Package ‘funModeling’

April 1, 2024

Type Package

Title Exploratory Data Analysis and Data Preparation Tool-Box

Description Around 10% of almost any predictive modeling project is spent in predictive modeling, 'funModeling' and the book Data Science Live Book (<https://livebook.datascienceheroes.com/>) are intended to cover remaining 90%: data preparation, profiling, selecting best variables 'dataViz', assessing model performance and other functions.

Version 1.9.5

Date 2024-03-30

Maintainer Pablo Casas <pcasas.biz@gmail.com>

License GPL-2

BugReports https://github.com/pablo14/funModeling/issues

URL https://livebook.datascienceheroes.com

LazyData true

Encoding UTF-8

Imports ROCR, ggplot2, gridExtra, pander, reshape2, scales, dplyr, lazyeval, utils, RColorBrewer, moments, entropy, cli, stringr

Depends R (>= 3.4.0), Hmisc (>= 3.17.1)

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Author Pablo Casas [aut, cre]

Repository CRAN

Date/Publication 2024-04-01 08:00:05 UTC

R topics documented:

`funModeling-package` *funModeling: Exploratory data analysis, data preparation and model performance*

Description

funModeling is intimately related to the Data Science Live Book -Open Source- (2017) in the sense that most of its functionality is used to explain different topics addressed by the book.

Details

To start using funModeling you can start by the vignette: ‘browseVignettes(package = "funModeling")‘

Or you can read the Data Science Live Book, fully accessible at: `https://livebook.datascienceheroes.com`

Author(s)

Maintainer: Pablo Casas `<pcasas.biz@gmail.com>`

See Also

Useful links:

- `https://livebook.datascienceheroes.com`
- Report bugs at `https://github.com/pablo14/funModeling/issues`

`auto_grouping` *Reduce cardinality in categorical variable by automatic grouping*

Description

Reduce the cardinality of an input variable based on a target -binary by now- variable based on attribitues of accuracy and representativity, for both input and target variable. It uses a cluster model to create the new groups.

Usage

```
auto_grouping(data, input, target, n_groups, model = "kmeans", seed = 999)
```

Arguments

`data`	data frame source
`input`	categorical variable indicating
`target`	string of the variable to optimize the re-grouping
`n_groups`	number of groups for the new category based on input, normally between 3 and 10.
`model`	is the clustering model used to create the grouping, supported models: "kmeans" (default) or "hclust" (hierarchical clustering).
`seed`	optional, random number used internally for the k-means, changing this value will change the model

Value

A list containing 3 elements: recateg_results which contains the description of the target variable with the new groups; df_equivalence is a data frame containing the input category and the new category; fit_cluster which is the cluster model used to do the re-grouping

Examples

```
# Reducing quantity of countries based on has_flu variable
auto_grouping(data=data_country, input='country', target="has_flu", n_groups=8)
```

`categ_analysis` *Profiling analysis of categorical vs. target variable*

Description

Retrieves a complete summary of the grouped input variable against the target variable. Type of target variable must be binary for now. A positive case will be the less representative one. It returns the total positive cases (sum_target)); pecentage of total positive cases (perc_target) that fell in that category (this column sums 1); likelihood or mean of positive cases (mean_target) measured by the total positive cases over total cases in that category; quantity of rows of that category (q_rows) and in percentage (perc_rows) -this column sums 1.

Usage

```
categ_analysis(data, input, target)
```

Arguments

`data`	input data containing the variable to describe
`input`	string input variable (if empty, it runs for all categorical variable), it can take a single character value or a character vector.
`target`	string target variable. Binary or two class is only supported by now.

Value

if input has 1 variable, it retrurns a data frame indicating all the metrics, otherwise prints in console all variable results.

