Package ‘varhandle’

February 21, 2020

Version 2.0.5

Date 2020-02-20

Title Functions for Robust Variable Handling

Author Mehrad Mahmoudian [aut, cre]

Maintainer Mehrad Mahmoudian <m.mahmoudian@gmail.com>

Depends R (>= 3.0.1),

Imports utils, graphics

Suggests

Description Variables are the fundamental parts of each programming language but handling them efficiently might be frustrating for programmers. This package contains some functions to help user (especially data explorers) to make more sense of their variables and take the most out of variables and hardware resources. These functions are written, collected and crafted over 7 years of experience in statistical data analysis on high-dimensional data, and for each of them there was a need. Functions in this package are suppose to be efficient and easy to use, hence they will be frequently updated to make them more convenient.

License GPL (>= 2)

URL https://bitbucket.org/mehrad_mahmoudian/varhandle

BugReports https://bitbucket.org/mehrad_mahmoudian/varhandle/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-21 00:30:02 UTC

R topics documented:

varhandle-package 2
check.numeric 2
inspect.na 4
pin.na 6
rm.all.but 7

save.var 9
to.dummy 10
unfactor 11
var.info 12

Index **15**

`varhandle-package` *A Package to Work and Handl Variables Easier, Convenient and Flawless*

Description

A collection of functions for working and handling variables easier, convenient and flawless in R programming environment. It is suitable for Projects: Small to semi-large projects with massive amount of data. Programmers: Programming and exploring data on the fly or preparing and assembling a pipeline.

Details

Package:	varhandle
Type:	Package
Version:	2.0.4
Date:	2019-10-21
License:	GPL (>=2)

Author(s)

Mehrad Mahmoudian [aut, cre] Who to complain to <m.mahmoudian@gmail.com> Reporting issues: https://bitbucket.org/mehrad_mahmoudian/varhandle/issues?status=new&status=open

`check.numeric` *Check the vector's possiblity to convert to numeric*

Description

This function gets a character or factor vector and checks if all the value can be safely converted to numeric.

Usage

```
check.numeric(v=NULL, na.rm=FALSE, only.integer=FALSE, exceptions=c(""),
              ignore.whitespace=TRUE)
```

Arguments

`v` The character vector or factor vector. (Mandatory)

`na.rm` logical. Should the function ignore NA? Default value is FLASE since NA can be converted to numeric. (Optional)

`only.integer` logical. Only check for integers and do not accept floating point. Default value is FALSE. (Optional)

`exceptions` A character vector containing the strings that should be considered as valid to be converted to numeric. (Optional)

`ignore.whitespace`
logical. Ignore leading and tailing whitespace characters before assessing if the vector can be converted to numeric. Default value is TRUE. (Optional)

Details

This function checks if it is safe to convert the vector to numeric and this conversion will not end up in producing NA. In nutshell this function tries to mak sure provided vector contains numbers but in a non-numeric class. see example for better understanding.

This function can be configured to only accept integer numbers (by setting the argument only.integer to TRUE). It can also ignore NA values (na.rm argument) and ignore heading/tailing whitespaces (ignore.whitespace argument).

There is also room to manually define exceptions to be concidered as numbers (exceptions argument).

Value

The function return a logical vector. TRUE for all the elements in the given vector if they are safe to be converted into numeric in R (does not turned into NA). Remember that "" and " " can be converted to numeric safely (without getting the "NAs introduced by coercion" error), but their value would be NA.

In case of a integer, numric or logical vector, the function simply returns all TRUE logical vector.

Author(s)

