Package 'Bullock'

November 25, 2021

Type Package

Title Tools for table-making and miscellaneous helper utilities

Version 2.4.0.3.9000

Date 2021-11-24

Imports knitr, lmtest, magrittr, rlang, rmarkdown, stringr

Suggests AER,
crayon,
dplyr,
estimatr,
ivpack,
multiwayvcov,
testthat,
tibble

Description These functions are used in John Bullock's code; they are typically needed for replication purposes. They range in complexity from a function that removes NA values from a vector prior to summing it (sumNA) to a trio of functions that produce well-formatted regression-table output (regTable, latexTable, and latexTablePDF).

License GPL (>= 2)

LazyLoad yes

URL https://github.com/jbullock35/Bullock

BugReports https://github.com/jbullock35/Bullock/issues

RoxygenNote 7.1.1

Encoding UTF-8

Language nob

VignetteBuilder knitr

R topics documented:

`.ls.objects` *List objects and their characteristics*

Description

`lsos()` and `ls.objects()` are like `ls()` in that they list the objects in a given environment. But they provide more detail: they show the class of each object, the amount of memory devoted to each objects, and the number of rows or columns of each object (if applicable).

Usage

```
.ls.objects(
  pos = 1,
  envir = as.environment(pos),
  pattern,
  order.by,
  decreasing = FALSE,
  MB = MB,
  n = NULL
)
```

```
lsos(..., MB = TRUE, n = 8)
```

Arguments

pos	Numeric. Specifies the position, in the search list, of the environment to search. Can be specified instead of envir.
envir	Environment to search. Can be specified instead of pos.
pattern	String. An optional regular expression; if it is specified, only objects whose names match the regular expression will be returned.
order.by	String, with value "Class", "Size", "Rows", or "Columns." The returned data frame will be sorted by the specified column.
decreasing	Logical variable. Should results be listed in decreasing order of the order.by column? In .ls.objects() the default is FALSE; in lsos(), it is TRUE.
MB	Logical. If TRUE (the default), object size is reported in megabytes. If FALSE, it is reported in kilobytes.
n	Numeric. Number of objects to list. In lsos(), the default is 8.
...	Additional arguments to .ls.objects().

Details

lsos() is a user-friendly wrapper for .ls.objects(). It is shorthand for .ls.objects(order.by="Size",decreasing

Both functions were created by Dirk Edelbuettel and modified by JD Long and John Bullock. See http://stackoverflow.com/questions/1358003/ for details.

Value

Data frame with columns "Class," "Size," "Row," and "Columns." The row names of the data frame are the names of objects in the environment.

Author(s)

Both functions were created by Dirk Edelbuettel and modified by JD Long and John Bullock. See http://stackoverflow.com/questions/1358003/ for details.

Examples

```
lsos()
lsos(pattern = '\\.df$')  # list only objects ending in ".df"
```

---

alpha_cronbach *Compute Cronbach's alpha*

Description

alpha_cronbach() takes a covariance matrix of data, S, that has been created from a matrix in which each column represents a variable. It returns Cronbach's alpha for the battery. It is not exported, but it is called by reliability().

Usage

```
alpha_cronbach(S)
```

Arguments

S	Covariance matrix generated by `var()`.

Value

The return value is a number (double): the estimated value of Cronbach's alpha.

Author(s)

Joseph F. Lucke

Bullock *Bullock: convenience functions and tools for table generation*

Description

This package provides tools that I use in my code. Most of the tools are simple convenience functions. Three of the functions—`regTable()`, `latexTable()`, and `latexTablePDF()`—are more elaborate tools for making tables, and they are designed to be used together. See the Building better tables in less time vignette for an overview.

Details

Of the convenience functions, I most often use `qw()`, `reliability()`, `meanNA()`, and `sumNA()`.

---

cutYears *Transform a vector of year values into an ordered factor of year groups.*

Description

This function is a wrapper around `cut()`. Given a vector of strings or integers that represent years, and a vector of breakpoints, it returns a factor in which each level represents a group of years. Unlike `cut()`, it returns pretty labels for the levels: "1975-79" instead of "[1975,1980)", and so on. Also unlike `cut()`, it ensures that all of the data are accounted for in the levels of the factor that it creates: data will never be dropped from a factor that `cutYears()` returns.

Usage

```
cutYears(x, breaks, levelsBoundedByData = TRUE, shortLabels = TRUE)
```

Arguments

x	Vector of four-digit integers, or of four-character strings that can be converted to integers, e.g., "1900".
breaks	Numeric vector of cutpoints
levelsBoundedByData	Logical. Ensures that the lowest and highest levels of the returned factor will contain some data. Also ensures that the label for the highest factor level reports the maximum year in x, rather than a higher year.
shortLabels	Logical. If FALSE, the second year in each label will always have four digits: for example, "1975-1999". If TRUE (the default), the second year in each label will typically have two digits: for example, "1975-99". But even if shortLabels is TRUE, the second year in a label will have four digits if it isn't in the same century as the first year. For example, cutYears() will always produce a label like "1975-2001" instead of "1975-01".

Details

By default, cutYears() differs from cut() in the following ways:

- Accepts only x vectors in which every value has four characters or four digits.
- Returns a factor that has better labels for groups of years: for example, "1975-80" rather than "[1975,1980)".
- Returns factor levels that encompass all values of x. Consequently, cutYears() will never convert year values to NA, as cut() will often do.
- Returns an ordered factor by default.
- By default, cutYears() drops levels that are outside the bounds of x. For example, if x ranges from 1975 to 1985, the factor returned by cut() may have an infinite number of levels, including, say, "(1900-1905]". (The exact levels returns by cut() depend on the arguments passed to it, especially the breaks argument.) But in a case like this, the lowest factor level returned by cutYears() will include 1975, and the highest factor level returned by cutYears() will contain 1985.

See Also

cut(), Hmisc::cut2()

Examples

```
years <- rep(1975:1993, each = 3)
fac1a <- cut(     years, breaks = seq(1975, 1993, by = 3))
fac1b <- cutYears(years, breaks = seq(1975, 1993, by = 3))
fac1c <- cutYears(years, breaks = seq(1975, 1993, by = 3), shortLabels = FALSE)

table(fac1a)
table(fac1b)
table(fac1c)

fac2a <- cut(     years, breaks = seq(1975, 1990, by = 3))
fac2b <- cutYears(years, breaks = seq(1975, 1990, by = 3))
table(fac2a)
table(fac2b)
```

```
fac3a <- cut(     years, breaks = seq(1955, 1990, by = 3))
fac3b <- cutYears(years, breaks = seq(1955, 1990, by = 3))
table(fac3a)
table(fac3b)
```

`latexTable` *Create a LaTeX table from a matrix.*

Description

`latexTable()` takes a single matrix or tibble, `mat`. By default, it returns a LaTeX macro that creates a well-formatted LaTeX table.

