Package ‘AMsnippets’

February 12, 2013

Type Package

Title Useful function ‘‘snippets’’

Version 1.0

Date 2013-02-08

Author Ananda Mahto, Rolf Turner, Akhil S Bhel

Maintainer Ananda Mahto <mrdwab@gmail.com>

Description Most of these are small utility functions designed to reduce
thinking about solutions to repetitive tasks.

License GPL-2

Collate ’aggregate2.R’’AMsnippets-
package.R’’CBIND.R’’dfcols.list.R’’LinearizeNestedList.R’’load.scripts.and.data.R’’mv.R’’round2.R’’SampleToSum.R’’

R topics documented:

`AMsnippets-package` *An assortment of R snippets*

Description

The *AMsnippets* package is a collection of utilities that make certain repetitious or annoying tasks less repetitious or less annoying. Some of the functions were written as answers to questions at Stack Overflow; in such cases, a link to the original question has been provided.

Details

Package:	AMsnippets
Type:	Package
Version:	1.0
Date:	2013-02-08
License:	GPL-2

Author(s)

Ananda Mahto, with functions by Rolf Turner, Akhil S Bhel, Anonymous

Maintainer: Ananda Mahto <mrdwab@gmail.com>

Examples

```
## aggregate2
aggregate2(ToothGrowth, "len", ".", c("sum", "mean"))

## CBIND
df1 <- data.frame(A = 1:5, B = letters[1:5])
df2 <- data.frame(C = 1:3, D = letters[1:3])
df3 <- data.frame(E = 1:8, F = letters[1:8], G = LETTERS[1:8])
#'CBIND(list(df1, df2, df3))

## FacsToChars
dat <- data.frame(title = c("title1", "title2", "title3"),
                 author = c("author1", "author2", "author3"),
                 customerID = c(1, 2, 1))
str(dat)
FacsToChars(dat, overwrite = TRUE)
str(dat)

## makemeNA
# Some sample data
temp <- data.frame(
V1 = c(1:3),
V2 = c(1, "*", 3),
V3 = c("a", "*", "c"),
V4 = c(".", "*", "3"))
temp
makemeNA(temp, c("*", "."))
```

aggregate2 *Perform multiple aggregation functions on grouped data*

Description

Base R's `aggregate` function allows you to specify multiple functions when aggregating. However, the output of such commands is a `data.frame` where the aggregated "columns" are actually matrices. `aggregate2` is a basic wrapper around `aggregate` that outputs a regular `data.frame` instead.

Usage

```
aggregate2(data, aggs, ids, funs = NULL, ...)
```

Arguments

`data`	Your `data.frame`
`aggs`	The variables that need to be aggregated, specified as a character vector.
`ids`	The variables that serve as grouping variables, specified as a character vector.
`funs`	The functions that you want to apply, specified as a character vector.
`...`	Further arguments to `aggregate`. Really only useful for the `subset` argument.

Note

This function essentially constructs a `formula` that can be used with `aggregate` and keeps track of the names of the aggregation functions you have applied to create new variable names. This function is not very useful when the output of `FUN` would already output a matrix (for example, if `FUN = fivenum` or `FUN = summary`). In such cases, it is recommended to use base R's `aggregate` with a `do.call`. For example: `do.call("data.frame", aggregate(. ~  Species, iris, summary))`.

Author(s)

Ananda Mahto

See Also

`aggregate`

Examples

```
# One-to-one, two functions
(temp1a <- aggregate(weight ~ feed, data = chickwts,
                     function(x) cbind(mean(x), sum(x))))
str(temp1a)
(temp1b <- aggregate2(chickwts, "weight", "feed", c("mean", "sum")))
str(temp1b)

# Many-to-many, two functions
(temp2a <- aggregate(cbind(ncases, ncontrols) ~ alcgp + tobgp, data = esoph,
                     function(x) cbind(sum(x), mean(x))))
str(temp2a)
(temp2b <- aggregate2(esoph, c("ncases", "ncontrols"),
                      c("alcgp", "tobgp"), c("sum", "mean")))
str(temp2b)

# Dot notation
(temp3a <- aggregate(len ~ ., data = ToothGrowth,
                     function(x) cbind(sum(x), mean(x))))
str(temp3a)
(temp3b <- aggregate2(ToothGrowth, "len", ".", c("sum", "mean")))
str(temp3b)
```

