Package 'ExPanDaR'

April 6, 2019

Type Package

Title Explore Panel Data Interactively

Version 0.4.0

Description Provides a shiny-based front end (the 'ExPanD' app) and a set of functions for exploratory panel data analysis. Run as a web-based app, 'ExPanD' enables users to assess the robustness of empirical evidence without providing them access to the underlying data. You can also use the functions of the package to support your exploratory data analysis workflow. Refer to the vignettes of the package for more information on how to use 'ExPanD' and/or the functions of this package.

Depends R (>= 3.3.0)

License MIT + file LICENSE

URL https://joachim-gassen.github.io/ExPanDaR

BugReports https://github.com/joachim-gassen/ExPanDaR/issues

Encoding UTF-8

LazyData true

Imports Hmisc, tidyr, dplyr, ggplot2, corrplot, lfe, multiwayvcov, lmtest, stargazer, scales, shiny, DT, openssl, tictoc, shinycssloaders, kableExtra, rio

RoxygenNote 6.1.1

Suggests gapminder, rmarkdown, htmltools, knitr, devtools, tidyquant, wbstats

VignetteBuilder knitr

NeedsCompilation no

Author Joachim Gassen [aut, cre] (<https://orcid.org/0000-0003-4364-2911>)

Maintainer Joachim Gassen <gassen@wiwi.hu-berlin.de>

Repository CRAN

Date/Publication 2019-04-06 10:12:45 UTC

R topics documented:

`ExPanD` *Explore Panel Data (ExPanD)*

Description

A shiny based web app that uses ExPanDaR functionality for interactive panel data exploration

Usage

```
ExPanD(df = NULL, cs_id = NULL, ts_id = NULL, df_def = NULL,
  var_def = NULL, config_list = NULL,
  title = "ExPanD - Explore panel data interactively", abstract = NULL,
  df_name = deparse(substitute(df)), long_def = TRUE,
  factor_cutoff = 10L, components = c(sample_selection = TRUE,
  subset_factor = TRUE, grouping = TRUE, bar_chart = TRUE, missing_values =
  TRUE, udvars = TRUE, descriptive_table = TRUE, histogram = TRUE, ext_obs
  = TRUE, by_group_bar_graph = TRUE, by_group_violin_graph = TRUE,
  trend_graph = TRUE, quantile_trend_graph = TRUE, corrplot = TRUE,
  scatter_plot = TRUE, regression = TRUE), html_blocks = NULL,
  store_encrypted = FALSE, key_phrase = "What a wonderful key",
  debug = FALSE, ...)
```

Arguments

df	A data frame or a list of data frames containing the panel data that you want to explore. If NULL, ExPanD will start up with a file upload dialog.
cs_id	A character vector containing the names of the variables that identify the cross-section in your data. Can only be NULL if df_def is provided instead.
ts_id	A character scalar identifying the name of the variable that identifies the time series in your data. The according variable needs to be coercible to an ordered vector. If you provide a time series indicator that already is an ordered vector, ExPanD will verify that it has the same levels for each data frame and throw an error otherwise. Can only be NULL if df_def is provided instead.
df_def	An optional dataframe (or a list of dataframes) containing variable names, definitions and types. If NULL (the default) ExPanD uses cs_id and ts_id to identify the panel structure and determines the variable types (factor, numeric, logical) based on the classes of the data. See the details section for further information.
var_def	If you specify here a dataframe containing variable names and variable definitions, ExPanD will use these on the provided sample(s) to create the analysis sample. In that case, the user gets the opportunity to add additional variables in the app. See the details section for the structure of the var_def dataframe. If NULL (default) the sample(s) provided by df will be used as analysis sample(s) directly.
config_list	a list containing the startup configuration for ExPanD to display. Take a look at data(ExPanD_config_russell_3000) for the format. The easiest way to generate a config list is to customize the display within the app and then save the configuration locally.
title	the title to display in the shiny web app.
abstract	An introductory text to display in the shiny web app. Needs to be formatted as clean HTML.
df_name	A character string or a vector of character strings characterizing the dataframe(s) provided in df (will be used in the selection menu of the app)
long_def	If you set this to TRUE (default) and are providing a var_def then ExPanD will add the definitions of the used variables of the underlying dataframe to the definitions provided for the analysis sample to make these more informative to the user. If set to FALSE only the variable definitions provided in the var_def sample will be provided to the user.
factor_cutoff	ExPanD treats factors different from numerical variables. Factors are available for sub-sampling data and for certain plots. Each variable classified as such will be treated as a factor. In addition, ExPanD classifies all logical values and all numerical values with less or equal than factor_cutoff unique values as a factor.
components	A named logical vector indicating the components that you want ExPanD to generate and their order. See the function head of ExpanD for the list of available components. By default, all components are reported. You can also exclude selected components from the standard order by setting then to FALSE. In addition, you can include an arbitrary number of html_block components. Each