Usage

```
latexTable(
  mat,
  SE_table = TRUE,
  headerFooter = TRUE,
  commandName = "myTable",
  callCommand = TRUE,
  label = commandName,
  floatPlacement = "p",
  landscape = if (SE_table) ncol(mat)/2 >= 6 else ncol(mat) >= 6,
  starredFloat = FALSE,
  horizOffset = "-0in",
  rowNames = rownames(mat),
  footerRows = lt_footer(),
  colNames = lt_colNames_default(),
  colNameExpand = FALSE,
  extraRowHeight = if (SE_table) "2pt" else "4pt",
  spacerColumns = lt_spacerColumns_default(),
  spacerColumnsWidth = ".67em",
  spacerRows = NULL,
  spacerRowsHeight = ".15in",
  tabColSep = "2.75pt",
  spaceBetweenColNameRows = "-.025in",
  columnTierSeparator = "  ",
  printCaption = TRUE,
  caption = paste0(label, " caption goes here."),
  captionMargins = NULL,
  formatNumbers = TRUE,
  decimalPlaces = 2,
  SE_fontSizeString = "\\fontsize{10.3bp}{10.3bp}\\selectfont",
  NA_text = "",
  clipboard = FALSE
)
```

Arguments

`mat`	Matrix or tibble of numbers to be displayed in a LaTeX table.

`SE_table` Logical variable that indicates whether `mat` contains pairs (or "tiers") of columns, with the first column in each pair containing estimates, and the second column containing standard errors. Defaults to `TRUE`. If `TRUE`, the even-numbered columns of `mat` will be rendered in smaller type than the odd-numbered columns. That is, the standard errors will be rendered in smaller type than their corresponding estimates. This default type sizing can be overridden by the `SE_fontSizeString` argument.

`headerFooter` Logical variable. If `TRUE`, which is the default, the output will be (or at least include) a LaTeX macro that generates a table. For example, you will be able to produce a table simply by calling `\myTable{p}` or `\myTable{h}` in your master LaTeX document.

If `headerFooter` is `FALSE`, the only output of the function will be LaTeX code for "data rows"—one row for each row of `mat`.

`commandName` A string. It is the name of the macro that produces the LaTeX table (if `headerFooter` is `TRUE`). By default, it is "myTable"; you can change it to something more descriptive, e.g., "mainEstimates".

`callCommand` Logical variable. Should the last line of the `latexTable` object be a call to the macro that creates the table? If `callCommand` is `TRUE`, which is the default, sourcing a file that contains `latexTable` output—that is, by using `\input` or `\include` in your master LaTeX document—will produce a table when that master LaTeX document is rendered. If `callCommand` is `FALSE`, sourcing the file will make the macro available in your LaTeX document, but it won't call the macro. (You will need to call the macro yourself by adding a line like `\myTable{p}` to your LaTeX document.)

`label` A string. Specifies the LaTeX label for a table. It is not printed anywhere in the table, but references to the figure in your LaTeX document (for example, references created by `\ref` or `\autoref`) must include the label name. For simplicity, the default `label` is the same as the `commandName` argument.

`floatPlacement` Character vector of length 1. Acceptable values are `p` (the default, which places each table on its own page), `h,H,t,b,` and `!`. See the LaTeX wikibook for more on float placement in LaTeX.

`landscape` Logical variable. Determines whether the table is printed in landscape or in portrait mode. Affects the output only if if `headerFooter == TRUE` and `callCommand == TRUE`.

`starredFloat` Logical variable that indicates whether the LaTeX table should be specified with `table*` instead of `table`. The default is FALSE, but you may want to set it to TRUE if you want you are using a multi-column page layout in LaTeX and want the table to cross both columns.

`horizOffset` A string that specifies a LaTeX length, e.g., ".25in". When the LaTeX code produced by `latexTable` is rendered, the table will be moved to the right by this length (or to the left if the length is negative, e.g., "-.25in").

`rowNames` Character vector of labels for the rows in `mat`. The labels will be printed to the left of each row in `mat`. `rowNames` can be `NULL`.

`footerRows` List, object that can be coerced to a list, or a function that creates a list. Each element in the list is a character vector that specifies entries for a row of the footer, or a function that creates such a character vector. The default is `lt_footer()`, which typically provides a "Number of observations" row. If a model is of class "lm," it will also provide an R^2 and "Std. error of regression" row.

The first entry in each `footerRows` list-element should be the row name for the corresponding footer row (e.g., 'F', 'R^2').

See the examples for various ways to specify the `footerRows` argument.

`colNames` List, or object that can be coerced to a list, of column headings. Typically, each element in the list is a character vector, and the elements of the character vector specify the names of the table's columns.

If `SE_table` is `TRUE` (the default), each column name will appear over a pair of columns. In this case, each element in the `colNames` list should contain `ncol(mat)/2` entries.

To specify multi-line column labels, use a list with multiple elements. The entries in the first list element will then appear in the top row of the column label, the entries in the second list element will appear in the next row of the column label, and so on.

By default, column names will be taken from `colnames(mat)`. If `colnames(mat)` is `NULL`, columns will be numbered "(1)", "(2)", etc. See `lt_colNames_default()` for more information.

`colNameExpand` Logical variable. By default, an entry of '' in a `colNames` list element—that is, an empty entry—indicates that a column should have no column heading. But if `colNameExpand` is `TRUE` and a text entry in a `colNames` list element is followed by one or more '' entries, the column name specified by the text entry will bridge the columns that have '' entries.

`colNameExpand` and `spacerColumns` do not play well together. If you run `latexTable` with `colNameExpand == TRUE` and a non-NULL `spacerColumns` argument, you will get LaTeX output, but you will probably need to edit the "\multicolumn" and "\cmidrule" commands in the output so that LaTeX can render the output.

`extraRowHeight` A string that specifies a length that LaTeX recognizes, e.g., '2pt' or '.25in'. The `extrarowheight` length in LaTeX will be set to `extraRowHeight`. In practice, this means that the vertical space between every row will be increased by `extraRowHeight`. This argument has no effect if `headerFooter` is `FALSE`.

`spacerColumns` A vector of integers. Specifies columns in `mat` after which to insert columns that contain no entries. These "spacer columns" are used to insert horizontal space into the typeset table. By default, spacerColumns are specified by a helper function, `spacerColumns_default()`:

- If `SE_table` is `FALSE`, there is a spacer column between every column in `mat`.
- If `SE_table` is `TRUE`, there is a spacer column after every even-numbered column in `mat`, except for the last column.
- If `rowNames` is not `NULL`, a spacer column is inserted between the table's row names and the first column of data.

  To add a spacerColumn between the rownames and the first data column, make 0 one of the values in spacerColumns.

colNameExpand and spacerColumns do not play well together. If you run latexTable with colNameExpand == TRUE and a non-NULL spacerColumns argument, you will get LaTeX output, but you will probably need to edit the "\multicolumn" and "\cmidrule" commands in the output so that LaTeX can render the output.

