Package ‘oetteR’

March 13, 2018

Type Package

Title Collection of personal R functions

Version 0.2.1

Author Bjoern Oettinghaus

Maintainer Bjoern Oettinghaus <bjoern.oettinghaus@gmail.com>

Description Collection of functions I frequently use

License None

LazyData true

RoxygenNote 6.0.1

Imports tidyverse
, DT
, HDtweedie
, MASS
, RColorBrewer
, broom
, dplyr
, fields
, forcats
, ggalluvial
, ggplot2
, ggpubr
, glmnet
, gridExtra
, htmltools
, lubridate
, magrittr
, modelr
, nycflights13
, pipelearner
, plotly
, purrr
, readr
, recipes
, rlang
, rmarkdown
, rminer
, stringr

, tabplot
, tibble
, tidyr
, webshot
, ISLR
, e1071
, gamlss
, randomForest
, rpart
, ROCR
, caret
, rpart.plot
, RColorBrewer

Suggests testthat,
knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

f_boxcox 4
f_clean_data 5
f_clean_data_no_changes 6
f_content 6
f_content_get_file_type 7
f_content_get_path 7
f_content_render 8
f_datatable_minimal 8
f_datatable_universal 9
f_html_breaks 10
f_html_filename_2_link 10
f_html_get_title_from_Rmd 11
f_html_padding 11
f_html_table_html_and_rmd_link 12
f_manip_append_2_list 13
f_manip_bin_numerics 13
f_manip_bring_to_pos_range 14
f_manip_data_2_model_matrix_format 15
f_manip_double_2_int 16
f_manip_factor_2_numeric 16
f_manip_get_most_common_level 17
f_manip_get_response_variable_from_formula 17
f_manip_get_variables_from_formula 18
f_manip_matrix_2_tibble 18
f_manip_summarize_2_median_and_most_common_factor 19
f_manip_transpose_tibble 20
f_model_add_predictions_2_grid_regression 20
f_model_data_grid 21
f_model_importance 22
f_model_importance_plot 22
f_model_importance_plot_tableplot 23

f_model_importance_pl_add_plots_regression . . . 24
f_model_importance_pl_plots_as_html . . . 26
f_model_importance_randomForest . . . 27
f_model_importance_rpart . . . 27
f_model_importance_svm . . . 28
f_model_plot_variable_dependency_regression . . . 29
f_model_plot_var_dep_over_spec_var_range . . . 30
f_model_seq_range . . . 32
f_pca . . . 33
f_pca_plot_components . . . 34
f_pca_plot_variance_explained . . . 35
f_plot_adjust_col_vector_length . . . 36
f_plot_alluvial . . . 36
f_plot_alluvial_1v1 . . . 38
f_plot_color_code_variables . . . 40
f_plot_col_vector74 . . . 41
f_plot_generate_comparison_pairs . . . 41
f_plot_hist . . . 42
f_plot_obj_2_html . . . 43
f_plot_pretty_points . . . 47
f_plot_profit_bars_plus_area . . . 48
f_plot_profit_lines . . . 49
f_plot_time . . . 50
f_predict_plot_model_performance_regression . . . 51
f_predict_plot_regression_alluvials . . . 52
f_predict_plot_regression_distribution . . . 53
f_predict_pl_regression . . . 54
f_predict_pl_regression_summarize . . . 55
f_predict_regression_add_predictions . . . 56
f_shiny_multiview . . . 57
f_shiny_som . . . 57
f_sim_profit . . . 58
f_stat_anova . . . 59
f_stat_chi_square . . . 59
f_stat_combine_anova_with_chi_square . . . 60
f_stat_diff_of_means_medians . . . 61
f_stat_group_ana . . . 62
f_stat_group_ana_taglist . . . 63
f_stat_group_counts_percentages . . . 64
f_stat_group_mean_medians . . . 65
f_stat_max_diff_of_freq . . . 65
f_stat_shapiro . . . 66
f_stat_stars . . . 66
f_train_lasso . . . 67
f_train_lasso_manual_cv . . . 68
make_container_for_function_calls . . . 69

Index **71**

f_boxcox	*f_boxcox*

Description

Takes a data_ls object generated by f_clean_data() and adds boxcox transformations of all numeric variables.

Usage

```
f_boxcox(data_ls)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)

Details

For a boxcox transformation all values mut be > 0. The function will automatically add the abs(min(x)) + 0.0001 to all columns if they contain values <= 0

Value

returns a list

`data`	the cleaned dataframe as tibble
`categoricals`	vector of column names containing categorical data
`numericals`	vector of column names containing numerical data
`ids`	vector of column names containing ids
`boxcox_names`	vector of column names containing boxcox transformed variables
`boxcox_data`	tibble containing boxcox transformed variables

See Also

`f_clean_data`

Examples

```
data_ls = f_clean_data(mtcars)
f_manip_get_most_common_level( data_ls$data$cyl)
```

`f_clean_data` *f_clean_data*

Description

Performs a number of cleaning operations on a dataframe, detects numerical and categorical columns and returns a list containing the cleaned dataframe and vectors naming the columns with a specific data type.

Usage

```
f_clean_data(data, max_number_of_levels_factors = 10,
  min_number_of_levels_nums = 6, exclude_missing = T,
  replace_neg_values_with_zero = T, allow_neg_values = c("null"),
  id_cols = c("null"))
```

Arguments

`data` a dataframe

`max_number_of_levels_factors`
If a factor variable contains more then the maximum number of levels the levels with the lowest frequency will be collapsed into 'others', Default: 10

`min_number_of_levels_nums`
If a numeric number contains less that the minimum of distinct values it will be converted to a factor, Default: 6

`exclude_missing`
exclude observations with missing values, Default: T

`replace_neg_values_with_zero`
all negative values will be set to 0, Default: T

`allow_neg_values`
specify columns for which negative values are allowed, Default: c("null")

`id_cols` specify columns containing ids.

Details

The list this function returns can be a bit tedious to work with. If you want to engineer a new feature you have to manually update the categoricals or the numericals vector. I suggest that you do all the feature engineering before applying this function. The advantage of this column is that when you get to the modelling or visualisation steps you have full control over which columns are used for the formula or for the type of visualisation even if you might have bloated your dataframe with some junk columns.

Value

returns a list

`data` the cleaned dataframe as tibble

`categoricals` vector of column names containing categorical data

`categoricals_ordered`
vector of column names containing all ordered categorical data

`numericals` vector of column names containing numerical data

`ids` vector of column names containing ids

See Also

f_boxcox

Examples

```
data_ls = f_clean_data( mtcars , id_cols = 'names')
str(data_ls)
```

f_clean_data_no_changes

wrapper for f_clean_data without modifications to data

Usage

```
f_clean_data_no_changes(data)
```

Arguments

data	a dataframe

Value

returns a list

data	the cleaned dataframe as tibble
categoricals	vector of column names containing categorical data
categoricals_ordered	vector of column names containing all ordered categorical data
numericals	vector of column names containing numerical data
ids	vector of column names containing ids

See Also

f_clean_data

f_content

opens an html index file for additional content

Description

opens a window in the default browser with links to all additional content files in oetteR, including vignettes and proof of concepts drafted as R and Rmd files. It includes the option of rendering all Rmd files into html. Which however can take around 30 minutes at least. You can find links to most of the additional content rendered as html on the github page of the package.

Usage

```
f_content(render_missing = F)
```

Arguments

`render_missing` logical, Default: T

Examples

```
## Not run:
 f_vignettes()

## End(Not run)
```

`f_content_get_file_type` *returns files of specified type from vignette path*

Usage

```
f_content_get_file_type(file_type = "html", folder = "vignettes")
```

Arguments

`file_type` character vector denoting file extensions

Examples

```
## Not run:
 f_content_get_file_type()

## End(Not run)
```

`f_content_get_path` *returns content path*

Usage

```
f_content_get_path(folder = "vignettes")
```

Examples

```
## Not run:
 f_content_get_path()

## End(Not run)
```

f_content_render *renders content*

Usage

```
f_content_render(overwrite = F, folder = "vignettes")
```

Arguments

overwrite	logical, Default: F

Examples

```
## Not run:
 f_content_render()

## End(Not run)
```

f_datatable_minimal *convert dataframe to DT:datatable with minimal features*

Usage

```
f_datatable_minimal(df, escape = F)
```

Arguments

df	PARAM_DESCRIPTION
escape	boolean, escape html content, Default:FALSE

Value

DT::datatable

See Also

datatable

Examples

```
f_datatable_minimal(mtcars)
```

`f_datatable_universal` *convert dataframe to DT:datatable inlcuding most usefull extensions and options*

Description

inludes the features for excel and clipboard export, hide and unhide (column visibility), reorder columns per drag and drop, navigate table with arrow keys and prefix/suffix directed rounding of numerical values.

- count_/ _count will round to 0
- p_val will round to Default: 2
- perc_/ _perc will round to Default: 1

Usage

```
f_datatable_universal(df, round_perc = 1, round_sign = 2,
  count_cols_as_int = T, round_other_nums = NULL, page_length = 10)
```

Arguments

`df`	dataframe
`round_perc`	digits for percentages, Default: 1
`round_sign`	digits for p_values, Default: 2
`count_cols_as_int`	detect count columns, Default: T
`round_other_nums`	round other numerical(double) columns to that digit. Will not do anything if NULL, Default: NULL
`page_length`	integer Default page length of table, Default: 10

Value

DT:datatable

See Also

`str_detect` `datatable`,`formatPercentage`,`formatSignif`,`formatRound`

Examples

```
data_ls = f_clean_data(mtcars)
f_stat_group_counts_percentages(data_ls, 'cyl') %>%
  f_datatable_universal()

f_stat_group_mean_medians(data_ls, 'cyl') %>%
  f_datatable_universal(round_other_nums = 2)
```

f_html_breaks *create a taglist with n lines of html line breaks*

Usage

```
f_html_breaks(n)
```

Arguments

n	number of line breaks

Value

taglist

See Also

br,tagList

Examples

```
f_html_breaks(5)
```

f_html_filename_2_link *convert a filename + path or a file_path to a html link*

Usage

```
f_html_filename_2_link(file_name = dir()[1], path = getwd(),
  file_path = NULL, link_text = file_name)
```

Arguments

file_name	character vector, Default: dir()[1]
path	character vector, Default: getwd()
file_path	file.path(path, file_name)
link_text	character vector

Value

link

See Also

str_replace_all

Examples

```
dir()[1]
f_html_filename_2_link()
dir()[1:5]
f_html_filename_2_link(dir()[1:5])

file_path = dir( getwd(), full.names=TRUE)[2]
file_path
f_html_filename_2_link(file_path = file_path)
```

f_html_get_title_from_Rmd *get title from Rmd file*

Description

usefull if we want to provide additional iformation if we wat to link to an html file that was generates from a Rmd file.

Usage

```
f_html_get_title_from_Rmd(file_path)
```

Arguments

`file_path`	path to Rmd file

Value

character vector

See Also

read_lines

f_html_padding *add some padding around html objects*

Usage

```
f_html_padding(obj, pad_before = 0, title = NULL, subtitle = NULL,
  caption = NULL, pad_after = 0, .f_htitle = htmltools::h3)
```

Arguments

obj — html object such as DT:datatable() or plotly::ggplotly()

pad_before — integer, Default: 0

title — character vector, Default: ”

subtitle — character vector, Default: ”

caption — character vector, Default: ”

pad_after — character vector, Default: 0

.f_htitle — function, Default: htmltools::h3

Value

taglist

See Also

tagList,h3,h4,h6

Examples

