Package ‘jwutil’

May 6, 2019

**Title**  Tools for Data Manipulation and Testing

**Version**  1.2.3

**Description**  This is a set of simple utilities for various data
manipulation and testing tasks. The goal is to use core R tools well,
without bringing in many dependencies. Main areas of interest are
semi-automated data frame manipulation, such as converting factors in
multiple binary indicator columns. There are testing functions which
provide ‘testthat’ expectations to permute arguments to function
calls. There are functions and data to test extreme numbers, dates,
and bad input of various kinds which should allow testing failure and
corner cases, which can be used for fuzzing your functions. The test
suite has many examples of usage.

**License**  GPL-3

**URL**  https://github.com/jackwasey/jwutil

**BugReports**  https://github.com/jackwasey/jwutil/issues

**Depends**  R (>= 3.4.0)

**Imports**  Rcpp

**Suggests**  clipr, devtools, knitr, methods, pkgbuild, stats, testthat,
tools, utils, withr

**LinkingTo**  Rcpp, testthat

**VignetteBuilder**  knitr

**Encoding**  UTF-8

**Language**  en-US

**RoxygenNote**  6.1.1.9000

**NeedsCompilation**  yes

**Author**  Jack O. Wasey [aut, cre, cph] (<https://orcid.org/0000-0003-3738-4637>)

**Maintainer**  Jack O. Wasey <jack@jackwasey.com>

**Repository**  CRAN

**Date/Publication**  2019-05-06 19:10:03 UTC
R topics documented:

- jwutil-package
- add_time_to_date
- affixFields
- as_char_no_warn
- as_numeric_nowarn
- bad_input
- binary_col_names
- build_formula
- combn_subset
- countIsNa
- countNonNaCumulative
- countNotNumeric
- countNumeric
dput_expect_equal
- drop_duplicate_fields
- drop_rows_with_na
- expect_that_combine_all_args
- extreme_numbers
- factor_nosort
- factor_to_df
fillMissingCombs
- filter_better
- fix_na_ish
- flattenList
getDropped
- get_factor_fields
get_na_fields
- get_numeric_char_field_names
invwhich
- is.Date
- isFlat
- isRowSorted
- isValidTime
- is_na_ish
- is_numeric_str
- jw_df_basics
- jw_scan_build
- listTrim
- listTrimFlat
- list_named
- logical_to_binary
- ls.objects
- lsf
- lsos
- lsp
- match_multi
mergeLists ........................................ 31
merge_better ....................................... 32
min_r_version ..................................... 33
npc ............................................... 34
numbers_to_long_and_float ....................... 34
opt_binary_brute ................................ 35
percentize ........................................ 35
percent_signif .................................... 36
permute .......................................... 36
permuteWithRepeats ................................ 37
platformIsLinux .................................. 38
propIsNaN ......................................... 38
propNaPerField .................................... 39
propRowSorted ..................................... 39
random_test_dates .................................. 40
random_test_numbers ................................ 40
read_xlsx_linux .................................... 41
read_zip_url ....................................... 42
reqinst ............................................ 42
rm_r ............................................... 42
save_in_data_dir .................................. 43
shuffle ............................................ 43
sort_clip_char ..................................... 44
source_purl ....................................... 45
strip .............................................. 45
strip_for_formula ................................. 46
str_multi_match ................................... 46
trim .............................................. 47
two_cat_to_logical ................................. 47
unzip_single ..................................... 48
unzip_to_data_raw ................................ 48
update_github_pkgs ................................ 49
zeroes .............................................. 49
zero_na ........................................... 50
%nin% .............................................. 50

Index 53

jwutil-package  Tools for testing and data manipulation not found elsewhere

Description

This is a set of simple utilities for various data manipulation and testing tasks. The goal is to use core
R tools well, without bringing in many dependencies. Main areas of interest are semi-automated
data frame manipulation, such as converting factors in multiple binary indicator columns. There
are testing functions which provide ‘testthat’ expectations to permute arguments to function calls.
There are functions and data to test extreme numbers, dates, and bad input of various kinds which
should allow testing failure and corner cases, which can be used for fuzzing your functions. The test suite has many examples of usage.

**Author(s)**

**Maintainer:** Jack O. Wasey <jack@jackwasey.com> (0000-0003-3738-4637) [copyright holder]

**See Also**

Useful links:

- [https://github.com/jackwasey/jwutil](https://github.com/jackwasey/jwutil)
- Report bugs at [https://github.com/jackwasey/jwutil/issues](https://github.com/jackwasey/jwutil/issues)

---

**add_time_to_date**

**convert separate lists of dates and times to POSIXlt objects**

**Description**

Some datetime data is presented as a separate dates and times. This function restores the full datetime.

**Usage**

```
add_time_to_date(tms, dts, verbose = FALSE)
```

**Arguments**

- `tms` vector of times, i.e. number in range 0 to 2400, as string or integer, with or without trailing zeros
- `dts` vector of dates, in string format `%Y-%m-%d` or simple R Date objects
- `verbose` single logical value, if TRUE then produce verbose messages

**Value**

vector of POSIXlt date-times
**affixFields**

*update a set of data frame field names*

**Description**

prefix or suffix

**Usage**

```r
affixFields(fields, affix, skip = NULL, renameHow = c("suffix", "prefix"), sep = ".")
```

**Arguments**

- **fields**: char vector
- **affix**: character
- **skip**: char vector, defaults to include all fields
- **renameHow**: should be "suffix" or "prefix", default is suffix
- **sep**: default "."

**Value**

character vector, same length as fields

**as_char_no_warn**

*convert to character vector without warning*

**Description**

convert to character vector without warning

**Usage**

```r
as_char_no_warn(x)
```

**Arguments**

- **x**: vector, typically numeric or a factor

**Value**

character vector
as_numeric_nowarn

convert factor or vector to numeric without warnings

Description

correctly converts factors to vectors, and then converts to numeric or integer, which may silently introduce NAs. Invisible rounding errors can be a problem going from numeric to integer, so consider adding tolerance to this conversion. asIntegerNoWarn silently floors.

Usage

as_numeric_nowarn(x)

as_integer_nowarn(x)

is_integerish(x, tol = 1e-09, na.ignore = FALSE)

areIntegers(x, tol = 1e-09, na.ignore = FALSE)

Arguments

x is a vector, probably of numbers of characters
tol tolerance when considering if two numbers are integers, default 1e-9
na.ignore logical, if TRUE will pass through NA values, otherwise, they are marked FALSE.

