Package 'collinear'

December 8, 2023

Title Seamless Multicollinearity Management

Version 1.1.1

Description System for seamless management of multicollinearity in data frames with numeric and/or categorical variables for statistical analysis and machine learning modeling. The package combines bivariate correlation (Pearson, Spearman, and Cramer's V) with variance inflation factor analysis, target encoding to transform categorical variables into numeric (Micci-Barreca, D. 2001 <DOI:10.1145/507533.507538>), and a flexible feature prioritization method, to deliver a comprehensive multicollinearity management tool covering a wide range of use cases.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports dplyr

Suggests ranger, mgcv, future, future.apply, testthat (>= 3.0.0), spelling

Config/testthat/edition 3

Depends R (>= 4.0)

LazyData true

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

Language en-US

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2023-12-08 08:50:02 UTC

R topics documented:

`auc_score` *Area Under the Receiver Operating Characteristic*

Description

Computes the AUC score of binary model predictions.

Usage

```
auc_score(observed = NULL, predicted = NULL)
```

Arguments

observed	(required, integer) Numeric vector with observations. Valid values are 1 and 0. Must have the same length as predicted. Default: NULL
predicted	(required, numeric) Numeric vector in the range 0-1 with binary model predictions. Must have the same length as observed.

Value

AUC value.

Examples

```
out <- auc_score(
  observed = c(0, 0, 1, 1),
  predicted = c(0.1, 0.6, 0.4, 0.8)
  )
```

---

collinear *Automated multicollinearity management*

Description

Automates multicollinearity management in data frames with numeric and categorical predictors by combining four methods:

- Pairwise correlation filtering: Pearson, Spearman, and Cramer's V statistics to identify pairs of highly correlated predictors.
- Variance Inflation Factor (VIF) filtering: identifies predictors that are linear combinations of other predictors.
- Target encoding: to transform categorical predictors to numeric using a numeric variable as reference.
- Flexible prioritization method: to help the user select a meaningful set of non-correlated predictors.

The pairwise correlation filtering is implemented in cor_select(). This function applies a recursive forward selection algorithm to keep predictors with a Pearson correlation with all other selected predictors lower than the value of the argument max_cor. When two predictors are correlated above this threshold, the one with the lowest preference order is removed. At this stage, if preference_order is NULL, predictors are ranked from lower to higher sum of absolute pairwise correlation with the other predictors.

The VIF-based filtering is implemented in vif_select(), which removes variables and recomputes VIF scores iteratively, until all variables in the resulting selection have a VIF below the value of the argument max_vif. The VIF for a given variable y is computed as 1/(1-R2), where R2 is the R-squared of a multiple regression model fitted using y as response against the other predictors. The

equation can be interpreted as "the rate of perfect model's R-squared to the unexplained variance of this model". The possible range of VIF values is (1, Inf], but the recommended thresholds for maximum VIF (argument `max_vif`) may vary, being 2.5, 5, and 10 the values most commonly mentioned in the relevant bibliography. At this stage, if `preference_order` is NULL, predictors are ranked from lower to higher Variance Inflation Factor.

When a 'response' argument is provided, categorical predictors are converted to numeric via target encoding with the function `target_encoding_lab()`, and all predictors are then handled as numeric during the multicollinearity filtering. When the 'response' argument is not provided, categorical variables are ignored. However, in such case, the function `cor_select()` can handle categorical variables, albeit with a lower computation speed.

The argument `preference_order` allows prioritizing variables that might be interesting or even required for a given analysis. If `preference_order` is not provided, then the predictors are ranked from lower to higher sum of their absolute correlations with the other predictors in `cor_select()`, and by their VIF in `vif_select()`, and removed one by one until the maximum R-squared of the correlation matrix is lower than `max_cor` and the maximum VIF is below `max_vif`.

Please note that near-zero variance columns are identified by `identify_zero_variance_predictors()`, and ignored by `collinear()`, `cor_select()`, and `vif_select()`.

Usage

```
collinear(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  cor_method = "pearson",
  max_cor = 0.75,
  max_vif = 5,
  encoding_method = "mean"
)
```

Arguments

`df`	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
`response`	(recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
`predictors`	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default: NULL
`preference_order`	(optional; character vector) vector with column names in 'predictors' in the desired preference order, or result of the function `preference_order()`. Allows defining a priority order for selecting predictors, which can be particularly useful when some predictors are more critical for the analysis than others. Predictors not included in this argument are ranked by their Variance Inflation Factor. Default: NULL.

cor_method (optional; character string) Method used to compute pairwise correlations. Accepted methods are "pearson" (with a recommended minimum of 30 rows in 'df') or "spearman" (with a recommended minimum of 10 rows in 'df'). Default: "pearson".

max_cor (optional; numeric) Maximum correlation allowed between any pair of predictors. Higher values return larger number of predictors with higher multicollinearity. Default: `0.75`

max_vif (optional, numeric) Numeric with recommended values between 2.5 and 10 defining the maximum VIF allowed for any given predictor in the output dataset. Higher VIF thresholds should result in a higher number of selected variables. Default: 5.

encoding_method (optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see `target_encoding_lab()` for further details). Default: "mean"

Value

Character vector with the names of uncorrelated predictors.

Author(s)

Blas M. Benito

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

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]

#without response
#without preference_order
#permissive max_cor and max_vif
#only numeric variables in output
selected.predictors <- collinear(
  df = vi,
  predictors = vi_predictors,
```

```
  max_cor = 0.8,
  max_vif = 10
  )

selected.predictors

#without response
#without preference_order
#restrictive max_cor and max_vif
#only numeric variables in output
selected.predictors <- collinear(
  df = vi,
  predictors = vi_predictors,
  max_cor = 0.5,
  max_vif = 2.5
)

selected.predictors

#with response
#without preference_order
#restrictive max_cor and max_vif
#numerics and categorical variables in output
selected.predictors <- collinear(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  max_cor = 0.5,
  max_vif = 2.5
)

selected.predictors

#with response
#with user-defined preference_order
#restrictive max_cor and max_vif
#numerics and categorical variables in output
selected.predictors <- collinear(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = c(
    "soil_temperature_mean",
    "swi_mean",
    "rainfall_mean",
    "evapotranspiration_mean"
  ),
  max_cor = 0.5,
  max_vif = 2.5
)

selected.predictors
```

```
#with response
#with automated preference_order
#restrictive max_cor and max_vif
#numerics and categorical variables in output
preference.order <- preference_order(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  f = f_rsquared, #cor(response, predictor)
  workers = 1
)

selected.predictors <- collinear(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = preference.order,
  max_cor = 0.5,
  max_vif = 2.5
)

selected.predictors
```

---

cor_df *Correlation data frame of numeric and character variables*

Description

Returns a correlation data frame between all pairs of predictors in a training dataset. Non-numeric predictors are transformed into numeric via target encoding, using the 'response' variable as reference.

Usage

```
cor_df(
  df = NULL,
  response = NULL,
  predictors = NULL,
  cor_method = "pearson",
  encoding_method = "mean"
)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors, and optionally, a response variable. Default: NULL.

