Package 'rrapply'

July 5, 2020

Type Package

Title Revisiting Base Rapply

Version 1.1.0

Date 2020-07-04

Description The 'rrapply'-package contains a single function rrapply(), providing an extended implementation of 'R'-base rapply() by allowing to recursively apply a function to elements of a nested list based on a general condition function and including the possibility to prune or aggregate nested list elements from the result. In addition, special arguments can be supplied to access the name and location in the nested list of the element under evaluation. The rrapply() function is implemented in 'R''s 'C' interface and requires no other package dependencies.

BugReports https://github.com/JorisChau/rrapply/issues

URL https://jorischau.github.io/rrapply/, https://github.com/JorisChau/rrapply

Depends R (>= 3.5)

Encoding UTF-8

License GPL-3

LazyData true

RoxygenNote 7.1.0

NeedsCompilation yes

Author Joris Chau [aut, cre]

Maintainer Joris Chau <joris.chau@openanalytics.eu>

Repository CRAN

Date/Publication 2020-07-04 22:50:03 UTC

R topics documented:

renewable_energy_by_country *UNSD renewable energy share by country in 2016*

Description

A nested list containing renewable energy shares as a percentage in the total energy consumption per country in 2016. The dataset is publicly available at the United Nations Open SDG Data Hub.

Usage

```
renewable_energy_by_country
```

Format

The 249 countries and areas are structured as a nested list based on their geographical location according to the United Nations M49 Standard (UNSD-M49). The numeric values listed for each country or area are percentages, if no data is available the value is `NA`. Each list element contains an `"M49-code"` attribute with the UN Standard Country or Area Codes for Statistical Use (Series M, No. 49).

Source

UNSD_SDG07

rrapply *Reimplementation of base-R's rapply*

Description

`rrapply` is a reimplemented and extended version of `rapply` to recursively apply a function `f` to a set of elements of a list and deciding *how* the result is structured.

Usage

```
rrapply(
  object,
  condition,
  f,
  classes = "ANY",
  deflt = NULL,
  how = c("replace", "list", "unlist", "prune", "flatten", "melt"),
  feverywhere = NULL,
  dfaslist = TRUE,
  ...
)
```

Arguments

object	a list or expression, i.e., "list-like".
condition	a condition function of one "principal" argument and optional special arguments .xname and/or .xpos (see 'Details'), passing further arguments via ....
f	a function of one "principal" argument and optional special arguments .xname and/or .xpos (see 'Details'), passing further arguments via ....
classes	character vector of class names, or "ANY" to match any class.
deflt	the default result (only used if how = "list" or how = "unlist").
how	character string partially matching the five possibilities given: see 'Details'.
feverywhere	character options "break" or "recurse" to override default behavior of f: see 'Details'. By default NULL, which applies f only to non "list-like" elements.
dfaslist	logical value to treat data.frames as "list-like" object.
...	additional arguments passed to the call to f and condition

Value

If how = "unlist", a vector as in rapply. If how = "list" or how = "replace", "list-like" of similar structure as object as in rapply. If how = "prune", a pruned "list-like" object of similar structure as object with pruned list elements based on classes and condition. If how = "flatten", an unnested pruned list with pruned list elements based on classes and condition. If how = "melt", a melted data.frame containing the node paths and values of the pruned list elements based on classes and condition.

List pruning

In addition to rapply's modes to set how equal to "replace", "list" or "unlist", three choices "prune", "flatten" and "melt" are available. how = "prune" filters all list elements not subject to application of f from the list object. The original list structure is retained, similar to the non-pruned options how = "replace" or how = "list". how = "flatten" is an efficient way to return a flattened unnested version of the pruned list. how = "melt" returns a melted data.frame of the pruned list, each row contains the path of a single terminal node in the pruned list at depth layers L1, L2, and so on. The list-column "value" contains the values at the terminal nodes and is equivalent to the flattened list returned by how = "flatten". If no list names are present, the node names in the data.frame default to the indices of the list elements "..1", "..2", etc.