Details

"are" functions return a value for each input, where is "allIs" functions return a single logical.

Value

numeric vector, may have NA values

logical vector

Functions

• areIntegers: Deprecated

Examples

stopifnot(is_integerish("1"))
**bad_input**

**bad_input data for tests**

---

**Description**

a variety of horrible data

**Usage**

bad_input

**Format**

An object of class list of length 43.

---

| binary_col_names | names of fields which are numeric, binary or combinations thereof |

**Description**

Doesn’t make any allowance for factors.

**Usage**

binary_col_names(x, invert = FALSE)

two_cat_col_names(x, invert = FALSE, ignore_na = FALSE, trim = TRUE)

binary_cols(x, invert = FALSE)

two_cat_cols(x, invert = FALSE)

**Arguments**

- **x** data frame
- **invert** single logical, if true, will return non-binary columns
- **ignore_na** If TRUE, then return columns with two distinct values in addition to NA. Default is FALSE, i.e. NA is counted as a distinct item.
- **trim** If character column found, then trim white space before assessing

**Value**

vector of column names
Functions

- `two_cat_col_names`: Get the columns which have exactly two categories therein, not including NA values. This would catch 0,1 "Yes", "No", etc.
- `binary_cols`: Get the data frame containing just the binary columns.
- `two_cat_cols`: Get the data frame containing only columns of input which have two categories.

Examples

```r
dat <- data.frame(  
c("a", "b"), c(TRUE, FALSE), c(1L, 0L),  
c(1L, 2L), c(0.1, 0.2), c("9", "8")  
)
names(dat) <- c(  
  "char", "bin", "binfloat", "binint",  
  "int", "float", "charint"  
)
binary_cols(dat)  
binary_col_names(dat)  
binary_col_names(dat, invert = TRUE)  
df <- data.frame(  
x = c("A", "B", "A", "B"),  
y = letters[1:4],  
z = c("y", NA, "y", NA),  
stringsAsFactors = FALSE  
)
two_cat_col_names(df)  
df[1, 1] <- NA  
df[2, 2] <- NA  
df  
stopifnot(two_cat_col_names(df) == "z")  
stopifnot(two_cat_col_names(df, ignore_na = TRUE) == "x")
```

---

**build_formula**

build simple linear formula from variable names

**Description**

build simple linear formula from variable names given by two character vectors. TODO: allow unquoted names.

**Usage**

```r
build_formula(left, right)
buildLinearFormula(left, right)
```
combn_subset

Arguments

left character vector
right character vector

Value

formula

Examples

print(f <- build_formula(left = "A", right = c("B", "C")))
class(f)
build_formula(left = "Species", right = names(iris)[1:4])

combn_subset           all unique combinations of a vector and all its non-zero subsets

Description

all unique combinations of a vector and all its non-zero subsets

Usage

combn_subset(x)

Arguments

x vector to be subsetted and combined

Value

list of vectors with all combinations of x and its subsets

Examples

combn_subset(c("a", "b"))
combn_subset(c(10, 20, 30))
combn_subset(NULL)
countIsNa

**Description**

count the number of NAs in a vector. also consider ‘base::anyNA’

**Usage**

countIsNa(x)

**Arguments**

x vector

**Value**

integer

countNonNaCumulative

**Description**

counts non-NA fields in first field, then progresses through fields, OR new field and saves running total for each field TODO: tests

**Usage**

countNonNaCumulative(d)

**Arguments**

d data.frame

**Value**

vector of cumulative non-NA counts with names corresponding to the given data frame
**countNotNumeric**

**Description**
counts the number of non-numeric elements in a vector, without throwing warnings

**Usage**
countNotNumeric(x)

**Arguments**

- `x` is usually a character vector

**Details**
did have extras = c(".", "NA")

**Value**
integer

---

**countNumeric**

**Description**
counts the number of numeric elements in a vector, without throwing warnings

**Usage**
countNumeric(x)

**Arguments**

- `x` is usually a character vector

**Value**
integer
dput_expect_equal  dput a testthat test

Description
Generate an R expression containing a testthat expectation for the given expression and its result. This is useful when you know that a certain output is correct, and wish to generate a test case to reflect this.

Usage
dput_expect_equal(...)  

Arguments
...  expressions  

Value
character vector with each element containing an R expression with expect_equal test case corresponding to the evaluated input expressions.

Examples
dput_expect_equal("a" %in% c("b", "c", "d"))

drop_duplicate_fields  Drop fields with duplicate data

Description
Compares all data in each field to every other field, and drops the latter match. Will find multiple matches. Doesn’t do any type conversions yet. This is purely by content, not by field name.

Usage
drop_duplicate_fields(df, verbose = FALSE)  
dropDuplicateFields(df, verbose = FALSE)  

Arguments
df  data.frame  
verbose  single logical value, if TRUE then produce verbose messages
**drop_rows_with_na**

**Value**

data frame without duplicate fields

**Functions**

- dropDuplicateFields: Deprecated

**Examples**

d <- data.frame(LETTERS, letters, letters)[1:10,]
drop_duplicate_fields(d)

---

**drop_rows_with_na**  
drops rows with NA values in specified fields

**Description**

employs `complete.cases` which is fast internal C code. Returns a data frame with unused factor levels dropped (these may have been introduced by dropping rows with some NA values)

**Usage**

drop_rows_with_na(x, fld = names(x), verbose = FALSE)
dropRowsWithNAField(x, fld = names(x), verbose = FALSE)

**Arguments**

- `x` data frame
- `fld` vector with names of fields which must have no NA values
- `verbose` single logical value, if TRUE then produce verbose messages

**Value**

data frame without rows containing NA in the specified data fields. There may be NA values in the resulting data frame in fields which are not listed in `fld`.

**Functions**

- dropRowsWithNAField: Deprecated, use drop_rows_with_na
**expect_that_combine_all_args**

Alternative `expect_that` from testthat which permutes all the inputs to a function which should give the same result where n args $\geq 2$ and the function is commutative.

