Package ‘easyr’

June 2, 2020

Type Package

Title Helpful Functions from Oliver Wyman Actuarial Consulting

Version 0.4-0

Maintainer Bryce Chamberlain <bryce.chamberlain@oliverwyman.com>

Description Makes difficult operations easy. Includes these types of functions:
    shorthand, type conversion, data wrangling, and workflow.
    Also includes some helpful data objects: NA strings, U.S. state list, color blind charting colors.
    Built and shared by Oliver Wyman Actuarial Consulting. Accepting proposed contributions through GitHub.

License GPL (>= 2)

LazyData true

URL https://github.com/oliver-wyman-actuarial/easyr

BugReports https://github.com/oliver-wyman-actuarial/easyr/issues

Depends R (>= 3.4.0)

Imports data.table, digest, dplyr, foreign, glue, Hmisc, lubridate,
    stringr, openssl, pdftools, qs, readxl, rlang, rprojroot, XML

RoxygenNote 6.1.1

Suggests testthat, rstudiaapi

NeedsCompilation no

Author Oliver Wyman Actuarial Consulting [aut, cph],
    Bryce Chamberlain [aut, cre],
    Scott Sobel [ctb],
    Rajesh Sahasrabuddhe [ctb]

Repository CRAN

Date/Publication 2020-06-02 14:50:02 UTC

R topics documented:

astext ................................................................. 3
atype ................................................................. 4
### R topics documented:

- begin ................................................................. 5
- binbyvol .............................................................. 6
- bindf ................................................................. 6
- cache.init ............................................................ 7
- cache.ok ............................................................. 8
- cblind ............................................................... 9
- cc ................................................................. 9
- char2fac ............................................................. 10
- charnum ............................................................. 11
- checkeq ............................................................. 12
- clear.cache .......................................................... 12
- coalf ............................................................... 13
- crun .............................................................. 14
- ddiff ............................................................. 14
- dict ............................................................... 15
- drows ............................................................. 16
- ecopy .............................................................. 16
- eq ............................................................... 17
- fac2char ............................................................. 18
- fjoinf ............................................................. 18
- fldict ............................................................. 19
- fmat .............................................................. 20
- getbetterint ......................................................... 21
- getinfo ............................................................ 22
- gr ............................................................... 23
- hashfiles ........................................................... 23
- headers_row ........................................................ 24
- ijoinf ............................................................. 24
- ischar ............................................................. 25
- isdate ............................................................ 26
- isfac ............................................................. 27
- isnum ............................................................. 27
- isval ............................................................. 28
- jrepl ............................................................. 29
- left .............................................................. 30
- likedate ............................................................ 30
- ljoinf ............................................................. 31
- match.factors ....................................................... 32
- mdiff ............................................................. 33
- mid .............................................................. 34
- na ............................................................... 35
- namesx ........................................................... 35
- nan .............................................................. 36
- nanull ............................................................. 36
- nastrings ............................................................ 37
- null .............................................................. 38
- pad0 .............................................................. 38
- qdiff ............................................................. 39
Description

Prints a vector as text you can copy and paste back into the code. Helpful for copying vectors into code for testing and validation. Author: Bryce Chamberlain.

Usage

astext(x)

Arguments

x Vector to represent as text.

Value

Vector represented as a character.
Examples

```r
c( 'a', 'b', 'c' )
astext( c( '1', '2', '4') )
```
Examples

```r
# create some data in all-characters.
x = data.frame(
  char = c( 'abc', 'def' ),
  num = c( '1', '2' ),
  date = c( '1/1/2018', '2018-2-01' ),
  na = c( NA, NA ),
  bool = c( 'TRUE', 'FALSE' ),
  stringsAsFactors = FALSE
)

# different atype options. Note how the output types change.
str( atype( x ) )
str( atype( x, exclude = 'date' ) )
str( atype( x, auto_convert_dates = FALSE ) )
str( atype( x, check_logical = FALSE ) )
```

Description

Perform common operations before running a script. Includes clearing environment variables, disabling scientific notation, loading common packages, and setting the working directory to the location of the current file.

Usage

```r
begin(wd = NULL, load = c("magrittr", "dplyr"), scipen = FALSE,
      verbose = TRUE, repos = "http://cran.us.r-project.org")
```

Arguments

- **wd**: Path to set as working directory. If blank, the location of the current file open in RStudio will be used if available. If FALSE, the working directory will not be changed.
- **load**: Packages to load. If not available, they’ll be installed.
- **scipen**: Do scientific notation in output?
- **verbose**: Print information about what the function is doing?
- **repos**: choose the URL to install from.

Examples

```r
begin()
```
binbyvol

Bin by Volume

Description

Bins a numerical column according to another numerical column’s volume. For example if I want to bin a column "Age" (of people) into 10 deciles according to "CountofPeople" then I will get Age breakpoints returned by my function such that there is 10 This function handles NA’s as their own separate bin, and handles any special values you want to separate out. Author: Scott Sobel. Tech Review: Bryce Chamberlain.

Usage

binbyvol(df, groupby, vol, numbins)

Arguments

df (Data Frame) Your data.
groupby (Character) Name of the column you’ll create cuts in. Must be the character name of a numeric column.
vol (Character) Name of the column for which each cut will have an equal percentage of volume.
numbins Number of bins to use.

Value

Age breakpoints returned by my function such that there is 10

Examples

# bin Sepal.Width according to Sepal.Length.
iris$bin <- binbyvol(iris, 'Sepal.Width', 'Sepal.Length', 5)

# check the binning success.
aggregate( Sepal.Length ~ bin, data = iris, sum )

bindf

Bind Rows with Factors

Description

Matches factor levels before binding rows. Author: Bryce Chamberlain.

Usage

bindf(..., sort.levels = TRUE)
**cache.init**

Initialize cache.

### Description

Set cache info so easyr can manage the cache.

### Usage

```r
cache.init(caches, at.path, verbose = TRUE, save.only = FALSE, skip.missing = TRUE)
```
Arguments

- **caches**: List of lists with properties name, depends.on. See example.
- **at.path**: Where to save the cache. If NULL, a cache/ folder will be created in the current working directory.
- **verbose**: Print via cat() information about cache operations.
- **save.only**: Choose not to load the cache. Use this if you need to check cache validity in multiple spots but only want to load at the last check.
- **skip.missing**: Passed to hashfiles, choose if an error occurs if a depends.on file isn’t found.

Examples

```r
# initialize a cache with 1 cache which depends on files in the current working directory.
# this will create a cache folder in your current working directory.
# then, you call functions to check and build the cache.
cache.init(
  caches = list(
    list(
      name = 'prep-files',
      depends.on = c('.'
    )
  ),
  at.path = tempdir()
)
)
```

---

**cache.ok**

*Check Cache Status*

Description

Check a cache and if necessary clear it to trigger a re-cache.

Usage

```r
cache.ok(cache.num, do.load = TRUE)
```

Arguments

- **cache.num**: The index/number for the cache we are checking in the cache.info list.
- **do.load**: Load the cache if it is found.
Value

Boolean indicating if the cache is acceptable. FALSE indicates the cache doesn’t exist or is invalid so code should be run again.

Examples

```r
# check the first cache to see if it exists and dependent files haven't changed.
# if this is TRUE, code in brackets will get skipped and the cache will be loaded instead.
# set do.load = FALSE if you have multiple files that build a cache,
# to prevent multiple cache loads.
# output will be printed to the console to tell you if the cache was loaded or re-built.
if( ! cache.ok(1) ){

    # do stuff

    # if this is the final file for this cache, end with save.cache to save passed objects as a cache.
    save.cache(iris)
}
```

Description

Color palette that is effective for color-blind clients.

Usage

cblind

Format

Named vector of hex colors.

Description

Shorthand function for paste. Author: Bryce Chamberlain.

Usage

```r
cc(..., sep = "")
```
char2fac

Arguments

... Arguments to be passed to paste0. Typically a list of vectors or values to be concatenated.

sep (Optional) Separator between concatenated items.

Value

Vector of pasted/concatenated values.

Examples

cc( 1, 2, 4 )
x = data.frame( c1 = c( 1, 2, 4 ), c2 = c( 3, 5, 7 ) )
cc( x$c1, x$c2 )
cc( x$c1, x$c2, sep = "-" )

char2fac

Characters to Factors

Description

Convert all character columns in a data frame to factors. Author: Bryce Chamberlain.