```
f_html_padding(DT::datatable(mtcars),5,'mtcars Data','subtitle', 'caption', 8 )
```

f_html_table_html_and_rmd_link

create a DT::datatable that pairs Rmd and rendered html files.

Description

Extracts the title from the Rmd file and generates one column with links to the Rmd and one column with links to the html files.

Usage

```
f_html_table_html_and_rmd_link(wd = getwd(), folder_Rmd = getwd(),
  folder_html = getwd())
```

Arguments

wd — working directory, Default: getwd()

folder_Rmd — path to folder with Rmd files relative to wd (use ’.’ for wd and ’..’ for parent directory of wd), Default: getwd()

folder_html — path to folder with html files relative to wd (use ’.’ for wd and ’..’ for parent directory of wd), Default: getwd()

Value

DT::datatable

Examples

```
wd = file.path( system.file(package = 'oetteR') )
folder_Rmd = 'vignettes'
folder_html = 'vignettes'

f_html_table_html_and_rmd_link( wd, folder_Rmd, folder_html )
```

`f_manip_append_2_list` *append object to list*

Description

convenience function to replace l[[length(l)+1]] = x

Usage

```
f_manip_append_2_list(l, x)
```

Arguments

`l`	list
`x`	object

Value

list

Examples

```
l = list('a', 'b')
l = f_manip_append_2_list(l, 'c')
str(l)
```

`f_manip_bin_numerics` *bin numerical columns*

Description

centers, scales and Yeo Johnson transforms numeric variables in a dataframe before binning into n bins of eqal range. Outliers based on boxplot stats are capped (set to min or max of boxplot stats).

Usage

```
f_manip_bin_numerics(df, bins = 5, bin_labels = c("LL", "ML", "M", "MH",
  "HH"), center = T, scale = T, transform = T)
```

Arguments

df	dataframe with numeric variables
bins	number of bins for numerical variables, Default: 5
bin_labels	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")#' @param center boolean, Default: T
center	boolean, Default: T
scale	boolean, Default: T
transform	boolean, Default: T

Value

dataframe

Examples

```
## Not run:
if(interactive()){
 #EXAMPLE1
 }

## End(Not run)
```

---

f_manip_bring_to_pos_range

bring vector to positice range

Description

if min < 0, add abs(min) to all values

Usage

```
f_manip_bring_to_pos_range(vec)
```

Arguments

vec	numeric vector

Value

vector

Examples

```
vec = c( -2,0,2,4,6)
vec = f_manip_bring_to_pos_range( vec )
vec
```

`f_manip_data_2_model_matrix_format`
brings data to model.matrix format

Description

model.matrix() creates dummy variables for factors. The names of these dummy variables however are not compatible with the formula syntax. This wrapper cleans up the names of the new variables.

Usage

```
f_manip_data_2_model_matrix_format(data, formula, scale_data = T,
  center_data = T, exclude_na_columns = T)
```

Arguments

`data`	a dataframe
`formula`	formula
`scale_data`	boolean
`center_data`	boolean
`exclude_na_columns`	boolean

Value

list with new dataframe and new formula

See Also

`str_replace_all`

Examples

```
data_ls = f_clean_data(mtcars)
data = data_ls$data
formula = hp ~ disp + am + gear
data_trans = f_manip_data_2_model_matrix_format( data, formula )
response_var =f_manip_get_response_variable_from_formula(data_trans$formula)
vars = f_manip_get_variables_from_formula(data_trans$formula)
x = as.matrix( select( data_trans$data, one_of(vars) ) )
y = data_trans$data[[response_var]]
glmnet::glmnet( x , y )
```

f_manip_double_2_int *converts columns of type double to integer if maximum number of decimal digits is zero*

Usage

```
f_manip_double_2_int(df)
```

Arguments

df	dataframe

Value

tibble

Examples

```
as_tibble(mtcars)
f_manip_double_2_int(mtcars)
```

f_manip_factor_2_numeric *converts factor to numeric preserving numeric levels and order in character levels*

Usage

```
f_manip_factor_2_numeric(vec)
```

Arguments

vec	vector

Value

vector

See Also

str_detect

Examples

```
fac_num = factor( c(1,3,8) )
fac_chr = factor( c('foo','bar') )
fac_chr_ordered = factor( c('a','b','c'), ordered = T )

f_manip_factor_2_numeric( fac_num )
f_manip_factor_2_numeric( fac_chr )
f_manip_factor_2_numeric( fac_chr_ordered )
```

f_manip_get_most_common_level

get most common level from vector

Usage

```
f_manip_get_most_common_level(x)
```

Arguments

x	factor vector

Value

character vector

See Also

tidy

Examples

```
data_ls = f_clean_data(mtcars)
f_manip_get_most_common_level( data_ls$data$cyl)
```

f_manip_get_response_variable_from_formula

get response variable from formula

Usage

```
f_manip_get_response_variable_from_formula(formula)
```

Arguments

formula	formula

Value

character vector

See Also

f_manip_get_variables_from_formula

f_manip_get_variables_from_formula

get variables from formula

Usage

```
f_manip_get_variables_from_formula(formula)
```

Arguments

`formula`	formula

Value

character vector

See Also

`f_manip_get_response_variable_from_formula`

Examples

```
f = foo~bar1 + bar2

vars = f_manip_get_variables_from_formula(f)
response_var = f_manip_get_response_variable_from_formula(f)
```

f_manip_matrix_2_tibble

converts matrices to tibble, preserving row.names

Description

row.names are added as row_names column as the first column of the tibble. Function does not fail when object cannot be converted to tibble thus can be used to map over lists with various variable types such as modells and objects.

Usage

```
f_manip_matrix_2_tibble(x)
```

Arguments

`x`	any variable

Value

a tibble or if the input variable is neither matrix dataframe or tibble the original input object.

Examples

```
mat = as.matrix(mtcars)
head( mat, 10)
f_manip_matrix_2_tibble( mat )

# convert all matrices from a list
pca = prcomp( mtcars ) %>%
 map( f_manip_matrix_2_tibble )
pca
```

---

f_manip_summarize_2_median_and_most_common_factor

takes a data_ls list created by f_clean_data() and returns a list with all medians for numerical and most common level for categorical variables.

Usage

```
f_manip_summarize_2_median_and_most_common_factor(data_ls)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)

Value

list

data summarized data as dataframe

data_boxcox summarized boxcox data as dataframe

Examples

```
summarized_ls = f_clean_data(mtcars) %>%
  f_boxcox() %>%
  f_manip_summarize_2_median_and_most_common_factor()

summarized_ls$data
summarized_ls$boxcox_data
```

f_manip_transpose_tibble

transpose a tibble

Description

transpose a tibble, values in first column will become column titles. Row names will be converted to first columns

Usage

```
f_manip_transpose_tibble(tib)
```

Arguments

tib	tibble

Value

tibble

Examples

```
tib = mtcars %>%
  as_tibble() %>%
  f_manip_transpose_tibble()
tib
```

f_model_add_predictions_2_grid_regression

add predictions to grid (regression models)

Description

wrapper for modelr::add_predictions

Usage

```
f_model_add_predictions_2_grid_regression(grid, m, var)
```

Arguments

grid	grid containing all variables used for the model
m	model
var	character vector denoting response variable

Value

grid

See Also

`add_predictions`

Examples

```
data_ls = f_clean_data(mtcars)
formula = disp~hp+mpg
m = lm(formula, data_ls$data)
grid = f_model_data_grid(data_ls, formula, 'hp', 10) %>%
  f_model_add_predictions_2_grid_regression( m, 'disp')
```

`f_model_data_grid` *generates a data grid based on a formula*

Description

the range of one specified variable is expanded, while all other variables are set to the most common values. Similar to modelr::data_grid but it can deal with factors.

Usage

```
f_model_data_grid(col_var, data_ls, formula, n = 500, set_manual = list())
```

Arguments

`col_var`	character vector, denoting variable that should be expanded
`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`formula`	formula
`n`	integer, length of grid, datapoints in between range of col_var
`set_manual`	named list, set some variables manually instead of defaulting to median or most common factor. !! Values need to be of the same variable type as in the original data.

Value

dataframe

Examples

```
data_ls = f_clean_data(mtcars)
formula = disp~cyl+mpg+hp
f_model_data_grid( 'mpg', data_ls, formula,  10 )
f_model_data_grid( 'mpg', data_ls, formula,  10 , set_manual = list( cyl = min(data_ls$data$cyl) ) )
```

f_model_importance *model importance*

Description

supports rpart, randomForest, svm, will return NULL for other models

Usage

```
f_model_importance(m, data)
```

Arguments

m	model
data	training data

Value

tibble

Examples

```
pl = pipelearner::pipelearner(mtcars) %>%
  pipelearner::learn_models( twidlr::rpart, disp~. ) %>%
  pipelearner::learn_models( twidlr::randomForest, disp~. ) %>%
  pipelearner::learn_models( twidlr::svm, disp~. ) %>%
  pipelearner::learn() %>%
  mutate( imp = map2(fit, train, f_model_importance) )
pl$imp
```

f_model_importance_plot *plot model importance*

Description

optimised for usage in pipelearner dataframe

Usage

```
f_model_importance_plot(importance, title, variable_color_code = NULL)
```

Arguments

importance	dataframe importance created by f_model_importance()
title	character vector (model column in pipelearner dataframe) will be pasted for plot title
variable_color_code	dataframe created by f_plot_color_code_variables()
...	additional character vectors to be pasted to plot title

Value

plotly graph

Examples

```
data_ls = f_clean_data(mtcars)
variable_color_code = f_plot_color_code_variables(data_ls)
m = twidlr::rpart(mtcars, disp~.)
imp = f_model_importance_rpart(m)
f_model_importance_plot(imp
                        , title = 'rpart'
                        , variable_color_code = variable_color_code
                        )
#pipelearner
pl = pipelearner::pipelearner(data_ls$data) %>%
  pipelearner::learn_models( twidlr::rpart, disp~. ) %>%
  pipelearner::learn_models( twidlr::randomForest, disp~. ) %>%
  pipelearner::learn_models( twidlr::svm, disp~. ) %>%
  pipelearner::learn() %>%
  mutate( imp = map2(fit, train, f_model_importance)
         , title = paste( model, models.id, cv_pairs.id, train_p )
         , plot = map2( imp
                       , title
                       , f_model_importance_plot
                       , variable_color_code = variable_color_code
                       )
         )
htmltools::tagList(pl$plot)
```

f_model_importance_plot_tableplot

tableplot of important variables

Description

takes the most important variables of a model and plots a tabplot::tableplot

Usage

```
f_model_importance_plot_tableplot(data, ranked_variables, response_var,
  limit = 10, print = F, ...)
```

Arguments

data	dataframe
ranked_variables	datafram as returned by f_model_importance()
response_var	character vector denoting response variable
limit	integer limit the number of variables , Default: 10
print	boolean, print tabplot when generating tabplot object
...	pass kwargs to tabplot::tableplot

Value

tabplot::tableplot object

See Also

tableplot

Examples

```
data = f_clean_data(mtcars) %>%
  .$data
m = rpart::rpart( disp~., data)
ranked_variables  = f_model_importance(m, data)
response_var = 'disp'

f_model_importance_plot_tableplot( data, ranked_variables, response_var, limit = 5 )

#pipe
form = as.formula('disp~cyl+mpg+hp')
pl = pipelearner::pipelearner(mtcars) %>%
 pipelearner::learn_models( twidlr::rpart, form ) %>%
 pipelearner::learn_models( twidlr::randomForest, form ) %>%
 pipelearner::learn_models( twidlr::svm, form ) %>%
 pipelearner::learn() %>%
 mutate( imp = map2(fit, train, f_model_importance)
         , tabplot = pmap( list( data = train
                                 , ranked_variables = imp
                                 , response_var = target
                                 , title = model
         )
         , f_model_importance_plot_tableplot
         , limit = 5
         )
  )
```

f_model_importance_pl_add_plots_regression

add plots based on variable importance to pipelearner dataframe

Description

adds a bar plot of the ranked variables, a tabplot sorted by the target variable and a dependency plot (response variable vs the sequential range of one of the predictor variables while all other predictors are kept constant at mean values).