**Description**

This makes a lot of assumptions, needs more testing. It can’t handle mixed error/no error outcomes after permutation, which is an important feature to consider. The command following this function attaches this function to the testthat namespace. This means that it can call internal testthat functions, but does not mean it appears as testthat::expect_that_combine

**Usage**

```r
expect_that_combine_all_args(object, condition, info = NULL, label = NULL)
```

```r
expect_that_combine_first_arg(object, condition, info = NULL, label = NULL)
```

**Arguments**

- **object**: See `expect_that`
- **condition**: See `expect_that`
- **info**: See `expect_that`
- **label**: See `expect_that`

**Value**

testthat result

**Examples**

```r
expect_that_combine_all_args(
    sum(1, 2, 3),
    testthat::equals(6)
)
## Not run:
expect_that_combine_all_args(stop("a", "b"), testthat::throws_error())
expect_that_combine_all_args(sum(1, 2), testthat::equals(3))
expect_that_combine_first_arg(sum(c(1, 2)), testthat::equals(3))
## End(Not run)
```
**extreme_numbers**

**Description**

very biggest and smallest non-zero numbers the current machine can handle, positive and negative.

**Usage**

extreme_numbers

**Format**

An object of class numeric of length 8.

---

**factor_nosort**

**Fast Factor Generation**

**Description**

This function generates factors more quickly, without leveraging fastmatch. The speed increase with fastmatch for ICD-9 codes was about 33 using Rcpp, and a hashed matching algorithm.

**Usage**

factor_nosort(x, levels = NULL, labels = levels)

**Arguments**

- **x**
  An object of atomic type integer, numeric, character or logical.
- **levels**
  An optional character vector of levels. Is coerced to the same type as x. By default, we compute the levels as sort(unique.default(x)).
- **labels**
  A set of labels used to rename the levels, if desired.

**Details**

NaNs are converted to NA when used on numeric values. Extracted from https://github.com/kevinushey/Kmisc.git

These feature from base R are missing: exclude = NA, ordered = is.ordered(x), nmax = NA

I don’t think there is any requirement for factor levels to be sorted in advance, especially not for ICD-9 codes where a simple alphanumeric sorting will likely be completely wrong.

**Author(s)**

Kevin Ushey, adapted by Jack Wasey
Examples

## Not run:
pts <- icd::random_unordered_patients(1e7)
u <- unique.default(pts$code)
# this shows that stringr (which uses stringi) sort takes 50% longer than
# built-in R sort.
microbenchmark::microbenchmark(sort(u), str_sort(u))

# this shows that \code{factor_} is about 50% faster than \code{factor} for
# big vectors of strings
# without sorting is much faster:
microbenchmark::microbenchmark(factor(pts$code),
   # factor_(pts$code),
   factor_nosort(pts$code),
   times = 25 )

## End(Not run)

---

**factor_to_df**

Convert factor into a data.frame of logicals

Description

Convert a single factor into a data.frame with multiple true or false fields, one for each factor. The ‘vtreat’ package may be a better choice for more comprehensive data preparation.

Usage

```r
factor_to_df(fctr, prefix = deparse(substitute(fctr)), sep = "", drop_empty = TRUE, na_as_col = TRUE, verbose = FALSE)
```

```r
factorToDataframeLogical(fctr, prefix = deparse(substitute(fctr)), sep = "", drop_empty = TRUE, na_as_col = TRUE, verbose = FALSE)
```

Arguments

- **fctr**: factor
- **prefix**: defaults to "f" to pre-pend the factor level when constructing the data frame columns names
- **sep**: scalar character, introduced between factor names and levels when forming new data frame column names
- **drop_empty**: logical, if ‘TRUE’ (the default) factor levels with no associated values are dropped.
- **na_as_col**: logical scalar: if NA data and/or NA levels, then covert to NA strings and expand these as for any other factor
- **verbose**: single logical value, if TRUE then produce verbose messages
fillMissingCombs

Value
data.frame with columns of logicals

Examples

```r
n <- 10
m <- 20
l <- LETTERS[seq_len(n)]
set.seed(1441)
f <- factor(sample(1, m, replace = TRUE), levels = l)
g <- factor_to_df(f, drop_empty = FALSE)
print(g)
stopifnot(nrow(g) == m, ncol(g) == n)
factor_to_df(
  shuffle(factor(shuffle(LETTERS[1:10]))),
  prefix = ""
)
factor_to_df(factor(c(NA, 1, 2, 3)))
factor_to_df(factor(c(NA, 1, 2, 3)), na_as_col = FALSE)
```

Description
fill out missing combinations of factors with NA

Usage

```r
fillMissingCombs(df)
```

Arguments

df data frame

Details
Adapated from [http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#using-aggregate](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#using-aggregate)
filter_better

filter data with diagnostics

Description

applies an expression to a data frame, and gives information about the numbers of dropped rows.

Usage

filter_better(x, expr, verbose = TRUE)

filterBetter(x, expr, verbose = TRUE)

Arguments

x        data frame
expr     expression in the context of the data frame, i.e. the terms should be column names.
verbose  logical default is TRUE

Value

filtered data frame

Functions

- filterBetter: Deprecated

fix_na_ish

Fix NA-like strings to be NA (or other value of choice)

Description

Fix NA-like strings to be NA (or other value of choice)

Usage

fix_na_ish(x, extra_na = NULL, new_val = NA)

Arguments

x        data frame
extra_na Additional values to consider equivalent to NA
new_val  New value to be used instead of NA-ish values, default is NA
flattenList

Examples

```r
df <- data.frame(
a = c("NA", "n/a", 1, NA),
b = c("three", "na", NaN, " N/A "),
stringsAsFactors = FALSE
)
df
fix_na_ish(df)
fix_na_ish(df, extra_na = "three", new_val = "0")
```

---

**flattenList**  
*flatten a list*

**Description**

Unlike `unlist`, this function returns a list of objects of different data types, but removes any depth.

**Usage**

`flattenList(..., na.rm = FALSE)`

**Arguments**

- `...`: list or any set of objects which will be made into a list, may include lists and nested lists.
- `na.rm`: will drop NA values if TRUE.

**Value**

List without nested lists, objects with preserved data types.

**Source**

https://stackoverflow.com/questions/8139677/how-to-flatten-a-list-to-a-list-without-coercion

---

**getDropped**  
*get items or numerics that would be dropped in a merge*

**Description**

Converts both vectors to numeric. This simulates merging when one key is character (but contains integer numbers), and another key is stored as integer.

**Usage**

`getDropped(x, y)`
Arguments

x vector or factor
y vector or factor

Value

list of two vectors

Description

Get the names of those fields in a data frame which are factors.

Usage

get_factor_fields(x, consider = names(x))
get_non_factor_fields(x, consider = names(x))
getFactorNames(x, consider = names(x))
getNonFactorNames(x, consider = names(x))

Arguments

x data frame
consider character vector of field names of the data frame to test, default is to use all of them.

