictors <- c(
  vi_predictors,
  "zero_variance",
  "near_zero_variance"
)

#identify zero variance predictors
x <- identify_zero_variance_variables(
  df = vi_smol,
  predictors = vi_predictors
)

x
```

---

model_formula *Build model formulas from response and predictors*

Description

Generates model formulas from a dataframe, a response name, and a vector of predictors that can be the output of a multicollinearity management function such as collinear_select() and the likes. Intended to help fit exploratory models from the result of a multicollinearity analysis.

The types of formulas it can generate are:

- additive: `y ~ x + z`
- polynomial: `y ~ poly(x, ...) + poly(z, ...)`
- GAM: `y ~ s(x) + s(z)`
- random effect: `y ~ x + (1 \ z)`

Usage

```
model_formula(
  df = NULL,
  response = NULL,
  predictors = NULL,
  term_f = NULL,
  term_args = NULL,
  random_effects = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`response`	(optional, character string) Name of a response variable in `df`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`term_f`	(optional; string). Name of function to apply to each term in the formula, such as "s" for `mgcv::s()` or any other smoothing function, "poly" for `stats::poly()`. Default: NULL
`term_args`	(optional; string). Arguments of the function applied to each term. For example, for "poly" it can be "degree = 2, raw = TRUE". Default: NULL
`random_effects`	(optional, string or character vector). Names of variables to be used as random effects. Each element is added to the final formula as `+(1 | random_effect_name)`. Default: NULL
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

list if `predictors` is a list or length of `response` is higher than one, and character vector otherwise.

See Also

Other modelling_tools: `case_weights()`, `score_auc()`, `score_cramer()`, `score_r2()`

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

#reduce collinearity
x <- collinear_select(
  df = vi_smol,
  predictors = vi_predictors_numeric
)

#additive formula
y <- model_formula(
  df = vi_smol,
  response = "vi_numeric",
  predictors = x
)

y

#using a formula in a model
m <- stats::lm(
 formula = y,
 data = vi_smol
 )

summary(m)

#classification formula (character response)
y <- model_formula(
  df = vi_smol,
  response = "vi_categorical",
  predictors = x
)

y


#polynomial formula (3rd degree)
y <- model_formula(
  df = vi_smol,
  response = "vi_numeric",
  predictors = x,
  term_f = "poly",
  term_args = "degree = 3, raw = TRUE"
)

y
```

```
#gam formula
y <- model_formula(
  df = vi_smol,
  response = "vi_numeric",
  predictors = x,
  term_f = "s"
)

y

#random effect
y <- model_formula(
  df = vi_smol,
  response = "vi_numeric",
  predictors = x,
  random_effects = "country_name" #from vi_smol$country_name
)

y
```

`prediction_cor_to_vif` *Prediction of the model* `gam_cor_to_vif` *across correlation values*

Description

Dataframe with predicted VIF threshold corresponding to a given correlation threshold..

Usage

```
data(prediction_cor_to_vif)
```

Format

A dataframe with 901 rows and 2 numeric columns:

max_cor Maximum allowed pairwise correlation, from 0.10 to 1.00 in steps of 0.001.

max_vif Predicted VIF threshold corresponding to each `max_cor`.

Details

Values were generated by applying `mgcv::predict.gam()` to the fitted model gam_cor_to_vif and rounding to three decimal places.

See Also

Other experiments: `experiment_adaptive_thresholds`, `experiment_cor_vs_vif`, `gam_cor_to_vif`

Examples

```
data(prediction_cor_to_vif)
head(prediction_cor_to_vif)
plot(max_vif ~ max_cor, data = prediction_cor_to_vif, type = "l")
```

`preference_order` *Rank predictors by importance or multicollinearity*

Description

Generates a valid input for the argument `preference_order` of the functions `vif_select()`, `cor_select()`, `collinear_select()`, and `collinear()`. This argument helps preserve important predictors during multicollinearity filtering.

The function works in two different ways:

- When `f` is NULL, it ranks the predictors from lower to higher multicollinearity, computed as one minus the average Pearson correlation between the given predictor against all others. This option is useful when the goal is to limit redundancy in a large dataset and there is not an specific model to train in mind.
- When `responses` and `f` are not NULL, it ranks the predictors by the strength of their association with a response based on the evaluation of univariate models. This is the best possible option when the end-goal is training a model.

The argument `f` (requires a valid `resopnses` argument) defines how the strength of association between the response and each predictor is computed. By default it calls `f_auto()`, which uses `f_auto_rules()` to select a suitable function depending on the types of the response and the predictors. This option is designed to provide sensible, general-purpose defaults optimized for speed and stability rather than any specific modeling approach.

For more fine-tuned control, the package offers the following `f` functions (see `f_functions()`):

- **Numeric response**:
  - `f_numeric_glm()`: Pearson's R-squared of response versus the predictions of a Gaussian GLM.
  - `f_numeric_gam()`: GAM model fitted with `mgcv::gam()`.
  - `f_numeric_rf()`: Random Forest model fitted with `ranger::ranger()`.
- **Integer counts response**:
  - `f_count_glm()`: Pearson's R-squared of a Poisson GLM.
  - `f_count_gam()`: Poisson GAM.
  - `f_count_rf()`: Random Forest model fitted with `ranger::ranger()`.
- **Binomial response (1 and 0)**:
  - `f_binomial_glm()`: AUC of Quasibinomial GLM with weighted cases.
  - `f_binomial_gam()`: AUC of Quasibinomial GAM with weighted cases.
  - `f_binomial_rf()`: AUC of a Random Forest model with weighted cases.
- **Categorical response**:

- `f_categorical_rf()`: Cramer's V of the response against the predictions of a classification Random Forest model.

These functions accept a cross-validation setup via the arguments `cv_iterations` and `cv_training_fraction`.

Additionally, the argument `f` accepts any custom function taking a dataframe with the columns "x" (predictor) and "y" (response) and returning a numeric indicator of association.

Accepts a parallelization setup via `future::plan()` and a progress bar via `progressr::handlers()` (see examples).

Accepts a character vector of response variables as input for the argument `responses`. When more than one response is provided, the output is a named list of preference data frames.

Usage