Examples

```
categ_analysis(data_country, "country", "has_flu")
```

`compare_df` *Compare two data frames by keys*

Description

Obtain differences between two data frames

Usage

```
compare_df(dfcomp_x, dfcomp_y, keys_x, keys_y = NA, compare_values = FALSE)
```

Arguments

`dfcomp_x`	first data frame to compare
`dfcomp_y`	second data frame to compare
`keys_x`	keys of the first dataframe
`keys_y`	(optional) keys of the second dataframe, if missing both data frames will be compared with the keys_x
`compare_values`	(optional) if TRUE it will not only compare keys, but also will check if the values of non-key matching columns have the same values

Value

Differences and coincident values

Examples

```
data(heart_disease)
a=heart_disease
b=heart_disease
a=subset(a, age >45)
b=subset(b, age <50)
b$gender='male'
b$chest_pain=ifelse(b$chest_pain ==3, 4, b$chest_pain)
res=compare_df(a, b, c('age', 'gender'))
# Print the keys that didn't match
res
# Accessing the keys not present in the first data frame
res[[1]]$rows_not_in_X
```

```
# Accessing the keys not present in the second data frame
res[[1]]$rows_not_in_Y
# Accessing the keys which coincide completely
res[[1]]$coincident
# Accessing the rows whose values did not coincide
res[[1]]$different_values
```

`concatenate_n_vars` *Concatenate 'N' variables*

Description

Concatenate 'N' variables using the char pipe: <|>. This function is used when there is the need of measuring the mutual information and/or the information gain between 'N' input variables an against a target variable. This function makes sense when it is used based on categorical data.

Usage

```
concatenate_n_vars(data, vars)
```

Arguments

`data`	data frame containing the two variables to concatenate
`vars`	character vector containing all variables to concatenate

Value

vector containing the concatenated values for the given variables

Examples

```
new_variable=concatenate_n_vars(mtcars, c("cyl", "disp"))
# Checking new variable
head(new_variable)
```

`convert_df_to_categoric` *Convert every column in a data frame to character*

Description

It converts all the variables present in 'data' to character. Criteria conversion is based on two functions, `discretize_get_bins` plus `discretize_df`, which will discretize all the numerical variables based on equal frequency criteria, with the number of bins equal to 'n_bins'. This only applies for numerical variables which unique valuesare more than 'n_bins' parameter. After this step, it may happen that variables remain non-character, so these variables will be converting directly into character.

Usage

```
convert_df_to_categoric(data, n_bins)
```

Arguments

`data`	input data frame to discretize
`n_bins`	number of bins/segments for each variable

Value

data frame containing all variables as character

Examples

```
# before
df_status(heart_disease)

# after
new_df=convert_df_to_categoric(data=heart_disease, n_bins=5)
df_status(new_df)
```

`coord_plot` *Coordinate plot*

Description

Calculate the means (or other function defined in 'group_func' parameter) per group to analyze how each segment behave. It scales each variable mean inti the 0 to 1 range to easily profile the groups according to its mean. It also calculate the mean regardless the grouping. This function is also useful when you want to profile cluster results in terms of its means.

Usage

```
coord_plot(data, group_var, group_func = mean, print_table = FALSE)
```

Arguments

`data`	input data source
`group_var`	variable to make the group by
`group_func`	the data type of this parameter is a function, not an string, this is the function to be used in the group by, the default value is: mean
`print_table`	False by default, if true it retrieves the mean table used to generate the plot.

Value

coordinate plot, if print_table=T it also prints a table with the average per column plus the average of the whole column

Examples

```
# calculating the differences based on function 'mean'
coord_plot(data=mtcars, group_var="cyl")
# printing the table used to generate the coord_plot
coord_plot(data=mtcars, group_var="cyl", print_table=TRUE)
# printing the table used to generate the coord_plot
coord_plot(data=mtcars, group_var="cyl", group_func=median, print_table=TRUE)
```

`correlation_table` *Get correlation against target variable*

Description

Obtain correlation table for all variables against target variable. Only numeric variables are analyzed (factor/character are skippted automatically).

Usage

```
correlation_table(data, target)
```

Arguments

`data`	data frame
`target`	string variable to predict

Value

Correlation index for all data input variable

Examples

```
correlation_table(data=heart_disease, target="has_heart_disease")
```

---

cross_plot *Cross-plotting input variable vs. target variable*

Description

The cross_plot shows how the input variable is correlated with the target variable, getting the likelihood rates for each input's bin/bucket .

Usage

```
cross_plot(data, input, target, path_out, auto_binning, plot_type = "both")
```

Arguments

`data`	data frame source
`input`	input variable name (if empty, it runs for all numeric variable), it can take a single character value or a character vector.
`target`	variable name to predict
`path_out`	path directory, if it has a value the plot is saved
`auto_binning`	indicates the automatic binning of input variable based on equal frequency (function 'equal_freq'), default value=TRUE
`plot_type`	indicates if the output is the 'percentual' plot, the 'quantity' or 'both' (default).