block will render clean HTML code as contained in the `html_blocks` parameter below. This allows you to customize your ExPanD report.

`html_blocks` A character vector containing the clean HTML code for each `html_block` that is included in `components`.

`store_encrypted`
Do you want the user-side saved config files to be encrypted? A security measure to avoid that users can inject arbitrary code in the config list. Probably a good idea when you are hosting sensitive data on a publicly available server.

`key_phrase` The key phrase to use for encryption. Change this from the default if you want to encrypt the config files.

`debug` Do you want ExPanD to echo some debug timing information to the console/log file and to store some diagnostics to the global environment? Probably not.

`...` Additional parameters that are passed on to `runApp`.

Details

If you start ExPanD without any options, it will start with an upload dialog so that the user (e.g., you) can upload a data file for analysis. Supported formats are as provided by the `rio` package.

If you provide variable definitions in `df_def` and/or `var_def`, ExPanD displays these as tooltips in the descriptive table of the ExPanD app.

When you provide more than one data frame in `df`, make sure that all have the same variables and variable types defined. If not, ExPanD will throw an error. When you provide only one `df_def` for multiple data frames, `df_def` will be recycled.

When you provide `var_def`, ExPanD starts up in the "advanced mode". The advanced mode uses (a) base sample(s) (the one(s) you provide via `df`) and the variable definitions in `var_def` to generate an analysis sample based on the active base sample. In the advanced mode, the app user can delete variables from the analysis sample within the app.

A `df_def` or `var_def` dataframe can contain the following variables

"var_name" Required: The names of the variables that are provided by the base sample or are to be calculated for the analysis sample

"var_def" Required: For a `var_def` data frame, the code that is passed to the data frame (grouped by cross-sectional units) in calls to `mutate` as right hand side to calculate the respective variable. For a `data_def` data frame, a string describing the nature of the variable.

"type" Required: One of the strings "cs_id", "ts_id", "factor", "logical" or "numeric", indicating the type of the variable. Please note that at least one variable has to be assigned as a cross-sectional identifier ("cs_id") and exactly one variable that is coercible into an ordered factor has to be assigned as the time-series identifier ("ts_id").

"can_be_na" Optional: If included, then all variables with this value set to FALSE are required to be non missing in the data set. This reduces the number of observations. If missing, it defaults to being TRUE for all variables other than cs_id and ts_id.

Examples

```
## Not run:
  ExPanD()
```

```
# Use this if you want to read very large files via the file dialog
options(shiny.maxRequestSize = 1024^3)
ExPanD()

data(russell_3000)
ExPanD(russell_3000, c("coid", "coname"), "period")
ExPanD(russell_3000, df_def = russell_3000_data_def)
ExPanD(russell_3000, df_def = russell_3000_data_def,
  components = c(ext_obs = T, descriptive_table = T, regression = T))
ExPanD(russell_3000, df_def = russell_3000_data_def,
  components = c(missing_values = F, by_group_violin_graph = F))
ExPanD(russell_3000, df_def = russell_3000_data_def,
  components = c(html_block = T, descriptive_table = T,
  html_block = T, regression = T),
  html_blocks = c(
  paste('<div class="col-sm-2"><h3>HTML Block 1</h3></div>',
  '<div class="col-sm-10">',
  "<p></p>This is a condensed variant of ExPanD with two additional HTML Blocks.",
  "</div>"),
  paste('<div class="col-sm-2"><h3>HTML Block 2</h3></div>',
  '<div class="col-sm-10">',
  "It contains only the descriptive table and the regression component.",
  "</div>")))
data(ExPanD_config_russell_3000)
ExPanD(df = russell_3000, df_def = russell_3000_data_def,
  config_list = ExPanD_config_russell_3000)
exploratory_sample <- sample(nrow(russell_3000), round(0.5*nrow(russell_3000)))
test_sample <- setdiff(1:nrow(russell_3000), exploratory_sample)
ExPanD(df = list(russell_3000[exploratory_sample, ], russell_3000[test_sample, ]),
  df_def = russell_3000_data_def,
  df_name = c("Exploratory sample", "Test sample"))
ExPanD(worldbank, df_def = worldbank_data_def, var_def = worldbank_var_def,
  config_list = ExPanD_config_worldbank)


## End(Not run)
```