`response` (recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.

`predictors` (optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'

`cor_method` (optional; character string) Method used to compute pairwise correlations. Accepted methods are "pearson" (with a recommended minimum of 30 rows in 'df') or "spearman" (with a recommended minimum of 10 rows in 'df'). Default: "pearson".

`encoding_method` (optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see `target_encoding_lab()` for further details). Default: "mean"

Details

This function attempts to handle correlations between pairs of variables that can be of different types:

- numeric vs. numeric: computed with stats::cor() with the methods "pearson" or "spearman".
- numeric vs. character, two alternatives leading to different results:
  - 'response' is provided: the character variable is target-encoded as numeric using the values of the response as reference, and then its correlation with the numeric variable is computed with stats::cor(). This option generates a response-specific result suitable for training statistical and machine-learning models
  - 'response' is NULL (or the name of a non-numeric column): the character variable is target-encoded as numeric using the values of the numeric predictor (instead of the response) as reference, and then their correlation is computed with stats::cor(). This option leads to a response-agnostic result suitable for clustering problems.
- character vs. character, two alternatives leading to different results:
  - 'response' is provided: the character variables are target-encoded as numeric using the values of the response as reference, and then their correlation is computed with stats::cor().
  - response' is NULL (or the name of a non-numeric column): the association between the character variables is computed using Cramer's V. This option might be problematic, because R-squared values and Cramer's V, even when having the same range between 0 and 1, are not fully comparable.

Value

data frame with pairs of predictors and their correlation.

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

#reduce size of vi to speed-up example execution
vi <- vi[1:1000, ]
vi_predictors <- vi_predictors[1:10]

#without response
#categorical vs categorical compared with cramer_v()
#categorical vs numerical compared wit stats::cor() via target-encoding
#numerical vs numerical compared with stats::cor()
df <- cor_df(
  df = vi,
  predictors = vi_predictors
)

head(df)

#with response
#different solution than previous one
#because target encoding is done against the response
#rather than against the other numeric in the pair
df <- cor_df(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors
)

head(df)
```

---

cor_matrix *Correlation matrix of numeric and character variables*

Description

Returns a correlation matrix between all pairs of predictors in a training dataset. Non-numeric predictors are transformed into numeric via target encoding, using the 'response' variable as reference.

Usage

```
cor_matrix(
  df = NULL,
  response = NULL,
  predictors = NULL,
```

```
  cor_method = "pearson",
  encoding_method = "mean"
)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors, and optionally, a response variable. Default: NULL.
response	(recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
predictors	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'
cor_method	(optional; character string) Method used to compute pairwise correlations. Accepted methods are "pearson" (with a recommended minimum of 30 rows in 'df') or "spearman" (with a recommended minimum of 10 rows in 'df'). Default: "pearson".
encoding_method	(optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see `target_encoding_lab()` for further details). Default: "mean"

Details

This function attempts to handle correlations between pairs of variables that can be of different types:

- numeric vs. numeric: computed with stats::cor() with the methods "pearson" or "spearman".
- numeric vs. character, two alternatives leading to different results:
  - 'response' is provided: the character variable is target-encoded as numeric using the values of the response as reference, and then its correlation with the numeric variable is computed with stats::cor(). This option generates a response-specific result suitable for training statistical and machine-learning models
  - 'response' is NULL (or the name of a non-numeric column): the character variable is target-encoded as numeric using the values of the numeric predictor (instead of the response) as reference, and then their correlation is computed with stats::cor(). This option leads to a response-agnostic result suitable for clustering problems.
- character vs. character, two alternatives leading to different results:
  - 'response' is provided: the character variables are target-encoded as numeric using the values of the response as reference, and then their correlation is computed with stats::cor().
  - response' is NULL (or the name of a non-numeric column): the association between the character variables is computed using Cramer's V. This option might be problematic, because R-squared values and Cramer's V, even when having the same range between 0 and 1, are not fully comparable.

Value

correlation matrix

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]
vi_predictors <- vi_predictors[1:5]

#convert correlation data frame to matrix
df <- cor_df(
  df = vi,
  predictors = vi_predictors
)

m <- cor_matrix(
  df = df
)

#show first three columns and rows
m[1:5, 1:5]

#generate correlation matrix directly
m <- cor_matrix(
  df = vi,
  predictors = vi_predictors
)

m[1:5, 1:5]

#with response (much faster)
#different solution than previous one
#because target encoding is done against the response
#rather than against the other numeric in the pair
m <- cor_matrix(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors
)

m[1:5, 1:5]
```

`cor_select` *Automated multicollinearity reduction via pairwise correlation*

Description

Applies a recursive forward selection algorithm algorithm to select predictors with a bivariate correlation with any other predictor lower than a threshold defined by the argument `max_cor`.

If the argument `response` is provided, all non-numeric variables in `predictors` are transformed into numeric using target encoding (see `target_encoding_lab()`). Otherwise, non-numeric variables are ignored.

The argument `preference_order` allows defining a preference selection order to preserve (when possible) variables that might be interesting or even required for a given analysis. If NULL, predictors are ordered from lower to higher sum of their absolute pairwise correlation with the other predictors.

For example, if `predictors` is `c("a", "b", "c")` and `preference_order` is `c("a", "b")`, there are two possibilities:

- If the correlation between `"a"` and `"b"` is below `max_cor`, both variables are selected.
- If their correlation is equal or above `max_cor`, then `"a"` is selected, no matter its correlation with `"c"`,

If `preference_order` is not provided, then the predictors are ranked by their variance inflation factor as computed by `vif_df()`.

Usage