Usage

char2fac(x, sortlevels = FALSE, na_level = "(Missing)"")

Arguments

x Data frame to modify.

sortlevels Choose whether to sort levels. This is the default R behavior and is therefore likely faster, but it may change the order of the data and this can be problematic so the default is FALSE.

na_level some functions don’t like factors to have NAs so we replace NAs with this value for factors only. Set NULL to skip.

Value

Data frame with converted factors.

Examples

char2fac( iris )
**charnum**

*Check for Number Formatted as Character.*

**Description**

Checks a vector or value to see if it is a number formatted as a character. Useful for checking columns formatted with $ or commas, etc. Author: Bryce Chamberlain. Tech review: Dominic Dillingham.

**Usage**

```r
charnum(x, na_strings = easyr::nastrings, run_unique = TRUE, 
check_date = TRUE)
```

**Arguments**

- `x` Vector to check.
- `na_strings` Strings to consider NA.
- `run_unique` Convert to unique variables before checking. In some cases, this can make it take longer than necessary. In most, it will make it faster.
- `check_date` Check for a date, in which case it isn’t a number. If you have already checked a date and know it isn’t, set this to FALSE to run faster.

**Value**

True/false value indicating if the vector is a number formatted as a character. Helpful for checking before calling easyr:tonum().

**Examples**

```r
charnum( c( 
  '123', '$50.02', '30%', '(300.01)', '-10', '1 230.4', NA, '- ', '', "3.7999999999999999E-2" 
))
charnum( c( '123', 'abc', '30%', NA ) )
# returns FALSE since this can be converted to a date:
charnum( c( '20180101' ))
```
checkeq  

**Check Value or Control Total**

**Description**
Check actual versus expected values and get helpful metrics back. Author: Bryce Chamberlain. Tech review: Lindsay Smeltzer.

**Usage**
```r
checkeq(expected, actual, desc = "", acceptable_pct_diff = 0.00000001, digits = 2)
```

**Arguments**
- `expected` The expected value of the metric.
- `actual` The actual value of the metric.
- `desc` (Optional) Description of the metric being checked.
- `acceptable_pct_diff` (Optional) Acceptable percentage difference when checking values. Checked as an absolute value.
- `digits` (Optional) Digits to round to. Without rounding you get errors from floating values. Set to NA to avoid rounding.

**Value**
Message (via cat) indicating success or errors out in case of failure.

**Examples**
```r
checkeq(expected=100, actual=100, desc='A Match')
```

clear.cache  

**Clear Cache**

**Description**
Clears all caches or the cache related to the passed cache info list.

**Usage**
```r
clear.cache(cache = NULL)
```

**Arguments**
- `cache` The cache list to clear.
Value

FALSE if a cache info list item is passed in order to assist other functions in returning this value, otherwise NULL.

Examples

# this will only have an effect if a current cache exists.
clear.cache()

---

coa  

Factor-friendly Coalesce

Description

Coalesce function that matches and updates factor levels appropriately. Checks each argument vector starting with the first until a non-NA value is found. Author: Bryce Chamberlain.

Usage

coalf(...)

Arguments

...  

Source vectors.

Value

Vector of values.

Examples

x <- sample(c(1:5, NA, NA, NA))
coalf(x, 0L)
**Description**

Concatenate arguments and run them as a command. Shorthand for `eval(parse(text = paste0(...)))`. Consider also using `base::get()` which can be used to get an object from a string, but only if it already exists. Author: Bryce Chamberlain.

**Usage**

```r
crun(...)
```

**Arguments**

```r
...
```

Character(s) to be concatenated and run as a command.

**Examples**

```r
crun("print("hello world!"), ")
crun("T", "RUE")
```

---

**ddiff**

*Date difference (or difference in days).*

**Description**

Date difference (or difference in days).

**Usage**

```r
ddiff(x, y, unit = "day", do.date.convert = TRUE, do.numeric = TRUE)
```

**Arguments**

```r
x
```

Vector of starting dates or items that can be converted to dates by `todate`.

```r
y
```

Vector of ending dates or items that can be converted to dates by `todate`.

```r
unit
```

Character indicating what to use as the unit of difference. Values like d, y, m or day, year, month will work. Takes just the first letter in lower-case to determine unit.

```r
do.date.convert
```

Convert to dates before running the difference. If you know your columns are already dates, setting to FALSE will make your code run faster.

```r
do.numeric
```

Convert the output to a number instead of a date difference object.
Value

Vector of differences.

Examples

diff( lubridate::mdy( '1/1/2018' ), lubridate::mdy( '3/4/2018' ) )

---

dict Get Data Dictionary

Description

Get information about a Data Frame or Data Table. Use getinfo to explore a single column instead. If you like, use ecopy function or argument to copy to the clipboard so that it can be pasted into Excel. Otherwise it returns a data frame. Author: Scott Sobel. Tech Review & Modifications: Bryce Chamberlain.

Usage

dict(x, topn = 5, botn = 5, na.strings = easyr::nastrings, 
do.atype = TRUE, ecopy = FALSE)

Arguments

x Data Frame or Data Table.
topn Number of top values to print.
botn Number of bottom values to print.
na.strings Strings to consider NA.
do.atype Auto-determine variable types. If your data already has types set, skip this to speed up the code.
ecopy Use ecopy function or argument to copy to the clipboard so that it can be pasted into Excel.

Examples

dict(iris)
drows  
*Get Rows with Duplicates*

**Description**

Pulls all rows with duplicates in a column, not just the duplicate row. Author: Bryce Chamberlain.

**Usage**

\[
drows(x, c, na = \text{FALSE})
\]

**Arguments**

- **x**: Data frame.
- **c**: Column as vector or string.
- **na**: Consider multiple NAs as duplicates?

**Value**

Rows from the data frame in which the column is duplicated.

**Examples**

\[
\text{ddt} = \text{bindf( cars, utilities::head( cars, 10 ) )}
\]

\[
drows( \text{ddt}, \text{`speed'})
\]

---

ecopy  
*Copy to Clipboard*

**Description**

Copies a data.frame or anything that can be converted into a data.frame. After running this, you can use ctrl+v or Edit > Paste to paste it to another program, typically Excel. A simple use case would be ecopy(names(df)) to copy the names of a data frame to the clipboard to paste to Excel or Outlook. Author: Scott Sobel. Tech Review: Bryce Chamberlain.

**Usage**

\[
ecopy(x, \text{showrowcolnames} = \text{c("cols", "rows", "both", "none"), show = \text{FALSE})}
\]
eq

Arguments

x Object you'd like to copy to the clipboard.
showrowcolnames (Optional) Show row and column names. Choose 'none', 'cols', 'rows', or 'both'.
show (Optional Boolean) Set to 'show' if you want to also print the object to the console.

Examples

ecopy(iris, showrowcolnames = "cols", show = 'show')
ecopy(iris)

eq NA-Friendly Equality Comparison

Description
Vectorized flexible equality comparison which considers NAs as a value. Returns TRUE if both values are NA, and FALSE when only one is NA. The standard == comparison returns NA in both of these cases and sometimes this is interpreted unexpectedly. Author: Bryce Chamberlain. Tech Review: Maria Gonzalez.

Usage

eq(x, y, do.nanull.equal = TRUE)

Arguments

x First vector/value for comparison.
y Second vector/value for comparison.
do.nanull.equal Return TRUE if both inputs are NA or NULL (tested via easyr::nanull).

Value
Boolean vector/value of comparisons.

Examples

c(NA,'NA',1,2,'c') == c(NA,NA,1,2,'a') # regular equality check.
eq(c(NA,'NA',1,2,'c'),c(NA,NA,1,2,'a')) # check with eq.
fac2char  

*Factors to Characters*

**Description**

Convert all factor columns in a data frame to characters. Author: Bryce Chamberlain.

**Usage**

```r
fac2char(x)
```

**Arguments**

- `x` Data frame to modify.

**Value**

Data frame with converted characters.

**Examples**

```r
char2fac( iris )
```

fjoinf  

*Full Join with Factors*

**Description**

Matches factor levels before full join via merge. Author: Bryce Chamberlain.