ExPanDaR *ExPanDaR: Explore Panel Data with R*

Description

ExPanDaR provides the code base for the ExPanD web app. ExPanD is a shiny based app supporting interactive exploratory data analysis.

Details

ExPanDaR has two main goals:

- Enable users to assess the robustness of empirical evidence without providing them with access to the underlying data.

- Provide a toolbox for researchers to explore panel data on the fly.

To learn more about ExPanDaR, start with the vignettes: `browseVignettes(package = "ExPanDaR")`

`ExPanD_config_russell_3000`

Default Configuration to use with ExPanD and the Russell 3000 Data Set

---

Description

List to use as a `list_config` parameter when starting ExPanD.

Usage

```
data(ExPanD_config_russell_3000)
```

Format

An object of class `"list"`.

Examples

```
data(russell_3000)
data(russell_3000_data_def)
data(ExPanD_config_russell_3000)
## Not run:
 ExPanD(russell_3000, df_def = russell_3000_data_def, config_list = ExPanD_config_russell_3000)

## End(Not run)
```

`ExPanD_config_worldbank`

Default Configuration to Use with ExPanD and the `worldbank` *Data Set*

Description

List to use as a `list_config` parameter when starting ExPanD.

Usage

```
data(ExPanD_config_worldbank)
```

Format

An object of class `"list"`.

Examples

```
data(worldbank)
data(worldbank_data_def)
data(worldbank_var_def)
data(ExPanD_config_worldbank)
## Not run:
  ExPanD(worldbank, df_def = worldbank_data_def,
    var_def = worldbank_var_def, config_list = ExPanD_config_worldbank)

## End(Not run)
```

`prepare_by_group_bar_graph` *Prepares a by Group Bar Graph*

Description

Reads a data frame containing a grouping factor and a numerical variable and plots a bar graph of a given statistic of the variable by the grouping factor.

Usage

```
prepare_by_group_bar_graph(df, by_var, var, stat_fun = mean,
  order_by_stat = FALSE, color = "red")
```

Arguments

`df`	Data frame containing the grouping factor and the numerical variable to be plotted
`by_var`	a string containing the column name of the grouping factor
`var`	a string containing the column name of the numerical variable
`stat_fun`	a function to be called on the numerical variable. Will be called with `na.rm = TRUE` to ignore missing values
`order_by_stat`	a logical value indicating whether you want your bars to be ordered the value of the statistic (defaults to FALSE)
`color`	bar color

Value

A list containing two items:

"df" A data frame containing the statistics by group

"plot" The plot as returned by `ggplot`

Examples

```
data(russell_3000)
graph <- prepare_by_group_bar_graph(russell_3000, "sector", "ni_sales", median)
graph$plot
```

`prepare_by_group_violin_graph` *Prepares a by Group Violin Graph*

Description

Reads a data frame containing a grouping factor and a numerical variable and plots a series of violin graphs by the grouping factor.