`CBIND` *cbind* `data.frames` *with different number of rows*

Description

`cbind` does not work when trying to combine `data.frames` with differing numbers of rows. This function takes a `list` of `data.frames`, identifies how many extra rows are required to make `cbind` work correctly, and does the combining for you.

Usage

```
CBIND(datalist)
```

Arguments

`datalist`	A `list` of `data.frames` that you want to combine by columns.

Details

The `CBIND` function also works with nested lists by first "flattening" them using the `LinearizeNestedList` function by Akhil S Bhel.

Author(s)

Ananda Mahto

See Also

`cbind`, `cbindX`, `LinearizeNestedList`

Examples

```
# Example data
df1 <- data.frame(A = 1:5, B = letters[1:5])
df2 <- data.frame(C = 1:3, D = letters[1:3])
df3 <- data.frame(E = 1:8, F = letters[1:8], G = LETTERS[1:8])

CBIND(list(df1, df2, df3))

# Nested lists:
test1 <- list(list(df1, df2, df3), df1)
str(test1)

CBIND(test1)
```

`dfcols.list` *Convert the columns of a* `data.frame` *to a* `list`

Description

Sometimes, it is useful to have the columns of a `data.frame` as separate list items or vectors. `unlist` is useful for creating a single vector, but not for creating multiple vectors. The `dfcols.list` function is a simple convenience function that allows for such transformations.

Usage

```
dfcols.list(data, vectorize = TRUE)
```

Arguments

`data`	The input `data.frame`
`vectorize`	Logical. Should the function return a list of single-column `data.frame`s, or a simple `vector` of values? Defaults to `TRUE`.

Author(s)

Ananda Mahto

Examples

```
dat <- data.frame(A = c(1:2), B = c(3:4), C = c(5:6))
dfcols.list(dat)
dfcols.list(dat, vectorize = FALSE)
```

FacsToChars *Convert all* `factor` *columns to* `character` *columns in a* `data.frame`

Description

Sometimes, we forget to use the `stringsAsFactors` argument when using `read.table` and related functions. By default, R converts character columns to factors. Instead of re-reading the data, the `FacsToChars` function will identify which columns are currently factors, and convert them all to characters.

Usage

```
FacsToChars(mydf, overwrite = FALSE)
```

Arguments

`mydf`	The name of your `data.frame`
`overwrite`	Logical. Should the current object be overwritten? Defaults to `FALSE`

Author(s)

Ananda Mahto

See Also

`read.table`

Examples

```
## Some example data
dat <- data.frame(title = c("title1", "title2", "title3"),
                  author = c("author1", "author2", "author3"),
                  customerID = c(1, 2, 1))
## Make a copy of dat for later
dat_copy <- dat
str(dat) # current structure
dat2 <- FacsToChars(dat)
str(dat2) # Your new object
str(dat)  # Original object is unaffected

## You can also overwrite the existing object
str(dat_copy) # Before applying the function
FacsToChars(dat_copy, overwrite=TRUE)
str(dat_copy) # After applying the function
```

`LinearizeNestedList` *Linearize (un-nest) nested lists*

Description

Implements a recursive algorithm to linearize nested `lists` upto any arbitrary level of nesting (limited by R's allowance for recursion-depth). By linearization, it is meant to bring all list branches emanating from any nth-nested trunk upto the top-level trunk such that the return value is a simple non-nested list having all branches emanating from this top-level branch.