**Usage**

```r
fjoinf(data.left, data.right, by, sort.levels = TRUE, restrict.levels = FALSE, na_level = "(Missing)")
```

**Arguments**

- `data.left` Left data. Only rows that match the join will be included (may still result in duplication).
- `data.right` Right data. All of this data will be preserved in the join (may also result in duplication).
- `by` Columns to join on.
- `sort.levels` Sort the factor levels after combining them.
- `restrict.levels` Often the joined data won’t use all the levels in both datasets. Set to TRUE to remove factor levels that aren’t in the joined data.
- `na_level` some functions don’t like factors to have NAs so we replace NAs with this value for factors only. Set NULL to skip.
Value

Joined data, with any factors modified to contain all levels in the joined data.

Examples

```r
df1 = data.frame(
  factor1 = c('a', 'b', 'c'),
  factor2 = c('high', 'medium', 'low'),
  factor.join = c('0349038u093843', '304359867893753', '3409783509735'),
  numeric = c(1, 2, 3),
  logical = c(TRUE, TRUE, TRUE)
)
df2 = data.frame(
  factor1 = c('d', 'e', 'f'),
  factor2 = c('low', 'medium', 'high'),
  factor.join = c('32532532536', '304359867893753', '32534745876'),
  numeric = c(4, 5, 6),
  logical = c(FALSE, FALSE, FALSE)
)
fjoin(f, df2, by = 'factor.join')
```

fdict

*Get Data Dictionary for Files in Folder*

Description

Get information about data files in a folder path. Use `dict()` on a single data frame or `getinfo(0)` to explore a single column. Author: Bryce Chamberlain.

Usage

```r
fldict(folder = NULL, file.list = NULL,
      pattern = "^[\^~]+[.](xls|xml|xmb|csv|rds|xml)", ignore.case = TRUE,
      recursive = TRUE, ...)
```

Arguments

- **folder**: File path of the folder to create a dictionary for. Pass either this or `file.list`. `file.list` will override this argument.
- **file.list**: List of files to create a combined dictionary for. Pass either this or `folder`. This will override folder.
- **pattern**: Pattern to match files in the folder. By default we use a pattern that matches read.any-compatible data files and skips temporary Office files. Passed to `list.files`.
- **ignore.case**: Ignore case when checking pattern. Passed to `list.files`.
recursive
  Check files recursively. Passed to list.files.
...
  Other arguments to read.any for reading in files. Consider using a first_column_name
vector, etc.

Value
List with the properties:

s
  Summary data of each dataset.
1
  Line data with a row for each column in each dataset.

Examples

fl = fldict( 'path/to/folder' )

names( fl )

fl$sheets
fl$columns

fmat

Number Formatter

Description
Flexible number formatter for easier formatting from numbers and dates into characters for display.

Usage

fmat(x = NULL, type = c("auto", ",", "$", ",\%", ",.\", "mdy", "ymd", "date", "dollar", "dollars", "count", "percentage", "decimal"),
do.return = c("formatted", "highcharter"), digits = NULL,
with.unit = FALSE, do.date.sep = "/", do.remove.spaces = FALSE,
digits.cutoff = NULL)

Arguments

x
  Vector of values to convert. If retu

type
  Type of format to return. If do.return == 'highcharter' this is not required.
do.return
  Information to return. "formatted" returns a vector of formatted values.
digits
  Number of digits for rounding. If left blank, the function will guess at the best
digits.
with.unit
  For large numbers, choose to add a suffix for fewer characters, like M for mil-
do.date.sep
  Separator for date formatting.
do.remove.spaces
   Remove extra spaces in return.
digits.cutoff Amount at which to show 0 digits. Allows for flexibility of rounding.

Value
   Information requested via do.return.

Examples

   fmat( 1000, 'dollar', digits = 2 )

getbetterint
   Get better Int

Description
   Takes bucket names of binned values such as [1e3, 2e3) or [0.1234567, 0.2) and formats the values
   nicely into values such as 1,000-2,000 or 0.12-0.20 Author: Scott Sobel. Tech Review: Bryce
   Chamberlain.

Usage

   getbetterint(int)

Arguments

   int Vector of character bucket names to transform.

Value

   Vector of transformed values.

Examples

   iris$bin <- binbyvol( iris, 'Sepal.Width', 'Sepal.Length', 5 )
   getbetterint( iris$bin )
getinfo

Get Info

Description

Get information about a Column in a Data Frame or Data Table. Use getdatadict to explore all columns in a dataset instead. Author: Scott Sobel. Tech Review: Bryce Chamberlain.

Usage

getinfo(df, colname, topn = 5, botn = 5, graph = TRUE, ordered = TRUE, display = TRUE, cutoff = 20, main = NULL, cex = 0.9, xcex = 0.9, bins = 50, col = "light blue")

Arguments

df (Data Frame or Data Table).

colname (Character) Name of the column to get information about.
topn (Optional) Number of top values to print.
botn (Optional) Number of bottom values to print.
graph (Boolean Optional) Output a chart of the column.
ordered (Optional)
display (Optional)
cutoff (Optional)
main (Optional)
cex (Optional)
xcex (Optional)
bins (Optional)
col (Optional)

Value

Only if display = FALSE, returns information about the column. Otherwise information comes through the graphing pane and the console (via cat/print).

Examples

getinfo(iris,"Sepal.Width")
getinfo(iris,"Species")
Golden Ratio

Description
Get the golden ratio. Author: Bryce Chamberlain. Tech Review: Maria Gonzalez.

Usage
gr()

Value
The golden ratio: \((1+\sqrt{5}) / 2\)

Examples
gr()

Hash Files

Description
Get a hash value representing a list of files. Useful for determining if files have changed in order to reset dependent caches.

Usage
hashfiles(x, skip.missing = FALSE, full.hash = FALSE, verbose = FALSE)

Arguments
- **x**: Input which specifies which files to hash. This can be a vector mix of paths and files.
- **skip.missing**: Skip missing files. Default is to throw an error if a file isn’t found.
- **full.hash**: By default we just hash the file info (name, size, created/modified time). Set this to TRUE to read the file and hash the contents.
- **verbose**: Print helpful messages from code.

Value
String representing hash of files.

Examples
hashfiles( '.' )
headers_row

*Identify headers row.*

**Description**

Identify the row with headers in a data frame. It should NOT be used directly (that’s why it isn’t exported), but will be called by function *[read.any]* as necessary, with the applicable defaults set by that function.

**Usage**

```r
define_headers_row(x, headers_on_row = NA, first_column_name = NA, field_name_map = NA)
```

**Arguments**

- **x**
  - Data frame to work with.
- **headers_on_row**
  - The specific row with headers on it.
- **first_column_name**
  - A known column(s) that can be used to find the header row. This is more flexible, but only used if headers_on_row is not available. If multiple are possible, use a vector argument here.
- **field_name_map**
  - *field_name_map* from read.any.

**Value**

List with `headers_already_column_names` (TRUE/FALSE); `headers_on_row` (1-indexed number of the to match standard R indexing).

---

**ijoinf**

*Inner Join with Factors*

**Description**

Matches factor levels before inner join via merge. Author: Bryce Chamberlain.

**Usage**

```r
ijoinf(data.left, data.right, by, sort.levels = TRUE, restrict.levels = FALSE, na_level = "(Missing)")
```
Arguments

- **data.left**: Left data. Only rows that match the join will be included (may still result in duplication).
- **data.right**: Right data. Only rows that match the join will be included (may also result in duplication).
- **by**: Columns to join on.
- **sort.levels**: Sort the factor levels after combining them.
- **restrict.levels**: Often the joined data won’t use all the levels in both datasets. Set to TRUE to remove factor levels that aren’t in the joined data.
- **na_level**: some functions don’t like factors to have NAs so we replace NAs with this value for factors only. Set NULL to skip.

Value

Joined data, with any factors modified to contain all levels in the joined data.

Examples

```r
df1 = data.frame(
  factor1 = c('a', 'b', 'c'),
  factor2 = c('high', 'medium', 'low'),
  factor.join = c('0349038u093843', '304359867893753', '3409783509735'),
  numeric = c(1, 2, 3),
  logical = c(TRUE, TRUE, TRUE)
)

df2 = data.frame(
  factor1 = c('d', 'e', 'f'),
  factor2 = c('low', 'medium', 'high'),
  factor.join = c('32532532536', '304359867893753', '32534745876'),
  numeric = c(4, 5, 6),
  logical = c(FALSE, FALSE, FALSE)
)

ljoinf( df1, df2, by = 'factor.join' )
```

---

**ischar**

*Shorthand for is.character*

**Description**

Shorthand for is.character
Usage
ischar(x)

Arguments
x Value to check.