Value

cross plot

Examples

```
## Example 1:
cross_plot(data=heart_disease, input="chest_pain", target="has_heart_disease")

## Example 2: Disabling auto_binning:
cross_plot(data=heart_disease, input="oldpeak",
target="has_heart_disease", auto_binning=FALSE)

## Example 3: Saving the plot into a folder:
#cross_plot(data=heart_disease, input="oldpeak",
# target="has_heart_disease", path_out = "my_folder")

## Example 4: Running with multiple input variables at the same time:
cross_plot(data=heart_disease, input=c("age", "oldpeak", "max_heart_rate"),
target="has_heart_disease")
```

`data_country` *People with flu data*

Description

Each row represents a person from different countries indicating if he or she has or not flu. Colmuns person: unique id country: country of the person, 70 different countries has_flu: character variable with values "yes" or "no" indicating if the person has flu

Usage

```
data_country
```

Format

A data frame with 910 rows and 3 variables

`data_golf` *Play golf*

Description

This well known small data frame containst 14 cases indicating wheter or not play golf based on wheather conditions. Target variable: 'play_golf.'

Usage

```
data_golf
```

Format

A data frame with 14 rows and 3 variables

`data_integrity` *Data integrity*

Description

A handy function to return different vectors of variable names aimed to quickly filter NA, categorical (factor / character), numerical and other types (boolean, date, posix). It also returns a vector of variables which have high cardinality. It returns an 'integrity' object, which has: 'status_now' (comes from status function), and 'results' list, following elements can be found:

vars_cat: Vector containing the categorical variables names (factor or character)

vars_num: Vector containing the numerical variables names

vars_char: Vector containing the character variables names

vars_factor: Vector containing the factor variables names

vars_other: Vector containing the other variables names (date time, posix and boolean)

vars_num_with_NA: Summary table for numerical variables with NA

vars_cat_with_NA: Summary table for categorical variables with NA

vars_cat_high_card: Summary table for high cardinality variables (where thershold = MAX_UNIQUE parameter)

vars_one_value: Vector containing the variables names with 1 unique different value

Explore the NA and high cardinality variables by doing summary(integrity_object), or a full summary by doing print(integrity_object)

Usage

```
data_integrity(data, MAX_UNIQUE = 35)
```

Arguments

`data`	data frame or a single vector
`MAX_UNIQUE`	max unique threshold to flag a categorical variable as a high cardinality one. Normally above 35 values it is needed to reduce the number of different values.

Value

An 'integrity' object.

Examples

```
# Example 1:
data_integrity(heart_disease)
# Example 2:
# changing the default minimum threshold to flag a variable as high cardiniality
data_integrity(data=data_country, MAX_UNIQUE=50)
```

`data_integrity_model` *Check data integrity model*

Description

Given a data frame, we need to create models (xgboost, random forest, regression, etc). Each one of them has its constraints regarding data types. Many errors appear when we are creating models just because of data format. This function returns, given a certain model, which are the constraints that the data is not satisfying. This way we can anticipate and correct errors before we call for model creation. This function is quite related to `data_integrity`.

Usage

```
data_integrity_model(data, model_name, MAX_UNIQUE = 35)
```

Arguments

`data`	data frame or a single vector
`model_name`	model name, you can check all the available models by printing ‘metadata_models‘ data frame.
`MAX_UNIQUE`	max unique threshold to flag a categorical variable as a high cardinality one. Normally above 35 values it is needed to reduce the number of different values. # Example 1: data_integrity_model(data=heart_disease, model_name="pca") # Example 2: # changing the default minimum threshold to flag a variable as high cardiniality data_integrity_model(data=iris, model_name="xgboost", MAX_UNIQUE=50)

Value

an ‘integritymodel‘ object

`desc_groups` *Profiling categorical variable*

Description

Calculate the means (or other function) per group to analyze how each segment behave. It scales each variable mean inti the 0 to 1 range to easily profile the groups according to its mean. It also calculate the mean regardless the grouping. This function is also useful when you want to profile cluster results in terms of its means. It automatically adds a row representing the sumarization of the column regardless the group_var categories, this is useful to compare each segement with the whole population. It will exclude all factor/character variables.