Usage

```
f_model_importance_pl_add_plots_regression(pl, data, m, ranked_variables,
  response_var, title,
  variable_color_code = f_plot_color_code_variables(data_ls), formula,
  data_ls, var_dep_limit = 10, var_dep_log_y = F, tabplot_limit = 12,
  formula_in_pl = F)
```

Arguments

`pl` a dataframe containing the columns for data, m, ranked_variables, response_var and title

`data` symbol (unquoted name) of data column in pl

`m` symbol (unquoted name) of data column in pl

`ranked_variables` symbol (unquoted name) of data column in pl

`response_var` symbol (unquoted name) of data column in pl

`title` symbol (unquoted name) of data column in pl

`variable_color_code` dataframe created by f_plot_color_code_variables()

`formula` fomula that was used to construct model

`data_ls` data_ls list object containing the whole of the original data

`var_dep_limit` number of variables to be plotted on dependency plot

`var_dep_log_y` should y axis of dependency plot be logarithmic

`tabplot_limit` number of variables to be plotted on tabplot

`formula_in_pl` boolean if formula is a column in pl?

Value

dataframe

See Also

`f_model_importance_plot` `f_model_importance_plot_tableplot` `f_model_plot_variable_dependency_regres`

Examples

```
data_ls = f_clean_data(mtcars)
form = disp~cyl+mpg+hp
variable_color_code = f_plot_color_code_variables(data_ls)

pl = pipelearner::pipelearner(data_ls$data) %>%
  pipelearner::learn_models( twidlr::rpart, form ) %>%
  pipelearner::learn_models( twidlr::randomForest, form ) %>%
  pipelearner::learn_models( twidlr::svm, form ) %>%
  pipelearner::learn() %>%
  mutate( imp = map2(fit, train, f_model_importance)
          , title = paste(model, models.id, train_p) ) %>%
  f_model_importance_pl_add_plots_regression(  data                 = train
                                             , m                    = fit
                                             , ranked_variables     = imp
                                             , title                = title
                                             , response_var         = target
                                            , variable_color_code = variable_color_code
                                             , formula              = form
                                             , data_ls              = data_ls
                                             , var_dep_limit        = 10
                                             , var_dep_log_y        = T
                                             , tabplot_limit        = 12 )
```

f_model_importance_pl_plots_as_html

print plots of variable importance in modelling dataframe to html

Description

should execute f_model_importance_pl_add_plots_regression() on modelling dataframe first

Usage

```
f_model_importance_pl_plots_as_html(pl, prefix = NULL, quiet = FALSE)
```

Arguments

pl	modelling dataframe containing the following columns 'imp_plot', 'imp_plot_dep', 'imp_tabplot', 'title'
prefix	character vector file name prefix for html files, Default: NULL
quiet	boolean, suppresses output to console by render function, Default: FALSE

Value

html files in working directory

See Also

tagList

Examples

```
## Not run:
    data_ls = f_clean_data(mtcars)
form = disp~cyl+mpg+hp
variable_color_code = f_plot_color_code_variables(data_ls)

pl = pipelearner::pipelearner(data_ls$data) %>%
 pipelearner::learn_models( twidlr::rpart, form ) %>%
 pipelearner::learn_models( twidlr::randomForest, form ) %>%
 pipelearner::learn_models( twidlr::svm, form ) %>%
 pipelearner::learn() %>%
 mutate( imp = map2(fit, train, f_model_importance)
        , title = paste(model, models.id, train_p) ) %>%
 f_model_importance_pl_add_plots_regression(  data                = train
                                             , m                   = fit
                                             , ranked_variables    = imp
                                             , title               = title
                                             , response_var        = target
                                            , variable_color_code = variable_color_code
                                             , formula             = form
                                             , data_ls             = data_ls
                                             , var_dep_limit       = 10
                                             , var_dep_log_y       = T
                                             , tabplot_limit       = 12) %>%
 f_model_importance_pl_plots_as_html( prefix = 'test_oetteR_html_')
```

```
files = dir() %>%
 .[ startsWith(., 'test_oetteR_html_') ]

file.remove( files )



## End(Not run)
```

f_model_importance_randomForest

extract variable importance for randomForest model

Usage

```
f_model_importance_randomForest(m)
```

Arguments

m	model of class randomForest

Value

dataframe

Examples

```
#regression
m = twidlr::randomForest(mtcars, disp~.)
f_model_importance_randomForest(m)

#classification
data_ls = f_clean_data(mtcars)
m = twidlr::randomForest(data_ls$data, cyl~.)
f_model_importance_randomForest(m)
```

f_model_importance_rpart

extract variable importance for rpart

Usage

```
f_model_importance_rpart(m)
```

Arguments

m	model of class rpart

Value

dataframe

Examples

```
#regression
m = twidlr::rpart(mtcars, disp~.)
f_model_importance_rpart(m)

#classification
data_ls = f_clean_data(mtcars)
m = twidlr::rpart(data_ls$data, cyl~.)
f_model_importance_rpart(m)
```

---

f_model_importance_svm

extract variable importance for svm

Usage

```
f_model_importance_svm(m, data)
```

Arguments

m	model of class svm
data	original training dataframe

Details

uses 1D-SA 1 dimensional sensitivity analysis using rminer::Importance()

Value

dataframe

Examples

```
#regression
m = twidlr::svm(mtcars, disp~.)
f_model_importance_svm(m, mtcars)

#classification
data = mtcars
data$cyl = factor(data$cyl, ordered = T)
m = twidlr::svm(data, cyl~.)
f_model_importance_svm(m, data)
```

f_model_plot_variable_dependency_regression
plot model dependency on most important variables

Description

response variable will be plotted against the entire range of each variable staring with the most important ones. All other variables will be set to median or most common factor. This function requires a ranked list of the most important variables as returned by f_model_importance()

Usage

```
f_model_plot_variable_dependency_regression(m, ranked_variables,
  title = unlist(stringr::str_split(class(m)[1], "\\."))[1], data = NULL,
  formula, data_ls,
  variable_color_code = f_plot_color_code_variables(data_ls), limit = 12,
  log_y = F, set_manual = list(), ...)
```

Arguments

`m`	a regression model
`ranked_variables`	datafram as returned by f_model_importance()
`title`	character vector as plot title, Default: unlist(stringr::str_split(class(m)[1], "\."))[1]
`data`	a dataframe, only necessary if it differs from data_ls$data, Default: NULL
`formula`	the formula used to train the model
`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`variable_color_code`	dataframe created by f_plot_color_code_variables()
`limit`	integer limit the number of variables to be plotted, Default: 12
`log_y`	boolean log_scale for y axis
`set_manual`	named list, set some variables manually instead of defaulting to median or most common factor. !! Values need to be of the same variable type as in the original data.
`...`	arguments passed to facet_wrap e.g. usefull for nrow, ncol

Value

plot

See Also

str_split

Examples

```
# regular version--------------------------------------
data_ls             = f_clean_data(mtcars)
data                = data_ls$data
formula             = disp~hp+mpg+cyl
m                   = randomForest::randomForest(formula, data)
ranked_variables    = f_model_importance( m, data)
variable_color_code = f_plot_color_code_variables(data_ls)
limit               = 12
f_model_plot_variable_dependency_regression( m
                                             , ranked_variables
                              , title = unlist( stringr::str_split( class(m)[1], '\\.') )[1]
                                             , formula = formula
                                             , data_ls = data_ls
                                             , variable_color_code = variable_color_code
                                             , limit = limit
                                             )

#pipe version ------------------------------------------

data_ls = f_clean_data(mtcars)
form = as.formula('disp~hp+cyl+wt')
variable_color_code = f_plot_color_code_variables(data_ls)
limit            = 10

 pl = pipelearner::pipelearner( data_ls$data ) %>%
  pipelearner::learn_models( rpart::rpart, form ) %>%
  pipelearner::learn_models( randomForest::randomForest, form ) %>%
  pipelearner::learn_models( e1071::svm, form ) %>%
  pipelearner::learn() %>%
  mutate( imp   = map2(fit, train, f_model_importance)
         ,plot = pmap( list( m = fit, ranked_variables = imp, title = model, data = train)
                       , .f = f_model_plot_variable_dependency_regression
                       , formula = form
                       , data_ls = data_ls
                       , variable_color_code = variable_color_code
                      , limit = limit
         )
   )
```

---

f_model_plot_var_dep_over_spec_var_range

plot vmodel varaible dependency over the range of a specified variable

---

Description

Some models are able to capture relative dependencies. In order to visualise them the dataset is split into three parts. 0-25,25-75,75-100 percentile or the three most common factors.Then variable dependencies for each of the three splits are plotted. In the mtcars example below we can see that the model predicts an increase in disp if drat increases for cars with 8 cylinders, while the opposite is true for cars with only 6 cylinders.