Value
logical indicator

Examples
ischar('a character')
ischar(1)

---

isdate Shorthand for lubridate::is.Date

Description
Shorthand for lubridate::is.Date

Usage
isdate(x)

Arguments
x Value to check.

Value
logical indicator

Examples
isdate( lubridate::mdy('10/1/2014') )
isdate(1)
isfac

Shorthand for is.factor

Description
Shorthand for is.factor

Usage
isfac(x)

Arguments
x Value to check.

Value
logical indicator

Examples
isfac(factor(c('a', 'b', 'c')))  # Not isfac(1)

isnum

Shorthand for is.numeric

Description
Shorthand for is.numeric

Usage
isnum(x)

Arguments
x Value to check.

Value
logical indicator

Examples
isnum(1)  # isnum(factor(c('a', 'b', 'c')))

isval

Is Valid / Is a Value / NA NULL Check

Description

Facilitates checking for missing values which may cause errors later in code. NULL values can cause errors on is.na checks, and is.na can cause warnings if it is inside if() and is passed multiple values. This function makes it easier to check for missing values before trying to operate on a variable. It will NOT check for strings like "" or "NA". Only NULL and NA values will return TRUE. Author: Bryce Chamberlain. Tech Review: Maria Gonzalez.

Usage

isval(x, na_strings = easyr::nastrings, do.test.each = FALSE)

Arguments

x
Object to check. In the case of a data frame or vector, it will check the first (non-NULL) value.

na_strings
(Optional) Set the strings you want to consider NA. These will be applied after stringr::str_trim on x.

do.test.each
Return a vector of results to check each element instead of checking the entire object.

Value

True/false indicating if the argument is NA, NULL, or an empty/NA string/vector. For specct, only the first value is checked.

Examples

isval( NULL )
isval( NA )
isval( c( NA , NULL ) )
isval( c( 1, 2, 3 ) )
isval( c( NA, 2, 3 ) )
isval( c( 1, 2, NA ) ) # only the first values is checked, so this will come back FALSE.
isval( c( NULL, 2, 3 ) ) # NULL values get skipped in a vector.
isval( data.frame() )
isval( dplyr::group_by( dplyr::select( cars, speed, dist ), speed ) ) # test a tibble.
isval( "#VALUE!" ) # test an excel error code.
**jrepl**

*Join and Replace Values.*

**Description**

Replace a columns values with matches in a different dataset. Author: Bryce Chamberlain.

**Usage**

```r
jrepl(x, y, by, replace.cols, na.only = FALSE, only.rows = NULL, verbose = FALSE)
```

**Arguments**

- `x`: Main dataset which will have new values. This data set will be returned with new values.
- `y`: Supporting dataset which has the id and new values.
- `by`: Vector of join column names. A character vector if the names match. A named character vector if they don't.
- `replace.cols`: Vector of replacement column names, similar format as `by`.
- `na.only`: Only replace values that are NA.
- `only.rows`: Select rows to be affected. Default checks all rows.
- `verbose`: Print via `cat` information about the replacement.

**Value**

- `x` with new values.

**Examples**

```r
df1 = utils::head( sleep )
group.reassign = data.frame(
  id.num = factor( c( 1, 3, 4 ) ),
group.replace = factor( c( 99, 99, 99 ) )
)

jrepl(  
  x = df1,  
  y = group.reassign,  
  by = c( 'ID' = 'id.num' ),  
  replace.cols = c( 'group' = 'group.replace' )  
)
```

# doesn't affect since there are no NAs in group.
```
```
\begin{verbatim}
y = group.reassign,
by = c('ID' = 'id.num'),
replace.cols = c('group' = 'group.replace'),
na.only = TRUE
\)
\end{verbatim}

---

**left**

**left**

---

**Description**

Behaves like Excel's LEFT, RIGHT, and MID functions Author: Dave. Tech review: Bryce Chamberlain.

**Usage**

\texttt{left(string, char)}

**Arguments**

- **string**: String to process.
- **char**: Number of characters.

**Examples**

\texttt{left( "leftmidright", 4 )}

---

**likedate**

**Like Date**

---

**Description**

Check if a column can be converted to a date. Helpful for checking a column before actually converting it. Author: Bryce Chamberlain. Tech review: Dominic Dillingham.

**Usage**

\texttt{likedate(x, na_strings = easyr::nastrings, run_unique = TRUE, aggressive.extraction = TRUE)}
ljoinf

**Arguments**

- **x**: Value or vector to check.
- **na_strings**: Vector of characters to consider NA. Like Date will treat these values like NA.
- **run_unique**: Convert to unique variables before checking. In some cases, this can make it take longer than necessary. In most, it will make it faster.
- **aggressive.extraction**: todate will take dates inside long strings (like filenames) and convert them to dates. This seems to be the preferred outcome, so we leave it as default (TRUE). However, if you want to avoid this you can do so via this option (FALSE).

**Value**

Boolean indicating if the entire vector can be converted to a date.

**Examples**

```r
x <- c('20171124','2017/12/24',NA,'12/24/2017','March 3rd, 2015','Mar 3, 2016')
likedate(x)
likedate(c(123,456,NA))
if(likedate(x)) t <- todate(x)
likedate(lubridate::mdy('1-1-2014'))
likedate( '3312019' )
likedate( '2019.1.3' )
```

---

ljoinf  
*Left Join with Factors*

**Description**

Matches factor levels before left join via merge. Author: Bryce Chamberlain.

**Usage**

```r
ljoinf(data.left, data.right, by, sort.levels = TRUE, restrict.levels = FALSE, na_level = "(Missing)")
```

**Arguments**

- **data.left**: Left data. All of this data will be preserved in the join (may still result in duplication).
- **data.right**: Right data. Only rows that match the join will be included (may also result in duplication).
- **by**: Columns to join on.
- **sort.levels**: Sort the factor levels after combining them.
restrict.levels

Often the joined data won’t use all the levels in both datasets. Set to TRUE to remove factor levels that aren’t in the joined data.

na_level

some functions don’t like factors to have NAs so we replace NAs with this value for factors only. Set NULL to skip.

Value

Joined data, with any factors modified to contain all levels in the joined data.

Examples

df1 = data.frame(
  factor1 = c( 'a', 'b', 'c' ),
  factor2 = c( 'high', 'medium', 'low' ),
  factor.join = c('0349038u093843', '304359867893753', '3409783509735' ),
  numeric = c( 1, 2, 3 ),
  logical = c( TRUE, TRUE, TRUE )
)

df2 = data.frame(
  factor1 = c( 'd', 'e', 'f' ),
  factor2 = c( 'low', 'medium', 'high' ),
  factor.join = c('32532532536', '304359867893753', '32534745876' ),
  numeric = c( 4, 5, 6 ),
  logical = c( FALSE, FALSE, FALSE )
)
ljoinf( df1, df2, by = 'factor.join' )

---

match.factors  Match Factors.

Description

Modifies two datasets so matching factor columns have the same levels. Typically this is used prior to joining or bind_rows in the easyr functions bindf, ijoinf, lfjoinf.

Usage

match.factors(df1, df2, by = NA, sort.levels = TRUE)

Arguments

df1  First data set.
df2  Second data set.
Columns to join on, comes from the function using match.factors (ljoinf, fjoinf, ijoinf).

Sort the factor levels after combining them.

Value

List of the same data but with factors modified as applicable. All factors are checked if no 'by' argument is passed. Otherwise only the 'by' argument is checked.