```
cor_select(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  cor_method = "pearson",
  max_cor = 0.75,
  encoding_method = "mean"
)
```

Arguments

`df`	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
`response`	(recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
`predictors`	(optional; character vector) Character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'

preference_order
: (optional; character vector) vector with column names in 'predictors' in the desired preference order, or result of the function `preference_order()`. Allows defining a priority order for selecting predictors, which can be particularly useful when some predictors are more critical for the analysis than others. Default: NULL (predictors ordered from lower to higher sum of absolute correlation with the other predictors).

cor_method
: (optional; character string) Method used to compute pairwise correlations. Accepted methods are "pearson" (with a recommended minimum of 30 rows in 'df') or "spearman" (with a recommended minimum of 10 rows in 'df'). Default: "pearson".

max_cor
: (optional; numeric) Maximum correlation allowed between any pair of predictors. Higher values return larger number of predictors with higher multicollinearity. Default: 0.75

encoding_method
: (optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see `target_encoding_lab()` for further details). Default: "mean"

Value

Character vector with the names of the selected predictors.

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]
vi_predictors <- vi_predictors[1:10]

#without response
#without preference_order
#permissive max_cor
selected.predictors <- cor_select(
  df = vi,
  predictors = vi_predictors,
  max_cor = 0.8
)

selected.predictors
```

```
#without response
#without preference_order
#restrictive max_cor
selected.predictors <- cor_select(
  df = vi,
  predictors = vi_predictors,
  max_cor = 0.5
)

selected.predictors

#with response
#without preference_order
#restrictive max_cor
#slightly different solution than previous one
#because here target encoding is done against the response
#while before was done pairwise against each numeric predictor
selected.predictors <- cor_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  max_cor = 0.5
)

selected.predictors

#with response
#with user-defined preference_order
#restrictive max_cor
#numerics and categorical variables in output
selected.predictors <- cor_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = c(
    "soil_type", #categorical variable
    "soil_temperature_mean",
    "swi_mean",
    "rainfall_mean",
    "evapotranspiration_mean"
  ),
  max_cor = 0.5
)

selected.predictors


#with response
#with automated preference_order
#restrictive max_cor and max_vif
#numerics and categorical variables in output
preference.order <- preference_order(
  df = vi,
```

```
  response = "vi_mean",
  predictors = vi_predictors,
  f = f_rsquared #cor(response, predictor)
)

head(preference.order)

selected.predictors <- cor_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = preference.order,
  max_cor = 0.5
)

selected.predictors
```

`cramer_v` *Bias Corrected Cramer's V*

Description

The `cramer_v()` function calculates bias-corrected Cramer's V, a measure of association between two categorical variables.

Cramer's V is an extension of the chi-squared test to measure the strength of association between two categorical variables. Provides values between 0 and 1, where 0 indicates no association, and 1 indicates a perfect association. In essence, Cramer's V assesses the co-occurrence of the categories of two variables to quantify how strongly these variables are related.

Even when its range is between 0 and 1, Cramer's V values are not directly comparable to R-squared values, and as such, a multicollinearity analysis containing both types of values must be assessed with care. It is probably preferable to convert non-numeric variables to numeric using target encoding rather before a multicollinearity analysis.

Usage

```
cramer_v(x = NULL, y = NULL, check_input = TRUE)
```

Arguments

`x`	(required; character vector) character vector representing a categorical variable. Default: NULL
`y`	(required; character vector) character vector representing a categorical variable. Must have the same length as 'x'. Default: NULL
`check_input`	(required; logical) If FALSE, disables data checking for a slightly faster execution. Default: TRUE

Value

Numeric, value of Cramer's V

Author(s)

Blas M. Benito

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

Examples

```
#loading example data
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#computing Cramer's V for two categorical predictors
v <- cramer_v(
  x = vi$soil_type,
  y = vi$koppen_zone
  )

v
```

---

f_gam_auc_balanced *AUC of Logistic GAM Model*

Description

Fits a binomial logistic Generalized Additive Model (GAM) `y ~ s(x, k = 3)` between a binary response and a numeric predictor and returns the Area Under the Curve of the observations versus the predictions.

Usage

```
f_gam_auc_balanced(x, y, df)
```

Arguments

x	(required, character string) name of the predictor variable.
y	(required, character string) name of the binary response variable.
df	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#this example requires "mgcv" installed
if(requireNamespace(package = "mgcv", quietly = TRUE)){

  f_gam_auc_balanced(
    x = "growing_season_length", #predictor
    y = "vi_binary",               #response
    df = vi
  )

}
```

---

f_gam_auc_unbalanced *AUC of Logistic GAM Model with Weighted Cases*

Description

Fits a quasibinomial logistic Generalized Additive Model (GAM) `y ~ s(x, k = 3)` with weighted cases between a binary response and a numeric predictor and returns the Area Under the Curve of the observations versus the predictions.

Usage

```
f_gam_auc_unbalanced(x, y, df)
```

Arguments

`x`	(required, character string) name of the predictor variable.
`y`	(required, character string) name of the binary response variable
`df`	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#this example requires "mgcv" installed
if(requireNamespace(package = "mgcv", quietly = TRUE)){

  f_gam_auc_unbalanced(
    x = "growing_season_length", #predictor
    y = "vi_binary",               #response
    df = vi
  )

}
```

`f_gam_deviance` *Explained Deviance from univariate GAM model*

Description

Computes the explained deviance of a response against a predictor via Generalized Additive Model (GAM). This option is slower than `f_rsquared()`, but suitable if you will be fitting GAMs with the resulting preference order.