Usage

```
f_model_plot_var_dep_over_spec_var_range(m, title, variables, range_variable,
  data, formula, data_ls, variable_color_code, log_y = F, limit = 12)
```

Arguments

`m` a model

`title` model title

`variables` character vector with variable names, or ranked variables as returned by f_model_importance()

`range_variable` character vector denoting range variable

`data` dataset

`formula` formula

`data_ls` data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>) - The data_ls object provides the entire dataset

`variable_color_code` dataframe created by f_plot_color_code_variables()

`log_y` boolean log_scale for y axis

`limit` integer limit the number of variables to be plotted, Default: 12

Value

grid can be printed with gridExtra::grid.arrange()

See Also

`arrangeGrob`

Examples

```
## Not run:

 # single output example ---------------------------------------
.f                  = randomForest::randomForest
data_ls             = f_clean_data(mtcars)
data                = data_ls$data
formula             = disp~mpg+cyl+am+hp+drat+qsec+vs+gear+carb
m                   = .f(formula, data)
variables           = f_model_importance( m, data)
title               = unlist( stringr::str_split( class(m)[1], '\\.') )[1]
variable_color_code = f_plot_color_code_variables(data_ls)
limit               = 10
log_y               = F

range_variable_num  = data_ls$numericals[1]
range_variable_cat  = data_ls$categoricals[1]

grid_num = f_model_plot_var_dep_over_spec_var_range(m
                                                    , title
                                                    , variables
                                                    , range_variable_num
                                                    , data
                                                    , formula
                                                    , data_ls
                                                    , variable_color_code
                                                    , log_y
                                                    , limit  )
gridExtra::grid.arrange(grid_num)

# pipe example ------------------------------------------------

data_ls = f_clean_data(mtcars)
form = as.formula('disp~cyl+mpg+hp+am+gear+drat+wt+vs+carb')
variable_color_code = f_plot_color_code_variables(data_ls)

grids = pipelearner::pipelearner(data_ls$data) %>%
  pipelearner::learn_models( twidlr::rpart, form ) %>%
  pipelearner::learn_models( twidlr::randomForest, form ) %>%
  pipelearner::learn_models( twidlr::svm, form ) %>%
  pipelearner::learn() %>%
  dplyr::mutate( imp = map2(fit, train, f_model_importance)
                , range_var = map_chr(imp, function(x) head(x,1)$row_names )
                , grid = pmap( list( m = fit
                                     , title = model
                                     , variables = imp
                                     , range_variable = range_var
                                     , data = test
                )
                , f_model_plot_var_dep_over_spec_var_range
                , formula = form
                , data_ls = data_ls
                , variable_color_code = variable_color_code
                , log_y = F
                , limit = 12
                )
  )  %>%
  .$grid

f_plot_obj_2_html( grids,  type = "grids", output_file =  'test_me', title = 'Grids', height = 30 )

file.remove('test_me.html')

## End(Not run)
```

f_model_seq_range *generates sequence of variable spanning from min to max*

Description

similar to modelr::seq_range but can handle categorical variables

Usage

```
f_model_seq_range(data_ls, col_var, n = 500)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_var`	character vector denoting variable
`n`	integer number of intermediate data points, Default: 500

Value

vector

Examples

```
data_ls = f_clean_data(mtcars)
col_var = 'disp'
f_model_seq_range( data_ls, col_var, 10)
```

`f_pca` *calculate principle components for a dataset*

Description

This function is an extended wrapper for prcomp(). I takes a data_ls object created by f_clean_data and calculates the contribution of each variable to each principle component in percent.

Usage

```
f_pca(data_ls, center = T, scale = T, use_boxcox_tansformed_vars = T,
  include_ordered_categoricals = T)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`center`	boolean, Default: T
`scale`	boolean, Default: T
`use_boxcox_tansformed_vars`	boolean, Default: T
`include_ordered_categoricals`	boolean, Default: T

Details

Blog post explaining how to calculate contributions

Value

a list with the original data complemented with the principle component vector data of each observation and an object returned by prcomp() supplemented with some extra features

`data`	dataframe
`pca`	pca object created by prcomp()

added features of pca:

`cos2`	The squared rotation vectors.A value between 0 and 1 denotes the amount of contribution of a variable to a specific principle component
`vae`	percent variance explained
`contrib_abs_perc`	The absolute contribution of one variable to the variance explained by one principle component in percent. The total contibution adds up to the total contibution of the principle componaent in percent.
`contrib_abs_perc_reduced`	as above but variables contibuting less than 2.5 percent are grouped
`threshold_vae_for_pc_perc`	principle components that explain less percent variance than this threshold are dropped

See Also

prcomp

Examples

```
pca_ls = f_clean_data(mtcars) %>%
f_boxcox() %>%
f_pca()
```

f_pca_plot_components *plot principle components as a dot plot*

Usage

```
f_pca_plot_components(pca_ls, x_axis = "PC1", y_axis = "PC2",
  group = NULL)
```

Arguments

`pca_ls`	list created by f_pca()
`x_axis`	character vector, Default: 'PC1'
`y_axis`	character vector, Default: 'PC2'
`group`	character vector denoting the grouping variable, determines dot colour, Default: NULL

Value

htmltools taglist containing a plotly graph and tow DT datatables will only show if printed in a rmarkdown document

Examples

```
## Not run:
tagls = f_clean_data(mtcars) %>%
  f_boxcox() %>%
  f_pca() %>%
  f_pca_plot_components(group = 'cyl')

## End(Not run)
```

---

f_pca_plot_variance_explained

plot varaince explained of principle components

Usage

```
f_pca_plot_variance_explained(pca_ls, threshold_vae_for_pc_perc = 2.5)
```

Arguments

pca_ls	list created by f_pca()
threshold_vae_for_pc_perc	double, filter principle components that explain a lesser percentage of the variance than this threshold, Default: 2.5

Value

plotly graph

Examples

```
p = f_clean_data(mtcars) %>%
  f_boxcox() %>%
  f_pca() %>%
  f_pca_plot_variance_explained()
p
```

f_plot_adjust_col_vector_length *adjust length of color vector, by repeating colors*

Usage

```
f_plot_adjust_col_vector_length(n = 74, col_vector = f_plot_col_vector74())
```

Arguments

n	length, Default: 74
col_vector	vector containing colors, Default: f_plot_col_vector74()

Value

vector containing colors of specified length

Examples

```
length( f_plot_adjust_col_vector_length(100) )
```

f_plot_alluvial *plot alluvial on tidy data*

Description

plots a dataframe as an alluvial plot. All numerical variables are scaled, centered and YeoJohnson transformed before binning.

Usage

```
f_plot_alluvial(data, variables = names(data), col_id = NULL,
  max_variables = 20, bins = 5, bin_labels = c("LL", "ML", "M", "MH",
  "HH"), NA_label = "NA", order_levels = NULL, fill_by = "first_variable",
  col_vector_flow = f_plot_col_vector74(faint = F, greys = F),
  col_vector_value = RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7, 5)])
```

Arguments

data	a dataframe
variables	vector denoting names and order of the plotted variables, Default: names(data)
col_id	character vector denoting id column
max_variables	maximum number of variables, Default: 20
bins	number of bins for numerical variables, Default: 5
bin_labels	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")
NA_label	character vector define label for missing data

`order_levels`	character vector denoting levels to be reorderer from low to high
`fill_by`	one_of(c('first_variable', 'last_variable', 'all_flows', 'values')), Default: 'first_variable'
`col_vector_flow`	HEX colors for flows, Default: f_plot_col_vector74(faint = F, greys = F)
`col_vector_value`	Hex colors for y levels/values, Default: RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7, 5)]

Details

DETAILS

Value

OUTPUT_DESCRIPTION

See Also

`brewer.pal` `fct_relevel` `geom_flow`,`geom_stratum`

Examples

```
## Not run:
if(interactive()){

data_ls = mtcars %>%
  f_clean_data()

data = data_ls$data
max_variables = 5
variables = c( data_ls$categoricals[1:3], data_ls$numericals[1:3] )

f_plot_alluvial( data = data
                , variables = variables
                , max_variables = max_variables
                , fill_by = 'first_variable' )

f_plot_alluvial( data = data
                , variables = variables
                , max_variables = max_variables
                , fill_by = 'last_variable' )

f_plot_alluvial( data = data
                , variables = variables
                , max_variables = max_variables
                , fill_by = 'all_flows' )

f_plot_alluvial( data = data
                , variables = variables
                , max_variables = max_variables
                , fill_by = 'first_variable' )

# manually order variable values

f_plot_alluvial( data = data
                 , variables = variables
                 , max_variables = max_variables
                 , fill_by = 'values'
                 , order_levels = c('1', '0') )
}

## End(Not run)
```

f_plot_alluvial_1v1 *plot alluvial of gathered data*

Description

Plots two variables of a dataframe on an alluvial plot. A third variable can be added either two the left or the right of the alluvial plot to provide coloring of the flows. All numerical variables are scaled, centered and YeoJohnson transformed before binning.

Usage

```
f_plot_alluvial_1v1(data, col_x, col_y, col_id, col_fill = NULL,
  fill_right = T, bins = 5, bin_labels = c("LL", "ML", "M", "MH", "HH"),
  NA_label = "NA", order_levels_y = NULL, order_levels_x = NULL,
  order_levels_fill = NULL, complete = TRUE, fill_by = "first_variable",
  col_vector_flow = f_plot_col_vector74(faint = F, greys = F),
  col_vector_value = RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7, 5)])
```

Arguments

`data` a dataframe

`col_x` character vector denoting column for the x axis variable

`col_y` character vector denoting column for the y axis variable

`col_id` character vector denoting id column

`col_fill` character vector denoting color fill variable for flows, Default: NULL

`fill_right` logical, TRUE fill variable is added to the right FALSE to the left, Default: T

`bins` number of bins for automatic binning of numerical variables, Default: 5

`bin_labels` labesl for bins, Default: c("LL", "ML", "M", "MH", "HH")

`NA_label` character vector define label for missing data

`order_levels_y` character vector denoting order of y levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL

`order_levels_x` character vector denoting order of x levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL

`order_levels_fill`
character vector denoting order of color fill variable levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL

`complete` boolean, insert implicitly missing observations, Default: TRUE

`fill_by` one_of(c('first_variable', 'last_variable', 'all_flows', 'values')), Default: 'first_variable'

`col_vector_flow`
: HEX colors for flows, Default: f_plot_col_vector74(faint = F, greys = F)

`col_vector_value`
: Hex colors for y levels/values, Default: RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7, 5)]

Value

plot

See Also

`brewer.pal` `fct_relevel`,`fct_rev` `UQ` `geom_flow`,`geom_stratum`

Examples

```
## Not run:
if(interactive()){
# sample data
 monthly_flights = nycflights13::flights %>%
 group_by(month, tailnum, origin, dest, carrier) %>%
 summarise() %>%
 group_by( tailnum, origin, dest, carrier) %>%
 count() %>%
 filter( n == 12 ) %>%
 select( - n ) %>%
 left_join( nycflights13::flights ) %>%
 .[complete.cases(.), ] %>%
 ungroup() %>%
 mutate( tailnum = pmap_chr(list(tailnum, origin, dest, carrier), paste )
         , qu = cut(month, 4)) %>%
 group_by(tailnum, carrier, origin, dest, qu ) %>%
 summarise( mean_arr_delay = mean(arr_delay) ) %>%
 ungroup() %>%
 mutate( mean_arr_delay = ifelse( mean_arr_delay < 10, 'on_time', 'late' ) )

levels(monthly_flights$qu) = c('Q1', 'Q2', 'Q3', 'Q4')

data = monthly_flights

col_x = 'qu'
col_y = 'mean_arr_delay'
col_fill = 'carrier'
col_id = 'tailnum'

# flow coloring variants
f_plot_alluvial_1v1( data, col_x, col_y, col_id, col_fill )
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'last_variable' )
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'first_variable' )
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'all_flows' )
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'value' )

# use same color coding for flows and y levels
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'last_variable'
                    , col_vector_flow = f_plot_col_vector74()
                    , col_vector_value = f_plot_col_vector74() )
# move fill variable to the left
f_plot_alluvial_1v1( data, col_x, col_y, col_id, col_fill, fill_right = F )

# reorder levels
f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'first_variable'
                    , order_levels_y = c('on_time', 'late') )

f_plot_alluvial_1v1( data, col_x, col_y, col_id, fill_by = 'first_variable'
                    , order_levels_x = c('Q4', 'Q3', 'Q2', 'Q1') )

order_by_carrier_size = data %>%
 group_by(carrier) %>%
 count() %>%
 arrange( desc(n) ) %>%
 .[['carrier']]

f_plot_alluvial_1v1( data, col_x, col_y, col_id, col_fill
                    , order_levels_fill = order_by_carrier_size )

}

## End(Not run)
```

f_plot_color_code_variables

color code all variables in a data_ls list.