Examples

df1 = data.frame(
  factor1 = c( 'a', 'b', 'c' ),
  factor2 = c( 'high', 'medium', 'low' ),
  factor.join = c( '0349038u093843', '304359867893753', '3409783509735' ),
  numeric = c( 1, 2, 3 ),
  logical = c( TRUE, TRUE, TRUE )
)

df2 = data.frame(
  factor1 = c( 'd', 'e', 'f' ),
  factor2 = c( 'low', 'medium', 'high' ),
  factor.join = c( '32532532536', '304359867893753', '32534745876' ),
  numeric = c( 4, 5, 6 ),
  logical = c( FALSE, FALSE, FALSE )
)

t = match.factors( df1, df2 )
levels( df1$factor1 )
levels( t[[1]]$factor1 )
levels( t[[2]]$factor1 )
Arguments

x Vector of starting dates or items that can be converted to dates by todate.
y Vector of ending dates or items that can be converted to dates by todate.
do.date.convert Convert to dates before running the difference. If you know your columns are already dates, setting to FALSE will make your code run faster.
do.numeric Convert the output to a number instead of a date difference object.

Value

Vector of differences.

Examples

mdiff( lubridate::mdy('1/1/2018'), lubridate::mdy('3/4/2018') )

mid

mid

Description

Behaves like Excel’s LEFT, RIGHT, and MID functions Author: Bryce Chamberlain.

Usage

mid(string, start, nchars)

Arguments

string String to process.
start Index (1-index) to start at.
nchars Number of characters to read in from start.

Examples

mid("leftmidright", 5, 3)
### na

**Description**

Shorthand for `is.na`

**Usage**

```r
na(x)
```

**Arguments**

- `x`: Value to check.

**Value**

- logical indicator

**Examples**

```r
na(NA)
na(1)
```

### namesx

**Description**

Get column names that match a pattern. Author: Scott Sobel. Tech review: Bryce Chamberlain.

**Usage**

```r
namesx(df, char, fixed = TRUE, ignore.case = TRUE)
```

**Arguments**

- `df`: Object with names you’d like to search.
- `char`: Regex character to match to columns.
- `fixed`: Match as a string, not a regular expression.
- `ignore.case`: Ignore case in matches.

**Value**

- Vector of matched names.
Examples

nan(iris,'len')
nan(iris,'Len')

nan \hspace{1em} \textit{Shorthand for is.nan}

Description

Shorthand for is.nan

Usage

nan(x)

Arguments

x \hspace{1em} \text{Value to check.}

Value

\text{logical indicator}

Examples

nan( NaN )
nan(1)

nanull \hspace{1em} \textit{NA / NULL Check}

Description

Facilitates checking for missing values which may cause errors later in code. NULL values can cause errors on is.na checks, and is.na can cause warnings if it is inside if() and is passed multiple values. This function makes it easier to check for missing values before trying to operate on a variable. It will \textbf{NOT} check for strings like "" or "NA". Only NULL and NA values will return TRUE. Author: Bryce Chamberlain. Tech Review: Maria Gonzalez.

Usage

nanull(x, na_strings = easyr::nastrings, do.test.each = FALSE)
Arguments

- **x**: Vector to check. In the case of a data frame or vector, it will check the first (non-NULL) value.

- **na_strings**: (Optional) Set the strings you want to consider NA. These will be applied after stringr::str_trim on x.

- **do.test.each**: Return a vector of results to check each element instead of checking the entire object.

Value

True/false indicating if the argument is NA, NULL, or an empty/NA string/vector. For specct, only the first value is checked.

Examples

```r
nanull( NULL )
nanull( NA )
nanull( c( NA, NULL ) )
nanull( c( 1, 2, 3 ) )
nanull( c( NA, 2, 3 ) )
nanull( c( 1, 2, NA ) ) # only the first values is checked, so this will come back FALSE.
nanull( c( NULL, 2, 3 ) ) # NULL values get skipped in a vector.
nanull( data.frame() )
nanull( dplyr::group_by( dplyr::select( cars, speed, dist ), speed ) ) # test a tibble.
```

---

### nastrings

#### NA Strings

**Description**

A list of strings to consider NA. Includes blank string, "NA", excel errors, etc. Used throughout easyr for checking NA.

**Usage**

```r
nastrings
```

**Format**

A vector of values.
null

*Shorthand for is.null*

**Description**

Shorthand for is.null

**Usage**

null(x)

**Arguments**

x Value to check.

**Value**

logical indicator

**Examples**

null( NULL )
null(1)

---

**pad0**

*Pad with Zeros*

**Description**

Adds leading zeros to a numeric vector to make each value a specific length. For values shorter than length passed, leading zeros are removed. Author: Scott Sobel. Tech Review: Bryce Chamberlain.

**Usage**

pad0(x, len)

**Arguments**

x Vector.

len Number of characters you want in each value.

**Value**

Character vector with padded values.
Examples

\texttt{pad0(} \texttt{c(123,00123,5), len = 5 \texttt{)}
\texttt{pad0(} \texttt{c(123,00123,5), len = 2 \texttt{)}
\texttt{pad0(} \texttt{'}1234\texttt{', 5 \texttt{)}}

\begin{verbatim}

qdiff

\textbf{Date Difference in Quarters}

\textbf{Description}

Date Difference in Quarters

\textbf{Usage}

\texttt{qdiff(x, y, do.date.convert = TRUE, do.numeric = TRUE)}

\textbf{Arguments}

\begin{itemize}
\item \texttt{x} Vector of starting dates or items that can be converted to dates by \texttt{todate}.
\item \texttt{y} Vector of ending dates or items that can be converted to dates by \texttt{todate}.
\item \texttt{do.date.convert} Convert to dates before running the difference. If you know your columns are already dates, setting to \texttt{FALSE} will make your code run faster.
\item \texttt{do.numeric} Convert the output to a number instead of a date difference object.
\end{itemize}

\textbf{Value}

Vector of differences.

\textbf{Examples}

\texttt{qdiff( \texttt{lubridate::mdy('1/1/2018'), lubridate::mdy('3/4/2018')} )}

\end{verbatim}

\begin{verbatim}

rany_fixColNames

\textbf{Fix column names.}

\textbf{Description}

Code to fix column names, since this has to be done up to twice will reading in files. It should NOT be used directly (that’s why it isn’t exported), but will be called by function \texttt{read.any} as necessary, with the applicable defaults set by that function.

\textbf{Usage}

\texttt{rany_fixColNames(col_names, fix.dup.column.names, nastrings)}

\end{verbatim}
Arguments

- `col_names`: Vector/value of column names/name.
- `fix.dup.column.names`: Adds 'DUPLICATE #’ to duplicated column names to avoid errors with duplicate names.
- `nastrings`: Characters/strings to read as NA.

Value

- Fixed names.

Description

Flexible read function to handle many types of files. Currently handles CSV, TSV, DBF, RDS, XLS (incl. when formatted as HTML), and XLSX. Also handles common issues like strings being read in as factors (strings are NOT read in as factors by this function, you’d need to convert them later).

Author: Bryce Chamberlain. Tech Review: Dominic Dillingham.

Usage

```r
read.any(filename = NA, folder = NA, sheet = 1, file_type = "", first_column_name = NA, header = TRUE, headers_on_row = NA, nrows = -1L, row.names.column = NA, row.names.remove = TRUE, make.names = FALSE, field_name_map = NA, require_columns = NA, all_chars = FALSE, auto_convert_dates = TRUE, allow_times = FALSE, check_numbers = TRUE, nazero = FALSE, check_logical = TRUE, stringsAsFactors = FALSE, na_strings = easyr::nastrings, na_level = "(Missing)", ignore_rows_with_na_at = NA, drop.na.cols = TRUE, drop.na.rows = TRUE, fix.dup.column.names = TRUE, do.trim.sheetname = TRUE, x = NULL, isexcel = FALSE, encoding = "unknown", verbose = TRUE)
```

Arguments

- `filename`: File path and name for the file to be read in.
- `folder`: Folder path to look for the file in.
- `sheet`: The sheet to read in.
- `file_type`: Specify the file type (CSV, TSV, DBF). If not provided, R will use the file extension to determine the file type. Useful when the file extension doesn’t indicate the file type, like .rpt, etc.
- `first_column_name`: Define headers location by providing the name of the left-most column. Alternatively, you can choose the row via the [headers_on_row] argument.
header
Choose if your file contains headers.

headers_on_row
Choose a specific row number to use as headers. Use this when you want to tell read.any exactly where the headers are.

nrows
Number of rows to read. Leave blank/NA to read all rows. This only speeds up file reads (CSV, XLSX, etc.), not compressed data that must be read all at once. This is applied BEFORE headers_on_row or first_column_name removes top rows, so it should be greater than those values if headers aren’t in the first row.