Usage

```
f_gam_deviance(x, y, df)
```

Arguments

`x`	(required, character string) name of the predictor variable.
`y`	(required, character string) name of the response variable
`df`	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Explained deviance

Examples

```
data(vi)

#subset to limit example run time
```

```
vi <- vi[1:1000, ]

#this example requires "mgcv" installed in the system
if(requireNamespace(package = "mgcv", quietly = TRUE)){

  f_gam_deviance(
    x = "growing_season_length", #predictor
    y = "vi_mean",               #response
    df = vi
  )

}
```

`f_logistic_auc_balanced` *AUC of Binomial GLM with Logit Link*

Description

Fits a logistic GLM model `y ~ x` when `y` is a binary response with values 0 and 1 and `x` is numeric. This function is suitable when the response variable is balanced. If the response is unbalanced, then `f_logistic_auc_unbalanced()` should provide better results.

Usage

```
f_logistic_auc_balanced(x, y, df)
```

Arguments

`x`	(required, character string) name of the predictor variable.
`y`	(required, character string) name of the binary response variable
`df`	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

f_logistic_auc_balanced(
  x = "growing_season_length", #predictor
  y = "vi_binary",             #binary response
```

```
  df = vi
)
```

`f_logistic_auc_unbalanced` *AUC of Binomial GLM with Logit Link and Case Weights*

Description

Fits a quasibinomial GLM model `y ~ x` with case weights when `y` is an unbalanced binary response with values 0 and 1 and `x` is numeric. It uses the function `case_weights()` to weight 0s and 1s according to their frequency within `y`.

Usage

```
f_logistic_auc_unbalanced(x, y, df)
```

Arguments

`x`	(required, character string) name of the predictor variable.
`y`	(required, character string) name of the binary response variable.
`df`	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

f_logistic_auc_unbalanced(
  x = "growing_season_length", #predictor
  y = "vi_binary",             #binary response
  df = vi
)
```

f_rf_auc_balanced *AUC of Random Forest model of a balanced binary response*

Description

Computes a univariate random forest model cases via \link[ranger]{ranger} and returns the Area Under the Curve on the out-of-bag data.

Usage

```
f_rf_auc_balanced(x, y, df)
```

Arguments

x	(required, character string) name of the predictor variable.
y	(required, character string) name of the binary response variable
df	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#this example requires "ranger" installed in the system
if(requireNamespace(package = "ranger", quietly = TRUE)){

  f_rf_auc_balanced(
    x = "growing_season_length", #predictor
    y = "vi_binary",               #response
    df = vi
  )

}
```

`f_rf_auc_unbalanced` *AUC of Random Forest model of an unbalanced binary response*

Description

Computes a univariate random forest model with weighted cases via `\link[ranger]{ranger}` and returns the Area Under the Curve on the out-of-bag data.

Usage

```
f_rf_auc_unbalanced(x, y, df)
```

Arguments

`x`	(required, character string) name of the predictor variable.
`y`	(required, character string) name of the binary response variable
`df`	(required, data frame) data frame with the columns 'x' and 'y'.

Value

Area Under the Curve

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#this example requires "ranger" installed in the system
if(requireNamespace(package = "ranger", quietly = TRUE)){

  f_rf_auc_unbalanced(
    x = "growing_season_length", #predictor
    y = "vi_binary",               #response
    df = vi
  )

}
```

f_rf_rsquared *R-squared of Random Forest model*

Description

Computes a univariate random forest model with \link[ranger]{ranger} and returns the R-squared on the out-of-bag data.

Usage

```
f_rf_rsquared(x, y, df)

f_rf_deviance(x, y, df)
```

Arguments

x	(required, character string) name of the predictor variable.
y	(required, character string) name of the response variable
df	(required, data frame) data frame with the columns 'x' and 'y'.

Details

f_rf_rsquared() and f_rf_deviance() are synonyms

Value

R-squared

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

#this example requires "ranger" installed in the system
if(requireNamespace(package = "ranger", quietly = TRUE)){

  f_rf_rsquared(
    x = "growing_season_length", #predictor
    y = "vi_mean",               #response
    df = vi
  )

}
```

f_rsquared *R-squared between a response and a predictor*

Description

Computes the R-squared between a response and a predictor. Fastest option to compute preference order.

Usage

```
f_rsquared(x, y, df)
```

Arguments

x	(required, character string) name of the predictor variable.
y	(required, character string) name of the response variable
df	(required, data frame) data frame with the columns 'x' and 'y'.

Value

R-squared

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

f_rsquared(
  x = "growing_season_length", #predictor
  y = "vi_mean",               #response
  df = vi
)
```

identify_non_numeric_predictors *Identify non-numeric predictors*

Description

Given 'df' and 'predictors' arguments, this function subsets and returns the non-numeric (character, factor, and logical) predictors.

Usage

```
identify_non_numeric_predictors(df = NULL, predictors = NULL)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
predictors	(optional; character vector) A vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'

Value

character vector with names of numeric predictors.

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

non.numeric.predictors <- identify_non_numeric_predictors(
  df = vi,
  predictors = vi_predictors
)

non.numeric.predictors
```

---

identify_numeric_predictors

Identify numeric predictors

Description

Given 'df' and 'predictors' arguments, this function subsets and returns the numeric predictors.

Usage

```
identify_numeric_predictors(df = NULL, predictors = NULL)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
predictors	(optional; character vector) A vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'

Value

character vector with names of numeric predictors.

Author(s)

Blas M. Benito

Examples

```
if (interactive()) {

data(
  vi,
  vi_predictors
)

numeric.predictors <- identify_numeric_predictors(
  df = vi,
  predictors = vi_predictors
)

numeric.predictors

}
```

---

identify_zero_variance_predictors

Identify zero and near-zero-variance predictors

Description

Predictors a variance of zero or near zero are highly problematic for multicollinearity analysis and modelling in general. This function identifies these predictors with a level of sensitivity defined by the 'decimals' argument. Smaller number of decimals increase the number of variables detected as near zero variance. Recommended values will depend on the range of the numeric variables in 'df'.

Usage

```
identify_zero_variance_predictors(df = NULL, predictors = NULL, decimals = 4)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
predictors	(optional; character vector) A vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'
decimals	(required, integer) number of decimal places for the zero variance test. Default: 4

Value

character vector with names of zero and near-zero variance columns.