Usage

```
LinearizeNestedList(NList, LinearizeDataFrames = FALSE,
  NameSep = "/", ForceNames = FALSE)
```

Arguments

`NList`	The input `list`
`LinearizeDataFrames`	Logical. Should columns in `data.frames` in the list be "linearized" as vectors? Defaults to `FALSE`.
`NameSep`	Character to be used when creating names. Defaults to "/" to mimic directory listings.
`ForceNames`	Logical. Should the present names be discarded and new simplified names be created? Defaults to `FALSE`

Details

Since `data.frames` are essentially `lists` a boolean option is provided to switch on/off the linearization of `data.frames`. This has been found desirable in the author's experience.

Also, one would typically want to preserve names in the lists in a way as to clearly denote the association of any list element to its nth-level history. As such we provide a clean and simple method of preserving names information of list elements. The names at any level of nesting are appended to the names of all preceding trunks using the `NameSep` option string as the seperator. The default "/" has been chosen to mimic the unix tradition of filesystem hierarchies. The default behavior works with existing names at any n-th level trunk, if found; otherwise, coerces simple numeric names corresponding to the position of a list element on the nth-trunk. Note, however, that this naming pattern does not ensure unique names for all elements in the resulting list. If the nested lists had non-unique names in a trunk the same would be reflected in the final list. Also, note that the function does not at all handle cases where *some* names are missing and some are not.

Clearly, preserving the n-level hierarchy of branches in the element names may lead to names that are too long. Often, only the depth of a list element may only be important. To deal with this possibility a boolean option called `ForceNames` has been provided. `ForceNames` shall drop all original names in the lists and coerce simple numeric names which simply indicate the position of an element at the nth-level trunk as well as all preceding trunk numbers.

Author(s)

Akhil S Bhel

References

https://sites.google.com/site/akhilsbehl/geekspace/articles/r/linearize_nested_lists_in_r

See Also

unlist

Examples

```
# Create some sample data
NList <- list(a = "a", # Atom
              b = 1:5, # Vector
              c = data.frame(x = runif(5), y = runif(5)), # Add a data.frame
              d = matrix(runif(4), nrow = 2), # Throw in a matrix for good measure
              e = list(l = list("a", "b"), # Introduce nesting
                       m = list(1:5, 5:10),
                       n = list(list(1), list(2)))) # More nesting

LinearizeNestedList(NList)
LinearizeNestedList(NList, LinearizeDataFrames = TRUE)
LinearizeNestedList(NList, ForceNames = TRUE)
LinearizeNestedList(NList, ForceNames = TRUE, NameSep = "_")
```

`load.scripts.and.data` *Load all script and data files from specified directories*

Description

A convenience function to read all the data files and scripts from specified directories. In general, should only need to specify the directories. Specify directories without trailing slashes.

Usage

```
load.scripts.and.data(path,
  pattern = list(scripts = "*.R$", data = "*.rda$|*.Rdata$"),
  ignore.case = TRUE)
```

Arguments

`path`	A character vector of file paths.
`pattern`	A named list of patterns to match for loading scripts and data files. See "Notes".
`ignore.case`	Logical. Should letter case be considered when searching for data files and script files? Defaults to `FALSE`.

Note

The pre-defined pattern is `list(scripts = "*.R$",  data = "*.rda$|*.Rdata$")`. This should match most conventionally used file extensions for R's native script and data files. Alternative patterns should be specified in the same form.

Author(s)

Ananda Mahto

Examples

```
## Not run:
load.scripts.and.data(c("~/Dropbox/Public",
                        "~/Dropbox/Public/R Functions"))

## End(Not run)
```

`makemeNA` *Make certain values in a* `data.frame` `NA`

Description

Sometimes, after having read in data, one needs to replace certain values by `NA`. One approach is to use `mydf[mydf == "some-character"] <- NA`. However, in many cases that results in a `data.frame` where variables which should be numeric end up as characters or factors if the NA string was a character to begin with. This function is a convenience wrapper around `type.convert` to address such problems.