row.names.column
Specify the column (by character name) to use for row names. This drops the columns and lets rows be referenced directly with this id. Must be unique values.

row.names.remove
If you move a column to row names, it is removed from the data by default. If you’d like to keep it, set this to FALSE.

make.names
Apply make.names function to make column names R-friendly (replaces non-characters with , starting numbers with x, etc.)

field_name_map
Rename fields for consistency. Provide as a named vector where the names are the file’s names and the vector values are the output names desired. See examples for how to create this input.

require_columns
List of required columns to check for. Calls stop() with helpful message if any aren’t found.

all_chars
Keep all column types as characters. This makes using bind_rows easier, then you can use atype() later to set types.

auto_convert_dates
Identify date fields and automatically convert them to dates

allow_times
Times are not allowed in reading data in to facilitate easy binding. If you need times though, set this to TRUE.

check_numbers
Identify numbers formatted as characters and convert them as such.

nazero
Convert NAs in numeric columns to 0.

check_logical
Identify logical columns formatted as characters (Yes/No, etc) or numbers (0,1) and convert them as such.

stringsAsFactors
Convert characters to factors to increase processing speed and reduce file size.

na_strings
Strings to treat like NA. By default we use the easyr NA strings.

na_level
dplyr doesn’t like factors to have NAs so we replace NAs with this value for factors only. Set NULL to skip.

ignore_rows_with_na_at
Vector or value, numeric or character, identifying column(s) that require a value. read.any will remove these rows after colname swaps and read, before type conversion. Especially helpful for removing things like page numbers at the bottom of an excel report that break type discovery. Suggest using the claim number column here.

drop.na.cols
Drop columns with only NA values.
drop.na.rows  Drop rows with only NA values.
fix.dup.column.names  Adds 'DUPLICATE #' to duplicated column names to avoid issues with multiple columns having the same name.
do.trim.sheetname  read.any will trim sheet names to get better matches. This will cause an error if the actual sheet name has spaces on the left or right side. Disable this functionality here.
x  If you want to use read.any functionality on an existing data frame, pass it with this argument.
iseexcel  If you want to use read.any functionality on an existing data frame, you can tell read.any that this data came from excel using isexcel manually. This comes in handy when excel-integer date conversions are necessary.
encoding  Encoding passed to fread and read.csv.
verbose  Print helpful information via cat.

Value
Data frame with the data that was read.

Examples

read.any( 'path/to/file.extension' )

# if dates are being converted incorrectly, disable date conversion:
read.any( 'path/to/file.extension', auto_convert_dates = FALSE )

# to handle type conversions manually:
read.any( 'path/to/file.extension', all_chars = TRUE )

---

**read.txt**

*Read File as Text*

**Description**

Read File as Text

**Usage**

read.txt(filename, folder = NA)

**Arguments**

filename  File path and name for the file to be read in.
folder  Folder path to look for the file in.
Value

Character variable containing the text in the file.

Examples

```r
# write a files.
path = tempfile()
cat( "some text", file = path )

# read the file.
read.txt( path )

# cleanum.
file.remove( path )
```

Description

Behaves like Excel’s LEFT, RIGHT, and MID functions Author: Dave. Tech review: Bryce Chamberlain.

Usage

```
right(string, char)
```

Arguments

- **string**: String to process.
- **char**: Number of characters.

Examples

```
right( "leftmidright", 5 )
```
runfolder 

**Run Folder**

**Description**

Run all the R scripts in a folder. Author: Bryce Chamberlain.

**Usage**

```r
runfolder(path, recursive = FALSE, is.local = TRUE, check.fn = NULL,
          run.files = NULL, verbose = TRUE, edit.on.err = TRUE,
          pattern = "\[.\][Rr]\$")
```

**Arguments**

- `path`: Folder to run.
- `recursive`: Run all folder children also.
- `is.local`: Code is running on a local machine, not a Shiny server. Helpful for skipping items that can be problematic on the server. In this case, printing to the log.
- `check.fn`: Function to run after each file is read-in.
- `run.files`: Optionally pass the list of files to run. Otherwise, list.files will be run on the folder.
- `verbose`: Print names of files and run-time via cat.
- `edit.on.err`: Open the running file if an error occurs.
- `pattern`: Passed to list.files. Pattern to match/filter files.

**Examples**

```r
# runfolder( 'R' )
```

---

**rx Read Excel**

**Description**

This gets a bit complex since many errors can occur when reading in excel files. We’ve done our best to handle common ones. Requires packages: openxlsx, readxl, XML (these are required by easyr). It should NOT be used directly (that’s why it isn’t exported), but will be called by function [read.any] as necessary, with the applicable defaults set by that function.

**Usage**

```r
rx(filename, sheet, first_column_name, nrows, verbose)
```
save.cache

Arguments

filename File path and name for the file to be read in.
sheet The sheet to read in.
first_column_name Pass a column name to help the function find the header row.
nrows Number of rows to read in.
verbose Print helpful messages via cat().

Value

Data object

---

save.cache Save Cache Saves the arguments to a cache file, using the cache.num last checked with cache.ok.

Description

Save Cache

Saves the arguments to a cache file, using the cache.num last checked with cache.ok.

Usage

save.cache(...)

Arguments

... Objects to save.

Examples

# check the first cache to see if it exists and dependent files haven't changed.
# if this check is TRUE, code in brackets will get skipped and the cache will be loaded instead.
# set do.load = FALSE if you have multiple files that build a cache,
# to prevent multiple cache loads.
# output will be printed to the console to tell you if the cache was loaded or re-built.
if( ! cache.ok(1) ){
    # do stuff

    # if this is the final file for this cache, end with save.cache to save passed objects as a cache.
    save.cache(iris)
}

# delete the cache folder to close out the example.
system( "rm -r cache" )
**sch**  

*Search a Data Frame.*

**Description**

Searches all columns for a term and returns all rows with at least one match. Author: Bryce Chamberlain.

**Usage**

```r
sch(x, pattern, ignore.case = FALSE, fixed = FALSE, pluscols = NULL, exact = FALSE, trim = TRUE, spln = NULL)
```

**Arguments**

- `x`  
  Data to search.
- `pattern`  
  Regex pattern to search. Most normal search terms will work fine, too.
- `ignore.case`  
  Ignore case in search (uses grepl).
- `fixed`  
  Passed to grepl to match string as-is instead of using regex. See ?grepl.
- `pluscols`  
  Choose columns to return in addition to those where matches are found. Can be a name, number or 'all' to bring back all columns.
- `exact`  
  Find exact matches instead of pattern matching.
- `trim`  
  Use trimws to trim columns before exact matching.
- `spln`  
  Sample data use easyr::spl() before searching. This will speed up searching in large datasets when you only need to identify columns, not all data that matches. See ?spln argument for more info.

**Value**

Matching rows.

**Examples**

```r
sch( iris, 'seto' )
sch( iris, 'seto', pluscols='all' )
sch( iris, 'seto', pluscols='Sepal.Width' )
sch( iris, 'seto', exact = TRUE ) # message no matches and return NULL
```
**Description**

Extracts a uniform random sample from a dataset or vector. Provides a simpler API than base R. 
Author: Bryce Chamberlain. Tech Review: Maria Gonzalez.

**Usage**

```
spl(x, n = 10, warn = TRUE, replace = FALSE, ...)
```

**Arguments**

- `x`: Data to sample from.
- `n`: Number or percentage of rows/values to return. If less than 1 it will be interpreted as a percentage.
- `warn`: Warn if sampling more than the size of the data.
- `replace`: Whether or not to sample with replacement.
- `...`: Other parameters passed to sample()

**Value**

Sample dataframe/vector.

**Examples**

```
spl( c(1:100) )
spl( c(1:100), n = 50 )
spl( iris )
```

---

**Description**

Helpful info for states. Right now, just a mapping of abbreviations to names.

**Usage**

```
states
```

**Format**

Data frame.
strx  

Structure with Like

Description

Runs str function but only for names matching a character value (regex). Author: Scott Sobel. Tech Review: Bryce Chamberlain.

Usage

strx(df, char, ignore.case = T)

Arguments

df  
Object with names you’d like to search.

char  
Regex (character value) to match.

ignore.case  
(Optional) Ignore case when matching.