Usage

```
prepare_by_group_violin_graph(df, by_var, var, order_by_mean = FALSE,
  group_on_y = TRUE, ...)
```

Arguments

`df`	Data frame containing the grouping factor and the numerical variable to be plotted
`by_var`	a string containing the column name of the grouping factor
`var`	a string containing the column name of the numerical variable
`order_by_mean`	a logical value indicating whether you want your violins to be ordered by group means (defaults to FALSE)
`group_on_y`	a logical value indicating whether you want your violins to be oriented horizontally (defaults to TRUE)
`...`	additional parameters that are passed to `geom_violin`

Value

The plot as returned by `ggplot2`

Examples

```
data(russell_3000)
df <- treat_outliers(russell_3000)
prepare_by_group_violin_graph(df, "sector", "nioa")
```

prepare_correlation_graph

Prepares a Correlation Graph

Description

Reads a data frame and presents Pearson correlations above and Spearman correlations the diagonal using a fancy graph prepared by the package `corrplot`.

Usage

```
prepare_correlation_graph(df)
```

Arguments

`df`	Data frame containing at least two variables that are either numeric or logical and at least five observations.

Value

The function directly renders the graph as produced by `corrplot`. In addition, it returns a list containing three items:

"df_corr" A data frame containing the correlations

"df_prob" A data frame containing the p-values of the correlations

"df_n" A data frame containing the number of observations used for the correlations

Examples

```
prepare_correlation_graph(mtcars)
```

prepare_correlation_table

Prepares a Correlation Table

Description

Reads a data frame and presents Pearson correlations above the diagonal and Spearman correlations below.

Usage

```
prepare_correlation_table(df, digits = 2, bold = 0.05,
  format = "html", ...)
```

Arguments

df	Data frame containing at least two variables that are either numeric or logical and at least five observations.
digits	The number of digits that you want to report.
bold	Indicate the p-Value for for identifying significant correlations in bold print. Defaults to 0.05. If set to 0, no bold print is being used.
format	The format that you want kable to produce ("html" or "latex")
...	Additional parameters that are passed on to kable

Value

A list containing four items:

"df_corr" A data frame containing the correlations

"df_prob" A data frame containing the p-values of the correlations

"df_n" A data frame containing the number of observations used for the correlations

"kable_ret" The return value provided by kable containing the formatted table

Examples

```
t <- prepare_correlation_table(mtcars)
t$df_corr
```

---

prepare_descriptive_table

Prepares a Table of Descriptive Statistics

Description

Reads a data frame and reports descriptive statistics (n, mean, standard deviation, minimum, first quartile, median, third quartile, maximum) for all members of the data frame that are either numeric or logical.

Usage

```
prepare_descriptive_table(df, digits = c(0, 3, 3, 3, 3, 3, 3, 3),
  format = "html")
```

Arguments

df	Data frame containing at least one variable that is either numeric or logical and at least two observations.
digits	Number of decimal digits that you want to be displayed for each column. If you provide NA, then the column is omitted from the output.
format	character scalar that is handed over to kable (e.g., "html" or "latex").

Details

The digits parameter from prepare_descriptive_table() uses the default method of kable to format numbers, calling round. This implies that trailing zeroes are just omitted.

Value

A list containing two items.

"df" A data frame containing the descriptive table

"kable_ret" The return value provided by kable containing the formatted table

Examples

```
t <- prepare_descriptive_table(mtcars)
t$df
```

---

prepare_ext_obs_table *Prepares a Table Displaying Extreme Observations*

Description

Reads a data frame, sorts it by the given variable and displays the top and bottom n observations.

Usage

```
prepare_ext_obs_table(df, n = 5, cs_id = NULL, ts_id = NULL,
  var = utils::tail(colnames(df[sapply(df, is.numeric) & (!colnames(df)
  %in% c(cs_id, ts_id))]), n = 1), ...)
```

Arguments

df	Data frame
n	The number of top/bottom observations that you want to report.
cs_id	The variable(s) identifying the cross-section in the data.
ts_id	The variable identifying the time-series in the data.
var	Variable to display. Defaults to the last numerical variable of the data frame.
...	Additional parameters that are passed to kable.

Details

When both cs_id and ts_id are omitted, all variables are tabulated. Otherwise, var is tabulated along with the identifiers. Infinite values in var are omitted. The default parameters for calling kable, are format = "html", digits = 3, format.args = list(big.mark = ','), row.names = FALSE.

