Package ‘mrdwabmisc’

January 27, 2013

Type Package

Title Miscellaneous R functions, mostly for data processing

Version 1.0

Date 2013-01-22

Author Ananda Mahto

Maintainer Ananda Mahto <mrdwab@gmail.com>

Description Miscellaneous R functions, some utility, and others to clean and organize data.

License CC-SA

Collate
'concat.split.R''df.sorter.R''multi.freq.table.R''random.names.R''row.extractor.R''sample.size.R'

R topics documented:

concat.split *Split concatenated cells in a* data.frame

Description

The concat.split function takes a column with multiple values, splits the values into a list or into separate columns, and returns a new data.frame.

Usage

```
concat.split(data, split.col, sep = ",",
  structure = "compact", mode = NULL, drop.col = FALSE,
  fixed = FALSE)
```

Arguments

data	The source data.frame
split.col	The variable that needs to be split; can be specified either by the column number or the variable name.
sep	The character separating each value (defaults to ",").
structure	Can be either "compact", "expanded", or "list". Defaults to "compact". See Details.
mode	Can be either binary or value (where binary is default and it recodes values to 1 or NA, like Boolean data, but without assuming 0 when data is not available). This setting only applies when structure = "expanded"; an warning message will be issued if used with other structures.
drop.col	Logical (whether to remove the original variable from the output or not). Defaults to TRUE.
fixed	Is the input for the sep value *fixed*, or a *regular expression*? See Details.

Details

structure

- "compact" creates as many columns as the maximum length of the resulting split. This is the most useful general-case application of this function.
- When the input is numeric, "expanded" creates as many columns as the maximum value of the input data. This is most useful when converting to mode = "binary".
- "list" creates a single new column that is structurally a list within a data.frame.

fixed When structure = "expanded" or structure = "list", it is possible to supply a a regular expression containing the characters to split on. For example, to split on ",", ";", or "|", you can set sep = ",|;|\|" or sep = "[,;|]", and fixed = FALSE to split on any of those characters.

Note

If using `structure = "compact"`, the value for `sep` can only be a single character. See the "Advanced Usage" example of how to specify multiple characters for batch conversion of columns.

Author(s)

Ananda Mahto

References

- See http://stackoverflow.com/q/10100887/1270695
- The "condensed" setting was inspired by an answer from David Winsemius to a question at Stack Overflow. See: http://stackoverflow.com/a/13924245/1270695

Examples

```
## Load some data
data(concatenated)
head(concat.test)

# Split up the second column, selecting by column number
head(concat.split(concat.test, 2))

# ... or by name, and drop the offensive first column
head(concat.split(concat.test, "Likes", drop.col = TRUE))

# The "Hates" column uses a different separator
head(concat.split(concat.test, "Hates", sep = ";", drop.col = TRUE))

# You'll get a warning here, when trying to retain the original values
head(concat.split(concat.test, 2, mode = "value", drop.col = TRUE))

# Try again. Notice the differing number of resulting columns
head(concat.split(concat.test, 2, structure = "expanded",
     mode = "value", drop.col = TRUE))

# Let's try splitting some strings... Same syntax
head(concat.split(concat.test, 3, drop.col = TRUE))

# Split up the "Likes column" into a list variable; retain original column
head(concat.split(concat.test, 2, structure = "list", drop.col=FALSE))

# View the structure of the output for the first 10 rows to verify
# that the new column is a list; note the difference between "Likes"
# and "Likes_list".
str(concat.split(concat.test, 2, structure = "list",
   drop.col=FALSE)[1:10, c(2, 5)])

# ADVANCED USAGE ###

# Show just the first few lines, compact structure
# Note that the split characters must be specified
#   in the same order that lapply will encounter them
head(do.call(cbind,
             c(concat.test[1],
               lapply(1:(ncol(concat.test)-1),
                      function(x) {
                          splitchars = c(",", ",", ";")
                          concat.split(concat.test[-1][x], 1,
                                       splitchars[x],
                                       drop.col=TRUE)
                                       }))))
```

---

df.sorter *Sort a* `data.frame` *by rows or columns*

Description

The `df.sorter` function allows you to sort a `data.frame` by columns or rows or both. You can also quickly subset data columns by using the `var.order` argument.