Value

vector

Functions

• get_non_factor_fields: Get the fields which are not factors, instead.
• getFactorNames: Deprecated
• getNonFactorNames: Deprecated
get_na_fields

get na_fields

get NA field names from data frame

Description

Get the names of any columns in a data frame which have NA values.

Usage

get_na_fields(x, na_ish = FALSE, extra_na = NULL)

getNAFields(x, na_ish = FALSE, extra_na = NULL)

get_non_na_fields(x)

getNonNAFields(x)

Arguments

x data.frame

na_ish Logical, if ‘TRUE’, also consider NA-like strings, using ‘is_na_ish’

extra_na passed on to ‘is_na_ish’

Value

vector of names of fields which contain any NA values, length zero if no matches

Functions

• getNAFields: Deprecated

• getNonNAFields: Deprecated

get_numeric_char_field_names

Find columns which are numeric

Description

Get field names, or the data itself, of fields in a data frame which are numeric, or numeric-like characters.
Usage

get_numeric_char_field_names(x, invert = FALSE, attrition = 0.05)

get_numeric_field_names(x, invert = FALSE)

get_numeric_fields(x, invert = FALSE)

Arguments

x              Data frame
invert         Logical, if FALSE – the default – the numeric fields are returned, otherwise, non-
numeric fields are returned.
attrition       If less than this proportion of rows become NA on conversion to numeric, then
accept this is a numeric column after all.

invwhich

inverse which

Description

for a given vector of ordinals which would reference items in a vector, list etc, invwhich returns a
logical vector with TRUE for the cited positions. If length is not provided, the maximum index is
used.

Usage

invwhich(which, len = max(which))

Arguments

which        integer vector of indices, as would be produced by which
len          integer scalar: length of return vector, defaults to max(which)

Value

logical vector of length length
is.Date

is the object a Date

Description

copied from lubridate

Usage

is.Date(x)

Arguments

x object to test

Value

logical

isFlat
determine whether a list is nested

Description

Returns TRUE if the given list is not nested.

Usage

isFlat(x)

Arguments

x list

Value

single logical
isRowSorted  is every row sorted?

Description
Quickly run through rows of a matrix looking for any non-ascending rows in C++

Usage
isRowSorted(x)

Arguments
x matrix, each row containing ordered or disordered numerics

isValidTime  check if a time is valid in 24h clock

Description
allow leading and trailing space, optional colon in middle, 2400 is not allowed. TODO: can lubricate do this better?

Usage
isValidTime(tms, na.rm = FALSE)

Arguments
tms is a vector of characters which may represent times
na.rm logical if true, will ignore NA values, otherwise these will test as invalid.

Value
logical vector, with NA out if NA given
is_na_ish

Determine whether a value is, or should be, ‘NA’

Description
Determine whether a value is, or should be, ‘NA’

Usage
is_na_ish(x, extra_na = NULL)

Arguments
x vector to test
extra_na Additional values to consider equivalent to NA

Examples
is_na_ish(c(NA, "1"))
is_na_ish(c("NA", "N/A", "NaN"))
is_na_ish(c(NA))
is_na_ish(c(NA))

is_numeric_str
Which elements of a character vector are numeric

Description
Takes a character vector and returns a logical vector of the same length, indicating which values are numeric. NA is considered non-numeric. NA is never returned from this function.

Usage
is_numeric_str(x, extras = c(".", "NA", NA))
areNumeric(x, extras = c(".", "NA", NA))

Arguments
x character vector
extras character vector containing acceptable alternatives to numeric values which will result in returning TRUE for that element. Default is c(".", "NA", NA).

Value
logical vector of same length as input
Functions

- areNumeric: Deprecated

Examples

```r
areNumeric(c("1", "2", "3"))
areNumeric(c("1L", "2.2"))
areNumeric(c("NA", NA, ".", ",", ","-1.9"))
```

### jw_df_basics

**Description**

minimal basic pre-processing metrics

**Usage**

```r
jw_df_basics(x, df_list)
```

**Arguments**

- `x`: data.frame input
- `df_list`: list of data frames

### jw_scan_build

**Description**

C and C++ compilers are replaced by `scan-build clang`, and restored afterwards. Other flags and anything else in `~/R/Makevars` is left alone.

**Usage**

```r
jw_scan_build(path = ".", clang = "clang-8",
               scan_build = "scan-build")
```

**Arguments**

- `path`: Path to package root, default is ".".
- `clang`: Path or name of clang compiler executable. Currently `clang-8` which is what MacOS homebrew currently (early 2019) provides.
- `scan_build`: Path or name of scan-build executable. Current `scan-build`, also from MacOS Homebrew. On linux, this has the LLVM version suffix, e.g., `scan-build-8`. 
listTrim

**Description**

delete null/empty entries in a list. Recursively looks through list if nested.

**Usage**

```plaintext
listTrim(x)
```

**Arguments**

- `x`

**Value**

trimmed list

---

listTrimFlat

**Description**

Trim NULL or empty values from a flat list.

**Usage**

```plaintext
listTrimFlat(x)
```

**Arguments**

- `x`

**Value**

trimmed list
list_named

Make a list using input argument names as names

Description

Make a list using input argument names as names

Usage

list_named(...)  

Arguments

... arguments whose names become list item names, and whose values become the values in the list

Examples

```r
a <- c(1, 2)
b <- c("c", "d")stopifnot(
  identical(
    list_named(a, b),
    list(a = a, b = b)
  )
)
```
### Examples

```r
d <- data.frame(
a = c(TRUE, FALSE, TRUE),
b = c(FALSE, TRUE, FALSE),
c = c(-1, 0, 1),
d = c("not", "logical", "values")
)
logical_to_binary(d)
```

---

### Description

Get type, size (bytes) and dimensions of objects

### Usage

```r
ls.objects(env = parent.frame(), pattern, order.by, decreasing = FALSE, head = FALSE, n = 5)
```

### Arguments

- **env**: Environment to search, default is the parent frame
- **pattern**: regex pattern to match objects of interest
- **order.by**: which column to order by
- **decreasing**: default is TRUE
- **head**: default is FALSE but if true, just show top n
- **n**: number to show if limiting to head

---

### lsf

#### list all functions in a package

### Description

List functions in a package

### Usage

```r
lsf(pkg)
```

### Arguments

- **pkg**: character string containing package name
**Value**

character vector of functions in given package

---

**lsos**  
*show largest objects*

**Description**


**Usage**

`lsos(..., n = 10)`

**Arguments**

- `...`: arguments passed on to `.ls.objects`
- `n`: scalar integer, number of objects to show

---

**lsp**  
*List all items in a package*

**Description**

By default includes names beginning with `.`

**Usage**

`lsp(package, all.names = TRUE, pattern)`

**Arguments**

- `package`: character scalar: name of the package
- `all.names` = TRUE, set to FALSE to ignore items beginning with a period
- `pattern` = optional pattern to match

**Value**

character vector of package contents

**Examples**

```r
lsp("jwutil")
tail(lsp("base"), 30L)
```
### match_multi

**Match across columns for multiple lookup values**

**Description**

This provides a succinct way to query a data frame for conditions, which is otherwise very verbose in base R or dplyr.