Value

A list containing two items:

"df" A data frame containing the top/bottom n observations

"kable_ret" The return value provided by `kable` containing the formatted table

Examples

```
t <- prepare_ext_obs_table(russell_3000, n = 10,
                           cs_id = c("coid", "coname"),
                           ts_id = "period", var = "sales")
t$df
```

`prepare_missing_values_graph`
Prepares a Graph Displaying Missing Values in Panel Data

Description

Displays a heatmap of missing value frequency across the panel

Usage

```
prepare_missing_values_graph(df, ts_id, no_factors = FALSE)
```

Arguments

`df`	Data frame containing the data.
`ts_id`	A string containing the name of the variable indicating the time dimension. Needs to be coercible into an ordered factor.
`no_factors`	A logical variable indicating whether you want to limit the plot to logical and numerical variables. Defaults to FALSE.

Details

This was inspired by a blog post of Rense Nieuwenhuis. Thanks!

Value

A `ggplot2` plot.

Examples

```
prepare_missing_values_graph(russell_3000, ts_id="period")
```

prepare_quantile_trend_graph *Prepares a Quantile Trend Graph*

Description

Reads a data frame and plots the quantiles of the specified variable by an ordered factor (normally the time-series indicator)

Usage

```
prepare_quantile_trend_graph(df, ts_id, quantiles = c(0.05, 0.25, 0.5,
  0.75, 0.95), var = utils::tail(colnames(df[sapply(df, is.numeric) &
  colnames(df) != ts_id]), n = 1))
```

Arguments

df	Data frame containing the ordered factor and the numerical variable to be plotted
ts_id	a string containing the column name of the ordered factor (normally the time-series indicator)
quantiles	a numerical vector containing the quantiles that are to be plotted
var	a string containing the column name of the variable to be plotted. Defaults to the last numerical variable of the data frame that is not ts_id.

Value

A list containing two items:

"df" A data frame containing the plotted quantiles

"plot" The plot as returned by ggplot

Examples

```
prepare_quantile_trend_graph(worldbank, "year", var = "SP.DYN.LE00.IN")$plot +
  ggplot2::ylab("Life expectancy at birth world-wide")
df <- data.frame(year = floor(stats::time(datasets::EuStockMarkets)),
                 DAX = datasets::EuStockMarkets[,"DAX"])
graph <- prepare_quantile_trend_graph(df, "year", c(0.05, 0.25, 0.5, 0.75, 0.95))
graph$plot
```

prepare_regression_table *Prepares a Regression Table*

Description

Builds a regression table based on a set of user-specified models or a single model and a partitioning variable.

Usage

```
prepare_regression_table(df, dvs, idvs, feffects = rep("", length(dvs)),
  clusters = rep("", length(dvs)), models = rep("auto", length(dvs)),
  byvar = "", format = "html")
```

Arguments

df	Data frame containing the data to estimate the models on.
dvs	A character vector containing the variable names for the dependent variable(s).
idvs	A character vector or a a list of character vectors containing the variable names of the independent variables.
feffects	A character vector or a a list of character vectors containing the variable names of the fixed effects.
clusters	A character vector or a a list of character vectors containing the variable names of the cluster variables.
models	A character vector indicating the model types to be estimated ('ols', 'logit', or 'auto')
byvar	A factorial variable to estimate the model on (only possible if only one model is being estimated).
format	A character scalar that is passed on stargazer as type to determine the presentation format ("html", "text", or "latex").

Details

This is a wrapper function calling the stargazer package. Depending on whether the dependent variable is numeric, logical or a factor with two levels, the models are estimated using felm (for numeric dependent variables) or glm (with family = binomial(link="logit")) (for two-level factors or logical variables). You can override this behavior by specifying the model with the models parameter. Multinomial logit models are not supported. For glm, clustered standard errors are estimated using cluster.vcov. If run with byvar, only levels that have more observations than coefficients are estimated.