Usage

```
desc_groups(data, group_var, group_func = mean, add_all_data_row = T)
```

Arguments

`data`	input data source
`group_var`	variable to make the group by
`group_func`	the data type of this parameter is a function, not an string, this is the function to be used in the group by, the default value is: mean
`add_all_data_row`	flag indicating if final data contains the row: 'All_Data', which is the function applied regardless the grouping. Useful to compare with the rest of the values.

Value

grouped data frame

Examples

```
# default grouping function: mean
desc_groups(data=mtcars, group_var="cyl")

# using the median as the grouping function
desc_groups(data=mtcars, group_var="cyl", group_func=median)

# using the max as the grouping function
desc_groups(data=mtcars, group_var="gear", group_func=max)
```

`desc_groups_rank` *Profiling categorical variable (rank)*

Description

Similar to 'desc_groups' function, this one computes the rank of each value in order to quickly know what is the value in each segment that has the highest value (rank=1). 1 represent the highest number. It will exclude all factor/character variables.

Usage

```
desc_groups_rank(data, group_var, group_func = mean)
```

Arguments

`data`	input data source
`group_var`	variable to make the group by
`group_func`	the data type of this parameter is a function, not an string, this is the function to be used in the group by, the default value is: mean

Value

grouped data frame, showing the rank instead of the absolute values/

Examples

```
# default grouping function: mean
desc_groups_rank(data=mtcars, group_var="gear")

# using the median as the grouping function
desc_groups(data=mtcars, group_var="cyl", group_func=median)

# using the max as the grouping function
desc_groups_rank(data=mtcars, group_var="gear", group_func=max)
```

`df_status` *Get a summary for the given data frame (o vector).*

Description

For each variable it returns: Quantity and percentage of zeros (q_zeros and p_zeros respectevly). Same metrics for NA values (q_NA/p_na), and infinite values (q_inf/p_inf). Last two columns indicates data type and quantity of unique values. This function print and return the results.

Usage

```
df_status(data, print_results)
```

Arguments

`data`	data frame or a single vector
`print_results`	if FALSE then there is not a print in the console, TRUE by default.

Value

Metrics data frame

Examples

```
df_status(heart_disease)
```

`discretize_df` *Discretize a data frame*

Description

Converts all numerical variables into factor or character, depending on 'stringsAsFactors' parameter, based on equal frequency criteria. The thresholds for each segment in each variable are generated based on the output of `discretize_get_bins` function, which returns a data frame containing the threshold for each variable. This result is must be the 'data_bins' parameter input. Important to note that the returned data frame contains the non-transformed variables plus the transformed ones. More info about converting numerical into categorical variables can be found at: `https://livebook.datascienceheroes.com/data-preparation.html#data_types`

Usage

```
discretize_df(data, data_bins, stringsAsFactors = TRUE)
```

Arguments

`data`	Input data frame
`data_bins`	data frame generated by 'discretize_get_bins' function. It contains the variable name and the thresholds for each bin, or segment.
`stringsAsFactors`	Boolean variable which indicates if the discretization result is character or factor. When TRUE, the segments are ordered. TRUE by default.

Value

Data frame with the transformed variables

Examples

```
# Getting the bins thresholds for each. If input is missing,
# will run for all numerical variables.
d_bins=discretize_get_bins(data=heart_disease,
input=c("resting_blood_pressure", "oldpeak"), n_bins=5)

# Now it can be applied on the same data frame,
# or in a new one (for example in a predictive model that
# change data over time)
heart_disease_discretized=
discretize_df(data=heart_disease,
data_bins=d_bins,
stringsAsFactors=TRUE)
```

`discretize_get_bins` *Get the data frame thresholds for discretization*

Description

It takes a data frame and returns another data frame indicating the threshold for each bin (or segment) in order to discretize the variable.

Usage

```
discretize_get_bins(data, n_bins = 5, input = NULL)
```

Arguments

`data`	Data frame source
`n_bins`	The number of desired bins (or segments) that each variable will have.
`input`	Vector of string containing all the variables that will be processed. If empty it will run for all numerical variables that match the following condition, the number of unique values must be higher than the ones defined at 'n_bins' parameter. NAs values are automatically handled by converting them into another category (more info about it at `https://livebook.datascienceheroes.com/data-preparation.html#treating-missing-values-in-numerical-variables`). This function must be used with discretize_df. If it is needed a different number of bins per variable, then the function must be called more than once.