See the end of the Building better tables in less time vignette for a technical note on spacerColumns and column spacing in LaTeX.

spacerColumnsWidth
: Either a single string of a recognizable LaTeX length (e.g., '.5em') or a character vector indicating the width of each spacer column. Has no effect unless headerFooter is TRUE.

spacerRows
: A vector of integers. After each row in mat whose number is in spacerRows, a vertical space of spacerRowsHeight will be printed. For example, if spacerRows == c(2,4), a vertical space will be added after rows 2 and 4 of mat.

spacerRowsHeight
: A string that specifies a recognizable LaTeX length, e.g., ".15in".

tabColSep
: Character vector indicating a length that LaTeX recognizes, e.g., ".25in". The tabcolsep value in LaTeX will be set to this value if headerFooter is TRUE. If SE_table is TRUE, tabColSep will be the default distance between the estimate and the SE column in each column pair, and it will be half of the distance between column pairs. If SE_table is FALSE, tabColSep will simply be half of the default distance between columns. These distances between columns can be increased by the spacerColumns argument.

spaceBetweenColNameRows
: String specifying a LaTeX length, e.g., "-.025in" (the default). When column names are to be split across multiple rows, a vertical space of this length will be inserted between the rows.

columnTierSeparator
: A string. In the LaTeX code generated by latexTable, all columns are separated from each other by " & ". Column tiers – that is, pairs of columns giving the estimate and the SE for a particular coefficient – are further separated by columnTierSeparator, which defaults to two spaces (' '). This option affects only the LaTeX code produced by latexTable; it exists to make the LaTeX code more readable. It does not affect the typeset (e.g., PDF) version of the table.

printCaption
: Logical variable.

caption
: A string. It can include LaTeX commands, e.g., "\\textitResults from a minimal specification." It can also include references to other labeled parts of your LaTeX document, e.g., "\\autorefSomeFigure". See the examples.

captionMargins
: A vector of two strings that specify the margins of the caption. The strings should be LaTeX lengths, e.g., ".25in" or ".67em". By default, captionMargins is NULL.

formatNumbers Logical variable, TRUE by default. Pretty-print the entries in mat by:

- Rounding all entries to the decimalPlaces digit.
- Padding out entries with trailing zeroes. For example, 3 will become 3.00 if decimalPlaces == 2.
- Removing leading zeroes. For example, 0.12 will become .12.

If formatNumbers is FALSE, formatting of entries in mat will be handled by print.default().

decimalPlaces Integer. If formatNumbers is TRUE, table entries will be shown to this decimal place. For example, if decimalPlaces==2, both "3.0035" and "3" will become "3.00."

If formatNumbers is FALSE, entries will not be adjusted, but decimalPlaces will still be used to determine the widths of columns and some aspects of column spacing. For example, even if formatNumbers is FALSE, data columns will be wider when decimalPlaces is 10 than when it is 2.

SE_fontSizeString A string. Indicates how standard errors are to be formatted when SE_table is TRUE. Defaults to \\fontsize{10.3bp}{10.3bp}\\selectfont, which renders standard errors in slightly smaller type than the corresponding estimates.

NA_text A string. NA entries in mat will be replaced by the string.

clipboard Logical variable. Copy entire output to clipboard. Useful if you want to paste the output directly into a .tex file. Works only on Windows.

Value

An object of classes latexTable and character. The returned object is a vector of strings of LaTeX code; each string is a line in a LaTeX macro that can create a table.

There is one small exception. If callCommand is TRUE, the last line is not part of the macro; instead, it calls the macro, thereby telling LaTeX to display the table. For example, if commandName and label are myTable, and if callCommand is TRUE, the last line of the returned object is \mytable{p}.

Note

Required LaTeX packages. The LaTeX code produced by the latexTable makes use of capabilities provided by the booktabs, caption, float, numprint, and ragged2e LaTeX packages—and, for landscaped tables, the pdflscape package. If you haven't installed those LaTeX packages, you won't be able to render the tables produced by latexTable.

The LaTeX code produced by latexTable also makes use of capabilities provided by the array and afterpage LaTeX packages—but these packages are included in every LaTeX distribution.

Changes from pre-release versions:

- The names of some arguments have changed slightly since the pre-release versions of this function. They have been changed to enforce consistency: camelCase is used for all arguments, and every acronym is followed by an underscore (_) character. We thus have SE_table instead of SEtable, tabColSep instead of tabcolsep, and so on.

- The hspace argument has been renamed to horizOffset.
- Some default arguments have changed. In particular, the default spacerColumns argument is no longer NULL. Instead, the default is to insert spacer columns in appropriate places. See documentation of the spacerColumns argument for details.

See Also

Other functions for making tables: regTable(), latexTablePDF(). See also the Building better tables in less time and Using latexTable() with R Markdown and Rnw documents vignettes.

Examples

```
data(iris)
lm1 <- lm(Sepal.Length ~ Petal.Length,                data = iris)
lm2 <- lm(Sepal.Length ~ Petal.Length + Petal.Width, data = iris)
lm3 <- lm(Sepal.Length ~ Petal.Length * Petal.Width, data = iris)
rT1 <- regTable(list(lm1, lm2, lm3))
latexTable(rT1)
latexTable(rT1, SE_table = FALSE, colNames = lt_colNumbers())
lt2 <- latexTable(
  mat      = rT1,
  colNames = list(qw("model model model"), qw("1 2 3")))
## Not run:
lt3 <- latexTable(
  mat         = rT1,
  colNames    = lt_colNumbers(),
  rowNames    = c(
    "Intercept",
    "Petal length",
    "Petal width",
    "Petal length $\\times$ petal width"),
  footerRows  = list(lt_nobsRow(), lt_rSquaredRow()),
  commandName = 'mainEstimates',
 caption    = "Each entry is an estimate or a standard error from a separate OLS regression.")
lt4 <- update(
  lt3,
  commandName = 'myEstimates',  # change name of LaTeX macro
  spacerRows  = 1,              # add vertical space after intercept row
  footerRows  = list(
    c("My first footer row",  "a",     "b",     "c"),
    c("My second footer row", "Lorem", "ipsum", "dolor")
  ))

## End(Not run)

# You can pass a previously created list to the "footerRows" argument. To
# store functions in that list, use alist():
footerList <- alist(lt_rSquaredRow, lt_SER_row)
latexTable(rT1, footerRows = footerList)
```

---

latexTablePDF *Renders a LaTeX table as a PDF file.*

Description

`latexTablePDF()` takes an object produced by `latexTable()` and writes a PDF file. It can also write the corresponding .tex file.