Usage

```
df.sorter(data, var.order = names(data), col.sort = NULL,
  at.start = TRUE)
```

Arguments

`data`	The source `data.frame`.
`var.order`	The new order in which you want the variables to appear. See Details
`col.sort`	The columns *within* which there is data that need to be sorted. See Details.
`at.start`	Should the pattern matching be from the start of the variable name? Defaults to `TRUE`.

Details

var.order

- Defaults to `names(data)`, which keeps the variables in the original order.
- Variables can be referred to either by a vector of their index numbers or by a vector of the variable name; partial name matching also works, but requires that the partial match identifies similar columns uniquely (see Examples). Basic subsetting can also be done using `var.order` simply by omitting the variables you want to drop.

col.sort

- Defaults to `NULL`, which means no sorting takes place. Variables can be referred to either by a vector of their index numbers or by a vector of the variable names; full names must be provided.

Note

If you are sorting both by variables and within the columns and using numeric indexes as opposed to variable names, the `col.sort` order should be based on the location of the columns in the new `data.frame`, not the original `data.frame`.

Author(s)

Ananda Mahto

Examples

```
# Make up some data
set.seed(1)
dat = data.frame(id = rep(1:5, each=3), times = rep(1:3, 5),
                 measure1 = rnorm(15), score1 = sample(300, 15),
                 code1 = replicate(15, paste(sample(LETTERS[1:5], 3),
                                            sep="", collapse="")),
                 measure2 = rnorm(15), score2 = sample(150:300, 15),
                 code2 = replicate(15, paste(sample(LETTERS[1:5], 3),
                                            sep="", collapse="")))
# Preview your data
dat

# Change the variable order, grouping related columns
# Note that you do not need to specify full variable names,
#    just enough that the variables can be uniquely identified
head(df.sorter(dat, var.order = c("id", "ti", "cod", "mea", "sco")))

# As above, but sorted by ’times’ and then ’id’
head(df.sorter(dat,
               var.order = c("id", "tim", "cod", "mea", "sco"),
               col.sort = c(2, 1)))

# Drop ’measure1’ and ’measure2’, sort by ’times’, and ’score1’
head(df.sorter(dat,
               var.order = c("id", "tim", "sco", "cod"),
               col.sort = c(2, 3)))

# Just sort by columns, first by ’times’ then by ’id’
head(df.sorter(dat, col.sort = c("times", "id")))

# Pattern matching anywhere in the variable name
head(df.sorter(dat, var.order= "co", at.start=FALSE))
```

`multi.freq.table` *Tabulates columns from a* `data.frame` *containing multiple-response data*

Description

The multi.freq.table function takes a data.frame containing Boolean responses to multiple response questions and tabulates the number of responses by the possible combinations of answers.

Usage

```
multi.freq.table(data, sep = "", boolean = TRUE,
  factors = NULL, NAto0 = TRUE, basic = FALSE,
  dropzero = TRUE, clean = TRUE)
```

Arguments

data The multiple responses that need to be tabulated.

sep The desired separator for collapsing the combinations of options; defaults to "" (collapsing with no space between each option name).

boolean Are you tabulating boolean data (see dat Examples)? Defaults to TRUE.

factors If you are trying to tabulate non-boolean data, and the data are not factors, you can specify the factors here (see dat2 Examples). Defaults to NULL and is not used when boolean = TRUE.

NAto0 Should NA values be converted to 0? Defaults to TRUE, in which case, the number of valid cases should be the same as the number of cases overall. If set to FALSE, any rows with NA values will be dropped as invalid cases. Only applies when boolean = TRUE.

basic Should a basic table of each item, rather than combinations of items, be created? Defaults to FALSE.

dropzero Should combinations with a frequency of zero be dropped from the final table? Defaults to TRUE. Does not apply when boolean = TRUE.

clean Should the original tabulated data be retained or dropped from the final table? Defaults to TRUE (drop). Does not apply when boolean = TRUE.