Value

Data frame containing the thresholds or cuts to bin every variable

`discretize_rgr` *Variable discretization by gain ratio maximization*

Description

Discretize numeric variable by maximizing the gain ratio between each bucket and the target variable.

Usage

```
discretize_rgr(input, target, min_perc_bins = 0.1, max_n_bins = 5)
```

Arguments

`input`	numeric input vector to discretize
`target`	character or factor multi-calss target variable
`min_perc_bins`	minimum percetange of rows for each split or segment (controls the sample size), 0,1 (or 10 percent) as default
`max_n_bins`	maximum number of bins or segments to split the input variable, 5 bins as default

Value

discretized variable (factor)

Examples

```
library(funModeling)
data=heart_disease
input=data$oldpeak
target=as.character(data$has_heart_disease)

input2=discretize_rgr(input, target)

# checking:
summary(input2)
```

`entropy_2` *Computes the entropy between two variables*

Description

It calculates the entropy between two categorical variables using log2. This log2 is mentioned in most of the Claude Shannon bibliography. Input/target can be numeric or character.

Usage

```
entropy_2(input, target)
```

Arguments

`input`	numeric/character vector
`target`	numeric/character vector

Value

Entropy measured in bits

Examples

```
# Measuring entropy between input and target variable
entropy_2(input=data_golf$outlook, target=data_golf$play_golf)
```

`equal_freq` *Equal frequency binning*

Description

Equal frequency tries to put the same quantity of cases per bin when possible. It's a wrapper of function cut2 from Hmisc package.

Usage

```
equal_freq(var, n_bins)
```

Arguments

`var`	input variable
`n_bins`	number of bins to split 'var' by equal frequency, if it not possible to calculate for the desired bins, it returns the closest number

Value

The binned variable.

Examples

```
## Example 1
summary(heart_disease$age)
age_2=equal_freq(var=heart_disease$age, n_bins = 10)
summary(age_2)

## Example 2
age_3=equal_freq(var=heart_disease$age, n_bins = 5)
summary(age_3)
```

`export_plot` *Export plot to jpeg file*

Description

Export 'object_plot' to jpeg file under the name 'file_name' in the directory 'path_out'

Usage

```
export_plot(object_plot, path_out, file_name)
```

Arguments

`object_plot`	Object plot to export (like ggplot2)
`path_out`	path directory to export the output, if it has a value the plot is saved, if the directory doesn't existis it will try to create it. To save in current directory path must be dot: "."
`file_name`	output file name

Value

none

`fibonacci` *Fibonacci series*

Description

It retrieves a vector containing the first N numbers specified in 'length' parameter of the Fibonacci series.

Usage

```
fibonacci(length, remove_first = FALSE)
```

Arguments

`length`	data frame
`remove_first`	removes the first value of the series, because first 2 elements are the same (number=1). False by default.

Value

vector

Examples

```
# Get the first 4 elements of Fibonacci series
fibonacci(4)
```

`freq` *Frequency table for categorical variables*

Description

Retrieves the frequency and percentage for input

Usage

```
freq(data, input = NA, plot = TRUE, na.rm = FALSE, path_out)
```

Arguments

`data`	input data containing the variable to describe
`input`	string input variable (if empty, it runs for all numeric variable), it can take a single character value or a character vector.
`plot`	flag indicating if the plot is desired, TRUE by default
`na.rm`	flag indicating if NA values must be included in the analysis, FALSE by default
`path_out`	path directory, if it has a value the plot is saved

Value

vector with the values scaled into the 0 to 1 range

Examples

```
freq(data=heart_disease$thal)
freq(data=heart_disease, input = c('thal','chest_pain'))
```

`gain_lift` *Generates lift and cumulative gain performance table and plot*

Description

It retrieves the cumulative positive rate -gain curve- and the lift chart & plot when score is divided in 5, 10 or 20 segments. Both metrics give a quality measure about how well the model predicts. Higher values at the beginning of the population implies a better model. More info at: `https://livebook.datascienceheroes.com/model-performance.html#scoring_data`