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

#create zero variance predictors
vi$zv_1 <- 1
vi$zv_2 <- runif(n = nrow(vi), min = 0, max = 0.0001)


#add to vi predictors
vi_predictors <- c(
  vi_predictors,
  "zv_1",
  "zv_2"
)

#identify zero variance predictors
zero.variance.predictors <- identify_zero_variance_predictors(
  df = vi,
  predictors = vi_predictors
)

zero.variance.predictors
```

preference_order *Compute the preference order for predictors based on a user-defined function.*

Description

This function calculates the preference order of predictors based on a user-provided function that takes a predictor, a response, and a data frame as arguments.

Usage

```
preference_order(
  df = NULL,
  response = NULL,
  predictors = NULL,
  f = f_rsquared,
  encoding_method = "mean",
  workers = 1
)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
response	(required, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
predictors	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'
f	(optional: function) A function that returns a value representing the relationship between a given predictor and the response. Higher values are ranked higher. The available options are: • f_rsquared() (default option): returns the R-squared of the correlation between a numeric response and a numeric predictor. • f_gam_deviance: fits a univariate GAM model between a numeric response and a numeric predictor to return the explained deviance. Requires the package mgcv. • f_rf_rsquared() also named f_rf_deviance(): fits a univariate random forest model with ranger::ranger() between a numeric response and a numeric predictor to return the R-squared of the observations against the out-of-bag predictions. Requires the package ranger. • f_logistic_auc_balanced(): fits a logistic univariate GLM of a balanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.

- f_logistic_auc_unbalanced(): fits a quasibinomial univariate GLM with weighted cases of an unbalanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.
- f_gam_auc_balanced(): fits a logistic univariate GAM of a balanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.
- f_gam_auc_unbalanced(): fits a quasibinomial univariate GAM with weighted cases of an unbalanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.
- f_rf_auc_balanced(): fits a random forest model of a balanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.
- f_rf_auc_unbalanced(): fits a random forest model with weighted cases of an unbalanced binary response (0s and 1s) against a numeric predictor to return the Area Under the Curve of the observations against the predictors.

`encoding_method`
: (optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see target_encoding_lab() for further details). Default: "mean"

`workers`
: (integer) number of workers for parallel execution. Default: 1

Value

A data frame with the columns "predictor" and "value". The former contains the predictors names in order, ready for the argument `preference_order` in cor_select(), vif_select() and collinear(). The latter contains the result of the function `f` for each combination of predictor and response.

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]

#computing preference order
#with response
#numeric and categorical predictors in the output
#as the R-squared between each predictor and the response
preference.order <- preference_order(
```

```
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  f = f_rsquared,
  workers = 1
  )

preference.order

#using it in variable selection with collinear()
selected.predictors <- cor_select(
  df = vi,
  response = "vi_mean", #don't forget the response!
  predictors = vi_predictors,
  preference_order = preference.order,
  max_cor = 0.75
  )

selected.predictors

#check their correlations
selected.predictors.cor <- cor_df(
  df = vi,
  response = "vi_mean",
  predictors = selected.predictors
)

#all correlations below max_cor
selected.predictors.cor

#USING A CUSTOM FUNCTION
#custom function to compute RMSE between a predictor and a response
#x is a predictor name
#y is a response name
#df is a data frame with multiple predictors and one response
#must return a single number, with higher number indicating higher preference
#notice we use "one minus RMSE" to give higher rank to variables with lower RMSE
f_rmse <- function(x, y, df){

  xy <- df[, c(x, y)] |>
    na.omit() |>
    scale()

  1 - sqrt(mean((xy[, 1] - xy[, 2])^2))

}

preference.order <- preference_order(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  f = f_rmse,
  workers = 1
```

```
)

preference.order
```

`target_encoding_lab` *Target encoding of non-numeric variables*

Description

Target encoding involves replacing the values of categorical variables with numeric ones from a "target variable", usually a model's response. Target encoding can be useful for improving the performance of machine learning models.

This function identifies categorical variables in the input data frame, and transforms them using a set of target-encoding methods selected by the user, and returns the input data frame with the newly encoded variables.

The target encoding methods implemented in this function are:

- "rank": Returns the rank of the group as a integer, starting with 1 as the rank of the group with the lower mean of the response variable. The variables returned by this method are named with the suffix "__encoded_rank". This method is implemented in the function `target_encoding_rank()`.
- "mean": Replaces each value of the categorical variable with the mean of the response across the category the given value belongs to. This option accepts the argument "white_noise" to limit potential overfitting. The variables returned by this method are named with the suffix "__encoded_mean". This method is implemented in the function `target_encoding_mean()`.
- "rnorm": Computes the mean and standard deviation of the response for each group of the categorical variable, and uses `rnorm()` to generate random values from a normal distribution with these parameters. The argument `rnorm_sd_multiplier` is used as a multiplier of the standard deviation to control the range of values produced by `rnorm()` for each group of the categorical predictor. The variables returned by this method are named with the suffix "__encoded_rnorm". This method is implemented in the function `target_encoding_rnorm()`.
- "loo": This is the leave-one-out method, that replaces each categorical value with the mean of the response variable across the other cases within the same group. This method supports the `white_noise` argument to increase limit potential overfitting. The variables returned by this method are named with the suffix "__encoded_loo". This method is implemented in the function `target_encoding_loo()`.

The methods "mean" and "rank" support the `white_noise` argument, which is a fraction of the range of the `response` variable, and the maximum possible value of white noise to be added. For example, if `response` is within 0 and 1, a `white_noise` of 0.25 will add to every value of the encoded variable a random number selected from a normal distribution between -0.25 and 0.25. This argument helps control potential overfitting induced by the encoded variable.

The method "rnorm" has the argument `rnorm_sd_multiplier`, which multiplies the standard deviation argument of the `\link[stats]{rnorm}` function to control the spread of the encoded values between groups. Values smaller than 1 reduce the spread in the results, while values larger than 1 have the opposite effect.

Usage