```
preference_order(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  f = f_auto,
  cv_training_fraction = 1,
  cv_iterations = 1,
  seed = 1,
  quiet = FALSE,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`responses`	(optional; character, character vector, or NULL) Name of one or several response variables in `df`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`f`	(optional: function name) Unquoted function name without parenthesis (see f_functions). By default calls to `f_auto()`, which selects a suitable function depending on the nature of the response and predictors. Set to NULL if `responses = NULL`. If NULL, predictors are ranked from lower to higher multicollinearity. Default: `f_auto`
`cv_training_fraction`	(optional, numeric) Value between 0.1 and 1 defining the training faction used in cross-validation. If 1 (default), no cross-validation is performed, and the resulting metric is computed from all observations and predictions. Automatically set to 1 when `cv_iterations = 1`. Default: 1
`cv_iterations`	(optional, integer) Number of cross-validation iterations to perform. The recommended range lies between 30 and 100. In general, smaller datasets and large

	values of cv_training_fraction require more iterations to achieve stability. Automatically set to 1 when cv_training_fraction = 1. Default: 1
seed	(optional, integer) Random seed, required for reproducibility when using cross-validation or random forest models. Default: 1
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

dataframe:

- response: character, response name, if any, or "none" otherwise.
- predictor: character, name of the predictor.
- f: name of the function used to compute the preference order. If argument f is NULL, the value "stats::cor()" is added to this column.
- metric: name of the metric used to assess strength of association. Usually one of "R-squared", "AUC" (Area Under the ROC Curve), or "Cramer's V". If f is a custom function not in f_functions(), then metric is set to "custom". If f is NULL, then "1 - R-squared" is returned in this column.
- score: value of the metric returned by f to assess the association between the response and each given predictor.
- rank: integer value indicating the rank of the predictor.

Author(s)

Blas M. Benito, PhD

See Also

Other preference_order_functions: f_binomial_gam(), f_binomial_glm(), f_binomial_rf(), f_categorical_rf(), f_count_gam(), f_count_glm(), f_count_rf(), f_numeric_gam(), f_numeric_glm(), f_numeric_rf()

Examples

```
#load example data
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

##OPTIONAL: parallelization setup
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
```

```
# )

##OPTIONAL: progress bar
##does not work in R examples
# progressr::handlers(global = TRUE)

#ranking predictors from lower to higher multicollinearity
#-------------------------------------------------
x <- preference_order(
  df = vi_smol,
  responses = NULL, #default value
  predictors = vi_predictors_numeric[1:10],
  f = NULL #must be explicit
)

x

#automatic selection of ranking function
#-------------------------------------------------
x <- preference_order(
  df = vi_smol,
  responses = c("vi_numeric", "vi_categorical"),
  predictors = vi_predictors_numeric[1:10],
  f = f_auto
  )

x

#user selection of ranking function
#-------------------------------------------------
#Poisson GLM for a integer counts response
x <- preference_order(
  df = vi_smol,
  responses = "vi_binomial",
  predictors = vi_predictors_numeric[1:10],
  f = f_binomial_glm
)

x

#cross-validation
#-------------------------------------------------
x <- preference_order(
  df = vi_smol,
  responses = "vi_binomial",
  predictors = vi_predictors_numeric[1:10],
  f = f_binomial_glm,
  cv_training_fraction = 0.5,
  cv_iterations = 10
)

x
```

```
#custom pairwise correlation function
#-----------------------------------------------
#custom functions need the ellipsis argument
f_rsquared <- function(df, ...){
    stats::cor(
      x = df$x,
      y = df$y,
      use = "complete.obs"
    )^2
}

x <- preference_order(
  df = vi_smol,
  responses = "vi_numeric",
  predictors = vi_predictors_numeric[1:10],
  f = f_rsquared
)

x

#resetting to sequential processing
#future::plan(future::sequential)
```

`print.collinear_output` *Print all collinear selection results of* `collinear()`

Description

Print all collinear selection results of `collinear()`

Usage

```
## S3 method for class 'collinear_output'
print(x = NULL, n = 5, ...)
```

Arguments

`x`	(required, list of class `class.collinear_output`) Object to print. Default: NULL
`n`	(optional, integer) Maximum printed vector length. Default: 5.
`...`	Ignored, kept for consistency with generic.

See Also

Other S3_methods: `print.collinear_selection()`, `summary.collinear_output()`, `summary.collinear_selection()`

print.collinear_selection

Print single selection results from collinear

Description

Print single selection results from collinear

Usage

```
## S3 method for class 'collinear_selection'
print(x = NULL, n = 5, ...)
```

Arguments

x	(required, sub-list in output of collinear()) Object to print. Default: NULL
n	(optional, integer) Maximum printed vector length. Default: 5.
...	Ignored, kept for consistency with generic.

See Also

Other S3_methods: print.collinear_output(), summary.collinear_output(), summary.collinear_selection()

score_auc

Compute area under the ROC curve between binomial observations and probabilistic predictions

Description

Internal function to compute the AUC of binomial models within preference_order(). Used within f_binomial_glm(), f_binomial_gam(), and f_binomial_rf(). This function is build for speed and it does not check the inputs.

Usage

```
score_auc(o = NULL, p = NULL, ...)
```

Arguments

o	(required, numeric vector) Binomial observations (values 0 and 1). Default: NULL
p	(required, numeric vector) Prediction of binomial model in the range 0-1. Default: NULL
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

numeric: Area Under the ROC Curve

See Also

Other modelling_tools: `case_weights()`, `model_formula()`, `score_cramer()`, `score_r2()`

Examples

```
score_auc(
  o = c(1, 1, 1, 1, 0, 0, 0),
  p = c(1, 0.8, 0.7, 0.6, 0.5, 0.6, 0.7)
)
```

`score_cramer` *Compute Cramer's V between categorical observations and predictions*

Description

Internal function to compute the Cramer's V of categorical observations versus categorical model predictions. Please read the help file of `cor_cramer()` for further details.

Usage

```
score_cramer(o = NULL, p = NULL, ...)
```

Arguments

`o`	(required; character vector) categorical observations. Default: NULL
`p`	(required; character vector) categorical predictions. Default: NULL
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

numeric: Cramer's V

See Also

Other modelling_tools: `case_weights()`, `model_formula()`, `score_auc()`, `score_r2()`

Examples

```
score_cramer(
 o = c("a", "a", "b", "c", "c"),
 p = c("a", "b", "b", "c", "c")
 )
```

score_r2 *Compute R-squared between numeric observations and predictions*

Description

Internal function to compute the R-squared of observations versus predictions via `stats::cor()`. Used within `f_numeric_glm()`, `f_numeric_gam()`, `f_numeric_rf()`, `f_count_glm()`, and `f_count_gam()`.