Examples

strx(iris,"length")

sumnum  

Summarize All Numeric Columns

Description

Easily summarize at all numeric variables. Helpful for flexibly summarizing without knowing the columns. Defaults to sum but you can send a custom function through also. Typically pass in a data frame after group_by.

Usage

sumnum(x, do.fun = NULL, except = c(), do.ungroup = TRUE, ...)

Arguments

x  
Grouped tibble to summarize.

do.fun  
Function to use for the summary. Passed to dplyr::summarize(). Can be a custom function. Defaults to sum().

except  
Columns names, numbers, or a logical vector indicating columns NOT to summarize.

do.ungroup  
Run dplyr::ungroup() after summarizing the prevent future issues with grouping.

...  
Extra args passed to dplyr::summarize() which are applied as arguments to the function passed in do.fun.
**Value**

Summarized data frame or tibble.

**Examples**

```r
require(dplyr)
require(easyr)

sumnum( group_by( cars, speed )
sumnum( group_by( cars, speed ), mean )
sumnum( cars )
```

**Description**

Easy Try/Catch implementation to return the same message on error or warning. Makes it easier to write tryCatches. Author: Bryce Chamberlain. Tech review: Lindsay Smelzter.

**Usage**

`tcmgs(code_block, ...)`

**Arguments**

- `code_block` Code to run in Try Catch.
- `...` Strings to concatenate to form the message that is returned.

**Examples**

```r
tryCatch({
  tcmgs({ NULL = 1 }, 'Cannot assign to NULL','variable' )
},
error = function(e) print( e )
)
```

```r
tryCatch({
  tcmgs({ as.numeric('abc') },'Issue in as.numeric()' )
},
warning = function(e) print( e )
)```
tcol

Transpose at Column.

Description
Transpose operation that sets column names equal to a column in the original data. Author: Bryce Chamberlain.

Usage
tcol(x, header, cols.colname = "col", do.atype = TRUE)

Arguments
- x: Data frame to be transposed.
- header: Column name/number to be used as column names of transposed data.
- cols.colname: Name to use for the column of column names in the transposed data.
- do.atype: Transpose converts to strings, since data types are uncertain. Run atype to automatically correct variable typing where possible. This will slow the result a bit.

Value
Transposed data frame.

Examples
# create a summary dataset from iris.
x = dplyr::summarize_at(dplyr::group_by(iris, Species),
  dplyr::vars(Sepal.Length, Sepal.Width), list(sum))
# run tcol
tcol(x, 'Species')

tcwarn
tryCatch with warning

Description
Easy Try/Catch implementation to return the same message as a warning on error or warning. Makes it easier to write tryCatches. Author: Bryce Chamberlain. Tech review: Lindsay Smelzter.

Usage
tcwarn(code_block, ...)

Arguments

code_block  
...  

Examples

tryCatch({
  tcwarn({ NULL = 1 },'Cannot assign to NULL','variable')
},
  warning = function(e) print( e )
)

tryCatch({
  tcwarn({ as.numeric('abc') },'Issue in as.numeric()')
},
  warning = function(e) print( e )
)

tobool

Convert to Logical/Boolean

Description

Flexible boolean conversion. Author: Bryce Chamberlain.

Usage

tobool(x, preprocessed.values = NULL, nastrings = easyr::nastrings,
  ifna = c("return-unchanged", "error", "warning", "return-na"),
  verbose = TRUE, true.vals = c("true", "1", "t", "yes"),
  false.vals = c("false", "0", "f", "no"))

Arguments

x  
preprocessed.values  
Strings need to have NAs set, lowercase and be trimmed before they can be checked. To avoid doing this multiple times, you can pass these processed values to the function.

nastrings  
Vector of characters to be considered NAs. todate will treat these like NAs. Defaults to the easyr::nastrings list.

ifna  
Action to take if NAs are created. 'return-unchanged' returns the sent vector unchanged; 'warning' results in a warning and returns the converted vector with new NAs; 'error' results in an error.

verbose  
Choose to view messaging.

ture.vals  
Values to consider as TRUE.

false.vals  
Values to consider as FALSE.
Value

Converted logical vector.

Examples

tobool(c( 'true', 'FALSE', 0, 1, NA, 'yes', 'NO' ))

tochar Shorthand for as.character

Description

Shorthand for as.character

Usage

tochar(x)

Arguments

x Value to check.

Value

as.character result

Examples

 tochar(NA)
tochar(1)

todate Convert to Date

Description

Flexible date conversion function using lubridate. Works with dates in many formats, without needing to know the format in advance. Only use this if you don’t know the format of the dates before hand. Otherwise, lubridate functions parse_date_time, mdy, etc. should be used. Author: Bryce Chamberlain. Tech review: Dominic Dillingham.
todate

Usage
todate(x, nastrings = easyr::nastrings, aggressive.extraction = TRUE,
preprocessed.values = NULL, ifna = c("return-unchanged", "error",
"warning", "return-na"), verbose = TRUE, allow_times = FALSE,
do.month.char = TRUE, do.excel = TRUE,
min.acceptable = lubridate::ymd("1920-01-01"),
max.acceptable = lubridate::ymd("2050-01-01"))

Arguments

x Value or vector to be converted.
nastrings Vector of characters to be considered NAs. todate will treat these like NAs.
Defaults to the easyr::nastrings list.
aggressive.extraction todate will take dates inside long strings (like filenames) and convert them to
dates. This seems to be the preferred outcome, so we leave it as default (TRUE).
However, if you want to avoid this you can do so via this option (FALSE).
preprocessed.values Strings need to have NAs set, lowercase and be trimmed before they can be
checked. To avoid doing this multiple times, you can pass these processed values
to the function.
ifna Action to take if NAs are created. 'return-unchanged' returns the sent vector
unchanged; 'warning' results in a warning and returns the converted vector with
new NAs; 'error' results in an error; 'return-na' returns new NAs without a
warning.
verbose Choose to view messaging.
allow_times Set to TRUE to allow DateTimes as output, otherwise this will always convert
to Dates (losing time information). This is better for binding data, hence the
default FALSE.
do.month.char Attempt to convert month names in text. lubridate does this by default, but some-
times it can result in inaccurate dates. For example, "Feb 2017" is converted to
2-20-2017 even though no day was given.
do.excel Check for excel-formatted numbers.
min.acceptable Set NA if converted value is less than this value. Helps to prevent numbers from
being assumed as dates. Set NULL to skip this check. Does not affect character
conversions.
max.acceptable Set NA if converted value is greater than this value. Helps to prevent numbers
from being assumed as dates. Set NULL to skip this check. Does not affect character
conversions.

Value

Converted vector using lubridate::parse_date_time(x,c(’mdy’,’ymd’,’dmy’))
Examples

```r
x <- c( '20171124', '2017/12/24', NA, '12/24/2017', '5/11/2017 1:51PM' )
x2 <- todate(x)
x2
```

---

### tonum Convert to Number

**Description**


**Usage**

```r
tonum(x, preprocessed.values = NULL, nastrings = easyr::nastrings, ifna = c("return-unchanged", "error", "warning", "return-na"), verbose = TRUE, nazero = FALSE, checkdate = TRUE, remove.chars = FALSE, do.logical = TRUE, do.try.integer = TRUE, multipliers = c("%" = 1/100, K = 1000, M = 1000^2, B = 1000^3))
```

**Arguments**

- `x` Vector to convert.
- `preprocessed.values` Strings need to have NAs set, lowercase and be trimmed before they can be checked. To avoid doing this multiple times, you can pass these processed values to the function.
- `nastrings` Vector of characters to be considered NAs. todate will treat these like NAs. Defaults to the easyr::nastrings list.
- `ifna` Action to take if NAs are created. 'return-unchanged' returns the sent vector unchanged; 'warning' results in a warning and returns the converted vector with new NAs; 'error' results in an error; return-na returns data with new NAs and prints via cat if verbose.
- `verbose` Choose to view messaging.
- `nazero` (Optional) Convert NAs to 0. Defaults to TRUE, if FALSE NAs will stay NA.
- `checkdate` Check if the column is a date first. If this has already been done, set this to FALSE so it doesn’t run again.
- `remove.chars` Remove characters for aggressive conversion to numbers.
- `do.logical` Check for logical-form vectors.
- `do.try.integer` Return an integer if possible. Integers are a more compact data type and should be used whenever possible.
- `multipliers` Named vector of factor symbols and values to check. Setting to NULL may speed up operations.
usepkg

Value

Converted vector.

Examples