Condition function

Both rapply and rrapply allow to apply f to list elements of certain classes via the classes argument. rrapply generalizes this concept via an additional condition argument, which accepts any function to use as a condition or predicate to select list elements to which f is applied. Conceptually, the f function is applied to all list elements for which the condition function exactly evaluates to TRUE similar to isTRUE. If the condition function is missing, f is applied to all list elements. Since the condition function generalizes the classes argument, it is allowed to use the deflt argument together with how = "list" or how = "unlist" to set a default value to all list elements for which the condition does not evaluate to TRUE.

Correct use of ...

The principal argument of the `f` and `condition` functions evaluates to the content of the list element. Any further arguments to `f` and `condition` (besides the special arguments `.xname` and `.xpos` discussed below) supplied via the dots `...` argument need to be defined as function arguments in *both* the `f` and `condition` function (if existing), even if they are not used in the function itself. See also the 'Examples' section.

Special arguments .xname and .xpos

The `f` and `condition` functions accept two special arguments `.xname` and `.xpos` in addition to the first principal argument. The `.xname` argument evaluates to the name of the list element. The `.xpos` argument evaluates to the position of the element in the nested list structured as an integer vector. That is, if `x = list(list("y","z"))`, then an `.xpos` location of `c(1,2)` corresponds to the list element `x[[c(1,2)]]`. The names `.xname` and `.xpos` need to be explicitly included as function arguments in `f` and `condition` (in addition to the principal argument). See the package vignette for example uses of the `.xname` and `.xpos` variables.

List node aggregation

By default, `rrapply` recurses into any "list-like" element. If `feverywhere = "break"`, this behavior is overridden and the `f` function is applied to any element of `object` (e.g. a sublist) that satisfies `condition` and `classes`. If the `condition` or `classes` arguments are not satisfied for a "list-like" element, `rrapply` will recurse further into the sublist, apply the `f` function to the nodes that satisfy `condition` and `classes`, and so on. The primary use of `feverywhere = "break"` is to aggregate list nodes or summarize sublists of `object`. Additional examples can be found in the package vignette.

List node updating

If `feverywhere = "recurse"`, `rrapply` applies the `f` function to any element of `object` (e.g. a sublist) that satisfies `condition` and `classes` similar to `feverywhere = "break"`, but recurses further into the *updated* list-like element after application of the `f` function. The primary use of `feverywhere = "recurse"` is to recursively update for instance the names of all nodes in a nested list. Additional examples are found in the package vignette.

Data.frames as lists

If `feverywhere = NULL` (default), `rrapply` recurses into all "list-like" objects equivalent to `rapply`. Since data.frames are "list-like" objects, the `f` function will descend into the individual columns of a data.frame. If `dfaslist = FALSE`, data.frames are no longer viewed as lists and the `f` and `condition` functions are applied directly to the data.frame itself and not its columns.

List attributes

In `rapply` intermediate list attributes (not located at the leafs) are kept when `how = "replace"`, but are dropped when `how = "list"`. To avoid unexpected behavior, `rrapply` always preserves intermediate list attributes when using `how = "replace"`, `how = "list"` or `how = "prune"`. If `how = "flatten"` or `how = "unlist"` intermediate list attributes cannot be preserved as the result is no longer a nested list.

Note

rrapply allows the f function argument to be missing, in which case no function is applied to the list elements.