Description

color coding is stable the same data_ls list gets the same coding with every function call. Assigns the same colors to the boxcox transformed and untransformed variant of a variable.

Usage

```
f_plot_color_code_variables(data_ls, col_vector = f_plot_col_vector74())
```

Arguments

data_ls	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
col_vector	character vector denoting Hexcode colors, Default: f_plot_col_vector74()

Value

tibble

- variable
- colorHEX code color

See Also

str_replace_all

Examples

```
f_clean_data(mtcars) %>%
  f_boxcox() %>%
  f_plot_color_code_variables() %>%
  print()
```

`f_plot_col_vector74` *generate a most distinctive color scale*

Description

based on RColorBrewer colours of length 74 for RGB colors see rapidtables(https://www.rapidtables.com/web/color/inde
Basically strings a couple of RColorBrewer palettes together.

Usage

```
f_plot_col_vector74(greys = T, reds = T, blues = T, greens = T,
  faint = T, only_unique = F)
```

Arguments

`greys`	boolean, include grey colors, Default: TRUE
`reds`	boolean, include red colors, Default: TRUE
`blues`	boolean, include blue colors, Default: TRUE
`greens`	boolean, include green colors, Default: TRUE
`faint`	boolean, include faint colors, Default: TRUE
`only_unique`	boolean, do not allow color repetitions, Default: FALSE

Value

vector with HEX colours

`f_plot_generate_comparison_pairs`
generates comparison pairs for ‘ggpubr::stat_compare_means()‘

Description

generates all possible pairs and filters according to wilcox p_value

Usage

```
f_plot_generate_comparison_pairs(data, col_var, col_group, thresh = 0.05)
```

Arguments

`data`	dataframe
`col_var`	character vector denoting variable column
`col_group`	character vector denoting grouping column
`thresh`	double, Default: 0.05

Value

list

See Also

`str_split` `UQ`

Examples

```
f_plot_generate_comparison_pairs( mtcars, 'disp', 'cyl' )
```

`f_plot_hist` *Plot Histograms*

Description

Function plots smart histograms for variables in a data_ls list generated by f_clean_data(). It supports three types of histograms: Bar histograms, density histograms and violin plots. We can further specify a categorical variable to group on. The function defaults to a sensible standard output if key word arguments are not applicable for variable type. Thus we can easily pipe through long lists of variables and thus generate histograms for all variables in the input (see examples).

Usage

```
f_plot_hist(variable, data_ls, group = "None", graph_type = "violin",
  y_axis = "count", auto_range = T, n_breaks = 30, rug = T, x_min = 0,
  x_max = 100, y_max = 100, title = "",
  col_vector = f_plot_adjust_col_vector_length(100,
  RColorBrewer::brewer.pal(name = "Dark2", n = 8)), p_val = T, add = "mean",
  ...)
```

Arguments

`variable`	character vector naming the variable to be plotted
`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`group`	character vector naming the column to be used as grouping variable, Default: NULL
`graph_type`	one of c("violin", "bar", "line"), Default: 'violin'
`y_axis`	one of c("count", "density"), Default: c("count", "density")

`auto_range`	boolean, Default: T
`n_breaks`	integer , Default: 30
`rug`	boolean
`x_min`	double, requires aut_range == F, Default: 0
`x_max`	double, requires aut_range == F, Default: 100
`y_max`	double, requires aut_range == F, Default: 100
`title`	character vector plot title
`col_vector`	vector with RGB colors, Default: f_plot_adjust_col_vector_length(100, RColorBrewer::brewer.pal(name = "Dark2", n = 8))
`p_val`	boolean, Default: T
`add`	character vector one_of(c('mean','median','none')) , Default: 'mean'
`...`	additional arguments passed to labs()

Value

plot object

Examples

```
## Not run:
#'
#plot single variable
data_ls = f_clean_data(mtcars)
f_plot_hist('disp', data_ls)
f_plot_hist('disp', data_ls, add = 'median')
f_plot_hist('disp', data_ls, add = 'none')
f_plot_hist('disp', data_ls, y_axis = 'density')
f_plot_hist('cyl', data_ls , group = 'gear' )
f_plot_hist('cyl', data_ls , group = 'gear', y_axis = 'density' )
f_plot_hist('cyl', data_ls, y_axis = 'density' )
f_plot_hist('cyl', data_ls, y_axis = 'count' )
f_plot_hist('disp', data_ls, graph_type = 'line', group = 'cyl')
f_plot_hist('disp', data_ls, graph_type = 'bar', group = 'cyl')
f_plot_hist('disp', data_ls, graph_type = 'violin', group = 'cyl'
             , caption ='caption', title = 'title', subtitle = 'subtitle')

#plot all variables
vars = data_ls$all_variables[ data_ls$all_variables != 'cyl' ] %>%
 map( f_plot_hist, data_ls, group = 'cyl')
vars

## End(Not run)
```

`f_plot_obj_2_html` *generate a separate html file from a list of various objects*

Description

lists of graphical objects like html(taglists), plots, tabplots, grids can be converted to html files

Usage

```
f_plot_obj_2_html(obj_list, type, output_file, title = "Plots",
  quiet = FALSE, ...)
```

Arguments

`obj_list` htmltools::tagList

`type` one of c('taglist','plots','tabplots','grids' , 'model_performance') some templates take additional arguments via the ... argument

taglist taglist ceated with htmltools::tagList, a good container for html widgets

plots a list with ggplot objects, takes additional arguments: *fig.height: Default 5, fig.width: Default 7*

tabplots a list of objects created with tabplot::tableplot, takes additional arguments: *fig.height: Default 5, fig.width: Default 7, titles: list of titles must be same length as obj_list*

grids a list of grids created with gridExtra::arrangeGrob, takes additional argument: *height: Default 30*

model_performance takes a taglist created with f_predict_plot_model_performance_regression, takes the additional arguments: *alluvial, plot objec created with f_predict_plot_regression_alluv dist, list of two plots created with f_predict_plot_regression_distribution, render_points_as_png: Default: TRUE, takes screenshots of point plots, otherwise plotly will load all points into memory which is not compatible with large data sets*

`output_file` file_name of the html file, without .html suffix

`title` character vector of html document title, Default: 'Plots'

`quiet` bollean, suppress markdown console print output, Default: FALSE

`...` additional arguments passed to rmarkdown::render argument params

Examples

```
# type = taglist---------------------------------------------------------------

taglist = f_clean_data(mtcars) %>%
  f_boxcox() %>%
  f_pca() %>%
  f_pca_plot_components()

f_plot_obj_2_html(taglist, type = "taglist", output_file =  'test_me', title = 'Plots')

file.remove('test_me.html')

#type = tabplot-----------------------------------------------------------------

form = as.formula('disp~cyl+mpg+hp')
pipelearner::pipelearner(mtcars) %>%
  pipelearner::learn_models( rpart::rpart, form ) %>%
  pipelearner::learn_models( randomForest::randomForest, form ) %>%
  pipelearner::learn_models( e1071::svm, form ) %>%
  pipelearner::learn() %>%
  dplyr::mutate( imp = map2(fit, train, f_model_importance)
                , tabplot = pmap( list( data = train
                                        , ranked_variables = imp
                                        , response_var = target
                                        , title = model
                 )
                 , f_model_importance_plot_tableplot
                 , limit = 5
                 )
  )  %>%
  .$tabplot %>%
  f_plot_obj_2_html( type = "tabplots", output_file =  'test_me', title = 'Plots')

file.remove('test_me.html')

#type = plots ---------------------------------------------------------------------

data_ls = f_clean_data(mtcars)
form = as.formula('disp~cyl+mpg+hp')
variable_color_code = f_plot_color_code_variables(data_ls)

pipelearner::pipelearner(data_ls$data) %>%
 pipelearner::learn_models( rpart::rpart, form ) %>%
 pipelearner::learn_models( randomForest::randomForest, form ) %>%
 pipelearner::learn_models( e1071::svm, form ) %>%
 pipelearner::learn() %>%
 dplyr::mutate( imp = map2(fit, train, f_model_importance)
                , plots = pmap( list( m = fit
                                       , ranked_variables = imp
                                       , title = model
                                       )
                                  , f_model_plot_variable_dependency_regression
                                  , formula = form
                                  , data_ls = data_ls
                                  , variable_color_code = variable_color_code
                                 )
 )  %>%
 .$plots %>%
 f_plot_obj_2_html( type = "plots"
                    , output_file =  'test_me'
                    , title = 'Plots'
                    , fig.width = 30
                    , fig.height = 21)

file.remove('test_me.html')

#type = grids --------------------------------------------------------------------

data_ls = f_clean_data(mtcars)

form = as.formula('disp~cyl+mpg+hp+am+gear+drat+wt+vs+carb')

variable_color_code = f_plot_color_code_variables(data_ls)

grids = pipelearner::pipelearner(data_ls$data) %>%
  pipelearner::learn_models( rpart::rpart, form ) %>%
  pipelearner::learn_models( randomForest::randomForest, form ) %>%
  pipelearner::learn_models( e1071::svm, form ) %>%
  pipelearner::learn() %>%
  dplyr::mutate( imp = map2(fit, train, f_model_importance)
                 , range_var = map_chr(imp, function(x) head(x,1)$row_names )
                 , grid = pmap( list( m = fit
                                      , title = model
                                      , variables = imp
                                      , range_variable = range_var
                                      , data = test
                 )
                 , f_model_plot_var_dep_over_spec_var_range
                 , formula = form
                 , data_ls = data_ls
                 , variable_color_code = variable_color_code
                 , log_y = F
                 , limit = 12
                 )
  )  %>%
  .$grid

f_plot_obj_2_html( grids
                   , type = "grids"
                   , output_file = 'test_me'
                   , title = 'Grids'
                   , height = 30 )

file.remove('test_me.html')

#' #type = model_performance -------------------------------------------------------

form = displacement ~ cylinders + mpg

df = ISLR::Auto %>%
 mutate( name = paste( name, row_number() ) ) %>%
 pipelearner::pipelearner() %>%
 pipelearner::learn_models( rpart::rpart, form ) %>%
 pipelearner::learn_models( randomForest::randomForest, form ) %>%
 pipelearner::learn_models( e1071::svm, form ) %>%
 pipelearner::learn() %>%
 f_predict_pl_regression( 'name' ) %>%
 unnest(preds) %>%
 mutate( bins = cut(target1, breaks = 3 , dig.lab = 4)
         , title = model )

dist = f_predict_plot_regression_distribution(df
                                              , col_title = 'title'
                                              , col_pred = 'pred'
                                              , col_obs = 'target1')


alluvial = f_predict_plot_regression_alluvials(df
                                               , col_id = 'name'
                                               , col_title = 'title'
                                               , col_pred = 'pred'
                                               , col_obs = 'target1')


taglist = f_predict_plot_model_performance_regression(df)
```

```
f_plot_obj_2_html( taglist
                 , type = 'model_performance'
                 , output_file = 'test_me'
                 , dist = dist
                 , alluvial = alluvial
                 , render_points_as_png = TRUE
                )


file.remove('test_me.html')
```

`f_plot_pretty_points` *plot prettier dot plot*

Description

color is contineoulsy scaled based on PC1 values and alpha values depend on point density.