```
tonum( c('123','$50.02','30%','(300.01)',NA,'-','' ) )
tonum( c('123','$50.02','30%','(300.01)',NA,'-','' ), nzero = FALSE )
tonum( c( '$(3,891)M', '4B', '3.41K', '30', '40K' ) )
```

usepkg

Use Package

Description

Installs a package if it needs to be installed, and calls require to load the package. Author: Scott Sobel. Tech Review: Bryce Chamberlain.

Usage

```
usepkg(packages, noCache = FALSE,
       repos = "http://cran.us.r-project.org")
```

Arguments

packages Character or character vector with names of the packages you want to use.
nCache When checking packages, you can choose to ignore the cached list, which will
       increase accuracy but decrease speed.
repos choose the URL to install from.

Examples

```
usepkg(c('dplyr','lubridate'))
usepkg('lubridate')
```

validate.equal

Validate Equal

Description

Check various properties of 2 data frames to ensure they are equivalent.
validate.equal

Usage

validate.equal(df1, df2, id.column = NULL,
regex.remove = "[^A-z0-9.+/,-]\", do.set.NA = TRUE,
nastrings = easyr::nastrings, match.round.to.digits = 4,
do.all.columns.before.err = FALSE, check.column.order = FALSE,
sort.by.id = TRUE, acceptable.pct.rows.diff = 0,
acceptable.pct.vals.diff = 0, return.summary = FALSE,
verbose = TRUE)

Arguments

df1 First data frame to compare.
df2 Second data frame to compare.
id.column If available, a column to use as an ID. Helpful in various checks and output.
regex.remove Pattern to remove from strings. Used in gsub to remove characters we don’t want to consider when comparing values. Set to NULL, NA, or "" to leave strings unchanged.
do.set.NA Remove NA strings.
nastrings Strings to consider NA.
match.round.to.digits Round numbers to these digits before checking equality.
do.all.columns.before.err Check all columns before returning an error. Takes longer but returns more detail. If FALSE, stops at first column that doesn’t match and returns mismatches.
check.column.order Enforce same column order.
sort.by.id Sort by the id column before making comparisons.
acceptable.pct.rows.diff If you are OK with differences in a few rows, set this value. If fewer rows in a column don’t match, the function will consider the columns equivalent. Interpreted as a percentage (it gets divided by 100).
acceptable.pct.vals.diff If you are OK with small differences in values, set this value. If the difference in numeric values is less, the function will consider the values equivalent. Interpreted as a percentage (it gets divided by 100) and compared to absolute value of percentage difference.
return.summary Return 2 items in a list, the row mismatches and a summary of row mismatches.
verbose Print helpful information via cat().

Value

May return information about mismatches. Otherwise doesn’t return anything (NULL).

Examples

validate.equal( iris, iris )
**w**

**Write**

**Description**

Improved write function. Writes to csv without row names and automatically adds .csv to the file name if it isn’t there already. Changes to .csv if another extension is passed. Easier to type than write.csv(row.names=F). Author: Bryce Chamberlain. Tech review: Maria Gonzalez.

**Usage**

\[
w(x, \text{filename} = "\text{out}", \text{row.names} = \text{FALSE}, \text{na} = "\"")\]

**Arguments**

- \(x\) Data frame to write to file.
- \(\text{filename}\) (Optional) Filename to use.
- \(\text{row.names}\) (Optional) Specify if you want to include row names/numbers in the output file.
- \(\text{na}\) (Optional) String to print for NAs. Defaults to an empty/blank string.

**Examples**

```r
# write the cars dataset.
p = \text{paste0( tempdir(), }'/\text{out.csv}')
w(cars, p)

# cleanup.
file.remove(p)
```

---

**xldate**

**Convert Excel Number to Date**

**Description**

Converts dates formatted as long integers from Excel to Date format in R, accounting for known Excel leap year errors. Author: Bryce Chamberlain. Tech review: Dominic Dillingham.

**Usage**

\[
xldate(x, \text{origin} = "1899-12-30", \text{nastrings} = \text{easyr::nastrings},
p\text{ressed.values} = \text{NULL}, \text{ifna} = \text{c("return-unchanged", "error",}
"\text{warning", "return-na"}, \text{verbose} = \text{TRUE}, \text{allow.times} = \text{FALSE},
\text{do.month.char} = \text{TRUE}, \text{min.acceptable} = \text{lubridate::ymd("1920-01-01"),}
\text{max.acceptable} = \text{lubridate::ymd("2050-01-01")})
\]
Arguments

x       Vector of values.
origin  Zero value to use in date conversion. Older version of excel might use a different value.
nastrings  Vector of characters to be considered NAs. todate will treat these like NAs. Defaults to the easyr::nastrings list.
preprocessed.values  Strings need to have NAs set, lowercase and be trimmed before they can be checked. To avoid doing this twice, you can tell the function that it has already been done.
ifna  Action to take if NAs are created. ‘return-unchanged’ returns the sent vector unchanged; ‘warning’ results in a warning and returns the converted vector with new NAs; ‘error’ results in an error.
verbose  Choose to view messaging.
allow_times  Return values with time, not just the date.
do.month.char  Convert month character names like Feb, March, etc.
min.acceptable  Set NA if converted value is less than this value. Helps to prevent numbers from being assumed as dates. Set NULL to skip this check.
max.acceptable  Set NA if converted value is greater than this value. Helps to prevent numbers from being assumed as dates. Set NULL to skip this check.

Value

Vector of converted values.

Examples

```r
xldate( c("7597", "42769", "47545", NA ) )
```

---

ydifff  

Date Difference in Years

Description

Date Difference in Years

Usage

```
ydiff(x, y, do.date.convert = TRUE, do.numeric = TRUE)
```
Arguments

- **x**: Vector of starting dates or items that can be converted to dates by todate.
- **y**: Vector of ending dates or items that can be converted to dates by todate.
- **do.date.convert**: Convert to dates before running the difference. If you know your columns are already dates, setting to FALSE will make your code run faster.
- **do.numeric**: Convert the output to a number instead of a date difference object.

Value

- Vector of differences.

Examples

```r
ydiff( lubridate::mdy( '1/1/2018' ), lubridate::mdy( '3/4/2018' ) )
```

---

%ni%  

**Not-In**

Description

Opposite of Author: Bryce Chamberlain.

Usage

```r
needle %ni% haystack
```

Arguments

- **needle**: Vector to search for.
- **haystack**: Vector to search in.

Value

- Boolean vector/value of comparisons.

Examples

```r
c(1,3,11) %ni% 1:10
```
Index

*Topic datasets
  cblind, 9
  nastrings, 37
  states, 47
%nix%, 59

astext, 3
atype, 4

begin, 5
binbyvol, 6
bindf, 6

cache.init, 7
cache.ok, 8
cblind, 9
cc, 9
char2fac, 10
charnum, 11
checkeq, 12
clear.cache, 12
colaf, 13
crun, 14

diff, 14
dict, 15
drows, 16

ecopy, 16
eq, 17

fac2char, 18
fjoinf, 18
f1dict, 19
fmat, 20

getbetterint, 21
getinfo, 22
gre, 23

hashfiles, 23

headers_row, 24

ijoinf, 24
ischar, 25
isdate, 26
isfac, 27
isnum, 27
isval, 28

jrepl, 29

left, 30
likedate, 30
ljoinf, 31

match.factors, 32
mdiff, 33
mid, 34

na, 35
namesx, 35
nan, 36
nanull, 36
nastrings, 37
null, 38

pad0, 38

qdiff, 39

rany_fixColNames, 39
read.any, 40
read.txt, 42
right, 43
runfolder, 44
rx, 44

save.cache, 45
sch, 46
spl, 47
states, 47
INDEX

strx, 48
sumnum, 48
tcmsg, 49
tcol, 50
tcwarn, 50
to bool, 51
to char, 52
to date, 52
tonum, 54
usepkg, 55
validate.equal, 55
w, 57
xldate, 57
ydiff, 58