Value

A list containing two items

"models" A list containing the model results and by values if appropriate

"table" The output of `stargazer` containing the table

Examples

```
df <- data.frame(year = as.factor(floor(stats::time(datasets::EuStockMarkets))),
                 datasets::EuStockMarkets)
dvs = c("DAX", "SMI", "CAC", "FTSE")
idvs = list(c("SMI", "CAC", "FTSE"),
            c("DAX", "CAC", "FTSE"),
            c("SMI", "DAX", "FTSE"),
            c("SMI", "CAC", "DAX"))
feffects = list("year", "year", "year", "year")
clusters = list("year", "year", "year", "year")
t <- prepare_regression_table(df, dvs, idvs, feffects, clusters, format = "text")
t$table
t <- prepare_regression_table(df, "DAX", c("SMI", "CAC", "FTSE"), byvar="year", format = "text")
print(t$table)
```

---

prepare_scatter_plot *Prepares a Scatter Plot*

Description

Reads a data frame and prepares a scatter plot.

Usage

```
prepare_scatter_plot(df, x, y, color = "", size = "", loess = 0,
  alpha = min(1, 1/((1 + (max(0, log(nrow(df)) - log(100)))))))
```

Arguments

`df`	Data frame containing the data
`x`	a string containing the column name of the x variable
`y`	a string containing the column name of the y variable
`color`	a string containing the column name of the variable providing the color aesthetic (can be numerical or a factor)
`size`	a string containing the column name of the variable providing the size aesthetic
`loess`	a numerical scalar **0** No loess curve **1** loess curve with equal weights **2** loess curve with weights based on `size` variable
`alpha`	The alpha value to be used. If missing, it calculates a default based on the sample size

Value

the plot as returned by ggplot

Examples

```
df <- data.frame(year = floor(stats::time(datasets::EuStockMarkets)),
                 datasets::EuStockMarkets[, c("DAX", "FTSE")])
prepare_scatter_plot(df, x="DAX", y="FTSE", color="year")
```

---

prepare_trend_graph *Prepares a Trend Graph*

Description

Reads a data frame and line plots all variables (which need to be numeric) by an ordered factor (normally the time-series indicator).

Usage

```
prepare_trend_graph(df, ts_id, var = colnames(df[sapply(df, is.numeric) &
  colnames(df) != ts_id]))
```

Arguments

df	Data frame containing the ordered factor and a set of numerical variables to be plotted
ts_id	a string containing the column name of the ordered factor (normally the time-series indicator)
var	a character vector containing the column names of the variables that should be plotted. Defaults to all numeric variables of the data frame besides the one indicated by ts_id.

Value

A list containing two items:

"df" A data frame containing the plotted means and standard errors

"plot The plot as returned by ggplot

Examples

```
df <- data.frame(year = floor(time(EuStockMarkets)), EuStockMarkets)
graph <- prepare_trend_graph(df, "year")
graph$plot
```

russell_3000 *Annual Financial Accounting and Stock Return Data for a Sample of Russell 3000 Firms (2013-2016)*

Description

Data collected from Google Finance and Yahoo finance using the package `tidyquant`.

Usage

```
data(russell_3000)
```

Format

An object of class `"data.frame"`.

Source

Has been collected using the `tq_get` function family in Summer 2017. The code to generate this data is available in the github repository of this package. As the Google Finance API providing financial statement data is currently unavailable, the data cannot be replicated by running the code. Use in scientific studies is not advised without prior cleaning/checking.

Examples

```
data(russell_3000)
prepare_missing_values_graph(russell_3000, ts_id = "period")
```

russell_3000_data_def *Data Definitions for* `russell_3000` *Data Set*

Description

A data frame containing variable definitions for the `russell_3000` data set. The data definitions can be passed to ExPanD via the `df_def` parameter.

Usage

```
data(russell_3000_data_def)
```

Format

An object of class `"data.frame"`.

Details

Data definitions are provided by the package maintainer and are somewhat superficial to make them both, short and informative. User discretion is advised when using this data outside of its didactic purpose.

Examples

```
data(russell_3000)
data(russell_3000_data_def)
data(ExPanD_config_russell_3000)
## Not run:
 ExPanD(russell_3000, df_def = russell_3000_data_def, config_list = ExPanD_config_russell_3000)

## End(Not run)
```

`treat_outliers` *Treats Outliers in Numerical Data*

Description

Treats numerical outliers either by winsorizing or by truncating.