Usage

```
latexTablePDF(
  latexTable,
  container = TRUE,
  containerFilename = "tableContainer.tex",
  outputFilenameStem = "latexTable",
  writePDF = TRUE,
  writeTex = FALSE,
  overwriteExisting = FALSE,
  verbose = FALSE,
  continuedFloat = FALSE,
  continuedFloatStar = FALSE,
  firstPageEmpty = container,
  firstTableNumber = NULL,
  openPDFOnExit = TRUE
)
```

Arguments

`latexTable`	Object of class `latexTable` (typically produced by `latexTable()`) or a list of such objects.
`container`	Logical variable. Should the LaTeX code in `latexTable` be inserted into a LaTeX file that contains a complete LaTeX preamble and the `\begin{document}` and `\end{document}` tags? If you want `latexTablePDF` to produce a PDF file, `container` must be `TRUE`. But if you just want to write `latexTable` to disk as a .tex file that you will insert into your own larger LaTeX document, `container` should probably be `FALSE`.
`containerFilename`	A string. Specifies the path of the "container" LaTeX file into which the `latexTable` table or tables will be inserted to make a complete LaTeX file that can be rendered as PDF. By default, it is "tableContainer.tex", which is included in this package.
`outputFilenameStem`	A string. It is the path and name of the file is to be saved to disk, up to the extension. For example, if you want to save "myTable.pdf" to disk, set `outputFilenameStem = "mytable"`.
`writePDF`	Logical variable. Should a PDF file be saved to disk?
`writeTex`	Logical variable. Should a .tex file be saved to disk?
`overwriteExisting`	Logical variable. Should files to be saved overwrite existing files that have the same names?
`verbose`	Logical variable. `latexTablePDF` calls `pdflatex` to render PDF files, and if `verbose` is `TRUE`, all of the output from `pdflatex` will be printed to screen. Useful for debugging.

`continuedFloat` Logical variable. Should be `TRUE` if the table or tables to be rendered are part of a series and should all share the same number. For example, `continuedFloat` should be true if `latexTable` is a list of tables and you all want them to be numbered as Table 3.

`continuedFloatStar`
Logical variable. Should be `TRUE` if the table or tables to be rendered are part of a series, and if all tables should share the same number but be distinguished by some secondary character. For example, `continuedFloatStar` should be true if `latexTable` is a list of tables and you all want them to be numbered as Table 3a, Table 3b, etc.

`continuedFloat` and `continuedFloatStar` cannot both be `TRUE`.

`firstPageEmpty` Logical variable. Defaults to the value of `container`. If `TRUE`, the page that contains the first table in `latexTable` will have an empty header and footer. (Technically, `latexTablePDF` will insert `\thispagestyle{empty}` into the code block that contains the first table in `latexTable`.)

`firstTableNumber`
Integer. What number should the first table in `latexTable` have?

By default, the table numbering will be "natural." That is, the number of the first `latexTable` table will be determined by the number of preceding tables in the document. For example, if you use `latexTablePDF()` to create a .tex file that you then insert into a larger LaTeX document, and if the .tex file is preceded in the LaTeX document by two tables, the first table created by `latexTable()` will be Table 3.

`openPDFOnExit` Logical variable. Open the PDF file after it is created by `latexTablePDF`? This argument has an effect only on Windows, and only if `writePDF` is `TRUE`.

Details

Although `latexTablePDF` produces PDF files by default, it is also useful for creating .tex files. For example, you may have a list of tables produced by `latexTable()` and a complex LaTeX document that contains many different sections and tables. When `latexTablePDF` is used with `writeTex = TRUE`, it will produce a single file that contains LaTeX code for all of the tables in your list. You can then insert those tables into your LaTeX document by adding a single `\input` or `\include` command to your LaTeX document. For details, see `vignette("tables", package = "Bullock")`.

Note

Required LaTeX tools. If `writePDF` is `FALSE`, you do not need to have LaTeX installed. But if `writePDF` is `TRUE`:

- `pdflatex` must be installed on your system. It is part of almost every LaTeX installation. The `array` and `afterpage` packages must also be installed; if you have a LaTeX installation, you already have these packages.
- The booktabs, caption, and `numprint` must be installed.

- If `containerFilename` is "tableContainer.tex" (the default), the fancyhdr, `footmisc`, geometry, and `ragged2e` packages must be installed.
- If you are producing a landscaped table, the `pdflscape` package must be installed.
- If your `latexTable()` output was produced with `floatPlacement = 'H'`, the `float` package must be installed.

See Also

Other functions for making tables: `latexTable()`, `regTable()`. See also the Building better tables in less time and Using `latexTable()` with R Markdown and Rnw documents vignettes.

Examples

```
## Not run:
  data(iris)
  lm1 <- lm(Sepal.Length ~ Petal.Length,               data = iris)
  lm2 <- lm(Sepal.Length ~ Petal.Length + Petal.Width, data = iris)
  lm3 <- lm(Sepal.Length ~ Petal.Length * Petal.Width, data = iris)
  lmList <- list(lm1, lm2, lm3)
  rT1 <- regTable(lmList)
  lT1 <- latexTable(
    mat        = rT1,
    colNames   = lt_colNumbers(),
    rowNames   = c(
      "Intercept",
      "Petal length",
      "Petal width",
      "Petal length $\\times$ petal width"),
    footerRows = list(lt_nobsRow(), lt_rSquaredRow()),
    spacerRows = 1,  # insert white space between Intercept row and other rows
    caption    = paste0(
      "\\textit{Sepal length as a function of petal length and petal width.} ",
      "Entries are estimates and standard errors from OLS regressions..."
    )
  )
  latexTablePDF(lT1, outputFilenameStem = "irisData")


  # Create a PDF or .tex file that contains two tables:
  lm1v <- update(lm1, subset = (Species == 'versicolor'))
  lm2v <- update(lm2, subset = (Species == 'versicolor'))
  lm3v <- update(lm3, subset = (Species == 'versicolor'))
  rT2  <- regTable(list(lm1v, lm2v, lm3v))
  lT2  <- update(lT1, mat = rT2, commandName = "tableVersicolor")

  latexTablePDF(                              # PDF with two pages
    lT2,
    outputFilenameStem = "irisData_twoTables")
  latexTablePDF(                              # add .tex file with code for two tables
    lT2,
    outputFilenameStem = "irisData_twoTables",
    writeTex           = TRUE)

## End(Not run)
```

lt_colNames_default *Compute default column names for latexTable objects.*

Description

If `colnames(mat)` is not `NULL`, this function will use `colnames(mat)` as the `colNames` argument in `latexTable()`. If `colnames(mat)` is `NULL`, column names will be determined by `lt_colNumbers()`.

Usage

```
lt_colNames_default(
  mat = parent.frame()$mat,
  SE_table = parent.frame()$SE_table
)
```

Arguments

`mat`	A matrix, typically a `regTable` object.
`SE_table`	Logical variable. See `latexTable()`.

Details

The function is not exported and is intended to be called only by `latexTable()`.

Value

A vector of strings. Each string is a column name.

lt_colNumbers *Automatically determine column names of the form (1), (2), etc.*

Description

Given `mat` and `SE_table`, this function determines appropriate column-number names of the form "(1)", "(2)", etc.

Usage

```
lt_colNumbers(mat = parent.frame()$mat, SE_table = parent.frame()$SE_table)
```

Arguments

`mat`	A matrix, typically a `regTable` object.
`SE_table`	Logical variable. See `latexTable()`.

Value

A vector of strings. If `SE_table` is `TRUE`, the vector elements are "(1)", "(2)", etc., where the last column number is `ncol(mat)/2`. If `SE_table` is `FALSE`, the vector elements are "(1)", "(2)", etc., where the last column number is simply `ncol(mat)`.

lt_footer *Compute default footers for latexTable() objects.*

Description

The default footer row or rows are determined in this way: if SE_table is FALSE or rowNames is NULL, no footer rows are produced. Otherwise, a footer row will be added for each of the following attributes of mat: "r.squared", "SER", and "N". If mat lacks one of those attributes, there will be no corresponding footer row. The function is not exported and is intended to be called only by latexTable().