Mehrad Mahmoudian

See Also

`as.numeric`

Examples

```
# Create a vector with NA
a <- as.character(c(1:5, NA, seq(from=6, to=7, by=0.2)))
# see what we created
print(a)
# check if the vector is all numbers (not ignoring NAs)
check.numeric(a)
```

```
# check if the vector is all numbers (ignoring NAs)
check.numeric(a, na.rm=TRUE)
# if all the items in vector a are safe for numeric conversion
if(all(check.numeric(a))){
    # convert the vector to numeric
    a <- as.numeric(a)
}

# create a complicated vector
b <- c("1", "2.2", "3.", ".4", ".5.", "..6", "seven", "00008",
   "90000", "-10", "+11", "12-", "--13", "++14", NA, "",
   " 7 ", "   ", "8e2", "8.6e-10", "-8.6e+10", "e3")

# show in propper format
print(data.frame(value=b, check.numeric=check.numeric(b), converted=as.numeric(b)))
    #       value check.numeric converted
    # 1         1          TRUE   1.0e+00
    # 2       2.2          TRUE   2.2e+00
    # 3        3.          TRUE   3.0e+00
    # 4        .4          TRUE   4.0e-01
    # 5       .5.         FALSE        NA
    # 6       ..6         FALSE        NA
    # 7     seven         FALSE        NA
    # 8     00008          TRUE   8.0e+00
    # 9     90000          TRUE   9.0e+04
    # 10      -10          TRUE  -1.0e+01
    # 11      +11          TRUE   1.1e+01
    # 12      12-         FALSE        NA
    # 13     --13         FALSE        NA
    # 14     ++14         FALSE        NA
    # 15     <NA>          TRUE        NA
    # 16                   TRUE        NA
    # 17       7           TRUE   7.0e+00
    # 18                   TRUE        NA
    # 19      8e2          TRUE   8.0e+02
    # 20  8.6e-10          TRUE   8.6e-10
    # 21 -8.6e+10          TRUE  -8.6e+10
    # 22       e3         FALSE        NA

# remember that "" and "   " can be converted to numeric safely, but their value would be NA.
```

`inspect.na` *inspect matrix or data.frame regarding NAs*

Description

This function provides a summary of NAs in a given matrix or data.frame either feature-wise (by column) or sample-wise (by row). It can also provide a barplot and/or histogram regarding this statistics.

Usage

```
inspect.na(d, hist=FALSE, summary=TRUE, byrow=FALSE, barplot=TRUE, na.value = NA)
```

Arguments

`d`	A data.frame or matrix which you want to get the summary of NAs in it (Mandatory)
`hist`	logical. Should the function plot histogram. Default is FALSE. (Optional)
`summary`	logical. Should the function returns the result dataframe. Default is TRUE. (Optional)
`byrow`	logical. Should the function perform row-wise. Default is FALSE. (Optional)
`barplot`	logical. Should the function plot barplot. Default is TRUE. (Optional)
`na.value`	A vector containing the value that should be concidered as missing value. The default is NA, but you can add to it or change it to your preference. See the example. (Optional)

Details

This function provides a quick and easy way to see how much missing values (e.g NA) exist in a data.frame or matrix. This function is designed to make the data exploration easier since missing values are one of the most problematic part in lated stages of analysis.

Value

The function prvides a data.frame (in case summary argument is set to TRUE) containing column or row index, name, number_of_NAs and ratio_of_NA. In case the function does not find any NA, it will return NULL in case it need to be checked by is.null().

The barplot generated by this function is presenting column names or row names which contain NAs with their NA ratio to the total number of items in that row or column. The plot also colors the bars based on their NA ratio: * Gray less than and equal to 10% * Yellow for >10% and <30% * Orange for >30% and <50% * Red for >50% The plot also has horizontal lines at 10%, 20%, 30% and 50% to make the plot easier to read.

The histogram generated by this function is meant to provide an overview of how NAs are distributed in the input data. This plot presents all the columns or rows regardless of having NA values or not. This plot is more useful when used for small number of rows or columns.

Author(s)

Mehrad Mahmoudian

See Also

`pin.na` `is.na`

Examples

```
# get some data
my_iris <- iris
# add 20 NAs randomly
for(i in 1:260){
    my_iris[sample(1:nrow(my_iris), 2), sample(c(1,2,3,1,3,3,3), 1)] <- NA
}

# now we can inspect the NAs
inspect.na(my_iris)
# plot the histogram
inspect.na(my_iris, hist=TRUE, barplot=FALSE)
```

pin.na *Pinpoint NAs in a vector, matrix or data.frame*

Description

This function finds NAs (or defined missing values) in a vector, data.frame or matrix and returns a data.frame which contains two columns that includes the row number and column number of each NA.

Usage

```
pin.na(x, na.value = NA)
```

Arguments

x	A vector, data.frame or matrix which you want to pinpoint its NAs. (Mandatory)
na.value	A vector containing the value that should be concidered as missing value. The default is NA, but you can add to it or change it to your preference. See the example. (Optional)

Details

This function provides a quick and easy way to locate and pinpoint NAs in a given vector, data.frame or matrix. This function is also used in inspect.na function of this package.