Usage

```
makemeNA(mydf, NAStrings)
```

Arguments

`mydf`	A `data.frame` in which some values need to be converted to `NA`
`NAStrings`	The values which have been used to represent NA

Author(s)

Ananda Mahto

See Also

`type.convert`

Examples

```
# Some sample data
temp <- data.frame(
V1 = c(1:3),
V2 = c(1, "*", 3),
V3 = c("a", "*", "c"),
V4 = c(".", "*", "3"))
temp
str(temp)

temp1 <- makemeNA(temp, c("*", "."))
temp1
str(temp1)

# Can make anything NA. Useful for -999 type of NA values
makemeNA(temp, "1")
```

`mv` *Rename an object in the workspace*

Description

Renames an object in the workspace, "removing" the orinal object. This does so without creating a copy of the original object. If an object in the workspace currently exists with the new name specified, the function prompts the user to verify that they want to overwrite that object before proceeding.

Usage

```
mv(currentName, newName)
```

Arguments

`currentName`	The current name of the object
`newName`	The new name for the object

Author(s)

Rolf Turner

References

A good amount of discussion on when R makes a copy in memory in this discussion thread: `https://stat.ethz.ch/pipermail/r-help/2008-March/156028.html`.

Examples

```
x <- runif(1e7)
ls()
x.add <- tracemem(x)
mv(x, y)
identical(x.add, tracemem(y))
ls()
```

RBIND *Append* `data.frame`*s by row, even when columns differ*

Description

The default `rbind` function will produce an error if you attempt to use it on `data.frame`s with differing numbers of columns. The `RBIND` function appends a list of `data.frames` together by row, filling missing columns with `NA`.

Usage

```
RBIND(datalist, keep.rownames = TRUE)
```

Arguments

`datalist`	A `list` of `data.frames` which need to be appended together by row.
`keep.rownames`	Logical. Should the original `rownames` be retained? Defaults to `TRUE`.

Author(s)

Ananda Mahto

See Also

`rbind` and `cbind` for other base R functions to combine `data.frames`; `rbind.fill` for a function with almost identical functionality (does not preserve the `rownames`); `CBIND`.

Examples

```
## Make up some data
x <- data.frame(a = 1:2, b = 2:3, c = 3:4, d = 4:5,
                row.names = c("row_1", "another_row1"))
y <- data.frame(a = c(10, 20), b = c(20, 30), c = c(30, 40),
                row.names=c("row_2", "another_row2"))
z <- data.frame(a = c(11, 21), b = c(22, 32), d = c(33, 43),
                row.names = c("row_3", "another_row3"))
xx <- data.frame(a = 1:2, b = 3:4)
yy <- data.frame(a = 5:6, b = 7:8)
zz <- data.frame(a = 9:10, b = 11:12)
```

```
zz2 <- data.frame(a = 9:10, w = 11:12)
temp1 <- list(x, y, z)
temp2 <- list(xx, yy, zz)
temp3 <- list(xx, yy, zz2)
temp4 <- list(x, y, z, xx, yy, zz, zz2)

## Apply the function
RBIND(temp1)
RBIND(temp1, keep.rownames = FALSE)
RBIND(temp2)
RBIND(temp3)
RBIND(temp4)
RBIND(temp4, keep.rownames = FALSE)
```

round2 — *Round numbers the way you learned in school*

Description

The round2 function rounds numbers in the way you probably learned in school, that is, round up to the next number for values of 5 and above.

Usage

```
round2(x, digits = 0)
```

Arguments

x	The number (or vector of numbers) that needs rounding.
digits	The number of decimal places in the output.

Details

To reduce bias in rounding, R's round function uses a "round-to-even" approach. Still, many people are surprised when they find that R's round function will return the same value for round(1.5) and round(2.5). This function uses the rounding approach found in most school lessons and in software like Excel to make the results comparable.