Usage

```
lt_footer(
  mat = parent.frame()$mat,
  rowNames = parent.frame()$rowNames,
  SE_table = parent.frame()$SE_table,
  decimalPlaces = parent.frame()$decimalPlaces
)
```

Arguments

mat	A matrix, typically a regTable object.
rowNames	Character vector. See latexTable().
SE_table	Logical variable. See latexTable().
decimalPlaces	Integer. See See latexTable().

Value

A list of string vectors. Each list element contains the strings needed to produce a footer row.

lt_nobsRow *Specify a footer row that indicates the number of observations for each regression.*

Description

Given a mat produced by regTable(), this function returns a footer row that indicates the number of observations for each model in mat.

Usage

```
lt_nobsRow(mat = parent.frame()$mat)
```

Arguments

mat	A regTable object.

Value

A vector of strings. The first element in the vector is "Number of observations". The remaining elements are the numbers of observations for each regression in `mat`.

`lt_rSquaredRow` *Specify a footer row that indicates R^2 for each regression.*

Description

Given a `mat` produced by `regTable()` in which all regressions are of class `lm`, this function returns a footer row that indicates R^2 for each model in `mat`.

Usage

```
lt_rSquaredRow(
  mat = parent.frame()$mat,
  decimalPlaces = parent.frame()$decimalPlaces
)
```

Arguments

`mat`	A matrix, typically a `regTable` object.
`decimalPlaces`	Integer. See `latexTable`.

Value

A vector of strings. The first element in the vector is "R^2". The remaining elements are strings that indicate R^2 for each model in `mat`. The strings are rounded to the number of digits specified by the `decimalPlaces` argument.

`lt_SER_row` *Specify a footer row that indicates the standard error of regression for each model*

Description

Given a `mat` produced by `regTable()` in which all regressions are of class `lm`, this function returns a footer row that indicates the standard error of regression (i.e., σ, the "residual standard error") for each model in `mat`.

Usage

```
lt_SER_row(
  mat = parent.frame()$mat,
  decimalPlaces = parent.frame()$decimalPlaces
)
```

Arguments

mat	A matrix, typically a regTable object.
decimalPlaces	Integer. See latexTable.

Value

A vector of strings. The first element in the vector is "Std. error of regression". The remaining elements are strings that indicate the SER for each model in mat. The strings are rounded to the number of digits specified by the decimalPlaces argument.

---

lt_spacerColumns_default

Compute default positions of spacer columns in calls to latexTable().

Description

spacerColumns_default specifies the default spacerColumns argument in calls to latexTable. It takes the values of mat, SE_table, and rowNames that are passed to latexTable. From these values, it computes the default positions of spacer columns:

- If SE_table is FALSE, there is a spacer column between every column in mat.
- If SE_table is TRUE, there is a spacer column after every even-numbered column in mat, except for the last column.
- If rowNames is not NULL, a spacer column is inserted between the table's row names and the first column of data.

Usage

```
lt_spacerColumns_default(
  mat = parent.frame()$mat,
  SE_table = parent.frame()$SE_table,
  rowNames = parent.frame()$rowNames
)
```

Arguments

mat	Matrix.
SE_table	Logical variable.
rowNames	Vector.

Details

The function is not exported and is intended to be called only by latexTable().

Value

A vector of integers.

makePercentage *Convert a number or a string to a percentage.*

Description

Takes an object x of class "character" or "numeric". Returns an object of class "character" in which the number is expressed as a percentage.

Usage

```
makePercentage(x, dp = 0)
```

Arguments

x	Object of class "character," "integer," or "numeric."
dp	Integer specifying the number of decimal places in the returned string.

Details

By default, the returned string will have no decimal digits: it will be "57%", not "57.0%". But if dp is greater than 0, the returned string will always have dp decimal places.

Value

A string, e.g., "57%".

Examples

```
makePercentage(.5)            # "50%"
makePercentage(.555)          # "56%"
makePercentage(.555, dp = 4)  # "55.5000%"
makePercentage(5)             # "500%"
makePercentage(5, dp = 2)     # "500.00%"

makePercentage(1:3)           # "100%" "200%" "300%"
```

mergeFac *Fill in missing values of one factor with corresponding values from another.*

Description

Given a factor of length n that contains some missing values, fill in the missing values with the corresponding values of others factors.

Usage

```
mergeFac(x, ...)
```

Arguments

x	Factor variable.
...	Other factor variables.

Details

x and the factors named in otherFactors must all have the same length. If they do, missing values in x will be filled in with the corresponding values of the first factor in otherFactors. If the corresponding values of that factor are also missing, mergeFac() will look to the corresponding values of the next factor, and so on.

Note

mergeFac() is deprecated. You should use dplyr::coalesce() instead. No problems with mergeFac() have been reported, but dplyr::coalesce() does all that mergeFac() does, is more powerful, and will be just as well maintained.

Merging factors as mergeFac() does is trickier than just using a command like fac1[is.na(fac1)] <-fac2[is.na(fac1)] because fac1 and fac2 may have different factor levels. This command takes care of the problem by merging the levels among different factors.

Examples

```
fac1 <- factor(c("a", NA,  "b", NA,  NA))
fac2 <- factor(c("y", "y", "y", NA,  NA))
fac3 <- factor(c(NA,  "z", "z", "z", NA))
mergeFac(fac1, fac2)        # [1] a    y    b    <NA> <NA>
mergeFac(fac1, fac2, fac3)  # [1] a    y    b    z    <NA>
```

---

missingPackageString *Checks for existence of required LaTeX packages.*

Description

The code produced by latexTable() can be rendered only if certain LaTeX packages are installed. (See the note at the end of the latexTablePDF() help file for details.) If packages are missing, this function generates an informative string that can be used in warning or error messages.

Usage

```
missingPackageString(
  installedPackageList,
  requiredPackageList,
  writePDF,
  writeTex
)
```

Arguments

installedPackageList, requiredPackageList	Character vectors.
writePDF, writeTex	Logical variables. See the latexTable() documentation for further information about these arguments.

Details

This function is not exported. It is called by `latexTablePDF()` only if packages are missing. It generates either an error (if `writePDF` is `TRUE`) or a warning (if `writePDF` is `FALSE` but `writeTex` is `TRUE`).

`missingValueFunctions` *Missing-value helper functions.*

Description

Functions to make code a little clearer. These are mainly ordinary functions, like `mean()`, with `na.rm` set to `TRUE`. For example, `meanNA()` is defined as `function(x) mean(x,na.rm = TRUE)`.

Usage

```
allNA(x)

lNA(x, verbose = FALSE)

lNAv(x)

maxNA(x)

meanNA(x)

medianNA(x)

minNA(x)

rangeNA(x)

sdNA(x)

sumNA(x)

varNA(x)
```

Arguments

`x`	An R object.
`verbose`	Logical variable. If `TRUE`, `lNA` will print the lengths of `x` before and after `NA` values have been removed.