Value

If a vector is given, the index of NAs will be returned in a numeric vector format. In case of a given matrix or data.frame the function will return a data.frame with two columns, one indicating the row number and one indicating the column number. Each row will represent a location of a NA. In case no NA is found, the function will return NULL which makes it easy to use in if conditions using is.null.

Author(s)

Mehrad Mahmoudian

See Also

is.na

Examples

```
## generate some variables
# create a vector
var1 <- 1:30
# add NA at random places
var1[runif(7, 1, 30)] <- NA
# pinpoint NAs
pin.na(var1)

# create a matrix
var2 <- matrix(runif(100, 10, 99), nrow = 10)
# add NA at random places
var2[runif(9, 1, 100)] <- NA
# pinpoint NAs
pin.na(var2)

## define your own missing values:
var2[runif(5, 1, 100)] <- "."
pin.na(var2, na.value = c(NA, "."))
```

rm.all.but *Remove all variables except those that you mention*

Description

This function removes all existing variables (in the defined environment) except the variables given to it.

Usage

```
rm.all.but(keep=NULL, envir=.GlobalEnv, keep_functions=TRUE, gc_limit=100,
            regex="auto")
```

Arguments

keep	A vector containing the name of variables which you wish to keep and not remove or a regular expression to match variables you want to keep. Variable names and regular expressions can be used combined if the argument ‘regex‘ is set to "auto". (Mandatory)
envir	The environment that this function should be functional in, search in and act in. (Optional)

keep_functions A logical vector of length 1 indicating exclusion of function variables from removal. (optional)

gc_limit A numeric vector of length 1 indicating the threshold for garbage collection in Megabyte (MB) scale. (Optional)

regex A vector with length 1 to define whether the function use regular expression in keep (TRUE or FALSE) or auto detect ("auto")

Details

While working with R it happens that users generates an accumulate many variables and at some point they just want to keep some of them and remove the rest to make the workspace clean and reduce memory usage. This is where this function comes in to keep those variables user wants and remove the rest.

Two criteria can be used as filtering the varibales you wish to keep:

* variable names * regular expression

These can be also used in combination in any order. You can also have multiple regular expressions to match different varibale names.

The garbage collection will run if the size of removed variables exceed the gc_limit parameter and will tell R to give back the amount of occupied memory by removed variables to the system. This comes handy since usually removed variables are created temprorily and removing them should free up the memory.

Author(s)

Mehrad Mahmoudian

See Also

remove gc object.size

Examples

```
# create some variable
for(i in  names(iris)){
 assign(i, iris[,i])
}
# see the list of variables
ls()
# remove every variable except Petal.Length, Petal.Width and i
rm.all.but(c("Petal*", "i"))
# see which variable are left
ls()
```

save.var *Save variables separate files*

Description

This function gets a list of variables as a character vector and save each variable name in a separate file, so that they can be loaded separately.

Usage

```
save.var(varlist = ls(envir = as.environment(.GlobalEnv)),
        path = getwd(), newdir = TRUE, newdirtag = NULL, envir = .GlobalEnv)
```

Arguments

`varlist`	Character vector containing variable names. If not provided the function will use all the variables of the environment. (Optional)
`path`	Path to folder that you want to save the files in. If not provided, the current working directory will be used. (Optional)
`newdir`	Logical vector of length 1 indicating whether you want to create a subdirectory to store your variable files in. This subdirectory will have the execution time (%Y%m%d-%H%M%S) as folder name in combination with a text tag (newdirtag) if provided. (Optional)
`newdirtag`	Character string used in combination with newdir=TRUE to tag the time based folder name with a custom name. If not provided no tag will be used and the folder name will just have the time format. (Optional)
`envir`	Character string providing target environment in R session. The default is the Global Environment. (Optional)

Details

This function is used for saving variables in batch into separate files in an organized way. This specifically comes handy when you generate many variables either dynamically or manually and you want to save them for later use and empty your memory. Saving variables in separate files help finding them as a file easier and faster and also reduces the loading time of the variable since you are also loading those that you want to use.

The save.var function has the feature to save the variables in subfolder to help user manage different version of same variable which are related to different runs.