Usage

```
f_plot_pretty_points(df, col_x, col_y, col_facet = NULL, size = 4,
  title = NULL, x_title = col_x, y_title = col_y, ...)
```

Arguments

`df`	datafram containing x,y pairs
`col_x`	character vector denoting x axis values
`col_y`	character vector denoting y axis values
`col_facet`	character vector denoting facetting column
`size`	size of points, Default: 4
`title`	character vector, Default: NULL
`x_title`	character vector, Default: col_x
`y_title`	character vector, Default: col_y
`...`	arguments passed to facet_wrap()

Details

Code adapted from `https://drsimonj.svbtle.com/pretty-scatter-plots-with-ggplot2`

Value

plot

See Also

`interp.surface` `kde2d`

Examples

```
df = ggplot2::diamonds %>%
  sample_n(2500)
col_x = 'carat'
col_y = 'price'
col_facet = 'cut'

f_plot_pretty_points(df, col_x, col_y, col_facet, title = 'price of diamonds by carat')
```

f_plot_profit_bars_plus_area

plot revenues cost and profit development over time with bars for revenue and costs and an area chart for profit.

Description

the function can graphically devide the chart into two periods e.g. past and future.

Usage

```
f_plot_profit_bars_plus_area(data, col_revenue, col_cost, col_time,
  now = max(data[, col_time]), unit_time = "years", unit_value = "CHF",
  title = "", alpha_past = 1, alpha_future = 0.5, alpha_past_area = 0.9,
  alpha_future_area = 0.7)
```

Arguments

`data` datafram

`col_revenue` character vector denoting revenue column

`col_cost` character vector denoting cost column

`col_time` character vector denoting time column

`now` integer denoting a time which should be regarded as the breakpoint, Default: max(data[, col_time])

`unit_time` character vector, will label y-axis, Default: ’years’

`unit_value` character vector, will label x-axis, Default: ’CHF’

`title` character vector, will be title label, Default: ”

`alpha_past` double between 0 and 1 will determine alpha value for fill under the curve before the breakpoint, Default: 1

`alpha_future` double between 0 and 1 will determine alpha value for fill under the curve after the breakpoint, Default: 0.5

`alpha_past_area` as alpha_past but for area only, Default: 0.9

`alpha_future_area` as alpha_future but for area only, Default: 0.7#’

Details

to some extent plotly compatible

Value

plot (to some extent plotly compatible)

Examples

```
data = tibble( time     = c(0,1,2,3,4,5,6,7,8,9,10,11,12)
             , revenue = - time^2 + time * 12
             , cost    = revenue * 0.4 * -1
             )
data[1,'cost'] = -10
data

print( f_plot_profit_bars_plus_area( data, 'revenue', 'cost', 'time') )
print( f_plot_profit_bars_plus_area( data, 'revenue', 'cost', 'time', now = 5) )

#clv figure for presenation
p = f_plot_profit_bars_plus_area( data, 'revenue', 'cost', 'time', now = 5, alpha_past_area = 0) +
  theme( panel.grid.major  = element_blank()
       , panel.grid.minor = element_blank()
       , axis.text        = element_blank()
       )+
  labs(x = '', y = '')
print(p)
```

---

f_plot_profit_lines *plot revenues cost and profit development over time as an area chart.*

Description

the function can graphically devide the chart into two periods e.g. past and future.

Usage

```
f_plot_profit_lines(data, col_revenue, col_cost, col_time, now = max(data[,
  col_time]), unit_time = "years", unit_value = "CHF", title = "",
  alpha_past = 1, alpha_future = 0.5)
```

Arguments

data	datafram
col_revenue	character vector denoting revenue column
col_cost	character vector denoting cost column
col_time	character vector denoting time column
now	integer denoting a time which should be regarded as the breakpoint, Default: max(data[, col_time])
unit_time	character vector, will label y-axis, Default: 'years'
unit_value	character vector, will label x-axis, Default: 'CHF'
title	character vector, will be title label, Default: "

`alpha_past` double between 0 and 1 will determine alpha value for fill under the curve before the breakpoint, Default: 1

`alpha_future` double between 0 and 1 will determine alpha value for fill under the curve after the breakpoint, Default: 0.5

Details

not plotly compatibel

Value

plot (is not plotly compatibel)

Examples

```
data = tibble( time     = c(0,1,2,3,4,5,6,7,8,9,10)
             , revenue = - time^2 + time * 12
             , cost    = revenue * 0.4 * -1
            )

print( f_plot_profit_lines( data, 'revenue', 'cost', 'time') )
print( f_plot_profit_lines( data, 'revenue', 'cost', 'time', now = 5) )
```

`f_plot_time` *plot variable distribution over time as reduced overlapping boxplots*

Description

It is difficult to compare two timeerieses when you have more than one observation per timepoint without reducing all observations to a single statistical variable such as average or mean. This visualisation plots the median and the upper and lower 25 contineuos line between the medians ot the timepoints.

Usage

```
f_plot_time(variable, time_variable, data_ls, time_variable_as_factor = F,
  group = NULL, normalize = F, time_unit = "day")
```

Arguments

`variable` character vector naming the variable to be plotted

`time_variable` character vector naming the timevariable to be plotted

`data_ls` data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)

`time_variable_as_factor`
If TRUE will convert time_variable to a factor, this will equalize the distance between timepoints on the plots and drops the connective line between timepoints, Default: F

`group` character vector naming the column to be used as grouping variable, Default: NULL `normalize` If TRUE y variable will be divided by x variable, usefull if y variable represents a cumulated sum, Default: F

`time_unit` character vector used as an x-axis lable , Default: ’day’

Value

plot

Examples

```
## Not run:

set.seed(1)
data       = dplyr::sample_n( nycflights13::flights, 1000 )
data$is_ua = ifelse( data$carrier == 'UA', 'UA', 'other')
data$date  = data$year * 10000 + data$month * 100 + data$day
data$date  = lubridate::as_date( data$date )
data_ls    = f_clean_data( data, replace_neg_values_with_zero = F)
f_plot_time( 'arr_delay', 'month', data_ls, group = 'is_ua', time_unit = 'month', time_variable_as_factor = T)

#without grouping
f_plot_time( 'arr_delay', 'month', data_ls, time_unit = 'month', time_variable_as_factor = F)


## End(Not run)
```

`f_predict_plot_model_performance_regression` *plot model performance*

Description

add predictions to modelling dataframe and unnest, create a title column and a bins column

Usage

```
f_predict_plot_model_performance_regression(data)
```

Arguments

`data` dataframe with the columns title, bins, resid_abs, resid_squ, ape

Value

taglist

[1] Headline summary Plots

[2] Residuals Pointplot

[3] Residuals Boxplot

[4] APE Pointplot

[5] APE Boxplot

[6] MAPE, MSE, MAE, Binning

[7] Headline Performance Measures Summary

[8] Summary MAPE, MSE, MAE with SE

[9] Summary MAPE, MSE, MAE with CI95

[10] Headline Summary Tables

[11] Summary Table

[12] Table Binning

See Also

tagList ggplotly datatable f_predict_pl_regression

Examples

```
## Not run:
form = as.formula( 'displacement~cylinders+mpg')

ISLR::Auto %>%
 pipelearner::pipelearner() %>%
 pipelearner::learn_models( twidlr::rpart, form ) %>%
 pipelearner::learn_models( twidlr::randomForest, form ) %>%
 pipelearner::learn_models( twidlr::svm, form ) %>%
 pipelearner::learn() %>%
 f_predict_pl_regression( 'name' ) %>%
 unnest(preds) %>%
 mutate( bins = cut(target1, breaks = 3 , dig.lab = 4)
         , title = paste(models.id, cv_pairs.id, train_p, target, model) ) %>%
 f_predict_plot_model_performance_regression() %>%
 f_plot_obj_2_html(type = 'taglist', 'test_me', title = 'Model Performance')

file.remove('test_me.html')

## End(Not run)
```

f_predict_plot_regression_alluvials

plot residuals of different models as aaluvials

Description

Residuals will be binned using boxplotstats with the modification, that the median will be set to zero. If quantile boundaries do not fall into sensible ranges they will be replaced by the standard error (SE). For example if we have the following boxplotstats -2, 1, 3, 5, 8 with an SE of 0.5 we will modify the boundaries as following. -2, -0.5, 0, 3, 5 . The reason is that we want to be able to follow observations with positive or negative residuals through the alluvial plot in order to judge emerging patterns. The models will be sorted by MSE and the alluvial plot will be flipped with the model with the lowest MSE on top.

Usage

```
f_predict_plot_regression_alluvials(data, col_id, col_title = "title",
  col_pred = "pred", col_obs = "target1", ...)
```

Arguments

`data`	dataframe
`col_id`	character vecotr dentoing id column
`col_title`	character vector denoting model title column, Default: 'title'
`col_pred`	character vector denoting prediciont column, Default: 'pred'
`col_obs`	character vector denoting column with observed values, Default: 'target1'
`...`	additional arguments passed to f_plot_alluvial_1v1

Value

plot

See Also

`brewer.pal`

Examples

```
  form = as.formula( 'displacement~cylinders+mpg')

df = ISLR::Auto %>%
  mutate( name = paste( name, row_number() ) ) %>%
  pipelearner::pipelearner() %>%
  pipelearner::learn_models( rpart::rpart, form ) %>%
  pipelearner::learn_models( randomForest::randomForest, form ) %>%
  pipelearner::learn_models( e1071::svm, form ) %>%
  pipelearner::learn() %>%
  f_predict_pl_regression( 'name' ) %>%
  unnest(preds)

f_predict_plot_regression_alluvials(df
                                    , col_id = 'name'
                                    , col_title = 'model'
                                    , col_pred = 'pred'
                                    , col_obs = 'target1')
```

`f_predict_plot_regression_distribution` *Plot distribution of model predictions vs observed*

Description

takes a dataframe with predictions and a title column and returns a list with one violin and one histogram plot to compare distributions.