Usage

```
gain_lift(data, score, target, q_segments = 10)
```

Arguments

`data`	input data source
`score`	the variable which contains the score number, or likelihood of being positive class
`target`	target binary variable indicating class label
`q_segments`	quantity of segments to split score variable, valid values: 5, 10 or 20

Value

lift/gain table, column: gain implies how much positive cases are catched if the cut point to define the positive class is set to the column "Score Point"

Examples

```
fit_glm=glm(has_heart_disease ~ age + oldpeak, data=heart_disease, family = binomial)
heart_disease$score=predict(fit_glm, newdata=heart_disease, type='response')
gain_lift(data=heart_disease, score='score', target='has_heart_disease')
```

`gain_ratio` *Gain ratio*

Description

Computes the information gain between an ’input’ and ’target’ variable (using log2). Similar to information gain but less sensitive to high cardinality variables.

Usage

```
gain_ratio(input, target)
```

Arguments

`input`	numeric/character vector
`target`	numeric/character vector

Value

gain ratio

Examples

```

gain_ratio(input=data_golf$outlook, target=data_golf$play_golf)
```

`get_sample` *Sampling training and test data*

Description

Split input data into training and test set, retrieving always same sample by setting the seed.

Usage

```
get_sample(data, percentage_tr_rows = 0.8, seed = 987)
```

Arguments

`data`	input data source
`percentage_tr_rows`	percentage of training rows, range value from 0.1 to 0.99, default value=0.8 (80 percent of training data)
`seed`	to generate the sample randomly, default value=987

Value

TRUE/FALSE vector same length as 'data' param. TRUE represents that row position is for training data

Examples

```
# Training and test data. Percentage of training cases default value=80%.
index_sample=get_sample(data=heart_disease, percentage_tr_rows=0.8)
# Generating the samples
data_tr=heart_disease[index_sample,]
data_ts=heart_disease[-index_sample,]
```

`hampel_outlier` *Hampel Outlier Threshold*

Description

Retrieves the bottom and top boundaries to flag outliers or extreme values, according to the Hampel method. This technique takes into account the median and MAD value, which is a is a robust measure of the variability of a univariate sample of quantitative data (Wikipedia). Similar to standard deviation but less sensitve to outliers. This function is used in 'prep_outliers' function. All 'NA's values are automatically excluded. More information at: `https://livebook.datascienceheroes.com/data-preparation.html#how_to_deal_with_outliers_in_r`.

Usage

```
hampel_outlier(input, k_mad_value = 3)
```

Arguments

`input`	Numeric variable vector
`k_mad_value`	'K' multiplier for the median absolute deviation. The higher the value, the more outliers will be detected. Default value=3 (it's an standad)

Value

A two-item vector, the first value represents the bottom threshold, while the second one is the top threshold

Examples

```
hampel_outlier(heart_disease$age)
```

`heart_disease` *Heart Disease Data*

Description

There are variables related to patient clinic trial. The variable to predict is 'has_heart_disease'.

Usage

```
heart_disease
```

Format

A data frame with 303 rows and 16 variables:

`https://archive.ics.uci.edu/ml/datasets/Heart+Disease`

`information_gain` *Information gain*

Description

Computes the information gain between an 'input' and 'target' variable (using log2). In general terms, the higher the more predictable the input is.

Usage

```
information_gain(input, target)
```

Arguments

`input`	numeric/character vector
`target`	numeric/character vector

Value

information gain

Examples

```
information_gain(input=data_golf$outlook, target=data_golf$play_golf)
```

`infor_magic` *Computes several information theory metrics between two vectors*

Description

It retrieves the same as `var_rank_info` but receiving two vectors. Metrics are: entropy (en), mutual information (mi), information gain (ig) and gain ratio (gr).

Usage

```
infor_magic(input, target)
```

Arguments

`input`	vector to be evaluated against the variable defined in 'target' parameter
`target`	vector containing the output variable.

Value

Matrix of 1 row and 4 columns, where each column represent the mentioned metrics

Examples

```
infor_magic(data_golf$outlook, data_golf$play_golf)
```

`metadata_models` *Metadata models data integrity*

Description

Metadata models data integrity

Usage

```
metadata_models
```

Format

Tibble

`plotar` *Correlation plots*

Description

Visual correlation analysis. Plot different graphs in order to expose the inner information of any numeric variable against the target variable

Usage

```
plotar(data, input, target, plot_type, path_out)
```

Arguments

`data`	data frame source
`input`	string input variable (if empty, it runs for all numeric variable), it can take a single character value or a character vector.
`target`	string of the variable to predict, it supports binary or multinominal values.
`plot_type`	Indicates the type of plot to retrieve, available values: "boxplot" or "histdens".
`path_out`	path directory, if it has a value the plot is saved. To save in current directory path must be dot: "."