Author(s)

Mehrad Mahmoudian

See Also

`save.image`, `save`

Examples

```
# generate variables dynamically
lapply(letters, function(x){assign(x=x, value=rnorm(1), env=globalenv())})

    ## Not run:
        # simple usage
        save.var()
        # specify a listof variables with tag
        save.var(varlist=c("a","e","i","o","u"), newdirtag="just_vowels")

## End(Not run)
```

to.dummy *Convert categorical vector into dummy binary dataframe*

Description

This function gets a vector that contains some categories and convert it to dummy columns (also known as binary columns). The number of output columns is equal to the input categories.

Usage

```
to.dummy(v, prefix)
```

Arguments

`v`	A character, numeric or factor vector that contains the categories. (Mandatory)
`prefix`	A character string to attach to the begining of the column names to prevent confusion or conflicts. (Mandatory)

Details

This function simplifies the procedure of making data ready for those learning algorithms or methods that cannot handle categorical columns. It works by getting a character, numeric or factor vector and convert it to some columns that each of which represent a category from the input vector. For example a vector of eye color with different categories like Black, Brown, Blue, Green will be transformed into a dataframe with 4 columns and each column has value of 1 for samples that have that specific eye color.

Value

A data.frame is returned which only contains 0 and 1 as values. Number of this data.frame columns is equal to number of categories in the original input vector.

Author(s)

Mehrad Mahmoudian

Examples

```
# load a dataframe (from base package)
data(iris)

# see the actual values of the categorical column
print(iris$Species)

# convert to dummy
binary_species <- to.dummy(iris$Species, "species")
# view the first few lines of the binary_species data.frame
head(binary_species)
```

`unfactor` *Convert factor into appropriate class*

Description

This function gets a factor vector, data.frame or matrix (that contains factor columns), detects the real class of the values and convert factor to the real class.

Usage

```
unfactor(obj, auto_class_conversion = TRUE, verbose = FALSE)
```

Arguments

`obj` The factor vector, data.fram or matrix. (Mandatory)

`auto_class_conversion` Whether or not the function should automatically convert numbers to numeric. If set to FALSE, it will return all columns as class characters. Default is TRUE. (Optional)

`verbose` Whether or not the function should be verbose, meaning it should message user about the details of operation. Default is FALSE. (Optional)

Details

This function turns factors to their real values. when a data.frame is given, the function detects factor columns and unfactor them, so you can give the whole data.frameand the function takes care of the rest. The values' real class detection mechanism is in a way that if everything in that column or vector are numbers and a decimal character, it change it to numeric otherwise it will be changed to character vector. This functionality can be turned off by setting the 'auto_class_conversion' argument to FALSE

Value

In case of providing a vector as an input, a character vector or numeric vector. This depends on the type of values the input variable contains. Check the details section for detailed information. In case of providing a data.frame, the same data.frame will be returned but with converted columns. In case there is nothing to get converted from factors, the function peacefully exits. You can get the details of the steps in form of message if you set the ‘verbose‘ argument to TRUE.

Note

In case you have any issues with the function, please report to: https://bitbucket.org/mehrad_mahmoudian/varhandle/issues

Author(s)

Mehrad Mahmoudian

See Also

`as.character`, `as.numeric`

Examples

```
 # load a dataframe (from base package)
 data(iris)

 # see the actual values of the categorical column
 class(iris$Species)

 # use vactor as input
 species <- unfactor(iris$Species)
 # check the class
 class(species)

 # use data.frame as input
my_iris <- data.frame(Sepal.Length=factor(iris$Sepal.Length), sample_id=factor(1:nrow(iris)))
 my_iris <- unfactor(my_iris)
 # check the class
 class(my_iris)
 class(my_iris$Sepal.Length)
 class(my_iris$sample_id)
```

var.info *Get a detailed list of variables*

Description

This function provides a detailed information of variables in the specified environment. If no environment and list of variables provided for the function, it will consider all existing variables in global environment.