Usage

```
score_r2(o = NULL, p = NULL, ...)
```

Arguments

`o`	(required, numeric vector) Observations. Default: NULL
`p`	(required, numeric vector) Predictions. Default: NULL
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

numeric: Pearson R-squared

See Also

Other modelling_tools: `case_weights()`, `model_formula()`, `score_auc()`, `score_cramer()`

Examples

```
score_r2(
  o = c(1, 1, 1, 0.5, 0.5, 0, 0),
  p = c(1, 0.8, 0.7, 0.6, 0.5, 0.1, 0)
)
```

step_collinear *Tidymodels recipe step for multicollinearity filtering*

Description

Adds a step to a recipe created by `recipes::recipe()]+` to apply multicollinearity filtering via `collinear()`.

This function requires the `recipes` package to be installed.

Unlike `collinear()`, this wrapper does not perform target encoding, and the default value for the argument `quiet` is `TRUE`.

Usage

```
step_collinear(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  options = list(),
  selected = NULL,
  skip = FALSE,
  keep_original_cols = FALSE,
  id = recipes::rand_id("collinear")
)

## S3 method for class 'step_collinear'
prep(x = NULL, training = NULL, info = NULL, ...)

## S3 method for class 'step_collinear'
bake(object = NULL, new_data = NULL, ...)
```

Arguments

recipe	(required, recipe) A recipe object to which this step will be added.
...	(optional) Additional arguments (currently ignored).
role	(optional, character) Not used by this step since no new variables are created. Default: NA
trained	(optional, logical) Indicates if the step has been trained. Default: FALSE
options	(optional, list) Named list of arguments passed to collinear(). Common options include: • max_cor: Maximum correlation threshold. If NULL (default), automatically set based on median correlation of predictors. • max_vif: Maximum VIF threshold. If NULL (default), automatically set to match the auto-computed max_cor. • preference_order: Vector of predictor names in priority order. • f: Function to compute preference order (default: f_auto) • quiet: Suppress messages (default: TRUE) Note: encoding_method is not supported in this step. The automatic threshold selection adapts to each dataset's correlation structure.
selected	(character vector) Predictors retained after filtering. Populated during training and used during baking. Default: NULL
skip	(optional, logical) Trigger to skip this step when baking. Default: FALSE.
keep_original_cols	(optional, logical) Whether to keep original columns. Default: FALSE.
id	(optional, character) Unique identifier for this step.
x	(required, step_collinear object) The step to be trained. Default: NULL

training	(required, data.frame) The training dataset used to estimate quantities.
info	(optional, data.frame) Preprocessed information about variables in training. Default: NULL
object	(required, step_collinear object) The trained step. Default: NULL
new_data	(required, data.frame) New data to apply the step to. Default: NULL

Value

Updated recipe with new step.

See Also

Other multicollinearity_filtering: collinear(), collinear_select(), cor_select(), vif_select()

Examples

```
## Not run:
if(requireNamespace("recipes", quietly = TRUE) &&
    requireNamespace("parsnip", quietly = TRUE) &&
    requireNamespace("workflows", quietly = TRUE)
    ){

data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

# model formula
vi_formula <- collinear::model_formula(
  df = vi_smol,
  response = "vi_numeric",
  predictors = vi_predictors_numeric
)

# recipe
vi_recipe <- recipes::recipe(
  formula = vi_formula,
  data = vi_smol
  ) |>
  #multicollinearity filtering
  collinear::step_collinear(
    recipes::all_predictors(),
    options = list(
      max_cor = 0.7,
      max_vif = 5,
      f = collinear::f_numeric_glm
      )
  ) |>
  #normalization
```

```
  recipes::step_normalize(
    recipes::all_predictors()
    )

# define linear regression model
vi_model <- parsnip::linear_reg() |>
  parsnip::set_engine("lm")

# create and fit workflow
vi_workflow <- workflows::workflow() |>
  workflows::add_recipe(vi_recipe) |>
  workflows::add_model(vi_model) |>
  workflows::fit(data = vi_smol)

vi_workflow

}

## End(Not run)
```

`summary.collinear_output`

Summarize all results of `collinear()`

Description

Summarize all results of `collinear()`

Usage

```
## S3 method for class 'collinear_output'
summary(object = NULL, ...)
```

Arguments

`object`	(required, list of class `collinear_output`) Object to summarize. Default: NULL
`...`	Ignored, kept for consistency with generic.

Value

list: If `object` was created with `responses = NULL`, a sublist named "result" containing a vector with the selected predictors. Otherwise, a list named after each response containing the corresponding variable selection.

See Also

Other S3_methods: `print.collinear_output()`, `print.collinear_selection()`, `summary.collinear_selection()`

summary.collinear_selection *Summarize single response selection results of* collinear

Description

Summarize single response selection results of collinear

Usage

```
## S3 method for class 'collinear_selection'
summary(object = NULL, ...)
```

Arguments

object	(sub-list in output of collinear()) Object to summarize. Default: NULL
...	Ignored, kept for consistency with generic.

Value

list: response name and character vector of selected predictors.

See Also

Other S3_methods: print.collinear_output(), print.collinear_selection(), summary.collinear_output()

target_encoding_lab *Convert categorical predictors to numeric via target encoding*

Description

Target encoding maps the values of categorical variables (of class character or factor) to numeric using another numeric variable as reference. The encoding methods implemented here are:

- "mean" (implemented in target_encoding_mean()): Maps each category to the average of reference numeric variable across the category cases. Variables encoded with this method are identified with the suffix "__encoded_mean". It has a method to control overfitting implemented via the argument smoothing. The integer value of this argument indicates a threshold in number of rows. Categories sized above this threshold are encoded with the group mean, while groups below it are encoded with a weighted mean of the group's mean and the global mean. This method is named "mean smoothing" in the relevant literature.
- "rank" (implemented in target_encoding_rank()): Returns the rank of the group as a integer, being 1 he group with the lower mean of the reference variable. Variables encoded with this method are identified with the suffix "__encoded_rank".

- "loo" (implemented in target_encoding_loo()): Known as the "leave-one-out method" in the literature, it encodes each categorical value with the mean of the response variable across all other group cases. This method controls overfitting better than "mean". Variables encoded with this method are identified with the suffix "__encoded_loo".

Accepts a parallelization setup via future::plan() and a progress bar via progressr::handlers() (see examples).

