Package 'rowr'

December 22, 2016

Title Row-Based Functions for R Objects

Version 1.1.3

Date 2016-12-19

Author Craig Varrichio <canthony427@gmail.com>

Maintainer Craig Varrichio <canthony427@gmail.com>

Description Provides utilities which interact with all R objects as
if they were arranged in rows. It allows more consistent and predictable
output to common functions, and generalizes a number of utility functions to
to be failsafe with any number and type of input objects.

Depends R (>= 3.0.1)

Imports methods

License GPL-3

URL https://github.com/cvarrichio/rowr

LazyData true

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-12-22 00:27:27

R topics documented:

`as2` *A more robust form of the R* `as` *function.*

Description

Alternative to `as` that allows any data object to be converted to any other.

Usage

```
as2(object, class)
```

Arguments

`object`	any `R` object
`class`	the name of the class to which `object` should be coerced

`buffer` *Pads an object to a desired length, either with replicates of itself or another repeated object.*

Description

Pads an object to a desired length, either with replicates of itself or another repeated object.

Usage

```
buffer(x, length.out = len(x), fill = NULL, preserveClass = TRUE)
```

Arguments

`x`	an R object
`length.out`	the desired length of the final output
`fill`	R object to fill empty rows in columns below the max size. If unspecified, repeats input rows in the same way as `cbind`.
`preserveClass`	determines whether to return an object of the same class as the original argument. Otherwise, returns a matrix.

Examples

```
buffer(c(1,2,3),20)
buffer(matrix(c(1,2,3,4),nrow=2),20)
buffer(list(1,2,3),20)
df<-data.frame(as.factor(c('Hello','Goodbye')),c(1,2))
buffer(df,5)
buffer((factor(x=c('Hello'))),5)
```

`cbind.fill` *Combine arbitrary data types, filling in missing rows.*

Description

Robust alternative to `cbind` that fills missing values and works on arbitrary data types. Combines any number of R objects into a single matrix, with each input corresponding to the greater of 1 or ncol. `cbind` has counterintuitive results when working with lists, cannot handle certain inputs of differing length, and does not allow the fill to be specified.

Usage

```
cbind.fill(..., fill = NULL)
```

Arguments

`...`	any number of R data objects
`fill`	R object to fill empty rows in columns below the max size. If unspecified, repeats input rows in the same way as `cbind`. Passed to `buffer`.

Examples

```
cbind.fill(c(1,2,3),list(1,2,3),cbind(c(1,2,3)))
cbind.fill(rbind(1:2),rbind(3:4))
df<-data.frame(a=c(1,2,3),b=c(1,2,3))
cbind.fill(c(1,2,3),list(1,2,3),cbind(c('a','b')),'a',df)
cbind.fill(a=c(1,2,3),list(1,2,3),cbind(c('a','b')),'a',df,fill=NA)
```

`coalesce` *A more versatile form of the T-SQL* `coalesce()` *function.*

Description

Little more than a wrapper for `vectorize`, allows for duplication of SQL coalesce functionality, certain types of if-else statements, and `apply`/`Reduce` combinations.

Usage

```
coalesce(..., fun = (function(x, y) if (!is.na(x)) x else y))
```

Arguments

...	an arbitrary number of R objects
fun	a two argument function that returns an atomic value

Examples

```
coalesce(c(NA,1,2))
coalesce(c(NA,1,2),c(3,4,NA))
df<-data.frame(a=c(NA,2,3),b=c(1,2,NA))
coalesce(df$a,df$b)
# Or even just:
coalesce(df)
# Coalesce can actually use any comparison.  For example, instead of non-NA
# values it could find the max in each row:
cbind(EuStockMarkets,Max=coalesce(EuStockMarkets,fun=function (x,y) if (x>y) x else y))
```

---

count *A more versatile form of the T-SQL* count() *function.*

Description

Implementation of T-SQL count and Excel COUNTIF functions. Shows the total number of elements in any number of data objects altogether or that match a condition.

Usage

```
count(..., condition = (function(x) TRUE))
```

Arguments

...	an arbitrary number of R objects
condition	a 1 argument condition

Examples

```
count(c(NA,1,2))
count(c(NA,1,2),is.na)
count(c(NA,1,2),list('A',4),cbind(1,2,3))
count(c(NA,1,2),list('A',4),cbind(1,2,3),condition=is.character)
```

insertRows — *Inserts a matrix into another matrix.*

Description

Inserts a matrix or data frame into another matrix or data frame. The new rows are placed together at the row index specified.