See Also

rapply

Examples

```
# Example data

## Nested list of renewable energy (%) of total energy consumption per country in 2016
data("renewable_energy_by_country")
## Subset values for countries and areas in Oceania
renewable_oceania <- renewable_energy_by_country[["World"]]["Oceania"]

# List pruning

## Drop all logical NA's while preserving list structure
na_drop_oceania <- rrapply(
  renewable_oceania,
  f = function(x) x,
  classes = "numeric",
  how = "prune"
)
str(na_drop_oceania, list.len = 3, give.attr = FALSE)

## Drop all logical NA's and return unnested list
na_drop_oceania2 <- rrapply(
  renewable_oceania,
  f = function(x) x,
  classes = "numeric",
  how = "flatten"
)
str(na_drop_oceania2, list.len = 10, give.attr = FALSE)

## Drop all logical NA's and return melted data.frame
na_drop_oceania3 <- rrapply(
  renewable_oceania,
  f = identity,
  classes = "numeric",
  how = "melt"
)
head(na_drop_oceania3)

# Condition function

## Drop all NA elements using condition function
na_drop_oceania3 <- rrapply(
  renewable_oceania,
  condition = Negate(is.na),
```

```
  f = function(x) x,
  how = "prune"
)
str(na_drop_oceania3, list.len = 3, give.attr = FALSE)

## Replace NA elements by a new value via the ... argument
## NB: the 'newvalue' argument should be present as function
## argument in both 'f' and 'condition', even if unused.
na_zero_oceania <- rrapply(
  renewable_oceania,
  condition = function(x, newvalue) is.na(x),
  f = function(x, newvalue) newvalue,
  newvalue = 0,
  how = "replace"
)
str(na_zero_oceania, list.len = 3, give.attr = FALSE)

## Filter all countries with values above 85%
renewable_energy_above_85 <- rrapply(
  renewable_energy_by_country,
  condition = function(x) x > 85,
  how = "prune"
)
str(renewable_energy_above_85, give.attr = FALSE)

# Special arguments .xname and .xpos

## Apply a function using the name of the node
renewable_oceania_text <- rrapply(
  renewable_oceania,
  f = function(x, .xname) sprintf("Renewable energy in %s: %.2f%%", .xname, x),
  condition = Negate(is.na),
  how = "flatten"
)
str(renewable_oceania_text, list.len = 10)

## Extract values based on country names
renewable_benelux <- rrapply(
  renewable_energy_by_country,
  condition = function(x, .xname) .xname %in% c("Belgium", "Netherlands", "Luxembourg"),
  how = "prune"
)
str(renewable_benelux, give.attr = FALSE)

## Filter European countries with value above 50%
renewable_europe_above_50 <- rrapply(
  renewable_energy_by_country,
  condition = function(x, .xpos) identical(.xpos[c(1, 2)], c(1L, 5L)) & x > 50,
  how = "prune"
)
str(renewable_europe_above_50, give.attr = FALSE)

## Return position of Sweden in list
```

```
(xpos_sweden <- rrapply(
  renewable_energy_by_country,
  condition = function(x, .xname) identical(.xname, "Sweden"),
  f = function(x, .xpos) .xpos,
  how = "flatten"
))
renewable_energy_by_country[[xpos_sweden$Sweden]]

# List node aggregation

## Calculate mean value of Europe
rrapply(
  renewable_energy_by_country,
  condition = function(x, .xname) .xname == "Europe",
  f = function(x) mean(unlist(x), na.rm = TRUE),
  how = "flatten",
  feverywhere = "break"
)

## Calculate mean value for each continent
renewable_continent_summary <- rrapply(
  renewable_energy_by_country,
  condition = function(x, .xpos) length(.xpos) == 2,
  f = function(x) mean(unlist(x), na.rm = TRUE),
  feverywhere = "break"
)

## Antarctica's value is missing
str(renewable_continent_summary, give.attr = FALSE)

# List node updating

## replace country names by M-49 codes
renewable_M49 <- rrapply(
  list(renewable_energy_by_country),
  condition = is.list,
  f = function(x) {
    names(x) <- vapply(x, attr, character(1L), which = "M49-code")
    return(x)
  },
  feverywhere = "recurse"
)

str(renewable_M49[[1]], max.level = 3, list.len = 3, give.attr = FALSE)

# List attributes

## how = "list" preserves all list attributes
na_drop_oceania_attr <- rrapply(
  renewable_oceania,
  f = function(x) replace(x, is.na(x), 0),
  how = "list"
)
```

```
str(na_drop_oceania_attr, max.level = 2)

## how = "prune" also preserves list attributes
na_drop_oceania_attr2 <- rrapply(
  renewable_oceania,
  condition = Negate(is.na),
  how = "prune"
)
str(na_drop_oceania_attr2, max.level = 2)

# Applying functions to data.frames

## Scale only Sepal columns in iris dataset
head(
  rrapply(
    iris,
    condition = function(x, .xname) grepl("Sepal", .xname),
    f = scale
  )
)

## Scale and keep only numeric columns
head(
  rrapply(
    iris,
    f = scale,
    classes = "numeric",
    how = "prune"
  )
)

## Summarize only numeric columns with how = "flatten"
rrapply(
  iris,
  f = summary,
  classes = "numeric",
  how = "flatten"
)
```

Index