Usage

```
target_encoding_lab(
  df = NULL,
  response = NULL,
  predictors = NULL,
  encoding_method = "loo",
  smoothing = 0,
  overwrite = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional, character string) Name of a numeric response variable in df. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
encoding_method	(optional; character vector or NULL). Name of the target encoding methods. One or several of: "mean", "rank", "loo". If NULL, target encoding is ignored, and df is returned with no modification. Default: "loo"
smoothing	(optional; integer vector) Argument of the method "mean". Groups smaller than this number have their means pulled towards the mean of the response across all cases. Default: 0
overwrite	(optional; logical) If TRUE, the original predictors in df are overwritten with their encoded versions, but only one encoding method, smoothing, white noise, and seed are allowed. Otherwise, encoded predictors with their descriptive names are added to df. Default: FALSE
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

dataframe

Author(s)

Blas M. Benito, PhD

References

- Micci-Barreca, D. (2001) A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems. SIGKDD Explor. Newsl. 3, 1, 27-32. doi: 10.1145/507533.507538

See Also

Other target_encoding: `target_encoding_loo()`

Examples

```
data(vi_smol, package = "spatialData")

#applying all methods for a continuous response
df <- target_encoding_lab(
  df = vi_smol,
  response = "vi_numeric",
  predictors = "koppen_zone",
  encoding_method = c(
    "mean",
    "loo",
    "rank"
  )
)

#identify encoded predictors
predictors.encoded <- grep(
  pattern = "*__encoded*",
  x = colnames(df),
  value = TRUE
)

head(df[, predictors.encoded])
```

---

target_encoding_loo *Encode categories as response means*

Description

Encode categories as response means

Usage

```
target_encoding_loo(
  df = NULL,
  response = NULL,
  predictor = NULL,
  encoded_name = NULL,
  smoothing = NULL,
  ...
)

target_encoding_mean(
  df = NULL,
  response = NULL,
  predictor = NULL,
  encoded_name = NULL,
  smoothing = 0,
  ...
)

target_encoding_rank(
  df = NULL,
  response = NULL,
  predictor = NULL,
  encoded_name = NULL,
  smoothing = NULL,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional, character string) Name of a numeric response variable in df. Default: NULL.
predictor	(required; string) Name of the categorical predictor to encode. Default: NULL
encoded_name	(optional, string) Name of the encoded predictor. Default: NULL
smoothing	(optional; integer) Groups smaller than this number have their means pulled towards the mean of the response across all cases. Ignored by target_encoding_rank() and target_encoding_loo(). Default: 0
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

dataframe

See Also

Other target_encoding: target_encoding_lab()

Examples

```
#  loading example data
data(vi_smol, package = "spatialData")

#mean encoding
#-------------

df <- target_encoding_mean(
  df = vi_smol,
  response = "vi_numeric",
  predictor = "soil_type", #categorical
  encoded_name = "soil_type_encoded"
)

if(interactive()){

  plot(
    x = df$soil_type_encoded,
    y = df$vi_numeric,
    xlab = "encoded variable",
    ylab = "response"
  )

}


#rank encoding
#----------

df <- target_encoding_rank(
  df = vi_smol,
  response = "vi_numeric",
  predictor = "soil_type",
  encoded_name = "soil_type_encoded"
)

if(interactive()){

  plot(
    x = df$soil_type_encoded,
    y = df$vi_numeric,
    xlab = "encoded variable",
    ylab = "response"
  )

}
```

```
#leave-one-out
#-------------

df <- target_encoding_loo(
  df = vi_smol,
  response = "vi_numeric",
  predictor = "soil_type",
  encoded_name = "soil_type_encoded"
)

if(interactive()){

  plot(
    x = df$soil_type_encoded,
    y = df$vi_numeric,
    xlab = "encoded variable",
    ylab = "response"
  )

}
```

`toy` *Toy dataframe with varying levels of multicollinearity*

Description

Dataframe with known relationship between responses and predictors useful to illustrate multicollinearity concepts.

Usage

```
data(toy)
```

Format

dataframe with 2000 rows and 5 columns.

Details

Columns:

- `y`: response variable generated from `a * 0.75 + b * 0.25 + noise`.
- `a`: most important predictor of `y`, uncorrelated with `b`.
- `b`: second most important predictor of `y`, uncorrelated with `a`.
- `c`: generated from `a + noise`.
- `d`: generated from `(a + b)/2 + noise`.

These are variance inflation factors of the predictors in `toy`. variable vif b 4.062 d 6.804 c 13.263 a 16.161

validate_arg_df *Check and prepare argument* df

Description

Internal function to validate the integrity of the argument df. It ensures that the dataframe has suitable dimensions for a multicollinearity analysis, transforms logical columns to numeric, character columns to factors, and converts NaN, Inf and -Inf to NA. Additionally, it checks the values of responses and predictors if these arguments are provided.

Usage

```
validate_arg_df(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  quiet = FALSE,
  function_name = NULL
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

dataframe

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df_not_null(), validate_arg_encoding_metho
validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(),
validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet(),
validate_arg_responses()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

df <- validate_arg_df(
  df = vi_smol,
  responses = "vi_numeric",
  predictors = vi_predictors_numeric,
  quiet = FALSE
)

attributes(vi)$validated
```

`validate_arg_df_not_null`

Ensure that argument `df` *is not* `NULL`

Description

Internal function to validate the default value of the argument `df`.

Usage

```
validate_arg_df_not_null(df = NULL, function_name = NULL)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`function_name`	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

dataframe

See Also

Other argument_validation: `drop_geometry_column()`, `validate_arg_df()`, `validate_arg_encoding_method()`, `validate_arg_f()`, `validate_arg_function_name()`, `validate_arg_max_cor()`, `validate_arg_max_vif()`, `validate_arg_predictors()`, `validate_arg_preference_order()`, `validate_arg_quiet()`, `validate_arg_responses()`

Examples

```
data(vi_smol, package = "spatialData")
df <- validate_arg_df_not_null(
  df = vi_smol
  )
```

`validate_arg_encoding_method`
Check and validate argument `encoding_method`

Description

Internal function to validate the argument `encoding_method` of `target_encoding_lab()`.

Usage

```
validate_arg_encoding_method(
  encoding_method = "loo",
  overwrite = NULL,
  quiet = FALSE,
  function_name = NULL
)
```

Arguments

`encoding_method`	(optional; character vector or NULL). Name of the target encoding methods. One or several of: "mean", "rank", "loo". If NULL, target encoding is ignored, and `df` is returned with no modification. Default: "loo"
`overwrite`	(optional; logical) If TRUE, the original predictors in `df` are overwritten with their encoded versions, but only one encoding method, smoothing, white noise, and seed are allowed. Otherwise, encoded predictors with their descriptive names are added to `df`. Default: FALSE
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`function_name`	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

character

See Also

Other argument_validation: `drop_geometry_column()`, `validate_arg_df()`, `validate_arg_df_not_null()`, `validate_arg_f()`, `validate_arg_function_name()`, `validate_arg_max_cor()`, `validate_arg_max_vif()`, `validate_arg_predictors()`, `validate_arg_preference_order()`, `validate_arg_quiet()`, `validate_arg_responses()`

Examples