```
target_encoding_lab(
  df = NULL,
  response = NULL,
  predictors = NULL,
  encoding_methods = c("mean", "rank", "loo", "rnorm"),
  smoothing = 0,
  rnorm_sd_multiplier = 0,
  seed = 1,
  white_noise = 0,
  replace = FALSE,
  verbose = TRUE
)
```

Arguments

`df`	(required; data frame, tibble, or sf) A training data frame. Default: NULL
`response`	(required; character string) Name of the response. Must be a column name of `df`. Default: NULL
`predictors`	(required; character vector) Names of all the predictors in `df`. Only character and factor predictors are processed, but all are returned in the "df" slot of the function's output. Default: NULL
`encoding_methods`	(optional; character string or vector). Name of the target encoding methods. Default: c("mean", "mean_smoothing, "rank", "loo", "rnorm")
`smoothing`	(optional; numeric) Argument of `target_encoding_mean()` (method "mean_smoothing"). Minimum group size that keeps the mean of the group. Groups smaller than this have their means pulled towards the global mean of the response. Default: 0
`rnorm_sd_multiplier`	(optional; numeric) Numeric with multiplier of the standard deviation of each group in the categorical variable, in the range 0-1. Controls the variability in the encoded variables to mitigate potential overfitting. Default: 1
`seed`	(optional; integer) Random seed to facilitate reproducibility when `white_noise` is not 0. Default: 1
`white_noise`	(optional; numeric) Numeric with white noise values in the range 0-1, representing a fraction of the range of the response to be added as noise to the encoded variable. Controls the variability in the encoded variables to mitigate potential overfitting. Default: `0`.
`replace`	(optional; logical) If `TRUE`, the function replaces each categorical variable with its encoded version, and returns the input data frame with the encoded variables instead of the original ones. Default: FALSE
`verbose`	(optional; logical) If TRUE, messages generated during the execution of the function are printed to the console Default: TRUE

Value

The input data frame with newly encoded columns if `replace` is `FALSE`, or the input data frame with encoded columns if `TRUE`

Author(s)

Blas M. Benito

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

Examples

```
data(
  vi,
  vi_predictors
  )

#subset to limit example run time
vi <- vi[1:1000, ]

#applying all methods for a continuous response
df <- target_encoding_lab(
  df = vi,
  response = "vi_mean",
  predictors = "koppen_zone",
  encoding_methods = c(
    "mean",
    "rank",
    "rnorm",
    "loo"
  ),
  rnorm_sd_multiplier = c(0, 0.1, 0.2),
  white_noise = c(0, 0.1, 0.2)
)

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

#correlation between encoded predictors and the response
stats::cor(
  x = df["vi_mean"],
  y = df[, predictors.encoded],
  use = "pairwise.complete.obs"
```

```
)
```

`target_encoding_mean` *Target-encoding methods*

Description

Methods to apply target-encoding to individual categorical variables. The functions implemented are:

- `target_encoding_mean()`: Each group is identified by the mean of the response over the group cases. The argument `smoothing` controls pushes the mean of small groups towards the global mean to avoid overfitting. White noise can be added via the `white_noise` argument. Columns encoded with this function are identified by the suffix "__encoded_mean". If `white_noise` is used, then the amount of white noise is also added to the suffix.
- `target_encoding_rank()`: Each group is identified by the rank of the mean of the response variable over the group cases. The group with the lower mean receives the rank 1. White noise can be added via the `white_noise` argument. Columns encoded with this function are identified by the suffix "__encoded_rank". If `white_noise` is used, then the amount of noise is also added to the suffix.
- `target_encoding_rnorm()`: Each case in a group receives a value coming from a normal distribution with the mean and the standard deviation of the response over the cases of the group. The argument `rnorm_sd_multiplier` multiplies the standard deviation to reduce the spread of the obtained values. Columns encoded with this function are identified by the suffix "__encoded_rnorm_rnorm_sd_multiplier_X", where X is the amount of `rnorm_sd_multiplier` used.
- `target_encoding_loo()`: The suffix "loo" stands for "leave-one-out". Each case in a group is encoded as the average of the response over the other cases of the group. Columns encoded with this function are identified by the suffix "__encoded_loo".

Usage

```
target_encoding_mean(
  df,
  response,
  predictor,
  smoothing = 0,
  white_noise = 0,
  seed = 1,
  replace = FALSE,
  verbose = TRUE
)

target_encoding_rnorm(
```

```
  df,
  response,
  predictor,
  rnorm_sd_multiplier = 1,
  seed = 1,
  replace = FALSE,
  verbose = TRUE
)

target_encoding_rank(
  df,
  response,
  predictor,
  white_noise = 0,
  seed = 1,
  replace = FALSE,
  verbose = TRUE
)

target_encoding_loo(
  df,
  response,
  predictor,
  white_noise = 0,
  seed = 1,
  replace = FALSE,
  verbose = TRUE
)

add_white_noise(df, response, predictor, white_noise = 0.1, seed = 1)
```

Arguments

df	(required; data frame, tibble, or sf) A training data frame. Default: NULL
response	(required; character string) Name of the response. Must be a column name of df. Default: NULL
predictor	(required; character) Name of the categorical variable to encode. Default: NULL
smoothing	(optional; numeric) Argument of target_encoding_mean(). Minimum group size that keeps the mean of the group. Groups smaller than this have their means pulled towards the global mean of the response. Default: 0.
white_noise	(optional; numeric) Numeric with white noise values in the range 0-1, representing a fraction of the range of the response to be added as noise to the encoded variable. Controls the variability in the encoded variables to mitigate potential overfitting. Default: 0.
seed	(optional; integer) Random seed to facilitate reproducibility. Default: 1
replace	(optional; logical) Advanced option that changes the behavior of the function. Use only if you really know exactly what you need. If TRUE, it replaces each

categorical variable with its encoded version, and returns the input data frame with the replaced variables.

`verbose` (optional; logical) If TRUE, messages and plots generated during the execution of the function are displayed. Default: TRUE

`rnorm_sd_multiplier` (optional; numeric) Numeric with multiplier of the standard deviation of each group in the categorical variable, in the range 0-1. Controls the variability in the encoded variables to mitigate potential overfitting. Default: 1

Value

The input data frame with a target-encoded variable.

Author(s)

Blas M. Benito

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

Examples

```
data(vi)

#subset to limit example run time
vi <- vi[1:1000, ]

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

#without noise
df <- target_encoding_mean(
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)

#with noise
df <- target_encoding_mean(
```

```
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  white_noise = 0.1,
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)


#group rank
#----------

df <- target_encoding_rank(
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)