Usage

```
treat_outliers(x, percentile = 0.01, truncate = FALSE, by = NULL,
  ...)
```

Arguments

`x`	Data that is coercible into a numeric vector or matrix. If it is a data frame then all numerical variables of the data frame are coerced into a matrix.
`percentile`	A numeric scalar. The percentile below which observations are considered to be outliers. Is treated symmetrical so that `c(percentile, 1-percentile)` are used as boundaries. Defaults to 0.01 and needs to be > 0 and < 0.5.
`truncate`	A logical scalar. If TRUE then data are truncated (i.e., set to NA if out of bounds). Defaults to FALSE.
`by`	NULL or either a factor vector or a character string identifying a factor variable in the data frame provided by x. The factor indicated by 'by' is being used to identify groups by which the outlier treatment is applied. Defaults to NULL (no grouping). If provided, the resulting vector must not contain NAs and needs to be such so that `length(byvec) == nrows(as.matrix(x))`.
`...`	Additional parameters forwarded to quantile (notably, `type`)

Details

All members of the numerical matrix are checked for finiteness and are set to NA if they are not finite. NAs are removed when calculating the outlier cut-offs.

Value

A numeric vector or matrix containing the outlier-treated x. if a data frame was provided in x, a data frame with its numeric variables replaced by their outlier-treated values.

Examples

```
treat_outliers(seq(1:100), 0.05)
treat_outliers(seq(1:100), truncate = TRUE, 0.05)

# When you like the percentiles calculated like STATA's summary or pctile:
treat_outliers(seq(1:100), 0.05, type = 2)

df <- data.frame(a = seq(1:1000), b = rnorm(1000), c = sample(LETTERS[1:5], 1000, replace=TRUE))
winsorized_df <- treat_outliers(df)
summary(df)
summary(winsorized_df)

winsorized_df <- treat_outliers(df, 0.05, by="c")
by(df, df$c, summary)
by(winsorized_df, df$c, summary)

hist(treat_outliers(rnorm(1000)), breaks=100)
```

---

worldbank — *A Snapshot of Macroeconomic Data as Provided by the World Bank API (1960 - 2016)*

Description

Data collected from the World Bank API using the package wbstats.

Usage

```
data(worldbank)
```

Format

An object of class "data.frame".

Source

Has been collected using the wb function from the World Bank API in March 2018. The code to generate this data is available in the github repository of this package. Use in scientific studies is not advised without prior cleaning/checking.

Examples

```
data(worldbank)
prepare_missing_values_graph(worldbank, ts_id = "year")

data(worldbank_data_def)
data(worldbank_var_def)
data(ExPanD_config_worldbank)
## Not run:
  ExPanD(worldbank, df_def = worldbank_data_def,
    var_def = worldbank_var_def, config_list = ExPanD_config_worldbank)

## End(Not run)
```

---

worldbank_data_def *Data Definitions for* worldbank *Data Set*

Description

A data frame containing variable definitions for the worldbank data set. The data definitions can be passed to ExPanD via the df_def parameter.

Usage

```
data(worldbank_data_def)
```

Format

An object of class "data.frame".

Details

Data definitions are as provided by the World Bank API and the code to generate them is available in the github repository of this package.

Examples

```
data(worldbank)
data(worldbank_data_def)
data(worldbank_var_def)
data(ExPanD_config_worldbank)
## Not run:
  ExPanD(worldbank,df_def = worldbank_data_def,
    var_def = worldbank_var_def, config_list = ExPanD_config_worldbank)

## End(Not run)
```

worldbank_var_def *Variable Definitions to Construct an Analysis Sample Based on the* worldbank *Data Set*

Description

A data frame containing variable definitions that can be passed to ExPanD via the var_def parameter.

Usage

```
data(worldbank_var_def)
```

Format

An object of class "data.frame".

Examples

```
data(worldbank)
data(worldbank_data_def)
data(worldbank_var_def)
data(ExPanD_config_worldbank)
## Not run:
  ExPanD(worldbank, df_def = worldbank_data_def,
    var_def = worldbank_var_def, config_list = ExPanD_config_worldbank)

## End(Not run)
```

Index