Usage

```
f_predict_plot_regression_distribution(data, col_title = "title",
  col_pred = "pred", col_obs = "target1", bins = 60, ...)
```

Arguments

data	dataframE
col_title	character vector denoting title column, Default: 'title'
col_pred	character vector denoting column with predictions, Default: 'preds'
col_obs	character vecor denoting column with observed values, Default: 'target1'
bins	number of bins used for histograms, Default: 60
...	additional arguments passed to the facet_wrap function of the histogramss

Value

list with two plots

See Also

UQ brewer.pal

Examples

```
form = as.formula( 'displacement~cylinders+mpg')

df = ISLR::Auto %>%
  pipelearner::pipelearner() %>%
  pipelearner::learn_models( rpart::rpart, form ) %>%
  pipelearner::learn_models( randomForest::randomForest, form ) %>%
  pipelearner::learn_models( e1071::svm, form ) %>%
  pipelearner::learn() %>%
  f_predict_pl_regression( 'name' ) %>%
  unnest(preds)

f_predict_plot_regression_distribution(df
                                       , col_title = 'model'
                                       , col_pred = 'pred'
                                       , col_obs = 'target1')
```

f_predict_pl_regression

adds predictions to learned pipelearner dataframe

Usage

```
f_predict_pl_regression(pl, cols_id = NULL, formula = NULL,
  col_model = "fit", col_target = "target", data_test = "test",
  data_train = "train")
```

Arguments

`pl`	learned pipelearner dataframe
`cols_id`	character vector naming id column
`formula`	Default: NULL
`col_model`	character vector naming model column, Default: 'fit'
`col_target`	character vector naming target column, Default: 'target'
`data_test`	character vector naming test data column, Default: 'test'
`data_train`	character vector naming train data column, Default: 'train'#' @return dataframe

See Also

`f_predict_regression_add_predictions`

Examples

```
form = as.formula( 'disp~cyl+mpg')

pl = mtcars %>%
  mutate(names = row.names(.)) %>%
  pipelearner::pipelearner() %>%
  pipelearner::learn_models( twidlr::rpart, form ) %>%
  pipelearner::learn_models( twidlr::randomForest, form ) %>%
  pipelearner::learn_models( twidlr::svm, form ) %>%
  pipelearner::learn_models( gamlss::gamlss, form ) %>%
  pipelearner::learn() %>%
  f_predict_pl_regression( cols_id = 'names' )
```

`f_predict_pl_regression_summarize` *summarize prediction by f_predict_pl_regression()*

Description

use this function to get a quick summary of pipelearner dataframe with unnested predictions. Will group by title

Usage

```
f_predict_pl_regression_summarize(pl)
```

Arguments

`pl`	pipelearner dataframe with nested predictions

Value

dataframe with mape, mea, rtmse and median versions

See Also

`f_predict_pl_regression`

Examples

```
form = as.formula( 'disp~cyl+mpg')

pl = mtcars %>%
  mutate(names = row.names(.)) %>%
  pipelearner::pipelearner() %>%
  pipelearner::learn_models( twidlr::rpart, form ) %>%
  pipelearner::learn_models( twidlr::randomForest, form ) %>%
  pipelearner::learn_models( twidlr::svm, form ) %>%
  pipelearner::learn() %>%
  f_predict_pl_regression( 'names' ) %>%
  unnest( preds , .drop = FALSE ) %>%
  mutate( title = model ) %>%
  f_predict_pl_regression_summarize()

pl
```

f_predict_regression_add_predictions

adds predictions, residuals, abolute residuals, squared residuals and absolute percent error to a dataframe.

Description

absolute percent error = (abs(resid/pred)*100)

Usage

```
f_predict_regression_add_predictions(data_test, m, col_target,
  data_train = NULL, cols_id = NULL, formula = NULL, ...)
```

Arguments

`data_test`	dataframe containing data to be used as the basis for prediction. Can also be a modelR resample object
`m`	regression model
`col_target`	character vector naming target/response variable
`data_train`	dataframe with trainig data, Default: 'NULL'
`cols_id`	character vector naming id columns, if specified non_id columns will be dropped from dataframe, in order to be more memory efficient.
`formula`	Default NULL
`...`	additional arguments passed to HDtweedie and glmnet predict functions

Details

works with HDtweedie, randomForest, rpart, e1071::svm, glmnet, gamlss

Value

dataframe

dataframe

Examples

```
df = mtcars %>%
mutate(names = row.names(.))
m = rpart::rpart(disp~., df)
pred = f_predict_regression_add_predictions(df, m, 'disp', 'names')
pred
```

---

f_shiny_multiview *run multiview shiny app*

Description

shiny tool to quickly analyze diffferences between groups in large datasets

Usage

```
f_shiny_multiview(data = NULL)
```

Arguments

data	dataframe, will let you choose between a variety of sample data sets if NULL, Default: NULL

See Also

run

---

f_shiny_som *self organizing map shiny app*

Description

shiny tool to build self organizing maps with subsequent clustering restricted to adjacent map segements. See http://rpubs.com/erblast/SOM.

Usage

```
f_shiny_som(data = NULL)
```

Arguments

data	dataframe, will let you choose between a variety of sample data sets if NULL, Default: NULL

See Also

run

f_sim_profit *simulate profit*

Description

using the following parameters:

- retention rate (retention_rate)
- retention common for business (retention_rate_common)
- new customers acquired per year (nca_per_year)
- expected increase in customers acquired (expected_increase_nca)
- present number of customers (n_customers)
- fixed cost (fix_cost)
- profit per customer per year (profit_cm1_per_customer)

Usage

```
f_sim_profit(output_file = "profit_simulation", path = ".",
  params = "ask")
```

Arguments

output_file	PARAM_DESCRIPTION, Default: 'profit_simulation'
path	PARAM_DESCRIPTION, Default: '.'
params	list params passed to rmarkdown::render(), see example, Default: 'ask'

Details

DETAILS

Value

OUTPUT_DESCRIPTION

See Also

render

Examples

```
## Not run:
if(interactive()){
 f_sim_profit( path = tempdir() )
 f_sim_profit( prefix, params = list( retention_rate = 0.82
                                      , retention_rate_common = 0.88
                                      , nca_per_year = 4000
                                      , expected_increase_nca = 2
                                      , n_customers = 150000
                                      , fix_cost = 20000000
                                      , profit_cm1_per_customer = 200 )
 }
```

```
## End(Not run)
```

`f_stat_anova` *generate a datatframe with anova results from a data_ls list*

Description

returns a dataframe with shapiro, anova und kruskal p values supplemented with maximum difference of means and medians between groups

Usage

```
f_stat_anova(data_ls, col_group, boxcox = F)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_group`	character vector denoting grouping variable
`boxcox`	perform analysis on boxcox-transformed or numerical variables

Value

dataframe

See Also

`str_c` `map`,`map_dbl` `f_stat_anova`

Examples

```
df_anova = data_ls = f_clean_data(mtcars) %>%
  f_stat_anova('cyl')

df_anova
```

`f_stat_chi_square` *generate a datatframe with chi square results from a data_ls list*

Description

FUNCTION_DESCRIPTION

Usage

```
f_stat_chi_square(data_ls, col_group)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_group`	character vector denoting grouping variable

Value

dataframe

See Also

`is_empty`,`map`,`map_dbl`

Examples

```
data_ls = f_clean_data(mtcars)
df_chi_squ = f_stat_chi_square(data_ls, 'cyl')
df_chi_squ
```

`f_stat_combine_anova_with_chi_square` *combines anova with chi square results into single dataframe*

Description

keeps wither the anova or the kruskal p value depending on the results of the shapiro test (shapiro_stat > 0.9 p_val > 0.05) and keeps the difference of percent of the mean. Still works if one of the input dataframes is NULL

Usage

```
f_stat_combine_anova_with_chi_square(df_anova = NULL, df_chi_square = NULL)
```

Arguments

`df_anova`	dataframe created with f_stat_anova, Default: NULL
`df_chi_square`	dataframe created with f_stat_chi_square(), Default: NULL

Details

DETAILS

Value

OUTPUT_DESCRIPTION

Examples

```
data_ls = f_clean_data(mtcars)
df_chi_squ = f_stat_chi_square(data_ls, 'cyl')
df_anova = f_stat_anova(data_ls, 'cyl')
df_comb = f_stat_combine_anova_with_chi_square(df_anova, df_chi_squ)
df_comb
df_comb = f_stat_combine_anova_with_chi_square(df_anova)
df_comb
df_comb = f_stat_combine_anova_with_chi_square(df_chi_square = df_chi_squ)
df_comb
```

`f_stat_diff_of_means_medians` *calculates maximum difference in group means and medians*

Description

used as a helper function for f_stat_anova

Usage

```
f_stat_diff_of_means_medians(df, col_group, col_variable)
```

Arguments

`df`	dataframe
`col_group`	character vector denoting grouping variable
`col_variable`	character vector denoting variable

Value

dataframe

Examples

```
set.seed(1)
df = tibble( fct = sample(LETTERS[1:5], 100, replace = T)
             , v1  = 1
             , v2  = rnorm(100, 4)
             , v3  = c( rep(3, 50), rep(8,50) )
)

col_group = 'fct'

f_stat_diff_of_means_medians(df, col_group, 'v1') %>%
  bind_rows( f_stat_diff_of_means_medians(df, col_group, 'v2') ) %>%
  bind_rows( f_stat_diff_of_means_medians(df, col_group, 'v3') )
```

f_stat_group_ana *analyse group difference of dataset*

Description

creates a html document with a group analysis including:

- P value table
- Dynamic Plots of all significant features
- static plots with brackets indicating statistical differences
- Tabplot
- Alluvial Plot
- table containing means and medians for numerical variables
- table containing counts and percentages for categorical variables

The function automatically renders three html pages one for the additional static plots, one for the tableplot and one for the alluvial plots. In the same direcotry that can be determined by the outputfile parameter. Default behaviour will also render the entire html document returning the filepath of the new html file. The other three html files will be linked to in the document. You can modify the function to return a htmltools taglist instead. The above mentioned 3 additional html files for the other types of plots will still be rendered though with default settings. These extra plots can be switched off though.

Usage

```
f_stat_group_ana(data_ls, col_group, thresh_p_val = 0.05,
  thresh_diff_perc = 3, output_file = "group_ana", static_plots = T,
  alluvial = T, alluvial_thresh_p_val = 0.05,
  alluvial_thresh_diff_perc = 7.5, max_alluvial_flows = 1500, tabplot = T,
  return_taglist = F, fig.width = 16, fig.height = 10, quiet = FALSE)
```

Arguments

`data_ls` data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)

`col_group` character vector denoting grouping columns

`thresh_p_val` p value threshold for plots, Default: 0.05

`thresh_diff_perc` minimum percent difference threshold for plots, Default: 3

`output_file` character vector containing output file name

`static_plots` boolean, render static plots indicating statistical differences with brackets, Default = TRUE

`alluvial` boolean, render alluvial plot, Default: TRUE

`alluvial_thresh_p_val` double, threshold for feature to be inlcuded in alluvial plot. Features that are not highly significant and convey a large percental difference will result in a high number of flows thus cluttering the plot. It is not recommended to set these thresholds lower than the default. Default: 0.05

`alluvial_thresh_diff_perc`
double, threshold for feature to be inlcuded in alluvial plot. Features that are not highly significant and convey a large percental difference will result in a high number of flows thus cluttering the plot. It is not recommended to set these thresholds lower than the default. Default: 7.5

`max_alluvial_flows`
integer, maximum number of alluvial flows. Alluvial Plots can take a long time to render. Rendering an alluvial plot with the default setting of 1500 should take at least 10 min. Default 1500

`tabplot` boolean, render tabplot threshold for features are the same as for the dynamic plots, Default: TRUE , static_plots = T

`return_taglist` boolean, return taglist instead of rendereing the final html document and returning the link to the html file. Usefull if analysis should be directly included into the current markdown document.