**Usage**

```r
match_multi(x, cols, table, incomparables = NULL)
```

**Arguments**

- `x` data.frame
- `cols` character vector of column names to be found in `x`
- `table` vector of items to find
- `incomparables` passed on to the base function `match`

**Value**

matrix with same number of rows as `x`, and a column for each of `cols`

**Examples**

```r
j <- cars[1:10,]
match_multi(j, "speed", 7)
match_multi(j, "dist", 22)
match_multi(j, c("speed", "dist"), 10)
match_multi(j, c("speed", "dist"), c(7, 17))
```

---

### mergeLists

**merge lists by names**

**Description**

merge lists by vector combining all the vector elements of the list items with the matching names. Unnamed vectors in the list will be dropped silently.

**Usage**

```r
mergeLists(x, y)
```
Arguments

x          unnested list with named elements, each of which is a vector
y          unnested list with named elements, each of which is a vector

Value

list of vectors

merge_better  Merge better

Description

Apply built-in R `merge` but with additional features for safety and information.

Usage

merge_better(x, y, by.x, by.y, all.x = FALSE, all.y = FALSE, 
  affix = NULL, renameConflict = c("suffix", "prefix"), 
  renameAll = c("no", "suffix", "prefix"), convert_factors = TRUE, 
  verbose = FALSE)

mergeBetter(x, y, by.x, by.y, all.x = FALSE, all.y = FALSE, 
  affix = NULL, renameConflict = c("suffix", "prefix"), 
  renameAll = c("no", "suffix", "prefix"), convert_factors = TRUE, 
  verbose = FALSE)

Arguments

x          data frame
y          data frame
by.x       field in x to merge on. Unlike merge, this is compulsory.
by.y       field in y to merge on. Unlike merge, this is compulsory.
all.x      outer join to keep all x values
all.y      outer join to keep all y values
affix      either prefix or suffix to disambiguate files. By default, this is the name of the 
            table specified in y. In all other respects in this function, x and y are symmetric.
renameConflict - determines whether prefix or suffix is added to disambiguate conflicting col-
            umn names. Value can be "suffix", "prefix". Suffix is the default.
renameAll   - regardless of column name clashes, "prefix" or "suffix" with every field with 
            original table name, or "no" for neither
convert_factors   Default is TRUE which causes factors to be converted to character before merge. 
                  This is almost certainly safer.
verbose     logical or numbers 0, 1 or 2. 1 or TRUE will give moderate verbosity, 2 will 
                  give full verbosity. 0 or FALSE turns off all messages.
**min_r_version**

Value

merged data frame

Examples

df <- data.frame(a = c("1", "2"), b = 1:2, stringsAsFactors = FALSE)
eg <- data.frame(a = c("1", "3"), b = 3:4, stringsAsFactors = FALSE)
mergeBetter(x = df, y = eg, by.x = "a", by.y = "a", verbose = TRUE)

---

**min_r_version**

Find minimum R version required for package

Description

Recursively search dependencies for R version, and find the highest stated R version requirement.

Usage

```r
min_r_version(pkg)
```

Arguments

pkg string with name of package to check

Source

Based on ideas from http://stackoverflow.com/questions/38686427/determine-minimum-r-version-for-all-package-dependencies

Examples

```r
base <- c("base", "compiler", "datasets", "grDevices", "graphics",
"grid", "methods", "parallel", "profile", "splines", "stats",
"stats4", "tcltk", "tools", "translations"
)
## Not run:
base_reqs <- lapply(base, min_r_version)
contrib <- c(
  "KernSmooth", "MASS", "Matrix", "boot",
  "class", "cluster", "codetools", "foreign", "lattice",
  "mgcv", "nlme", "nnet", "rpart", "spatial", "survival"
)
contrib_reqs <- lapply(contrib, min_r_version)
min_r_version("icd")
## End(Not run)
```
**npc**

*Print integers with percentage of total rounded to integer*

**Description**

Intended for succinctly printing summary data in a scientific publication.

**Usage**

```r
cmp(x, n, fmt = "%d (%s)"
```

**Arguments**

- `x` numeric number
- `n` numeric total
- `fmt` `sprintf` format, default being `%d (%s)

**Examples**

```r
cmp(1, 100)
cmp(1, 1)
cmp(2, 1)
cmp(1.321, 7.7432)
cmp(7239, 234897)
cmp(-10, 1000)
```

**numbers_to_long_and_float**

*convert numbers to long and float types*

**Description**

intended for generating values for stress testing functions

**Usage**

```r
numbers_to_long_and_float(..., na.rm = TRUE)
```

**Arguments**

- `...` list of values to convert to long and double
- `na.rm` logical, defaults to TRUE, so output contains only long and float values.

**Value**

list of long and double versions of convertable values from the input
### opt_binary_brute

**selects columns from a data frame using an optimization function**

**Description**

The optimization function is called with the data frame \( x \) and the names of each combination of the names of \( x \)'s columns. An example of real-world usage is to automate selection of columns according to the optimization function.

**Usage**

```r
opt_binary_brute(x, fun = opt_binary_fun, verbose = FALSE)
```

**Arguments**

- **x**: data frame
- **fun**: function which takes parameters \( x = \) data.frame, \( n = \) columns
- **verbose**: single logical value, if TRUE then produce verbose messages

**Examples**

```r
j <- data.frame(a = 1:5, b = 6:2, c = c(0, 2, 4, 6, 8))
opt_binary_brute(j)
j[1, 1] <- NA
j[1:4, 2] <- NA
my_opt_fun <- function(x, n) sum(lunlist(lapply(x, is.na)))
opt_binary_brute(j, fun = my_opt_fun)
```

### percentize

**Convert a number into rounded integer percentage string**

**Description**

The number is converted into a percentage, then rounded.