Usage

```
var.info(list=”ALL”, regex = NULL, envir=.GlobalEnv, human.readable=TRUE,
         sortby=”size”, decreasing=TRUE, n=Inf, beautify = FALSE,
         progressbar = FALSE)
```

Arguments

`list` A list of variables which you want to get information for. If not specified, it gets all variables (Optional)

`regex` A regular expression to be applied on the list of variables. This is very useful for example when ‘list = "ALL"‘. (Optional)

`envir` The environment in which you want this function to be functional. (Optional)

`human.readable` If you want to have the variable size in humar readable format (Kb, Mb, etc.). (Optional)

`sortby` The name of a column that you wish to sort the output with. If not specified the result will be sorted by "size". Valid options are "name", "class", "size" or "detail". (Optional)

`decreasing` A logical parameter (TRUE/FALSE) indicating that you want the sort to be done decreasingly or increasingly. If not specified it is TRUE. (Optional)

`n` Number of desired rows in output If you want to have top 10, n should be equal to 10. If not specified the output will include all variables. (Optional)

`beautify` A Logical parameter indicating whether the output should beautified. At the moment it just adds a up/down triangle in the column name, showing the sort direction and based which column the table is sorted. Default value is FALSE. See Details for more information. (optional)

`progressbar` A Logical parameter indicating whether user wants to see progressbar or not. Default value is FALSE. See Details for more information. (optional)

Details

This function is a quick way to have some basic information about a list of variables. By modifying and changing the default parameters, you can narrow down the variables you are investigating. The main objective of this fuction is providing the following information about the variables in an easy and intuitive way: class size (amount of memory the variable has occupied) detail (dimention for data.frame and matrices and length of vectors)

In case the variable is a matrix or data.frame, in detail column the dimention will be provided, if it is a vector, the length will be reported, otherwise you will see NA in detail column for that specific variable.

This function is usually very quick but in case of having many variables in the environment, it might take some time, hence a progressbar is implemented to inform user about the process.

Value

The output will be a sorted data.frame with 4 columns. The "name" column contains the name of each variable, column "class" contains the class of each variable, columns "size" show the amount of memory the variable is occupying (it can be configured to be in bytes or human readable format. for this use the human.readable parameter.) and "detail" column includes variable-specific information (for matrices and data.frames the dimention and for vectors their list will be reported.)

Author(s)

Mehrad Mahmoudian

See Also

class, object.size

Examples

```
###### generate some variables ######
# a data.frame
data(iris)

# some character vector
for(i in 1:5){
    assign(letters[i], paste("some random text:",
                             paste0(letters[runif(5, 1, 26)],
                             collapse="")))
}

# a list
f <- lapply(5:10, function(x){paste("some random text:",
                              paste0(letters[runif(5, 1, 26)],
                              collapse=""))})


###### demo of this function ######
# basic usage
var.info()

# the sorting
var.info(sortby="name", decreasing=FALSE)

# select using regular expression
var.info(regex="^i")

# having the top 5 objects that use most memory in a beautified output
var.info(n=5, sortby="size", decreasing=TRUE, beautify=TRUE)
```

Index

∗Topic **Binary**
    to.dummy, 10
∗Topic **NA**
    inspect.na, 4
    pin.na, 6
∗Topic **check numeric**
    check.numeric, 2
∗Topic **defactor**
    unfactor, 11
∗Topic **dummy**
    to.dummy, 10
∗Topic **information**
    var.info, 12
∗Topic **inspect**
    inspect.na, 4
∗Topic **keep variables**
    rm.all.but, 7
∗Topic **locate**
    inspect.na, 4
    pin.na, 6
∗Topic **memory usage**
    var.info, 12
∗Topic **missing**
    inspect.na, 4
    pin.na, 6
∗Topic **package**
    varhandle-package, 2
∗Topic **pinpoint**
    pin.na, 6
∗Topic **remove variables**
    rm.all.but, 7
∗Topic **save to file**
    save.var, 9
∗Topic **save variable**
    save.var, 9
∗Topic **size**
    var.info, 12
∗Topic **summary**
    inspect.na, 4
∗Topic **unfactor**
    unfactor, 11
∗Topic **variable**
    var.info, 12

as.character, *12*
as.numeric, *3*, *12*

check.numeric, 2
class, *14*

gc, *8*

inspect.na, 4, *6*
is.na, *5*, *7*
is.null, *6*

object.size, *8*, *14*

pin.na, *5*, 6

remove, *8*
rm.all.but, 7

save, *9*
save.image, *9*
save.var, 9

to.dummy, 10

unfactor, 11

var.info, 12
varhandle *(*varhandle-package*)*, 2
varhandle-package, 2