Details

`lNA(x)` returns its value silently. `lNAv` is shorthand for `lNA(x,verbose = TRUE)`; it returns the same value as `lNA(x)` but also prints the lengths of the vector before and after NAs are removed.

Examples

```
lNA(NA)    # 0

x <- c(1:3, NA, 5)
allNA(x <= 5)  # TRUE
lNA(x)         # 4
lNAv(x)        # 4 (length with NA=5)

sum(NA)        # NA
sumNA(NA)      # 0

sum(x)         # NA
sumNA(x)       # 11
meanNA(x)      # 2.75

sdNA(x)
varNA(x)
```

`modalValue` *Find modal value of a vector.*

Description

Find modal value of a vector. If there are multiple modal values, all will be returned.

Usage

```
modalValue(x, na.rm = FALSE)
```

Arguments

`x`	Vector.
`na.rm`	Logical variable.

Details

Before this package was released, it returned only the first mode if there were multiple modes. It now returns all modes. See the examples.

Author(s)

Ken Williams. See http://stackoverflow.com/a/8189441/697473.

Examples

```
modalValue(qw("a b b"))      # [1] "b"
modalValue(qw("a a b b c"))  # [1] "a" "b"
modalValue(1:3)              # [1] 1 2 3
```

moveToDF — *Move or copy "freestanding" variables into a data frame.*

Description

Variables are specified by `pattern`, which is a regular expression. The modal length of all variables in the calling environment that match `pattern` is determined. Matching variables are then moved to a data frame (or copied to a data frame if `move` is `FALSE`). `pattern` may be `NULL`, in which case all variables in the calling environment will be examined.

Usage

```
moveToDF(pattern = NULL, move = TRUE)
```

Arguments

`pattern`	String that specifies a regular expression. It is `NULL` by default, in which case all variables in the environment are examined for inclusion in the data frame.
`move`	Logical variable.

Details

If there are multiple modal lengths of the objects in the calling environment, all modes will be used. For example, if the calling environment has 20 objects of length 1, 20 objects of length 2, and one object of length 3, the returned data frame may have as many as 40 columns.

Variables that have the "dim" attribute – for example, arrays and matrices – will not be moved into the new data frame. Functions will never be moved into the new data frame, either.

Value

Data frame containing all one-dimensional variables that have names matching `pattern` and that have the modal length of those variables.

PDF_crop — *Crop white margins of a PDF file.*

Description

`PDF_crop()` removes ("crops") the white margins around a PDF. It requires that pdfcrop.exe be installed and in your path. To get pdfcrop.exe, install MikTeX or Tex Live, or just use `tinytex::tlmgr_install('pdfcro`

Usage

```
PDF_crop(
  input,
  dirOutput = NULL,
  outputExtension = "_crop",
  deleteOriginal = TRUE,
  openCropped = FALSE,
  verbose = FALSE
)
```

Arguments

input	A string. Path and name of file to be cropped. If input does not end with ".pdf", ".pdf" will be appended to it. This is the only required argument.
dirOutput	A string. Path of directory into which cropped file should be placed.
outputExtension	A string. The new file will have the same name as the old file, except that outputExtension will be inserted just before the ".pdf" extension.
deleteOriginal	Logical, defaulting to TRUE. Should the original file be deleted?
openCropped	Logical, defaulting to FALSE. After the new file is produced, should it be opened in your default PDF viewer? Ignored on non-Windows, non-Macintosh systems.
verbose	Logical, defaulting to FALSE. If it is TRUE, two lines of information from pdfcrop.exe will be printed. They will show your version of pdfcrop.exe, the number of pages in the file that it produced, and the path and filename of the file that it produced.

See Also

knitr::plot_crop(), which is less flexible in its handling of PDF files but can handle other file formats.

Examples

```
## Not run:
tmpFile <- tempfile(fileext = ".pdf")
pdf(tmpFile)
  plot(1:10, 1:10)
dev.off()
PDF_crop(tmpFile)

## End(Not run)
```

---

print.regTable *Print regTable objects.*

Description

print.regTable() is the default print method for objects produced by regTable(). The output that it produces differs from print.table() output because print.regTable()

- right-aligns row names;
- inserts one space between each estimate and SE column, but two spaces between estimate-SE column tiers, thereby making it easier to distinguish between results from different regressions;
- uses two rows of column headings: one to show the outcome for each regression, and another to distinguish columns of coefficient estimates from columns of SE estimates;
- does not print attributes (e.g., "N", "r.squared").

Usage

```
## S3 method for class 'regTable'
print(x, decimalPlaces = getOption("Bullock.print.regTable.dp"), ...)
```

Arguments

x	regTable object.
decimalPlaces	Integer. Table entries will be rounded to this number of digits after the decimal point. By default, it is the value of `getOption("Bullock.print.regTable.dp")`, which is set to 2 when the Bullock package is loaded. You can change the default by running `options("Bullock.print.regTable.dp") <-N` at any time, where `N` is your preferred number of digits after the decimal place.
...	Arguments passed to `print.table()`.

---

printAll *Print all rows of a tibble.*

Description

By default, tibbles—the default data frames created by tidyverse packages—print only as many rows as your console window can display. `printAll()` instead prints all of the tibble's rows. If it is called on an object other than a tibble, it just calls `print()`.

Usage

```
printAll(x)
```

Arguments

x	Object, perhaps of class "tbl_df"

Examples

```
x <- tibble::tibble(a = 1:100, b = 101:200)
printAll(x)
```

---

qw *Perl-like function for quoting a list of words*

Description

`qw` stands for *quote words*. The function takes a string of words separated by whitespace characters. It returns a vector in which each element is a word. The point of the function is to speed the creation of vectors of words and to make for more readable code.

Usage

```
qw(x)
```

Arguments

x	A string. It may contain newline characters; `x` will be split by these characters just as it will be split by ordinary spaces. See the examples.

Details

This function is an R implementation of Perl's qw() operator. Sadly, the operator has no equivalent in Python.

Value

Character vector.

Author(s)

Florent Delmotte ("flodel"). See http://stackoverflow.com/questions/520810/.

See Also

https://perldoc.perl.org/5.30.0/perlop.html#Quote-Like-Operators.

Examples

```
qw("You can type    text
    with    line breaks if you
    wish")
#  [1] "You"        "can"        "type"       "text"
#  [5] "here"       "with"       "linebreaks" "if"
#  [9] "you"        "wish"
```

---

regTable *Create a matrix of regression output from a list of regression models.*

Description

regTable() takes a list of regression objects, such as those created by lm(). It returns a matrix in which the columns are estimates and standard errors – two columns for each model. Together, the two columns that represent a regression are a *column-pair* or a *column-tier.*

Usage

```
regTable(
  objList,
  colNames = NULL,
  rowsToRemove = NULL,
  rowsToKeep = NULL,
  clusterVar = NULL
)
```

Arguments

objList	List of regression objects. This is the only required argument. regTable() has been tested with objects of classes ivreg,glm,lm, and plm, and with the objects produced by the estimatr package. It should work with other regression objects, too – that is, with any regression objects for which the coef() and vcov() functions can be used.