Details

In addition to tabulating the *frequency* (Freq), there are two other columns in the output: *Percent of Responses* (Pct.of.Resp) and *Percent of Cases* (Pct.of.Cases).

Percent of Responses is the frequency divided by the total number of answers provided; this column should sum to 100 table is generated and there are cases where a respondent did not select any option, the Percent of Responses value would be more than 100 frequency divided by the total number of valid cases; this column would most likely sum to more than 100 a basic table is produced since each respondent (case) can select multiple answers, but should sum to 100 other tables.

Author(s)

Ananda Mahto

References

apply shortcut for creating the Combn column in the output by Justin. See: http://stackoverflow.com/q/11348391/1270695 and http://stackoverflow.com/q/11622660/1270695

Examples

```
# Make up some data
set.seed(1)
dat <- data.frame(A = sample(c(0, 1), 20, replace=TRUE),
                 B = sample(c(0, 1, NA), 20,
                            prob=c(.3, .6, .1), replace=TRUE),
                 C = sample(c(0, 1, NA), 20,
                            prob=c(.7, .2, .1), replace=TRUE),
                 D = sample(c(0, 1, NA), 20,
                            prob=c(.3, .6, .1), replace=TRUE),
                 E = sample(c(0, 1, NA), 20,
                            prob=c(.4, .4, .2), replace=TRUE))

# View your data
dat

# How many cases have ”NA” values?
table(is.na(rowSums(dat)))

# Apply the function with all defaults accepted
multi.freq.table(dat)

# Tabulate only on variables ”A”, ”B”, and ”D”, with a different
# separator, keep any zero frequency values, and keeping the
# original tabulations. There are no solitary ”D” responses.
multi.freq.table(dat[c(1, 2, 4)], sep=”-”, dropzero=FALSE, clean=FALSE)

# As above, but without converting ”NA” to ”0”.
# Note the difference in the number of valid cases.
multi.freq.table(dat[c(1, 2, 4)], NAto0=FALSE,
                 sep=”-”, dropzero=FALSE, clean=FALSE)

# View a basic table.
multi.freq.table(dat, basic=TRUE)

#================================#

# NON-BOOLEAN DATA

# Make up some data
dat2 <- structure(list(Reason.1 = c(”one”, ”one”, ”two”, ”one”, ”two”,
                                    ”three”, ”one”, ”one”, NA, ”two”),
                      Reason.2 = c(”two”, ”three”, ”three”, NA, NA,
                                    ”two”, ”three”, ”two”, NA, NA),
                      Reason.3 = c(”three”, NA, NA, NA, NA,
                                    NA, NA, ”three”, NA, NA)),
                      .Names = c(”Reason.1”, ”Reason.2”, ”Reason.3”),
                      class = ”data.frame”,
                      row.names = c(NA, -10L))

# View your data
dat2
```

```
# The following will not work.
# The data are not factored.
multi.freq.table(dat2, boolean=FALSE)

# Factor create the factors.
multi.freq.table(dat2, boolean=FALSE,
                 factors = c("one", "two", "three"))

# And, a basic table.
multi.freq.table(dat2, boolean=FALSE,
                 factors = c("one", "two", "three"),
                 basic=TRUE)
```

RandomNames *Generate random names*

Description

The `RandomNames` function uses data from the *Genealogy Data: Frequently Occurring Surnames from Census 1990–Names Files* web page to generate a `data.frame` with random names.

Usage

```
RandomNames(N = 100, cat = NULL, gender = NULL,
  MFprob = NULL, dataset = NULL)
```

Arguments

`N` The number of random names you want. Defaults to 100.