Value

Single or multiple plots specified by 'plot_type' parameter

Examples

```
## It runs for all numeric variables automatically
plotar(data=heart_disease, target="has_heart_disease", plot_type="histdens")

plotar(heart_disease, input = 'age', target = 'chest_pain', plot_type = "boxplot")
```

`plot_num` *Plotting numerical data*

Description

Retrieves one plot containing all the histograms for numerical variables. NA values will not be displayed.

Usage

```
plot_num(data, bins = 10, path_out = NA)
```

Arguments

`data`	data frame
`bins`	number of bars (bins) to plot each histogram, 10 by default
`path_out`	path directory to export the output, if it has a value the plot is saved, if the directory doesn't existis it will try to create it. To save in current directory path must be dot: "."

Value

plot containing all numerical variables

Examples

```
plot_num(mtcars)
# changing the bins parameter and exporting the plot
# plot_num(data=mtcars, bins=5, path_out="my_folder")
```

`prep_outliers` *Outliers Data Preparation*

Description

Deal with outliers by setting an 'NA value' or by 'stopping' them at a certain. There are three supported methods to flag the values as outliers: "bottom_top", "tukey" and "hampel". The parameters: 'top_percent' and/or 'bottom_percent' are used only when method="bottom_top".

For a full reference please check the official documentation at: https://livebook.datascienceheroes.com/data-preparation.html#treatment_outliers> Setting NA is recommended when doing statistical analysis, parameter: type='set_na'. Stopping is recommended when creating a predictive model without biasing the result due to outliers, parameter: type='stop'.

The function can take a data frame, and returns the same data plus the transformations specified in the input parameter. Or it can take a single vector (in the same 'data' parameter), and it returns a vector.

Usage

```
prep_outliers(
  data,
  input = NA,
  type = NA,
  method = NA,
  bottom_percent = NA,
  top_percent = NA,
  k_mad_value = NA
)
```

Arguments

`data`	a data frame or a single vector. If it's a data frame, the function returns a data frame, otherwise it returns a vector.
`input`	string input variable (if empty, it runs for all numeric variable).
`type`	can be 'stop' or 'set_na', in the first case all falling out of the threshold will be converted to the threshold, on the other case all of these values will be set as NA.
`method`	indicates the method used to flag the outliers, it can be: "bottom_top", "tukey" or "hampel".
`bottom_percent`	value from 0 to 1, represents the lowest X percentage of values to treat. Valid only when method="bottom_top".
`top_percent`	value from 0 to 1, represents the highest X percentage of values to treat. Valid only when method="bottom_top".
`k_mad_value`	only used when method='hampel', 3 by default, might seem quite restrictive. Set a higher number to spot less outliers.

Value

A data frame with the desired outlier transformation

Examples

```
# Creating data frame with outliers
set.seed(10)
df=data.frame(var1=rchisq(1000,df = 1), var2=rnorm(1000))
df=rbind(df, 1135, 2432) # forcing outliers
df$id=as.character(seq(1:1002))

# for var1: mean is ~ 4.56, and max 2432
summary(df)

########################################################
### PREPARING OUTLIERS FOR DESCRIPTIVE STATISTICS
########################################################

#### EXAMPLE 1: Removing top 1%% for a single variable
# checking the value for the top 1% of highest values (percentile 0.99), which is ~ 7.05
quantile(df$var1, 0.99)

# Setting type='set_na' sets NA to the highest value specified by top_percent.
# In this case 'data' parameter is single vector, thus it returns a single vector as well.
var1_treated=prep_outliers(data = df$var1, type='set_na', top_percent  = 0.01,method = "bottom_top")

# now the mean (~ 1) is more accurate, and note that: 1st, median and 3rd
#  quartiles remaining very similar to the original variable.
summary(var1_treated)

#### EXAMPLE 2: Removing top and bottom 1% for the specified input variables.
vars_to_process=c('var1', 'var2')
df_treated3=prep_outliers(data = df, input = vars_to_process, type='set_na',
 bottom_percent = 0.01, top_percent  = 0.01, method = "bottom_top")
summary(df_treated3)

########################################################
### PREPARING OUTLIERS FOR PREDICTIVE MODELING
########################################################

data_prep_h=funModeling::prep_outliers(data = heart_disease,
input = c('age','resting_blood_pressure'),
 method = "hampel",  type='stop')

# Using Hampel method to flag outliers:
summary(heart_disease$age);summary(data_prep_h$age)
# it changed from 29 to 29.31, and the max remains the same at 77
hampel_outlier(heart_disease$age) # checking the thresholds

data_prep_a=funModeling::prep_outliers(data = heart_disease,
input = c('age','resting_blood_pressure'),
```

```
 method = "tukey",  type='stop')

max(heart_disease$age);max(data_prep_a$age)
# remains the same (77) because the max thers for age is 100
tukey_outlier(heart_disease$age)
```