`colNames` A vector of strings as long as `length(objList)`.

`rowsToRemove` A vector of strings, which may specify regular expressions. Variables in the regressions whose names match the strings will be omitted from the `regTable` output. This argument overrides `rowsToKeep`.

`rowsToKeep` A vector of strings, which may specify regular expressions. Variables in the regressions whose names match the strings will be kept in the `regTable` output. All other variables will be omitted. If `rowsToRemove` is specified, this argument has no effect.

`clusterVar` Either `NULL`, a list of length 1, or a list of length `length(objList)`. If `NULL` (the default), `regTable()` will report the standard errors indicated by your regression objects. Otherwise, the argument to `clusterVar` should indicate the clusters for the corresponding regression objects in `objList`. Clustered SEs for "lm" objects will be produced by `multiwayvcov::cluster.vcov`, and clustered SEs for "ivreg" objects will be produced by `ivpack::cluster.robust.se`. `regTable()` does not support clustering for other kinds of regression objects.

Value

A matrix in which the columns are estimates and standard errors – two columns for each model. The matrix has an "N" attribute that indicates the number of observations for each regression. If all regressions were of class `lm` (but not also class `glm`), it also has the "r.squared" and "SER" attributes. (The "SER" attribute indicates the standard error of regression – AKA σ or the "residual standard error" — for each model.)

Subsetting regTable objects with the `[` operator

You can take subsets of regTable objects with the `[` operator. It subsets intelligently. That is, it knows whether the subsetted object contains only intact column tiers, and it modifies the class and attributes of the subsetted object accordingly. To wit:

- If you remove only certain column-pairs from your original regTable object, such that the remaining columns all form intact column-pairs, all attributes for the remaining column-tiers are preserved. For example, the `N` and `r.squared` attributes are preserved. In other words, the `[` operator knows which regressions you have removed from the table, and it removes attribute information for only those regressions.
- If you remove only rows from your original regTable object, all attributes are preserved. This result has the potential to be misleading: for example, you may remove a row that reports information for a given predictor, but the `N`, `r.squared`, and `SER` attributes were all computed from regressions that included that predictor. Consequently, a message will be printed to remind you that the attributes have been preserved.
- If you remove columns from your original regTable object such that the remaining columns do *not* all form intact column-pairs, the `N`, `r.squared`, and `SER` attributes are stripped, and the returned object is a matrix without the "regTable" class.

See the examples for an illustration.

Change from the pre-publication version

Before `regTable()` was incorporated into this package, it used the `rowsToKeep` argument differently: variables were kept only if the *beginnings* of their names matched the strings in `rowsToKeep`.

See Also

Other functions for making tables: `latexTable()`, `latexTablePDF()`. See also the Building better tables in less time vignette.

Examples

```
data(iris)
lm1 <- lm(Sepal.Length ~ Petal.Length,                data = iris)
lm2 <- lm(Sepal.Length ~ Petal.Length + Petal.Width, data = iris)
regTable(list(lm1, lm2))
regTable(list(lm1, lm2), colNames = c("Sepal length", "Sepal width"))
regTable(list(lm1, lm2), rowsToKeep = 'Length')
regTable(list(lm1, lm2), rowsToKeep = c('Intercept', 'Length'))
regTable(list(lm1, lm2), clusterVar = list(iris$Species))

# illustrate subsetting
rT <- regTable(list(lm1, lm2))
ncol(rT)           # 4
attributes(rT)$N   # 150 150
rT2 <- rT[1:2, ]
attributes(rT2)$N  # 150 150
rT3 <- rT[, 3:4]
attributes(rT3)$N  # 150
rT4 <- rT[, 2:3]
attributes(rT4)$N  # NULL
class(rT4)         # "matrix"
```

`reliability` *Compute and report Cronbach's alpha*

Description

Given a matrix `x` in which every column represents a variable, this function reports (a) the estimated reliability (Cronbach's alpha) for the battery of `n` variables, and (b) the estimated reliability for each possible combination of `n -1` variables. If `x` contains missing values, listwise deletion will be done via `na.omit(x)` so that alpha can be calculated.

Usage

```
reliability(x, ...)
```

Arguments

`x`	Matrix of numbers
`...`	additional arguments passed to `var()`

Value

The function prints output to the screen and invisibly returns a list with five elements:

- `nrowBeforeListwiseDeletion` is the number of rows of `x`.
- `nrowAfterListwiseDeletion` is the number of rows of `x` after listwise deletion.

- alpha is Cronbach's alpha for x after listwise deletion.
- alphaSmall is a vector of ncol(x) numbers. It shows what Cronbach's alpha is when we remove one variable at a time from na.omit(x).
- alphaSmallAfterListwiseDeletion is a vector of ncol(x) numbers. It shows what Cronbach's alpha is when we remove one variable at a time from x and then use listwise deletion on the new matrix.

Author(s)

Peter Ellis

John G. Bullock

Examples

```
data(iris)
reliability(iris[, 1:4])
```

---

rescale *Rescale a vector to have a specified minimum and maximum.*

Description

Rescale a vector to have a specified minimum and maximum.

Usage

```
rescale(x, newRange = c(0, 1))
```

Arguments

x	Numeric object.
newRange	Numeric vector of length 2.

Author(s)

Simon D. Jackman

See Also

scale() and scales::rescale()

Examples

```
vec <- 1:10
rescale(vec, c(2, 5))
```

snakeToCamel *Transform a string from snake_case to camelCase.*

Description

Given a string like "mean_attitude", the function returns "meanAttitude". But for any given string segment involving an underscore character, the the segment will be transformed only if the character before the underscore is a lower-case letter. For example, "MEAN_attitude" won't be transformed. Nor will "PID3_early."

Usage

```
snakeToCamel(x)
```

Arguments

x	A string.

See Also

spaceToCamel()

Examples

```
snakeToCamel("party_ID")      # [1] "partyID"
snakeToCamel("hours_9_to_5")  # [1] "hours9_to_5"
```

sourcing *Detect whether a file is being sourced from another file.*

Description

sourcing() returns TRUE if it seems that it is called from a file that is being sourced, e.g., with source() or sys.source(). Otherwise, it returns FALSE. The command is helpful when we want a file to behave differently depending on whether it is sourced.

Usage

```
sourcing()
```

Details

sourcing() is just an alias for sys.nframe() != 1L.

See Also

interactive(), sys.nframe()

The function was inspired by inspired by interactive(), which detects whether a command is being run interactively. Stack Overflow user r2evans wrote a helpful post: see https://stackoverflow.com/a/47932989/697473.

Examples

```
if (sourcing()) {
  print("It seems that this code is being sourced. It's not being
    executed directly from the global environment.")
} else {
  print("It seems that this code is not being sourced. It's being executed
    directly from the global environment.")
}
```

---

spaceToCamel *Transform a string that has spaces to camelCase.*

Description

Given a string like "mean attitude", the function returns "meanAttitude". But if the character just before the space is uppercase, the space will be replaced by an underscore (_): "TV news" will become "TV_news".