`cat` Do you want "common" names, "rare" names, names with an "average" frequency, or some combination of these? Should be specified as a character vector (for example, `c("rare", "common")`). Defaults to `NULL`, in which case all names are used as the sample frame.

`gender` Do you want first names from the "male" dataset, the "female" dataset, or from all available names? Should be specified as a quoted string (for example, `"male"`). Defaults to `NULL`, in which case all available first names are used as the sample frame.

`MFprob` What proportion of the sample should be male names and what proportion should be female? Specify as a numeric vector that sums to 1 (for example, `c(.6, .4)`). The first number represents the probability of sampling a "male" first name, and the second number represents the probability of sampling a "female" name. This argument is not used if only one gender has been specified in the previous argument. Defaults to `NULL`, in which case, the probability used is `c(.5, .5)`.

`dataset` What do you want to use as the dataset of names from which to sample? A default dataset is provided that can generate over 400 million unique names. See the "Dataset Details" note for more information.

Note

Dataset Details This function samples from a provided dataset of names. By default, it uses the data from the *Genealogy Data: Frequently Occurring Surnames from Census 1990–Names Files* web page. Those data have been converted to `list` named `"CensusNames1990"` containing three `data.frames` (named `"surnames"`, `"malenames"`, and `"femalenames"`).

Alternatively, you may provide your own data in a `list` formatted according to the following specifications (see the `"myCustomNames"` data in the "Examples*" section). *Please remember that R is case sensitive!*

- This must be a named list with three items: `"surnames"`, `"malenames"`, and `"femalenames"`.
- The contents of each list item is a `data.frame` with at least the following named columns: `"Name"` and `"Category"`.
- Acceptable values for `"Category"` are `"common"`, `"rare"`, and `"average"`.

Author(s)

Ananda Mahto

References

- See `http://www.census.gov/genealogy/www/data/1990surnames/names_files.html` for source of data.
- Inspired by the online Random Name Generator `http://random-name-generator.info/`

Examples

```
# Generate 20 random names
RandomNames(N = 20)

# Generate a reproducible list of 100 random names with approximately
#   80% of the names being female names, and 20% being male names.
set.seed(1)
temp <- RandomNames(cat = "common", MFprob = c(.2, .8))
list(head(temp), tail(temp))
table(temp$Gender)

# Cleanup
rm(.Random.seed, envir=globalenv()) # Resets your seed
rm(temp)

# Generate 10 names from the common and rare categories of names
RandomNames(N = 10, cat = c("common", "rare"))

## ==================================== ##
## ======== USING YOUR OWN DATA ======== ##

myCustomNames <- list(
 surnames = data.frame(
   Name = LETTERS[1:26],
   Category = c(rep("rare", 10), rep("average", 10), rep("common", 6))),
 malenames = data.frame(
   Name = letters[1:10],
   Category = c(rep("rare", 4), rep("average", 4), rep("common", 2))),
 femalenames = data.frame(
   Name = letters[11:26],
   Category = c(rep("rare", 8), rep("average", 4), rep("common", 4))))
str(myCustomNames)

RandomNames(N = 15, dataset = myCustomNames)
```

`row.extractor` *Extract min/median/max/quantile rows from a* `data.frame`

Description

The `row.extractor` function takes a `data.frame` and extracts rows with the min, median, or max values of a given variable, or extracts rows with specific quantiles of a given variable.

Usage

```
row.extractor(data, extract.by, what = "all")
```

Arguments

`data` The source `data.frame`.

`extract.by` The column which will be used as the reference for extraction; can be specified either by the column number or the variable name.

`what` Options are `"min"` (for all rows matching the minimum value), `"median"` (for the median row or rows), `"max"` (for all rows matching the maximum value), or `"all"` (for min, median, and max); alternatively, a numeric vector can be specified with the desired quantiles, for instance `c(0, .25, .5, .75, 1)`.