```
x <- validate_arg_encoding_method(
  encoding_method = "wrong_method"
  )
```

`validate_arg_f` *Check and validate argument* `f`

Description

Check and validate argument `f`

Usage

```
validate_arg_f(f = NULL, f_name = NULL, function_name = NULL)
```

Arguments

`f`	(optional: function name) Unquoted function name without parenthesis (see f_functions). By default calls to `f_auto()`, which selects a suitable function depending on the nature of the response and predictors. Set to NULL if `responses = NULL`. If NULL, predictors are ranked from lower to higher multicollinearity. Default: `f_auto`
`f_name`	(optional, string) Name of the function `f`, as returned by `deparse(substitute(f))`. Default: NULL
`function_name`	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

function

See Also

Other argument_validation: `drop_geometry_column()`, `validate_arg_df()`, `validate_arg_df_not_null()`, `validate_arg_encoding_method()`, `validate_arg_function_name()`, `validate_arg_max_cor()`, `validate_arg_max_vif()`, `validate_arg_predictors()`, `validate_arg_preference_order()`, `validate_arg_quiet()`, `validate_arg_responses()`

Examples

```
x <- validate_arg_f(f = f_auto)
```

validate_arg_function_name *Build hierarchical function names for messages*

Description

Concatenates parent and child function names for a better message, warning, and error tracing.

Usage

```
validate_arg_function_name(default_name = NULL, function_name = NULL, ...)
```

Arguments

default_name	(optional, character) Name of the calling function. Default: NULL
function_name	(optional, character) Name of the parent function. Default: NULL
...	(optional) Used to pass function_name within these functions that don't have this argument.

Value

character

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet(), validate_arg_responses()

Examples

```
x <- validate_arg_function_name(
  default_name = "child_function",
  function_name = "parent_function"
)

message(x)
```

validate_arg_max_cor *Check and constrain argument* max_cor

Description

Check and constrain argument max_cor

Usage

```
validate_arg_max_cor(max_cor = NULL, quiet = FALSE, function_name = NULL)
```

Arguments

max_cor	(optional; numeric or NULL) Maximum correlation allowed between pairs of predictors. Valid values are between 0.01 and 0.99, and recommended values are between 0.5 (strict) and 0.9 (permissive). Default: 0.7
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

numeric or NULL

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet(), validate_arg_responses()

Examples

```
x <- validate_arg_max_cor(
  max_cor = 1.5, #wrong value
  quiet = FALSE
)

x
attributes(x)$validated
```

validate_arg_max_vif *Check and constrain argument* max_vif

Description

Check and constrain argument max_vif

Usage

```
validate_arg_max_vif(max_vif = NULL, quiet = FALSE, function_name = NULL)
```

Arguments

max_vif	(optional, numeric or NULL) Maximum Variance Inflation Factor allowed for predictors during multicollinearity filtering. Recommended values are between 2.5 (strict) and 10 (permissive). Default: 5
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

numeric or NULL

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet(), validate_arg_responses()

Examples

```
max_vif <- validate_arg_max_vif(
  max_vif = 11, #wrong value
  quiet = FALSE
)

max_vif
attributes(max_vif)$validated
```

validate_arg_predictors *Check and validate argument* predictors

Description

Validates the argument predictors by ensuring that all provided predictors are in df and don't intersect with responses, if any.

Usage

```
validate_arg_predictors(
  df = NULL,
  responses = NULL,
  predictors = NULL,
  quiet = FALSE,
  function_name = NULL
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

character vector: predictor names

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_preference_order(), validate_arg_quiet(), validate_arg_responses()

Examples

```
data(vi_smol, vi_predictors, package = "spatialData")

x <- validate_arg_predictors(
  df = vi_smol,
  predictors = vi_predictors
)

attributes(x)$validated
```

`validate_arg_preference_order`
Check and complete argument `preference_order`

Description

Internal function to validate the argument `preference_order` in `cor_select()`, `vif_select()`, `collinear_select()`, `collinear()`, and `collinear()`. Predictors not in `preference_order` are ranked from lower to higher sum of Pearson correlations with all other predictors.

Usage

```
validate_arg_preference_order(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  quiet = FALSE,
  function_name = NULL,
  ...
)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`response`	(optional, character string) Name of a numeric response variable in `df`. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`preference_order`	(optional; character vector, dataframe from `preference_order`, or NULL) Prioritizes predictors to preserve.

quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

character vector: ranked variable names

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_quiet(), validate_arg_responses()

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

#input arguments must be validated first
df <- validate_arg_df(
  df = vi_smol,
  response = "vi_numeric",
  predictors = vi_predictors_numeric,
  quiet = TRUE
)

response <- validate_arg_responses(
  df = df,
  responses = "vi_numeric"
)

predictors <- validate_arg_predictors(
  df = df,
  response = response,
  predictors = vi_predictors_numeric[1:10]
)



#no preference order
#no response
#ranks predictor from lower to higher multicollinearity
y <- validate_arg_preference_order(
  df = df,
```

```
  predictors = predictors,
  preference_order = NULL
)

y
attributes(y)$validated


#validate character vector
y <- validate_arg_preference_order(
  df = df,
  predictors = predictors,
  preference_order = c(
    "swi_max",
    "swi_min",
    "swi_deviance" #does not exist
  )
)

y
attributes(y)$validated

#validate output of preference order
x <- preference_order(
  df = df,
  responses = response,
  predictors = predictors
)

x

y <- validate_arg_preference_order(
  df = df,
  response = response,
  predictors = predictors,
  preference_order = x
)

y
attributes(y)$validated
```

`validate_arg_quiet` *Check and validate argument* `quiet`

Description

Internal function to validate the logical argument `quiet`, which triggers messaging when `FALSE`

Usage

```
validate_arg_quiet(quiet = FALSE, function_name = NULL)
```

Arguments

quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

logical

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_responses()

Examples

```
x <- validate_arg_quiet(
  quiet = TRUE
  )

attributes(x)$validated
```

---

validate_arg_responses

Check and validate arguments response *and* responses

Description

Internal function validate the arguments response and responses. It checks that its value exists as a column name of df,