Usage

```
spaceToCamel(x)
```

Arguments

x	A string.

See Also

snakeToCamel()

Examples

```
spaceToCamel("colleges and universities")  # [1] "collegesAndUniversities"
spaceToCamel("TV news")                    # [1] "TV_news"
spaceToCamel("TV News")                    # [1] "TV_News"
```

---

stackUtilities *Perl-like stack utilities for R.*

Description

You can "pop" data off the end of a vector, one element at a time, with pop(). You can also "push" new data onto the end of the vector with push(). The analogous functions for working at the start of a vector are shift() and unshift(): shift() removes the first element of your vector, and unshift() prepends new elements to your vector.

Usage

```
push(x, values)

pop(x)

unshift(x, values)

shift(x)
```

Arguments

x	Object, typically a vector or a list.
values	Object to be added to x.

Details

An important and unusual feature of push(),pop(),shift(), and unshift() is that they modify objects "in place"—that is, even when no explicit assignment is done. For example, pop(x) will return the last value of x, but it will also remove the last value from the x object. The examples illustrate this point.

These functions are adapted from Matt Pettis's code at https://gist.github.com/mpettis/b7bfeff282e3b052684f.

Previous versions of these functions were adapted from Jeffrey A. Ryan's code at http://www.lemnica.com/esotericR/Introducing-Closures/. That code works but is based on the creation of "stack" objects that contain their own environments. One consequence is that changing a copy of a stack object changes the original stack object, and vice versa. Note too that, in Ryan's code, the traditional meanings of shift() and unshift() are reversed: he uses shift() to concatenate objects, unshift() to remove a value from an object.

Value

pop() and shift() will return a scalar, that is, an object of length 1. push() and unshift() don't return anything.

Author(s)

Matt Pettis

John G. Bullock

See Also

Thomas Leeper's Gist describes his own implementation of pop() and push() and includes links to six other implementations. Some of these implementations of pop() and push() do not have the modify-in-place characteristic of the corresponding Perl functions. This quality is also absent from the constructor function at https://stackoverflow.com/a/14489296/697473.

Examples

```
myStack <- 1:3
push(myStack, 4)
myStack  # [1] 1 2 3 4
```

```
pop(myStack)    # [1] 4
shift(myStack)  # [1] 1
myStack         # [1] 2 3

unshift(myStack, "hello")
myStack         # [1] "hello" "2" "3"

myList <- list(1, 2, 3)  # list with three elements
push(myList, 4)
myList                   # list with 4 elements
shift(myList)            # returns 1
myList[[1]]              # first element of myList is now 2
```

`strip0` *Strip "0" from beginning of numbers printed by R.*

Description

Takes an object `x` of class "character" or "numeric". Returns an object of class "character" in which the leading zero has been removed. By default, it also rounds the number-string that it returns to two decimal places. Example: both 0.3445 and "0.3445" are transformed to ".34".

Usage

```
strip0(x, dp = 2)
```

Arguments

`x`	Object of class "character" or "numeric."
`dp`	Integer specifying the number of decimal places in the returned string. Can be NULL, in which case no rounding is performed.

Details

If `x` has no leading zero, the returned object is still rounded to `dp` decimal places.

Examples

```
strip0("0.346")  #  ".35"
strip0(0.346)    #  ".35"
strip0(.346)     #  ".35"
strip0(5.346)    # "5.35"

strip0(c("0.789", ".2346", 53.3, 53.346))  # c(".79", ".23", "53.3", "53.35")
```

suppress_warnings *Suppress specific warning messages*

Description

Sometimes R throws warning messages that we don't want to see. The base `suppressWarnings()` function permits one to suppress warnings, but it is tricky to selectively suppress only certain warnings on the basis of a regular expression or another condition. This function allows one to do that.

Usage

```
suppress_warnings(.expr, .f, ...)
```

Arguments

`.expr`	Expression to be evaluated.
`.f`	String or function. If a string (possibly representing a regular expression), any warning message generated when `.expr` is evaluated will be suppressed if `grepl{}` finds that the string matches the warning message. If a function, the warning message will be passed to the function, and the function must return `TRUE` or `FALSE`. See the examples for details.
`...`	Additional arguments to be passed to `rlang::as_function()`.

Note

Most functions in the Bullock package have camelCase names. This one does not: it would ordinarily have been called `suppressWarnings`, but that name is taken by a function in base R.

Author(s)

The function was created by Antoine Fabri ("Moody_Mudskipper"): see https://stackoverflow.com/a/55182432/697473.

Examples

```
suppress_warnings( {sqrt(-1); warning("oops", call. = FALSE)}, startsWith, "o" )
  # Warning message:
  # In sqrt(-1) : NaNs produced
suppress_warnings( {sqrt(-1); warning("oops", call. = FALSE)}, ~nchar(.)>10 )
  # Warning message:
  # oops
suppress_warnings( {sqrt(-1); warning("ooops", call. = FALSE)}, "NaN" )
  # Warning message:
  # oops
suppress_warnings( {sqrt(-1); invisible()}, "NaN" )
  # Nothing is printed.
```

update.latexTable *Update a latexTable object with new arguments.*

Description

Each latexTable object stores, as an attribute, the call that produced it. update.latexTable() updates the call by replacing arguments or adding new ones. It then calls latexTable() to produce a new latexTable object.

Usage

```
## S3 method for class 'latexTable'
update(object, ...)
```

Arguments

object	A latexTable object
...	Arguments to latexTable(), e.g., colNames, caption.

Details

update.latexTable() is adapted from stats::update.default(). It is a method for the generic update().

Value

A latexTable object.

Examples

```
lT1 <- latexTable(matrix(1:16, nrow = 4))
lT2 <- update(lT1, mat = matrix(2:17, nrow = 8), commandName = "intTable")
```

%IN% *Value matching*

Description

%IN% returns a logical vector indicating whether there is a match for its left operand. It is like %in%, but it has one crucial difference: if there are NA values in the left operand, the corresponding values in the returned vector will also be NA (rather than FALSE, as with %in%.)

Usage

```
x %IN% table
```

Arguments

x	vector or NULL: the values to be matched.
table	vector or NULL: the values to be matched against

Details

The ordinary binary match operator, `%in%`, can be misleading because it seems more closely related to == than it is. The problem is that == will return NA in some (expected) cases, but `%in%` will never return NA. Instead, when using `%in%`, the returned vector will be FALSE for every NA value in the left operand.

Like ==, `%IN%` will return NA when there are NA values in the left operand. See below for an example.

`%IN%` will always return TRUE values when `%in%` would do so, and vice versa. The two operators differ only in the sense that `%IN%` returns FALSE in some cases where `%in%` returns NA.

Value

A logical vector of the same length as x. It indicates whether a match was found for each non-NA element of x. NA elements of x are matched by NA elements in the returned vector.

See Also

`%in%`

Examples

```
tmp <- c(1, 2, 3, NA)
tmp == 1      # TRUE FALSE FALSE NA
tmp %in% 1:2  # TRUE TRUE  FALSE FALSE
tmp %IN% 1:2  # TRUE TRUE  FALSE NA
```

Index