Author(s)

Unknown (see "References")

References

Function originally found in an anonymous comment at the Statistically Significant blog. See http://www.webcitation.org/68djeLBtJ

See Also

round

Examples

```
input <- seq(from = 0.5, by = 1, length.out = 10)
round(input)
round2(input)
round(input/10, digits = 1)
round2(input/10, digits = 1)
```

SampleToSum *Draw a random sample that sums to a specified amount*

Description

This function creates a random sample of numbers drawn from a specified range which sum to a specified amount.

Usage

```
SampleToSum(Target = 100, VecLen = 10, InRange = 1:100,
  Tolerance = 2, writeProgress = NULL)
```

Arguments

`Target`	The desired sum of all the samples
`VecLen`	How many numbers should be in your resulting vector?
`InRange`	What is the acceptable range of values to be sampled from?
`Tolerance`	What is the maximum difference allowed between the target and the sum? Set to "0" to match the target exactly. In general, the difference is within 5 anyway, which is reasonable.
`writeProgress`	If you want a log-file to be written that includes *all* the variations tried before arriving at a vector that satisfies all the user's conditions, specify the output file name (quoted) with this argument. Note that in some cases, this might be quite a large file with tens-of-thousands of lines!

Note

This function can be notoriously slow, particularly if your range is too narrow and your tolerance is too high.

Author(s)

Ananda Mahto

References

This function was written as a response to the following Stack Overflow question: http://stackoverflow.com/q/14684539/1270695

See Also

sample, runif

Examples

```
set.seed(1)
SampleToSum()
SampleToSum(Tolerance = 0)
replicate(5,
         SampleToSum(Target = 1376,
                     VecLen = 13,
                     InRange = 10:200,
                     Tolerance = 0),
         simplify = FALSE)
replicate(5,
         SampleToSum(Target = 1376,
                     VecLen = 13,
                     InRange = 10:200),
         simplify = FALSE)
```

---

subsequence *Identify sequences in a vector*

Description

The subsequence function is like the inverse of rep, and is somewhat related to rle. It detects the sequence in a vector and returns the period of the sequence, the actual sequence, the number of times the sequence is repeated, and optionally, a "Groups" vector the same length as the input vector that can be used as a grouping variable.

Usage

```
subsequence(data, groups = FALSE)
```

Arguments

data	The input vector
groups	Logical. Should the grouping vector be returned?

Author(s)

Ananda Mahto

References

This function was written as an answer to the following Stack Overflow question: http://stackoverflow.com/q/12824931/1270695

See Also

`rep`, `rle`,

Examples

```
## Some sample data
s1a <- rep(c(1, 2, 3), 3)
s1b <- c(s1a, 1)
s2 <- rep(c(1, 2, 3), 50)
s3 <- c(1, 2, 3, 4, 2, 3, 4, 1, 2, 3, 4, 2, 3, 4)
set.seed(1)
s4 <- rep(sample(300, 15), 5)

subsequence(s1a)
## Note the creation of a grouping variable
subsequence(s1b, groups = TRUE)
subsequence(s2)
subsequence(s3)
subsequence(s4)
```

`unlistDF` *"Unlist" a `list` of `data.frames` to your workspace*

Description

Many people like the convenience that a `list` of `data.frames` offer; however, some would prefer to have each `data.frame` as a separate object in their workspace. This function "unlists" a `list` of `data.frames`, creating objects named after the `list` and the list item's names (or index position, if names are not available).

Usage

```
unlistDF(mylist)
```

Arguments

`mylist`	The name of your `list` object

Author(s)

Ananda Mahto

See Also

unlist

Examples

```
## Some example data
## List with named items
qwerty <- list(A = data.frame(A = 1:2, B = 3:4),
              B = data.frame(C = 5:6, D = 7:8))

## List with unnamed items
ytrewq <- list(data.frame(A = 1:2, B = 3:4),
              data.frame(C = 5:6, D = 7:8))

ls(pattern = "qwer|ytre")

unlistDF(qwerty)
unlistDF(ytrewq)

ls(pattern = "qwer|ytre")
```

Index