Usage

```
validate_arg_responses(
  df = NULL,
  responses = NULL,
  max_responses = NULL,
  quiet = FALSE,
  function_name = NULL
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
responses	(optional; character, character vector, or NULL) Name of one or several response variables in df. Default: NULL.
max_responses	(required, integer or NULL) Maximum number of responses to consider. Default: NULL
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
function_name	(optional, character string) Name of the function performing the argument check. Default: NULL

Value

character string: response name

See Also

Other argument_validation: drop_geometry_column(), validate_arg_df(), validate_arg_df_not_null(), validate_arg_encoding_method(), validate_arg_f(), validate_arg_function_name(), validate_arg_max_cor(), validate_arg_max_vif(), validate_arg_predictors(), validate_arg_preference_order(), validate_arg_quiet()

Examples

```
data(vi_smol, package = "spatialData")

x <- validate_arg_responses(
  df = vi_smol,
  responses = "vi_numeric"
)

attributes(x)$validated
```

---

vif *Compute variance inflation factors from a correlation matrix*

Description

Computes the Variance Inflation Factors from a correlation matrix in two steps:

- Applies base::solve() to transform the correlation matrix into a precision matrix, which is the inverse of the covariance matrix between all variables in predictors.
- Applies base::diag() to extract the diagonal of the precision matrix, which contains the variance of the regression of each predictor against all other predictors, also known as Variance Inflation Factor

Usage

```
vif(m = NULL, quiet = FALSE, ...)
```

Arguments

m	(required, matrix) Correlation matrix generated via `stats::cor()` or `cor_matrix()`. Must have named dimensions. Default: NULL
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

named numeric vector

Variance Inflation Factors

VIF for predictor a is computed as $1/(1-R^2)$, where R^2 is the multiple R-squared from regressing a on the other predictors. Recommended maximums commonly used are 2.5, 5, and 10.

References

- David A. Belsley, D.A., Kuh, E., Welsch, R.E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons. DOI: 10.1002/0471725153.

See Also

Other multicollinearity_assessment: `collinear_stats()`, `cor_clusters()`, `cor_cramer()`, `cor_df()`, `cor_matrix()`, `cor_stats()`, `vif_df()`, `vif_stats()`

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

m <- cor_matrix(
  df = vi_smol,
  predictors = vi_predictors_numeric[1:5]
)

vif(m)
```

`vif_df` *Compute variance inflation factors dataframe*

Description

Computes the pairwise correlation matrix between all pairs of predictors via `cor_df()` and `cor_matrix()`, applies `vif()` to the resulting matrix to compute Variance Inflation Factors, and returns the result as a dataframe.

Usage

```
vif_df(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.
`...`	(optional) Internal args (e.g. `function_name` for `validate_arg_function_name`, a precomputed correlation matrix `m`, or cross-validation args for `preference_order`).

Value

dataframe with columns:

- `predictor`: Character, predictor name.
- `vif`: Numeric, variance inflation factor

Variance Inflation Factors

VIF for predictor a is computed as $1/(1 - R^2)$, where R^2 is the multiple R-squared from regressing a on the other predictors. Recommended maximums commonly used are 2.5, 5, and 10.

References

- David A. Belsley, D.A., Kuh, E., Welsch, R.E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons. DOI: 10.1002/0471725153.

See Also

Other multicollinearity_assessment: `collinear_stats()`, `cor_clusters()`, `cor_cramer()`, `cor_df()`, `cor_matrix()`, `cor_stats()`, `vif()`, `vif_stats()`

Examples

```
data(vi_smol, package = "spatialData")

# ## OPTIONAL: parallelization setup
# ## irrelevant when all predictors are numeric
# ## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

# ## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

#predictors
predictors = c(
  "koppen_zone", #character
  "soil_type", #factor
  "topo_elevation", #numeric
  "soil_temperature_mean" #numeric
)

x <- vif_df(
  df = vi_smol,
  predictors = predictors
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`vif_select` *Multicollinearity filtering by variance inflation factor threshold*

Description

Wraps `collinear_select()` to automatize multicollinearity filtering via variance inflation factors (VIF) in dataframes with numeric and categorical predictors.

The argument `max_vif` determines the maximum variance inflation factor allowed in the resulting selection of predictors.

The argument `preference_order` accepts a character vector of predictor names ranked from first to last index, or a dataframe resulting from `preference_order()`. When two predictors in this vector or dataframe are highly collinear, the one with a lower ranking is removed. This option helps protect predictors of interest. If not provided, predictors are ranked from lower to higher multicollinearity.

Please check the sections **Variance Inflation Factors** and **VIF-based Filtering** at the end of this help file for further details.

Usage

```
vif_select(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  max_vif = 5,
  quiet = FALSE,
  ...
)
```

Arguments

df	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and 10 * (length(predictors) - 1) for VIF. Default: NULL.
response	(optional; character or NULL) Name of one response variable in df. Used to exclude columns when predictors is NULL, and to filter preference_order when it is a dataframe and contains several responses. Default: NULL.
predictors	(optional; character vector or NULL) Names of the predictors in df. If NULL, all columns except responses and constant/near-zero-variance columns are used. Default: NULL.
preference_order	(optional; character vector, dataframe from preference_order, or NULL) Prioritizes predictors to preserve.
max_vif	(optional, numeric or NULL) Maximum Variance Inflation Factor allowed for predictors during multicollinearity filtering. Recommended values are between 2.5 (strict) and 10 (permissive). Default: 5
quiet	(optional; logical) If FALSE, messages are printed. Default: FALSE.
...	(optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

character vector of selected predictors

Variance Inflation Factors

VIF for predictor a is computed as $1/(1-R^2)$, where R^2 is the multiple R-squared from regressing a on the other predictors. Recommended maximums commonly used are 2.5, 5, and 10.

VIF-based Filtering

vif_select ranks numeric predictors (user preference_order if provided, otherwise from lower to higher VIF) and sequentially adds predictors whose VIF against the current selection is below max_vif.

Author(s)

Blas M. Benito, PhD

References

- David A. Belsley, D.A., Kuh, E., Welsch, R.E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons. DOI: 10.1002/0471725153.