**Usage**

```r
percentize(x)
```

**Arguments**

- **x**: numeric
Examples

percentize(-1)
percentize(1L)
percentize(7.7)
percentize(0.01)
percentize(0.001)

percent_signif

Return percentage string to given significant figures

Description

From jwutil development version

Usage

percent_signif(x, figures = 3, sep = "")

Arguments

x numeric or integer values
figures integer number of significant figures to format
sep character used to separate number from percent symbol, default is empty string

permute

Generate all permutations of input

Description

Systematically permute the input vector or list, which is very slow for long x. Am amazed something this simple isn’t either in base R, or in a straightforward form in a package.

Usage

permute(x)

Arguments

x list or vector

Details

TODO: limit to a certain cut-off, after which we randomly sample
**permuteWithRepeats**

Generate all permutations of input, reusing values in each result row

**Value**

data frame, each row being one permutation

**Examples**

```r
ltr <- c("a", "b", "c")
x <- permute(ltr)
print(x)
stopifnot(nrow(x) == factorial(length(ltr)))
ltr <- c("a", "b")
x <- permute(ltr)
print(x)
stopifnot(nrow(x) == factorial(length(ltr)))
```

**Description**

Expand the given vector into all possible values in each location, with or without duplicates.

**Usage**

```r
permuteWithRepeats(x, unique = TRUE)
```

**Arguments**

- `x` : list or vector
- `unique` : logical, if TRUE, the default, only unique results are returned

**Value**

data frame, each row being one permutation

**Examples**

```r
ltr <- c("a", "b", "c")
x <- permuteWithRepeats(ltr, unique = FALSE)
print(x)
stopifnot(nrow(x) == length(ltr)^length(ltr))
# duplicate results are dropped
y <- permuteWithRepeats(c("X", "Y", "Y"))
print(y)
stopifnot(nrow(y) == 2^3)
z <- permuteWithRepeats(c("X", "Y", "Y", "Y"))
stopifnot(nrow(z) == 2^4)
a <- permuteWithRepeats(c(1, 2, 3, 1))
stopifnot(nrow(a) == 3^4)
```
**platformIsLinux**  
Are we running on Linux, Mac or Windows?

**Description**
Are we running on Linux, Mac or Windows?

**Usage**
```
platformIsLinux()
platformIsWindows()
platformIsMac()
```

**Value**
logical

---

**propIsNa**  
Proportion of NA values in a vector

**Description**
get fraction of NA in a vector

**Usage**
```
propIsNa(x)
```

**Arguments**
- **x**  
is a vector which may have NA values

**Value**
numeric proportion of NAs in the supplied vector
propNaPerField  

天使比例的NA值每个字段

**Description**

Return proportion of values which are NA in each field of the given data frame.

**Usage**

propNaPerField(x)

**Arguments**

x is a data frame

**Value**

numeric vector

---

propRowSorted  

比例非降序行在矩阵

**Description**

first performs isRowSorted to get a logical vector, then sums TRUE values and takes fraction of total

**Usage**

propRowSorted(x)

**Arguments**

x matrix, each row containing ordered or disordered numerics

**Value**

double, the proportion from 0 to 1
random_test_dates  generate random Dates or POSIXlt test datetimes

Description

generate random Dates and POSIXlt test datetimes

Usage

random_test_dates(n = n_rnd, origin = as.Date("2000-01-01"),
dayspread = 365 * 150)

random_test_posixlt_datetimes(n = n_rnd,
origin = as.Date("2000-01-01"), dayspread = 365 * 150)

Arguments

n  integer number to generate
origin  Date defaults to Jan 1, 2000.
dayspread  integer number of days either side of origin to pick random dates from, defaults to 150 years.

Value

vector of POSIXlt datetimes or Dates

random_test_numbers  create extreme random numbers

Description

create random Dates, POSIX dates, letters and numbers. The numbers explore limits of R precision and floating point and integer ranges. Zero, negatives, positives.

Usage

random_test_numbers(n = n_rnd, min = NULL, max = NULL, hole = NULL)

random_test_integers(n = n_rnd, min = -.Machine$integer.max,
max = .Machine$integer.max, hole = NULL)

random_test_letters(n = n_rnd, max_str_len = 257)
Arguments

- \( n \) integer number of each group to generate
- \( \text{min} \) optional minimum number
- \( \text{max} \) optional maximum number
- \( \text{hole} \) is a closed range of numbers not to include, e.g. \( c(1,2) \) would discard 1, 1.1 \( \pi/2 \) and 2
- \( \text{max_str_len} \) integer scalar, maximum length of possible strings created, as distinct from number of strings given by \( n \)

Value

vector length \( 5n+1 \) containing variety of difficult numbers for testing purposes

Description

currently relies on Linux xlsx2csv command, but could potentially be done with VB script in Windows. This offers a different backend to other Excel parsing functions in R

Usage

\[
\text{read_xlsx_linux} \text{(file)}
\]

Arguments

- \( \text{file} \) is the path to the .xlsx file

Value

data frame

See Also

readxl package by Hadley Wickham
read_zip_url  
**read file from zip at URL**

**Description**

Downloads zip file, and opens named file `filename`, or the single file in zip if `filename` is not specified. `FUN` is a function, with additional arguments to `FUN` given by ...  
@details TODO: update from icd package

**Usage**

```r
read_zip_url(url, filename = NULL, FUN = readLines, ...)
read_zip_url(url, filename = NULL, FUN = readLines, ...)
```

**Arguments**

- `url` character vector of length one containing URL of zip file.
- `filename` character vector of length one containing name of file to extract from zip. If not specified, and the zip contains a single file, then this single file will be used.
- `FUN` function used to process the file in the zip, defaults to `readLines`. The first argument to `FUN` will be the path of the extracted `filename`.
- `...` further arguments to `FUN`

**Functions**

- `read_zip_url`: Deprecated

---

reqinst  
**Load packages with library, installing any which are missing**

**Description**

Load packages with library, installing any which are missing

**Usage**

```r
reqinst(pkgs)
```

**Arguments**

- `pkgs` character vector of packages to load and attach, with installation if necessary
**Description**

search through environments until the variables in the list \( x \) are all gone. This doesn’t delete functions.

**Usage**

\[
\text{rm}\_r(x, \text{envir} = \text{parent.frame()})
\]

**Arguments**

- \( x \) variables to annihilate
- \( \text{envir} \) environment to start at, defaults to calling frame.