`fig.width` integer Width of Alluvial and Tabplot in inches. Default values can be comfortably viewed on a 1920 x 1080 screen resolution. Default: 16

`fig.height` integer height of Alluvial and Tabplot in inches. Default values can be comfortably viewed on a 1920 x 1080 screen resolution. Default: 10

`quiet` booloean, suppress render markdown output to console, Default: FALSE

Value

file path to html file / or taglist

See Also

`ggplotly` `tagList`,`h1`,`h2`

Examples

```
## Not run:
 data_ls = f_clean_data(mtcars)
 f_stat_group_ana(data_ls, 'cyl', output_file = 'test_me')
 file.remove('test_me.html')
 file.remove('test_me_stat_plots.html')
 file.remove('test_me_alluvial.html')
 file.remove('test_me_tabplots.html')

## End(Not run)
```

`f_stat_group_ana_taglist` *analyse group difference of dataset*

Description

returns a full analysis as a taglist inluding all features with p_values, medians, means, percentages and counts, as well as plots passing the treshold values

Usage

```
f_stat_group_ana_taglist(data_ls, col_group, tresh_p_val = 0.05,
  thresh_diff_perc = 3)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_group`	character vector denoting grouping columns
`tresh_p_val`	p value threshold for plots, Default: 0.05
`thresh_diff_perc`	minimum percent difference threshold for plots, Default: 3

Value

taglist

See Also

`ggplotly` `tagList`,`h1`,`h2`

Examples

```
## Not run:
 data_ls = f_clean_data(mtcars)
 taglist = f_stat_group_ana_taglist(data_ls, 'cyl')
 f_plot_obj_2_html(taglist, type = "taglist", output_file = 'test_me', title = 'Plots')
 file.remove('test_me.html')

## End(Not run)
```

`f_stat_group_counts_percentages`

create a aggregated data frame with percentagers and counts for categorical variables

Usage

```
f_stat_group_counts_percentages(data_ls, col_group)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_group`	character vector denoting grouping columns

Value

dataframe

Examples

```
f_clean_data( mtcars) %>%
 f_stat_group_counts_percentages('cyl')
```

`f_stat_group_mean_medians`

create a aggregated data frame with means and medians for numerical variables

---

Usage

```
f_stat_group_mean_medians(data_ls, col_group)
```

Arguments

`data_ls`	data_ls object generated by f_clean_data(), or a named list list(data = <dataframe>, numericals = < vector with column names of numerical columns>)
`col_group`	character vector denoting grouping columns

Value

dataframe

Examples

```
f_clean_data( mtcars) %>%
 f_stat_group_mean_medians('cyl')
```

`f_stat_max_diff_of_freq`

calculate the maximal difference in frequencies between to categorical variables

---

Description

used as a helper function for f_stat_chi_square

Usage

```
f_stat_max_diff_of_freq(df, col_var1, col_var2)
```

Arguments

`df`	dataframe containing both avariables
`col_var1`	character vector denoting variable column 1
`col_var2`	character vector denoting variable column 2

Value

dataframe

Examples

```
data_ls = f_clean_data(mtcars)
df_chi_squ = f_stat_max_diff_of_freq(data_ls$data, 'cyl', 'gear')
```

f_stat_shapiro *wrapper for shapiro.test()*

Description

shapiro.test i slimited to <5000 sample size and raises an error if sd(x) == 0. Wrapper samples from input vector and returns a list object with NA parameters if sd(x) == 0.

Usage

```
f_stat_shapiro(vec)
```

Arguments

vec	numeric vector

Value

shapiro.test object or list(statistic = NA, p.value = NA)

Examples

```
f_stat_shapiro( rnorm(1000, 10, 1) )
f_stat_shapiro( runif(1000, 1, 10) )
```

f_stat_stars *calculate significant level from p value*

Description

* P:0.05, ** P:0,005, *** P:0.001

Usage

```
f_stat_stars(p_value)
```

Arguments

p_value	numeric

Value

character vector

Examples

```
f_stat_stars(0.06)
f_stat_stars(0.05)
f_stat_stars(0.005)
f_stat_stars(0.001)
```

`f_train_lasso` *wrapper for cv.glmnet and cv.HDtweedie*

Description

performs lasso for different distributions, returns a list of formulas that result in the lowest rtmse for at least one of the distributions. Graphical output allows side-by-side comparison of lasso behaviour for all distributions.

Usage

```
f_train_lasso(data, formula, p = c(1, 1.25, 1.5, 1.75, 2), k = 5,
  family = "gaussian", ...)
```

Arguments

`data`	dataframe
`formula`	formula
`p`	p parameter for tweedie distributions, set p = NULL for not performing lasso for tweedie distributions, Default: c(1, 1.25, 1.5, 1.75, 2)
`k`	fold cross validation, Default: 5
`family`	family parameter for glmnet, can be a vector, Default: 'gaussian'
`...`	arguments passed to cv.glmnet, cv.HDtweedie such as lambda or n_lambda

Details

Columns containing NA will be removed, formula cannot be constructed with '.'

Value

list()

See Also

,`HDtweedie` ,`glmnet` ,`cv.HDtweedie` ,`cv.glmnet` ,`pipelearner` ,`learn_models` ,`learn_cvpairs` ,`learn`

Examples

```
data = MASS::quine
formula = Days~.

lasso = f_train_lasso(data, formula, p = NULL, k = 1
                      , lambda = 10^seq(3,-3,length= 25) )
lasso = f_train_lasso(data, formula, p = 1.5, k = 2
                      , lambda = 10^seq(3,-3,length= 25) )

lasso
```

---

f_train_lasso_manual_cv *wrapper for glmnet and HDtweedie*

Description

performs lasso for different distributions, returns a list of formulas that result in the lowest rtmse for at least one of the distributions. Graphical output allows side-by-side comparison of lasso behaviour for all distributions.

Usage

```
f_train_lasso_manual_cv(data, formula, grid = 10^seq(4, -4, length = 100),
  p = c(1, 1.25, 1.5, 1.75, 2), k = 5, family = "gaussian")
```

Arguments

data	dataframe
formula	formula
grid	grid values for lambda, Default: 10^seq(4, -4, length = 100)
p	p parameter for tweedie distributions, set p = NULL for not performing lasso for tweedie distributions, Default: c(1, 1.25, 1.5, 1.75, 2)
k	fold cross validation, set to 1 for testing against training data, Default: 5
family	family parameter for glmnet, can be a vector, Default: 'gaussian'

Details

Columns containing NA will be removed, formula cannot be constructed with '.'

Value

list()

See Also

HDtweedie glmnet pipelearner,learn_models,learn_cvpairs,learn

Examples

```
data = MASS::quine
formula = Days~.

lasso = f_train_lasso(data, formula, p = NULL, k = 1
                      , grid = 10^seq(3,-3,length= 25) )
lasso = f_train_lasso(data, formula, p = 1.5, k = 2
                      , grid = 10^seq(3,-3,length= 25) )

lasso
```

---

make_container_for_function_calls

container for function calls, can be used as a progress bar

Description

creates a closure with a make_call() method that wraps any function call. When the wrapper is used the function call is saved and the calls are counted and the progress is being printed. Use the method set_total() to input the total number of function calls. Based on the total an ETA is estimated and a percentage calculated.

Usage

```
make_container_for_function_calls()
```

Details

DETAILS

Value

container

See Also

`now`,`time_length`

Examples

```

.f = randomForest::randomForest
call_cont = make_container_for_function_calls()
call_cont$set_total(4)
m_wr = call_cont$make_call( .f = .f, formula = disp~., data = mtcars )

#pipe version
call_cont = make_container_for_function_calls()
call_cont$set_total(5)

pl = pipelearner::pipelearner(mtcars) %>%
  pipelearner::learn_models( models = c( call_cont$make_call )
                            , formulas = c(disp~.)
```

```
                              , .f = c( randomForest::randomForest )
                              , function_name = 'randomForest'
                              , print_call = c(T)
                            ) %>%
pipelearner::learn_cvpairs( pipelearner::crossv_kfold, k = 5 ) %>%
pipelearner::learn()
```

Index

add_predictions, *21*
arrangeGrob, *31*

br, *10*
brewer.pal, *37*, *39*, *53*, *54*

cv.glmnet, *67*
cv.HDtweedie, *67*

datatable, *8*, *9*, *52*

f_boxcox, 4, *6*
f_clean_data, *4*, 5, *6*
f_clean_data_no_changes, 6
f_content, 6
f_content_get_file_type, 7
f_content_get_path, 7
f_content_render, 8
f_datatable_minimal, 8
f_datatable_universal, 9
f_html_breaks, 10
f_html_filename_2_link, 10
f_html_get_title_from_Rmd, 11
f_html_padding, 11
f_html_table_html_and_rmd_link, 12
f_manip_append_2_list, 13
f_manip_bin_numerics, 13
f_manip_bring_to_pos_range, 14
f_manip_data_2_model_matrix_format, 15
f_manip_double_2_int, 16
f_manip_factor_2_numeric, 16
f_manip_get_most_common_level, 17
f_manip_get_response_variable_from_formula, 17, *18*
f_manip_get_variables_from_formula, *17*, 18
f_manip_matrix_2_tibble, 18
f_manip_summarize_2_median_and_most_common_factor, 19
f_manip_transpose_tibble, 20
f_model_add_predictions_2_grid_regression, 20
f_model_data_grid, 21
f_model_importance, 22
f_model_importance_pl_add_plots_regression, 24
f_model_importance_pl_plots_as_html, 26
f_model_importance_plot, 22, *25*
f_model_importance_plot_tableplot, 23, *25*
f_model_importance_randomForest, 27
f_model_importance_rpart, 27
f_model_importance_svm, 28
f_model_plot_var_dep_over_spec_var_range, 30
f_model_plot_variable_dependency_regression, *25*, 29
f_model_seq_range, 32
f_pca, 33
f_pca_plot_components, 34
f_pca_plot_variance_explained, 35
f_plot_adjust_col_vector_length, 36
f_plot_alluvial, 36
f_plot_alluvial_1v1, 38
f_plot_col_vector74, 41
f_plot_color_code_variables, 40
f_plot_generate_comparison_pairs, 41
f_plot_hist, 42
f_plot_obj_2_html, 43
f_plot_pretty_points, 47
f_plot_profit_bars_plus_area, 48
f_plot_profit_lines, 49
f_plot_time, 50
f_predict_pl_regression, *52*, 54, *55*
f_predict_pl_regression_summarize, 55
f_predict_plot_model_performance_regression, 51
f_predict_plot_regression_alluvials, 52
f_predict_plot_regression_distribution, 53
f_predict_regression_add_predictions, *55*, 56
f_shiny_multiview, 57
f_shiny_som, 57
f_sim_profit, 58

f_stat_anova, *59*, 59
f_stat_chi_square, 59
f_stat_combine_anova_with_chi_square, 60
f_stat_diff_of_means_medians, 61
f_stat_group_ana, 62
f_stat_group_ana_taglist, 63
f_stat_group_counts_percentages, 64
f_stat_group_mean_medians, 65
f_stat_max_diff_of_freq, 65
f_stat_shapiro, 66
f_stat_stars, 66
f_train_lasso, 67
f_train_lasso_manual_cv, 68
fct_relevel, *37*, *39*
fct_rev, *39*
formatPercentage, *9*
formatRound, *9*
formatSignif, *9*

geom_flow, *37*, *39*
geom_stratum, *37*, *39*
ggplotly, *52*, *63*, *64*
glmnet, *67*, *68*

h1, *63*, *64*
h2, *63*, *64*
h3, *12*
h4, *12*
h6, *12*
HDtweedie, *67*, *68*

interp.surface, *47*
is_empty, *60*

kde2d, *47*

learn, *67*, *68*
learn_cvpairs, *67*, *68*
learn_models, *67*, *68*

make_container_for_function_calls, 69
map, *59*, *60*
map_dbl, *59*, *60*

now, *69*

pipelearner, *67*, *68*
prcomp, *34*

read_lines, *11*
render, *58*
run, *57*

str_c, *59*
str_detect, *9*, *16*
str_replace_all, *10*, *15*, *40*
str_split, *29*, *42*

tableplot, *24*
tagList, *10*, *12*, *26*, *52*, *63*, *64*
tidy, *17*
time_length, *69*

UQ, *39*, *42*, *54*