Author(s)

Ananda Mahto

References

- `which.quantile` function by cbeleites: `http://stackoverflow.com/users/755257/cbeleites`
- See: `http://stackoverflow.com/q/10256503/1270695`

See Also

`min`, `max`, `median`, `which.min`, `which.max`, `quantile`

Examples

```
# Make up some data
set.seed(1)
dat = data.frame(V1 = 1:50, V2 = rnorm(50),
                V3 = round(abs(rnorm(50)), digits=2),
                V4 = sample(1:30, 50, replace=TRUE))
# Get a sumary of the data
summary(dat)

# Get the rows corresponding to the ’min’, ’median’, and ’max’ of ’V4’
row.extractor(dat, 4)

# Get the ’min’ rows only, referenced by the variable name
row.extractor(dat, "V4", "min")

# Get the ’median’ rows only. Notice that there are two rows
#    since we have an even number of cases and true median
#    is the mean of the two central sorted values
row.extractor(dat, "V4", "median")

# Get the rows corresponding to the deciles of ’V3’
row.extractor(dat, "V3", seq(0.1, 1, 0.1))
```

sample.size *Determine the optimal sample size of a given population*

Description

The sample.size function either calculates the optimum survey sample size when provided with a population size, or the confidence interval of using a certain sample size with a given population. It can be used to generate tables (data.frames) of different combinations of inputs of the following arguments, which can be useful for showing the effect of each of these in sample size calculation.

Usage

```
sample.size(population, samp.size = NULL, c.lev = 95,
  c.int = NULL, what = "sample", distribution = 50)
```

Arguments

`population` The population size for which a sample size needs to be calculated.

`samp.size` The sample size. This argument is only used when calculating the confidence interval, and defaults to `NULL`.

`c.lev` The desired confidence level. Defaults to a reasonable 95%.

`c.int` The confidence interval. This argument is only used when calculating the sample size. If not specified when calculating the sample size, defaults to 5% and a message is provided indicating this; this is also the default action if `c.int = NULL`.

`what` Should the function calculate the desired sample size or the confidence interval? Accepted values are `"sample"` and `"confidence"` (quoted), and defaults to `"sample"`.

`distribution` Response distribution. Defaults to 50% (`distribution = 50`), which will give you the largest sample size.

Note

From a teaching perspective, the function can be used to easily make tables which demonstrate how the sample size or confidence interval change when different inputs change. See the "Advanced Usage" examples. The following formulae were used in this function:

$$ss = \frac{-Z^2 \times p \times (1-p)}{c^2}$$

$$pss = \frac{ss}{1 + \frac{ss-1}{pop}}$$

Author(s)

Ananda Mahto

References

- See the 2657 Productions News site for how this function progressively developed: http://news.mrdwab.com/2010/09/10/a-sample-size-calculator-function-for-r/
- The `sample.size` function is based on the following formulas from the Creative Research Systems web page *Sample size formulas for our sample size calculator*: http://www.webcitation.org/69kNjMuKe

Examples

```
# What should our sample size be for a population of 300?
# All defaults accepted.
sample.size(population = 300)

# What sample should we take for a population of 300
#   at a confidence level of 97%?
sample.size(population = 300, c.lev = 97)

# What about if we change our confidence interval?
sample.size(population = 300, c.int = 2.5, what = "sample")

# What about if we want to determine the confidence interval
#   of a sample of 140 from a population of 300? A confidence
#   level of 95% is assumed.
sample.size(population = 300, samp.size = 140, what = "confidence")

## ========================================= ##
## =========== ADVANCED USAGE ============== ##

# What should the sample be for populations of 300 to 500 by 50?
sample.size(population=c(300, 350, 400, 450, 500))

# How does varying confidence levels or confidence intervals
#   affect the sample size?
sample.size(population=300,
            c.lev=rep(c(95, 96, 97, 98, 99), times = 3),
            c.int=rep(c(2.5, 5, 10), each=5))

# What is are the confidence intervals for a sample of
#   150, 160, and 170 from a population of 300?
sample.size(population=300,
            samp.size = c(150, 160, 170),
            what = ”confidence”)
```

Index