`profiling_num` *Profiling numerical data*

Description

Get a metric table with many indicators for all numerical variables, automatically skipping the non-numerical variables. Current metrics are: mean, std_dev: standard deviation, all the p_XX: percentile at XX number, skewness, kurtosis, iqr: inter quartile range, variation_coef: the ratio of sd/mean, range_98 is the limit for which the 98

Usage

```
profiling_num(data)
```

Arguments

`data`	data frame

Value

metrics table

Examples

```
profiling_num(mtcars)
```

`range01` *Transform a variable into the [0-1] range*

Description

Range a variable into [0-1], assigning 0 to the min and 1 to the max of the input variable. All NA values will be removed.

Usage

```
range01(var)
```

Arguments

`var`	numeric input vector

Value

vector with the values scaled into the 0 to 1 range

Examples

```
range01(mtcars$cyl)
```

`status` *Get a summary for the given data frame (o vector).*

Description

For each variable it returns: Quantity and percentage of zeros (q_zeros and p_zeros respectevly). Same metrics for NA values (q_NA/p_na), and infinite values (q_inf/p_inf). Last two columns indicates data type and quantity of unique values. 'status' function is the evolution of 'df_status'. Main change is to have the decimal points as it is, except in percentage. For example now p_na=0.04 means 4 This time it's easier to embbed in a data process flow and to take actions based on this number.

Usage

```
status(data)
```

Arguments

`data`	data frame, tibble or a single vector

Value

Tibble with metrics

Examples

```
status(heart_disease)
```

`tukey_outlier` *Tukey Outlier Threshold*

Description

Retrieves the bottom and top boundaries to flag outliers or extreme values, according to the Tukey's test. More info at `https://en.wikipedia.org/wiki/Outlier#Tukey.27s_test` This function is used in 'prep_outliers' function. All 'NA's values are automatically excluded. More information at: `https://livebook.datascienceheroes.com/data-preparation.html#how_to_deal_with_outliers_in_r`.

Usage

```
tukey_outlier(input)
```

Arguments

`input`	Numeric variable vector

Value

A two-item vector, the first value represents the bottom threshold, while the second one is the top threshold

Examples

```
tukey_outlier(heart_disease$age)
```

`var_rank_info` *Importance variable ranking based on information theory*

Description

Retrieves a data frame containing several metrics related to information theory. Metrics are: entropy (en), mutual information (mi), information gain (ig) and gain ratio (gr).

Usage

```
var_rank_info(data, target)
```

Arguments

`data`	input data frame, all the variables will be evaluated against the variable defined in 'target' parameter
`target`	string variable name containing the output variable.

Value

data frame ordered by gain ratio metric

Examples

```
var_rank_info(data_golf, "play_golf")
```

`v_compare` *Compare two vectors*

Description

Obtaing coincident and not coincident elements between two vectors.

Usage

```
v_compare(vector_x, vector_y)
```

Arguments

`vector_x`	1st vector to compare
`vector_y`	2nd vector to compare

Value

Correlation index for all data input variable

Examples

```
v1=c("height","weight","age")
v2=c("height","weight","location","q_visits")
res=v_compare(vector_x=v1, vector_y=v2)
# Print the keys that didn't match
res
# Accessing the keys not present in
```

Index