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

#without noise
df <- target_encoding_loo(
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)

#with noise
df <- target_encoding_loo(
```

```
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  white_noise = 0.1,
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)


#rnorm
#-----

#without sd multiplier
df <- target_encoding_rnorm(
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)

#with sd multiplier
df <- target_encoding_rnorm(
  df = vi,
  response = "vi_mean",
  predictor = "soil_type",
  rnorm_sd_multiplier = 0.1,
  replace = TRUE
)

plot(
  x = df$soil_type,
  y = df$vi_mean,
  xlab = "encoded variable",
  ylab = "response"
)
```

toy *One response and four predictors with varying levels of multicollinearity*

Description

Data frame with known relationship between responses and predictors useful to illustrate multicollinearity concepts. Created from vi using the code shown in the example.

Usage

```
data(toy)
```

Format

Data frame 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

Examples

```
library(collinear)
library(dplyr)
data(vi)
set.seed(1)
toy <- vi |>
  dplyr::slice_sample(n = 2000) |>
  dplyr::transmute(
    a = soil_clay,
    b = humidity_range
  ) |>
  scale() |>
  as.data.frame() |>
  dplyr::mutate(
    y = a * 0.75 + b * 0.25 + runif(n = dplyr::n(), min = -0.5, max = 0.5),
    c = a + runif(n = dplyr::n(), min = -0.5, max = 0.5),
```

```
  d = (a + b) / 2 + runif(n = dplyr::n(), min = -0.5, max = 0.5)
) |>
dplyr::transmute(y, a, b, c, d)
```

`validate_df` *Validate input data frame*

Description

Internal function to validate and prepare the input data frame for a multicollinearity analysis.

Validates a data frame to ensure it complies with the requirements of the package functions. The function performs the following actions:

- Stops if 'df' is NULL.
- Stops if 'df' cannot be coerced to data frame.
- Stops if 'df' has zero rows.
- Removes geometry column if the input data frame is an "sf" object.
- Removes non-numeric columns with as many unique values as rows df has.
- Raise warning if number of rows of 'df' is lower than 'min_rows'.
- Converts logical columns to numeric.
- Converts factor and ordered columns to character.
- Tags the data frame with the attribute `validated = TRUE` to let the package functions skip the data validation.

Usage

```
validate_df(df = NULL, min_rows = 30)
```

Arguments

`df`	(required; data frame or matrix) Input data frame. Default: NULL
`min_rows`	(required; integer) Minimum number of rows required for a pairwise correlation or a variance inflation factor analysis. Default: 30

Value

The input data frame modified to comply with the requirements of the functions in this package

Author(s)

Blas M. Benito

Examples

```
data(vi)

#validating example data frame
vi <- validate_df(
  df = vi
)

#tagged as validated
attributes(vi)$validated
```

`validate_predictors` *Validate the 'predictors' argument for analysis*

Description

Requires the argument 'df' to be validated with `validate_df()`.

Validates the 'predictors' argument to ensure it complies with the requirements of the package functions. It performs the following actions:

- Stops if 'df' is NULL.
- Stops if 'df' is not validated.
- If 'predictors' is NULL, uses column names of 'df' as 'predictors' in the 'df' data frame.
- Raise a warning if there are names in 'predictors' not in the column names of 'df', and returns only the ones in 'df'.
- Stop if the number of numeric columns in 'predictors' is smaller than 'min_numerics'.
- Raise a warning if there are zero-variance columns in 'predictors' and returns a new 'predictors' argument without them.
- Tags the vector with the attribute `validated = TRUE` to let the package functions skip the data validation.

Usage

```
validate_predictors(
  df = NULL,
  response = NULL,
  predictors = NULL,
  min_numerics = 0,
  decimals = 4
)
```

Arguments

df	(required; data frame) A validated data frame with numeric and/or character predictors, and optionally, a response variable. Default: NULL.
response	(optional, character string) Name of a numeric response variable. Used to remove the response from the predictors when predictors is NULL. Character response variables are ignored. Default: NULL.
predictors	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:NULL
min_numerics	(required, integer) Minimum number of numeric predictors required. Default: 1
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

Value

A character vector of validated predictor names

Author(s)

Blas M. Benito

Examples

```
data(
  vi,
  vi_predictors
  )

#validating example data frame
vi <- validate_df(
  df = vi
)

#validating example predictors
vi_predictors <- validate_predictors(
  df = vi,
  predictors = vi_predictors
)

#tagged as validated
attributes(vi_predictors)$validated
```

validate_response *Validate the 'response' argument for target encoding of non-numeric variables*

Description

Requires the argument 'df' to be validated with `validate_df()`.

Usage

```
validate_response(df = NULL, response = NULL, decimals = 4)
```

Arguments

df	(required; data frame) A validated data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
response	(optional, character string) Name of a numeric response variable. Character response variables are ignored. Default: NULL.
decimals	(required, integer) number of decimal places for the zero variance test. Default: 4

Value

character string with name of the response

Author(s)

Blas M. Benito

Examples

```
data(
  vi
)

#validating example data frame
vi <- validate_df(
  df = vi
)

#validating example predictors
response <- validate_response(
  df = vi,
  response = "vi_mean"
)

#tagged as validated
attributes(response)$validated
```

vi *30.000 records of responses and predictors all over the world*

Description

30.000 records of responses and predictors all over the world

Usage

```
data(vi)
```

Format

Data frame with 30.000 rows and 68 columns.

See Also

vi_predictors

vif_df *Variance Inflation Factor*

Description

Computes the Variance Inflation Factor of all variables in a training data frame.

Warning: predictors with perfect correlation might cause errors, please use `cor_select()` to remove perfect correlations first.

The Variance Inflation Factor for a given variable `y` is computed as `1/(1-R2)`, where `R2` is the multiple R-squared of a multiple regression model fitted using `y` as response and all the remaining variables of the input data set as predictors. The equation can be interpreted as "the rate of perfect model's R-squared to the unexplained variance of this model".

The possible range of VIF values is (1, Inf]. A VIF lower than 10 suggest that removing `y` from the data set would reduce overall multicollinearity.

This function computes the Variance Inflation Factor (VIF) in two steps:

- Applies `\link[base]{solve}` to obtain the precision matrix, which is the inverse of the covariance matrix.
- Uses `\link[base]{diag}` to extract the diagonal of the precision matrix, which contains the variance of the prediction of each predictor from all other predictors.

Usage

```
vif_df(df = NULL, response = NULL, predictors = NULL, encoding_method = "mean")
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
response	(recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
predictors	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'
encoding_method	(optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see target_encoding_lab() for further details). Default: "mean"

Value

Data frame with predictor names and VIF values

Author(s)

Blas M. Benito

- 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.