---

**Description**

File is named \( \text{varname.RData} \) with an optional suffix before \( .\text{RData} \)

**Usage**

\[
\text{save}\_\text{in}\_\text{data}\_\text{dir}(\text{var}\_\text{name}, \text{suffix} = "", \text{data}\_\text{path} = "data", \text{package}\_\text{dir} = \text{getwd()}, \text{envir} = \text{parent.frame()})
\]

**Arguments**

- \( \text{var}\_\text{name} \) character or symbol, e.g. "myvar" or \( \text{myvar} \), either of which would find \( \text{myvar} \) in the parent environment, and save it as \( \text{myvar.RData} \) in \( \text{package_root/data} \).
- \( \text{suffix} \) character scalar
- \( \text{data}\_\text{path} \) path to data directory, default is data in current directory.
- \( \text{package}\_\text{dir} \) character containing the directory root of the package tree in which to save the data. Default is the current working directory.
- \( \text{envir} \) environment in which to look for the variable to save

**Value**

invisibly returns the data
shuffle  

Shuffle a vector

Description
Randomly shuffle the order of a vector or list. This is to improve quality of bad data to throw at functions when testing.

Usage
shuffle(x)

Arguments
x  list or vector

Value
list or vector of same length as input, (probably) in a different order

Examples
set.seed(1441)
shuffle(LETTERS)

sort_clip_char  

Take clipboard contents, and write sorted character vector back

Description
Take clipboard contents, and write sorted character vector back

Usage
sort_clip_char(cl = NULL)

Arguments
c1  Name of class to give to data before sorting, default is NULL.
**source_purl**

*Extract code from knitr vignette and source it*

**Description**

Extract code from knitr vignette and source it.

**Usage**

```r
source_purl(input, documentation = 1L, ...)
```

**Arguments**

- `input`: path to file as single character string
- `documentation`: single integer value passed on to `knitr::purl`. An integer specifying the level of documentation to go the tangled script: 0 means pure code (discard all text chunks); 1 (default) means add the chunk headers to code; 2 means add all text chunks to code as roxygen comments
- `...`: further parameters passed to `source`

---

**strip**

*strip all whitespace*

**Description**

could do this with regular expression, but slow, and this function is called frequently. My only use case works with removal of all space character whitespace, and I don’t expect `<TAB>`. This uses non-unicode aware matching for speed. This can be changed by setting `useBytes` to FALSE.

**Usage**

```r
strip(x, pattern = " ", useBytes = TRUE)
```

**Arguments**

- `x`: is a character vector to strip
- `pattern`: is the non-regex of the character to strip, default " "
- `useBytes`: logical scalar. Unlike `gsub`, this will default to TRUE here, therefore breaking unicode.

**Details**

gsub is probably quicker than `stringr/stringi`. For comorbidity processing, this package prefers the faster `base` functions, whereas `stringr` is used for tasks which are not time critical, e.g. parsing source data to be included in the distributed icd package.
strip_for_formula

Value

character vector

Examples

```r
## Not run:
requireNamespace("microbenchmark")
requireNamespace("stringr")
x <- random_string(25000)
microbenchmark::microbenchmark(
  gsub(x = x, pattern = "A", replacement = "", fixed = TRUE, useBytes = TRUE),
  gsub(x = x, pattern = "A", replacement = "", fixed = TRUE, useBytes = TRUE, perl = TRUE),
  gsub(x = x, pattern = "A", replacement = ""),
  stringr::str_replace_all(x, "A", "")
)
```

## End(Not run)

strip_for_formula  strip a string so that it can be used as a variable name in a formula.

Description

This excludes many symbols, so just strip all symbols leaving alphanumeric, and no whitespace.

Usage

`strip_for_formula(x)`

Arguments

- `x`  character vector of potential formula variables

Value

character vector of length x
**str_multi_match**

return the actual matches from a bracketed regex

**Description**

Be careful: this may throw funny results for exotic regex, but so far, it seems okay. It also drops the first result which always seems to be a duplicate or whole-string match.

**Usage**

```r
str_multi_match(pattern, text, dropEmpty = FALSE, ...)

strMultiMatch(pattern, text, dropEmpty = FALSE, ...)
```

**Arguments**

- `pattern`: regular expression: if it has bracketed sections, these submatches are returned.
- `text`: is the string to match against. This vector should be the same length as the pattern vector, or the pattern vector should be length one.
- `dropEmpty`: logical whether to drop rows with no matches.
- `...`: are additional parameters passed to regexec and regmatches. I haven’t tried this: it may need two separate variables containing lists of params, since this will send everything to both functions.

**Value**

list of character vectors, list length being the length of the input text vector.

**Functions**

- `strMultiMatch`: Deprecated

---

**trim**

strip whitespace from ends of each string in given character vector

**Description**

slower than `strip`.

**Usage**

```r
trim(x)
```

**Arguments**

- `x`: is a character vector to trim
two_cat_to_logical  

Take dataframe, and convert any columns with just two categories into logical

Description

E.g. "Yes" would be converted to TRUE, "0" to FALSE, etc. If heuristics fail, then the function stops with an error message. NA values are counted, unless ignore_na is TRUE. When they are considered, na_val indicates whether they are attributed TRUE or FALSE.

Usage

two_cat_to_logical(x, ignore_na = FALSE, na_val = FALSE)

Arguments

x  
input data frame

ignore_na  
logical

na_val  
Single value to use in place of NA, default is FALSE

Value

data frame with two categories columns replaced by logical columns

Examples

df <- data.frame(  
a = c("y", "n", "y", "y", "n"),  
b = c(FALSE, TRUE, FALSE, TRUE, TRUE),  
c = c(NA, NA, NA, NA, NA),  
d = c(NA, "yes", NA, NA, "yes"),  
e = c("y ", "n ", NA, "y ", "n "),  
f = c("YES ", "NO ", "NO ", " YES"," NO"),
stringsAsFactors = FALSE  
)
df  
res <- two_cat_to_logical(df)  
stopifnot(identical(res$a, c(TRUE, FALSE, TRUE, TRUE, FALSE)))  
stopifnot(identical(res$b, c(FALSE, TRUE, FALSE, TRUE, TRUE)))
two_cat_to_logical(df, ignore_na = TRUE)
unzip_single

unzip a single file from URL

Description

take a single file from zip located at a given URL, unzip into temporary directory, and copy to the given save_path

Usage

unzip_single(url, file_name, save_path)

Arguments

url URL of a zip file
file_name file name of the resource within the zip file
save_path file path to save the first file from the zip

unzip_to_data_raw

Unzip file to data_raw

Description

Get a zip file from a URL, extract contents, and save file in data_raw. If the file already exists there, it is only retrieved if force is set to TRUE. If offline is FALSE, then NULL is returned if the file isn’t already downloaded.