Usage

```
insertRows(existing, insert, r)
```

Arguments

existing	table to insert into
insert	rows to insert
r	index at which to insert

Examples

```
df1<-data.frame(a=c(1,2,3),b=c(1,2,3),c=c(1,2,3))
insertRows(df1,data.frame(list('a','a','a')),5)
insertRows(df1,data.frame(list('a','a','a')),4)
insertRows(df1,data.frame(list('a','a','a')),3)
insertRows(df1,data.frame(list('a','a','a')),2)
insertRows(df1,data.frame(list('a','a','a')),1)
insertRows(df1,df1,3)
```

len — *Allows finding the 'length' without knowledge of dimensionality.*

Description

Allows finding the 'length' without knowledge of dimensionality.

Usage

```
len(data)
```

Arguments

data	any R object

Examples

```
len(list(1,2,3))
len(c(1,2,3,4))
df<-data.frame(a=c(1,2,3),b=c(1,2,3))
len(df)
```

`rollApply` *Applies a function over a rolling window on any data object.*

Description

Simple generalized alternative to `rollapply` in package `zoo` with the advantage that it works on any type of data structure (vector, list, matrix, etc) instead of requiring a `zoo` object.

Usage

```
rollApply(data, fun, window = len(data), minimum = 1, align = "left", ...)
```

Arguments

`data`	any R object
`fun`	the function to evaluate
`window`	window width defining the size of the subset available to the fun at any given point
`minimum`	minimum width of the window. Will not return results if the window is truncated below this value at the end of the data set
`align`	whether to align the window right or left
`...`	additional arguments to pass to `fun`

Examples

```
rollApply(1:100,sum,minimum=2,window=2)
rollApply(c(1,2,3),sum)
##6 5 3
rollApply(c(1,2,3,4,5,6,7,8,9),sum)
##45 44 42 39 35 30 24 17  9
rollApply(c(1,2,3,4,5,6,7,8,9),sum,window=2)
##3  5  7  9 11 13 15 17  9
rollApply(list(1,2,3,4,5,6,7,8,9),function(x) sum(unlist(x)),window=2,minimum=2)
##3  5  7  9 11 13 15 17
cbind(women,Rolling3=rollApply(women,fun=function(x) mean(x$weight),window=3,align='right'))
```

rowApply *Applies a function row-wise on any data object.*

Description

Essentially functions as a MARGIN=1 apply apply but also works on data objects without 2 dimensions such as lists and vectors.

Usage

```
rowApply(data, fun, ...)
```

Arguments

data	any R object
fun	the function to evaluate
...	additional arguments to pass to fun

Examples

```
rowApply(list(1,2,3),function (x) sum(unlist(x)))
df<-data.frame(a=c(1,2,3),b=c(1,2,3))
rowApply(df,sum)
```

rowr *Row-Based Functions for R Objects*

Description

Provides utilities which interact with all R objects as if they were arranged in rows. It allows more consistent and predictable output to common functions, and generalizes a number of utility functions to to be failsafe with any number and type of input objects.

rows *Allows row indexing without knowledge of dimensionality or class.*

Description

Allows row indexing without knowledge of dimensionality or class.

Usage

```
rows(data, rownums)
```

Arguments

data	any R object
rownums	indices of target rows

Examples

```
rows(c('A','B','C'),c(1,3))
rows(list('A','B','C'),c(1,3))
df<-data.frame(a=c(1,2,3),b=c(1,2,3))
rows(df,3)
```

vectorize *Vectorize a scalar function to work on any R object.*

Description

Robust alternative to Vectorize function that accepts any function with two or more arguments. Returns a function that will work an arbitrary number of vectors, lists or data frames, though output may be unpredicatable in unusual applications. The results are also intended to be more intuitive than Vectorize.

Usage

```
vectorize(fun, type = NULL)
```

Arguments

fun	a two or more argument function
type	like MARGIN in apply, except that c(1,2) is represented as a 3 instead. By default, will Reduce single dimensional data handle everything else row-wise.

Examples

```
vectorize(`+`)(c(1,2,3))
vectorize(sum)(c(1,2,3),c(1,2,3))
# Compare these results to Vectorize, which does not vectorize sum at all.
Vectorize(sum)(c(1,2,3),c(1,2,3))
# Across data frame columns.
df<-data.frame(a=c(1,2,3),b=c(1,2,3))
vectorize(sum)(df$a,df$b)
# Once again, Vectorize gives a different result
Vectorize(sum)(df$a,df$b)
# Any combination of vectors, lists, matrices, or data frames an be used.
vectorize(`+`)(c(1,2,3),list(1,2,3),cbind(c(1,2,3)))
```

Index