See Also

Other multicollinearity_filtering: collinear(), collinear_select(), cor_select(), step_collinear()

Examples

```
data(vi_smol, package = "spatialData")

## OPTIONAL: parallelization setup
## irrelevant when all predictors are numeric
## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

#predictors
predictors = c(
  "koppen_zone", #character
  "soil_type", #factor
  "topo_elevation", #numeric
  "soil_temperature_mean" #numeric
)

#predictors ordered from lower to higher multicollinearity
x <- vif_select(
  df = vi_smol,
  predictors = predictors,
  max_vif = 5
)

x


#with custom preference order
x <- vif_select(
  df = vi_smol,
  predictors = predictors,
  preference_order = c(
    "koppen_zone",
    "soil_type"
```

```
  ),
  max_vif = 5
)

x


#with automated preference order
df_preference <- preference_order(
  df = vi_smol,
  response = "vi_numeric",
  predictors = predictors
)

df_preference

x <- cor_select(
  df = vi_smol,
  predictors = predictors,
  preference_order = df_preference,
  max_cor = 0.7
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

`vif_stats` *VIF Statistics*

Description

Computes the the minimum, mean, maximum, and quantiles 0.05, 0.25, median (0.5), 0.75, and 0.95 of the column "vif" in the output of `vif_df()`.

Usage

```
vif_stats(df = NULL, predictors = NULL, quiet = FALSE, ...)
```

Arguments

`df`	(required; dataframe, tibble, or sf) A dataframe with responses (optional) and predictors. Must have at least 10 rows for pairwise correlation analysis, and `10 * (length(predictors) - 1)` for VIF. Default: NULL.
`predictors`	(optional; character vector or NULL) Names of the predictors in `df`. If NULL, all columns except `responses` and constant/near-zero-variance columns are used. Default: NULL.
`quiet`	(optional; logical) If FALSE, messages are printed. Default: FALSE.

... (optional) Internal args (e.g. function_name for validate_arg_function_name, a precomputed correlation matrix m, or cross-validation args for preference_order).

Value

dataframe with columns method with the value "vif", statistic with the statistic name, and value.

See Also

Other multicollinearity_assessment: collinear_stats(), cor_clusters(), cor_cramer(), cor_df(), cor_matrix(), cor_stats(), vif(), vif_df()

Examples

```
data(vi_smol, package = "spatialData")
data(vi_predictors, package = "spatialData")
vi_predictors_numeric <- identify_numeric_variables(
  df = vi_smol,
  predictors = vi_predictors
)$valid

# ## OPTIONAL: parallelization setup
# ## irrelevant when all predictors are numeric
# ## only worth it for large data with many categoricals
# future::plan(
#   future::multisession,
#   workers = future::availableCores() - 1
# )

# ## OPTIONAL: progress bar
# progressr::handlers(global = TRUE)

x <- vif_stats(
  df = vi_smol,
  predictors = vi_predictors_numeric
)