Usage

unzip_to_data_raw(url, file_name, force = FALSE, verbose = FALSE, offline = TRUE, data_raw_path = "data_raw")

download_to_data_raw(url, file_name = regmatches(url, regexpr("^[^/]*\$, url)), offline = TRUE, data_raw_path = "data_raw")

Arguments

url URL of a zip file
file_name file name of a single file in that zip
force logical, if TRUE, then download even if already in data_raw
verbose single logical value, if TRUE then produce verbose messages
offline single logical, if TRUE then don’t pull the file from internet, only return path and file name if the file already exists in data_raw. This is helpful for testing without using the internet.

data_raw_path path where the data_raw directory is.
Details
The file name is changed to a conservative cross platform name using make_names.

Value
path of unzipped file in data-raw

update_github_pkgs   Update github_install packages

Description
Update github_install packages

Usage
update_github_pkgs()

Value
Returns invisibly the names of packages which need updating. The function outputs the commands to run to actually update them (by reinstalling from github). Doesn’t do this automatically because it would mean bringing in a lot of dependencies.

zeroes   zeroes

Description
long, float and complex types

Usage
zeroes

Format
An object of class list of length 3.
Zero NA values in a data.frame

Description
Zero NA values in a data.frame, including cols and excluding ignore. Also does not replace Date or POSIXt fields.

Usage
```r
zero_na(x, cols = names(x), ignore = character(), verbose = FALSE, 
        na_ish = TRUE, new_val = 0)
```

Arguments
- `x` data.frame
- `cols` names of columns to work on, default is all columns
- `ignore` character vector of columns names to ignore
- `verbose` TRUE or FALSE
- `na_ish` Logical, default ‘TRUE‘ which will convert NA-like strings, too
- `new_val` ‘0’

Examples
```r
d <- data.frame(1:5, 6:10, 11:15)
d[2, 3] <- NA
d[5, 2] <- NA
d[1, 1] <- NA
print(d)
zero_na(d)
d[1, 1] <- "NA"
zero_na(d, na_ish = TRUE)
```

%nin% inverse of %in%

Description
borrowed from Hmisc. See nomatch = 0L > 0L

Usage
```r
x %nin% table
```
Arguments

- x is the vector of values to be matched
- table is actually a vector, to be matched against

Value

logical vector of length of x
Index

*Topic manip
  logical_to_binary, 28
*Topic sysdata
  bad_input, 7
  extreme_numbers, 15
  zeroes, 50
%nin, 51
add_time_to_date, 4
affixFields, 5
areIntegers (as_numeric_nowarn), 6
areNumeric (is_numeric_str), 25
as_char_no_warn, 5
as_integer_nowarn (as_numeric_nowarn), 6
as_numeric_nowarn, 6
asIntegerNoWarn (as_numeric_nowarn), 6
bad_input, 7
base, 45
binary_col_names, 7
binary_cols (binary_col_names), 7
build_formula, 8
buildLinearFormula (build_formula), 8
combn_subset, 9
complete cases, 13
countIsNa, 10
countNonNaCumulative, 10
countNotNumeric, 11
countNumeric, 11
download_to_data_raw
  (unzip_to_data_raw), 49
dput_expect_equal, 12
drop_duplicate_fields, 12
drop_rows_with_na, 13
dropDuplicateFields
  (drop_duplicate_fields), 12
dropRowsWithNAField
  (drop_rows_with_na), 13
  expect_that, 14
  expect_that_combine_all_args, 14
  expect_that_combine_first_arg
    (expect_that_combine_all_args), 14
  extreme_numbers, 15
factor_nosort, 15
factor_to_df, 16
factorToDataFrameLogical
  (factor_to_df), 16
fillMissingCombs, 17
filter_better, 18
filterBetter (filter_better), 18
fix_na_isn, 18
flattenList, 19
floor, 6
get_factor_fields, 20
get_na_fields, 21
get_non_factor_fields
  (get_factor_fields), 20
get_non_na_fields (get_na_fields), 21
get_numeric_char_field_names, 21
get_numeric_field_names
  (get_numeric_char_field_names), 21
get_numeric_fields
  (get_numeric_char_field_names), 21
getDropped, 19
getFactorNames (get_factor_fields), 20
getNAFields (get_na_fields), 21
getNonFactorNames (get_factor_fields), 20
getNonNAFields (get_na_fields), 21
invwhich, 22
is.Date, 23
is_integerish (as_numeric_nowarn), 6
is_na_ish, 25
is_numeric_str, 25
isFlat, 23
isRowSorted, 24
isValidTime, 24
jw_df Basics, 26
jw_scan_build, 26
jwutil (jwutil-package), 3
jwutil-package, 3
list_named, 28
listTrim, 27
listTrimFlat, 27
logical_to_binary, 28
logicalToBinary (logical_to_binary), 28
ls.objects, 29
lsf, 29
lsos, 30
lsp, 30
match_multi, 31
merge, 32
merge_better, 32
mergeBetter (merge_better), 32
mergeLists, 31
min_r_version, 33

cmp, 34
numbers_to_long_and_float, 34

opt_binary_brute, 35
percent_signif, 36
percentize, 35
permute, 36
permuteWithRepeats, 37
platformIsLinux, 38
platformIsMac (platformIsLinux), 38
platformIsWindows (platformIsLinux), 38
propIsNa, 38
propNaPerField, 39
propRowSorted, 39

random_test posixlt_datetimes
  (random_test_datetimes), 40
read.zip.url (read_zip_url), 42
read_xlsx_linux, 41
read_zip_url, 42
reqinst, 42
rm_r, 43

save_in_data_dir, 43
shuffle, 44
sort_clip_char, 44
source_purl, 45
str_multi_match, 47
strip, 45
strip_for_formula, 46
strMultiMatch (str_multi_match), 47
trim, 47
two_cat_col_names (binary_col_names), 7
two_cat_cols (binary_col_names), 7
two_cat_to_logical, 48

unzip_single, 49
unzip_to_data_raw, 49
update_github_pkgs, 50

zero_na, 51
zeroes, 50