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]

#reduce correlation in predictors with cor_select()
vi_predictors <- cor_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  max_cor = 0.75
)

#without response
#only numeric predictors are returned
df <- vif_df(
  df = vi,
  predictors = vi_predictors
)
```

```
df

#with response
#categorical and numeric predictors are returned
df <- vif_df(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors
)

df
```

`vif_select` *Automated multicollinearity reduction via Variance Inflation Factor*

Description

Automates multicollinearity management by selecting variables based on their Variance Inflation Factor (VIF).

Warning: predictors with perfect correlation might cause errors, please use `cor_select()` to remove perfect correlations first.

The `vif_select()` function is designed to automate the reduction of multicollinearity in a set of predictors by using Variance Inflation Factors.

If the 'response' argument is provided, categorical predictors are converted to numeric via target encoding (see `target_encoding_lab()`). If the 'response' argument is not provided, categorical variables are ignored.

The Variance Inflation Factor for a given variable `y` is computed as `1/(1-R2)`, where `R2` is the multiple R-squared of a multiple regression model fitted using `y` as response and all other predictors in the input data frame as predictors. The VIF equation can be interpreted as the "rate of perfect model's R-squared to the unexplained variance of this model".

The possible range of VIF values is (1, Inf]. A VIF lower than 10 suggest that removing `y` from the data set would reduce overall multicollinearity. The recommended thresholds for maximum VIF may vary depending on the source consulted, being the most common values, 2.5, 5, and 10.

The function `vif_select()` applies a recursive algorithm to remove variables with a VIF higher than a given threshold (defined by the argument `max_vif`).

If the argument `response` is provided, all non-numeric variables in `predictors` are transformed into numeric using target encoding (see `target_encoding_lab()`). Otherwise, non-numeric variables are ignored.

The argument `preference_order` allows defining a preference selection order to preserve (when possible) variables that might be interesting or even required for a given analysis.

For example, if `predictors` is `c("a", "b", "c")` and `preference_order` is `c("a", "b")`, there are two possibilities:

- If the VIF of "a" is higher than the VIF of "b", and both VIF values are above max_vif, then "a" is selected and "b" is removed.
- If their correlation is equal or above max_cor, then "a" is selected, no matter its correlation with "c",

If preference_order is not provided, then the predictors are ranked by their variance inflation factor as computed by vif_df().

Usage

```
vif_select(
  df = NULL,
  response = NULL,
  predictors = NULL,
  preference_order = NULL,
  max_vif = 5,
  encoding_method = "mean"
)
```

Arguments

df	(required; data frame) A data frame with numeric and/or character predictors predictors, and optionally, a response variable. Default: NULL.
response	(recommended, character string) Name of a numeric response variable. Character response variables are ignored. Please, see 'Details' to better understand how providing this argument or not leads to different results when there are character variables in 'predictors'. Default: NULL.
predictors	(optional; character vector) character vector with predictor names in 'df'. If omitted, all columns of 'df' are used as predictors. Default:'NULL'
preference_order	(optional; character vector) vector with column names in 'predictors' in the desired preference order, or result of the function preference_order(). Allows defining a priority order for selecting predictors, which can be particularly useful when some predictors are more critical for the analysis than others. Predictors not included in this argument are ranked by their Variance Inflation Factor. Default: NULL.
max_vif	(optional, numeric) Numeric with recommended values between 2.5 and 10 defining the maximum VIF allowed for any given predictor in the output dataset. Higher VIF thresholds should result in a higher number of selected variables. Default: 5.
encoding_method	(optional; character string). Name of the target encoding method to convert character and factor predictors to numeric. One of "mean", "rank", "loo", "rnorm" (see target_encoding_lab() for further details). Default: "mean"

Value

Character vector with the names of the selected predictors.

Author(s)

Blas M. Benito

- 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.

Examples

```
data(
  vi,
  vi_predictors
)

#subset to limit example run time
vi <- vi[1:1000, ]
vi_predictors <- vi_predictors[1:10]

#reduce correlation in predictors with cor_select()
vi_predictors <- cor_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  max_cor = 0.75
)

#without response
#without preference_order
#permissive max_vif
#only numeric predictors are processed
selected.predictors <- vif_select(
  df = vi,
  predictors = vi_predictors,
  max_vif = 10
)

selected.predictors

#without response
#without preference_order
#restrictive max_vif
#only numeric predictors are processed
selected.predictors <- vif_select(
  df = vi,
  predictors = vi_predictors,
  max_vif = 2.5
)

selected.predictors

#with response
#without preference_order
```

```
#restrictive max_cor
#slightly different solution than previous one
#because categorical variables are target-enccoded
selected.predictors <- vif_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  max_vif = 2.5
)

selected.predictors

#with response
#with user-defined preference_order
#restrictive max_cor
#numerics and categorical variables in output
selected.predictors <- vif_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = c(
    "soil_type", #categorical variable
    "soil_temperature_mean",
    "swi_mean",
    "rainfall_mean",
    "evapotranspiration_mean"
  ),
  max_vif = 2.5
)

selected.predictors


#with response
#with automated preference_order
#restrictive max_cor and max_vif
#numerics and categorical variables in output
preference.order <- preference_order(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  f = f_rsquared #cor(response, predictor)
)

head(preference.order)

selected.predictors <- vif_select(
  df = vi,
  response = "vi_mean",
  predictors = vi_predictors,
  preference_order = preference.order,
  max_vif = 2.5
)
```

```
selected.predictors
```

`vi_predictors` *Predictor names in data frame 'vi'*

Description

Predictor names in data frame 'vi'

Usage

```
data(vi_predictors)
```

Format

Character vector with predictor names.

See Also

vi

Index