x

## OPTIONAL: disable parallelization
#future::plan(future::sequential)
```

Index

∗ **S3_methods**
print.collinear_output, 59
print.collinear_selection, 60
summary.collinear_output, 65
summary.collinear_selection, 66
∗ **argument_validation**
drop_geometry_column, 23
validate_arg_df, 72
validate_arg_df_not_null, 73
validate_arg_encoding_method, 74
validate_arg_f, 75
validate_arg_function_name, 76
validate_arg_max_cor, 77
validate_arg_max_vif, 78
validate_arg_predictors, 79
validate_arg_preference_order, 80
validate_arg_quiet, 82
validate_arg_responses, 83
∗ **data_types**
identify_categorical_variables, 43
identify_logical_variables, 44
identify_numeric_variables, 45
identify_response_type, 47
identify_valid_variables, 48
identify_zero_variance_variables, 50
∗ **datasets**
experiment_adaptive_thresholds, 24
experiment_cor_vs_vif, 25
gam_cor_to_vif, 42
prediction_cor_to_vif, 54
toy, 71
∗ **example_data**
toy, 71
∗ **experiments**
experiment_adaptive_thresholds, 24
experiment_cor_vs_vif, 25
gam_cor_to_vif, 42
prediction_cor_to_vif, 54
∗ **modelling_tools**
case_weights, 3
model_formula, 51
score_auc, 60
score_cramer, 61
score_r2, 62
∗ **multicollinearity_assessment**
collinear_stats, 10
cor_clusters, 12
cor_cramer, 14
cor_df, 16
cor_matrix, 17
cor_stats, 22
vif, 84
vif_df, 86
vif_stats, 90
∗ **multicollinearity_filtering**
collinear, 4
collinear_select, 8
cor_select, 19
step_collinear, 62
vif_select, 87
∗ **preference_order_functions**
f_binomial_gam, 28
f_binomial_glm, 30
f_binomial_rf, 31
f_categorical_rf, 32
f_count_gam, 34
f_count_glm, 35
f_count_rf, 36
f_numeric_gam, 38
f_numeric_glm, 39
f_numeric_rf, 40
preference_order, 55
∗ **preference_order_tools**
f_auto, 26
f_auto_rules, 28
f_functions, 37
∗ **target_encoding**

target_encoding_lab, 66
target_encoding_loo, 68

bake.step_collinear *(*step_collinear*)*, 62

case_weights, 3
case_weights(), *28*, *30–32*, *52*, *61*, *62*
collinear, 4
collinear(), *10*, *20*, *24*, *25*, *55*, *60*, *62–64*, *66*, *80*, *89*
collinear_select, 8
collinear_select(), *7*, *19*, *20*, *51*, *55*, *64*, *80*, *87*, *89*
collinear_stats, 10
collinear_stats(), *13*, *15*, *17*, *18*, *22*, *85*, *86*, *91*
cor_clusters, 12
cor_clusters(), *11*, *15*, *17*, *18*, *22*, *85*, *86*, *91*
cor_cramer, *5*, 14
cor_cramer(), *11*, *13*, *16–18*, *22*, *32*, *61*, *85*, *86*, *91*
cor_df, *5*, 16
cor_df(), *11–13*, *15*, *17*, *18*, *22*, *85*, *86*, *91*
cor_matrix, *5*, 17
cor_matrix(), *11–13*, *15*, *17*, *22*, *85*, *86*, *91*
cor_select, *5*, *6*, *9*, 19, *20*
cor_select(), *7*, *10*, *25*, *26*, *42*, *55*, *64*, *80*, *89*
cor_stats, *6*, 22
cor_stats(), *10*, *11*, *13*, *15*, *17*, *18*, *85*, *86*, *91*

drop_geometry_column, 23
drop_geometry_column(), *72–79*, *81*, *83*, *84*

experiment_adaptive_thresholds, 24, *26*, *42*, *54*
experiment_cor_vs_vif, *25*, 25, *42*, *54*

f_auto, *5*, 26
f_auto(), *28*, *38*, *47*, *55*, *56*, *75*
f_auto_rules, 28
f_auto_rules(), *26*, *27*, *38*, *55*
f_binomial_gam, 28
f_binomial_gam(), *30*, *32–35*, *37*, *38*, *40*, *41*, *55*, *57*, *60*
f_binomial_glm, 30
f_binomial_glm(), *29*, *32–35*, *37*, *38*, *40*, *41*, *55*, *57*, *60*
f_binomial_rf, 31
f_binomial_rf(), *29*, *30*, *33–35*, *37*, *38*, *40*, *41*, *55*, *57*, *60*
f_categorical_rf, 32
f_categorical_rf(), *29*, *30*, *32*, *34*, *35*, *37*, *38*, *40*, *41*, *56*, *57*
f_count_gam, 34
f_count_gam(), *29*, *30*, *32*, *33*, *35*, *37*, *38*, *40*, *41*, *55*, *57*, *62*
f_count_glm, 35
f_count_glm(), *29*, *30*, *32–34*, *37*, *38*, *40*, *41*, *55*, *57*, *62*
f_count_rf, 36
f_count_rf(), *29*, *30*, *32–35*, *38*, *40*, *41*, *55*, *57*
f_functions, *6*, 37, *56*, *75*
f_functions(), *27*, *28*, *55*, *57*
f_numeric_gam, 38
f_numeric_gam(), *29*, *30*, *32–35*, *37*, *40*, *41*, *55*, *57*, *62*
f_numeric_glm, *5*, 39
f_numeric_glm(), *29*, *30*, *32–35*, *37*, *38*, *41*, *55*, *57*, *62*
f_numeric_rf, 40
f_numeric_rf(), *29*, *30*, *32–35*, *37*, *38*, *40*, *55*, *57*, *62*

gam, *42*
gam_cor_to_vif, *6*, *25*, *26*, 42, *54*

identify_categorical_variables, 43
identify_categorical_variables(), *45*, *46*, *48*, *49*, *51*
identify_logical_variables, 44
identify_logical_variables(), *43*, *46*, *48*, *49*, *51*
identify_numeric_variables, 45
identify_numeric_variables(), *43*, *45*, *48*, *49*, *51*
identify_response_type, 47
identify_response_type(), *43*, *45*, *46*, *49*, *51*
identify_valid_variables, 48
identify_valid_variables(), *43*, *45*, *46*, *48*, *51*
identify_zero_variance_variables, 50
identify_zero_variance_variables(), *43*, *45*, *46*, *48*, *49*

model_formula, 51
model_formula(), *4*, *61*, *62*

prediction_cor_to_vif, *25*, *26*, *42*, 54

preference_order, *4–6*, *9*, *11*, *12*, *14*, *16*, *18*, *20*, *22*, *23*, *27*, *43*, *44*, *46*, *47*, *49*, *50*, *52*, *55*, *57*, *60–62*, *67*, *69*, *80*, *81*, *85*, *86*, *88*, *91*
preference_order(), *8*, *19*, *28–41*, *60*, *87*
prep.step_collinear *(*step_collinear*)*, 62
print.collinear_output, 59
print.collinear_output(), *60*, *65*, *66*
print.collinear_selection, 60
print.collinear_selection(), *59*, *65*, *66*

score_auc, 60
score_auc(), *4*, *28*, *30*, *31*, *52*, *61*, *62*
score_cramer, 61
score_cramer(), *4*, *52*, *61*, *62*
score_r2, 62
score_r2(), *4*, *34–36*, *38*, *39*, *41*, *52*, *61*
spatialData::vi, *16*
spatialData::vi_predictors, *16*
stats::hclust(), *12*
step_collinear, 62
step_collinear(), *7*, *10*, *20*, *89*
summary.collinear_output, 65
summary.collinear_output(), *59*, *60*, *66*
summary.collinear_selection, 66
summary.collinear_selection(), *59*, *60*, *65*

target_encoding_lab, *4*, *5*, 66
target_encoding_lab(), *16*, *70*, *74*
target_encoding_loo, 68
target_encoding_loo(), *67–69*
target_encoding_mean *(*target_encoding_loo*)*, 68
target_encoding_mean(), *66*
target_encoding_rank *(*target_encoding_loo*)*, 68
target_encoding_rank(), *66*, *69*
toy, 71

validate_arg_df, 72
validate_arg_df(), *24*, *73–79*, *81*, *83*, *84*
validate_arg_df_not_null, 73
validate_arg_df_not_null(), *24*, *72*, *74–79*, *81*, *83*, *84*
validate_arg_encoding_method, 74
validate_arg_encoding_method(), *24*, *72*, *73*, *75–79*, *81*, *83*, *84*
validate_arg_f, 75
validate_arg_f(), *24*, *72–74*, *76–79*, *81*, *83*, *84*
validate_arg_function_name, *4*, *6*, *9*, *11*, *12*, *14*, *16*, *18*, *20*, *22*, *23*, *27*, *43*, *44*, *46*, *47*, *49*, *50*, *52*, *57*, *60–62*, *67*, *69*, 76, *81*, *85*, *86*, *88*, *91*
validate_arg_function_name(), *24*, *72–75*, *77–79*, *81*, *83*, *84*
validate_arg_max_cor, 77
validate_arg_max_cor(), *24*, *72–76*, *78*, *79*, *81*, *83*, *84*
validate_arg_max_vif, 78
validate_arg_max_vif(), *24*, *72–77*, *79*, *81*, *83*, *84*
validate_arg_predictors, 79
validate_arg_predictors(), *24*, *72–78*, *81*, *83*, *84*
validate_arg_preference_order, 80
validate_arg_preference_order(), *24*, *72–79*, *83*, *84*
validate_arg_quiet, 82
validate_arg_quiet(), *24*, *72–79*, *81*, *84*
validate_arg_responses, 83
validate_arg_responses(), *24*, *72–79*, *81*, *83*
vif, *5*, 84
vif(), *11*, *13*, *15*, *17*, *18*, *22*, *86*, *91*
vif_df, *5*, 86
vif_df(), *11*, *13*, *15*, *17*, *18*, *22*, *85*, *90*, *91*
vif_select, *5*, *6*, *9*, 87, *88*
vif_select(), *7*, *10*, *20*, *25*, *26*, *42*, *55*, *64*, *80*
vif_stats, 90
vif_stats(), *10*, *11*, *13*, *15*, *17*, *18*, *22*, *85*, *86*