Package 'AggregateR'

May 6, 2026

Type Package

Title Aggregate Numeric, Date and Categorical Variables

Version 0.1.1

Date 2020-11-18

Imports stats, utils, methods, data.table, tibble, NCmisc

Author Matthias Bogaert, Michel Ballings, Dirk Van den Poel

Maintainer Matthias Bogaert <matthias.bogaert@ugent.be>

Description Convenience functions for aggregating a data frame or data table.
Currently mean, sum and variance are supported. For Date variables, the recency
and duration are supported. There is also support for dummy variables in
predictive contexts. Code has been completely re-written in data.table for computational speed.

License GPL (>= 2)

LazyData TRUE

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2020-11-20 10:30:26 UTC

Contents

Aggregate *Aggregate numeric, Date and categorical variables*

Description

The Aggregate function (not to be confounded with aggregate) prepares a data frame or data table for merging by computing the sum, mean and variance of all continuous (integer and numeric) variables by a given variable. For all categorical variabes (character and factor), it creates dummies and subsequently computes the sum and the mode by a given variable. For all Date variables, it computes the recency and duration by a given variable with repsect the an end date variable. For computational speed, all the calculations are done with data.table. This functions aims at maximum information extraction with a minimum amount of code.

Usage

```
Aggregate(
  x,
  by,
  end_ind = Sys.Date(),
  format = "%Y-%m-%d",
  tibble = FALSE,
  verbose = TRUE,
  object = NULL,
  p = "all"
)
```

Arguments

x	A data frame or data table. Categorical variables have to be of type character or factor and continuous variables have to be of type integer or numeric. Date variables should be in the Date format.
by	A character string specifying the variable on which to aggregate the results. Note that 'by' should be a variable of the table 'x'.
end_ind	A Date object, or something which can be coerced by as.Date(origin, ...) to such an object. If not specified, we take the Sys.Date() as end date.
format	A character string. If not specified, the ISO 8601 international standard which expresses a day "%Y-%m-%d" is taken.
tibble	Should the output be a tibble, data frame or data table? By default, the function returns a data frame or data table depending on the input. To return a tibble, the user must set the tibble = TRUE.
verbose	indicator Used to show the progress.
object	Parameter related to the dummy function. See ?dummy for more information.
p	Parameter related to the dummy function. See ?dummy for more information.

Value

A data frame, data table or tibble with the aforementioned variables aggregated by the given ID variables. If the input is a data frame, a data frame is returned else a data table is returned.

Author(s)

Authors: Matthias Bogaert, Michel Ballings, Dirk Van den Poel, Maintainer: <matthias.bogaert@UGent.be>

Examples

```
# Example
# Create some data
data <- data.frame(V1=sample(as.factor(c('yes','no')), 200000, TRUE),
                   V2=sample(as.character(c(1,2,3,4,5)),200000, TRUE),
                   V3=sample(1:20000,200000, TRUE),
                   V4=sample(300:1000, 200000, TRUE),
                   V5 = sample(as.Date(as.Date('2014-12-09'):Sys.Date()-1,
                   origin = "1970-01-01"),200000,TRUE),
                   ID=sample(x = as.character(1:4), size = 200000, replace = TRUE))

Aggregate(x=data,by='ID')

# Examples of how to use the object and p argument. See dummy and categories function for details.
# Aggregate(x=data,by='ID',object=categories(data))
# Aggregate(x=data,by='ID',p=2)
```

`categories` *Extraction of Categorical Values as a Preprocessing Step for Making Dummy Variables*

Description

`categories` stores all the categorical values that are present in the factors and character vectors of a data frame. Numeric and integer vectors are ignored. It is a preprocessing step for the `dummy` function. This function is appropriate for settings in which the user only wants to compute dummies for the categorical values that were present in another data set. This is especially useful in predictive modeling, when the new (test) data has more or other categories than the training data.

Usage

```
categories(x, p = "all")
```

Arguments

`x`	data frame or data table containing factors or character vectors that need to be transformed to dummies. Numerics, dates and integers will be ignored.
`p`	select the top p values in terms of frequency. Either "all" (all categories in all variables), an integer scalar (top p categories in all variables), or a vector of integers (number of top categories per variable in order of appearance.

Value

A list containing the variable names and the categories

Author(s)

Authors: Michel Ballings, and Dirk Van den Poel, Maintainer: <Michel.Ballings@GMail.com>

See Also

dummy

Examples

```
#create toy data
(traindata <- data.frame(var1=as.factor(c("a","b","b","c")),
                         var2=as.factor(c(1,1,2,3)),
                         var3=c("val1","val2","val3","val3"),
                         stringsAsFactors=FALSE))
(newdata <- data.frame(var1=as.factor(c("a","b","b","c","d","d")),
                       var2=as.factor(c(1,1,2,3,4,5)),
                       var3=c("val1","val2","val3","val3","val4","val4"),
                       stringsAsFactors=FALSE))

categories(x=traindata,p="all")
categories(x=traindata,p=2)
categories(x=traindata,p=c(2,1,3))
```

---

dummy *Fast-automatic Dummy Variable Creation with Support for Predictive Contexts*

Description

dummy creates dummy variables of all the factors and character vectors in a data frame or data table. It also supports settings in which the user only wants to compute dummies for the categorical values that were present in another data set. This is especially useful in the context of predictive modeling, in which the new (test) data has more or other categories than the training data.For computational speed, the code is written in data.table.

Usage

```
dummy(x, p = "all", object = NULL, num = TRUE, verbose = FALSE, ref = FALSE)
```

Arguments

x	a data frame or data table containing at least one factor or character vector
p	Only relevant if object is NULL. Select the top p values in terms of frequency. Either "all" (all categories in all variables), an integer scalar (top p categories in all variables), or a vector of integers (number of top categories per variable in order of appearance).
object	output of the categories function. This parameter is to be used when dummies should be created only of categories present in another data set (e.g., training set)
num	should the dummies be of class numeric (TRUE) or factor (FALSE). Setting this to TRUE will speed up execution considerably.
verbose	logical. Used to show progress. Does not work when parallel="variable".
ref	logical. Only relevant when x is a data.table. If TRUE x will be overwritten by the dummy output (called transformed x), and a reference (i.e., not a copy) to the transformed x will be returned invisibly. If FALSE, x will be left untouched, and the output will be returned as usual. The difference between ref=TRUE and ref=FALSE is that the former uses less memory equal to the amount of the original x (not transformed x). If x=TRUE only the transformed x survives the function. If x=FALSE both the original x and the output (equal in size as transformed x) will survive. The difference is hence the size of the original x, and therefore ref=TRUE is more memory efficient.

Value

A data frame or data table containing dummy variables. If ref=TRUE then the output will be invisible and x will contain the output. NOTE: data.table currently has a print bug. In some cases the output does not print. Running the output object multiple times or running it once with [] appended will make it print. In either case, the output will be produced. str() also always works.

Author(s)

Authors: Michel Ballings, and Dirk Van den Poel, Maintainer: <Michel.Ballings@GMail.com>

See Also

categories

Examples

```
#create toy data
(traindata <- data.frame(var1=as.factor(c("a","b","b","c")),
                         var2=as.factor(c(1,1,2,3)),
                         var3=c("val1","val2","val3","val3"),
                         stringsAsFactors=FALSE))
(newdata <- data.frame(var1=as.factor(c("a","b","b","c","d","d")),
                       var2=as.factor(c(1,1,2,3,4,5)),
                       var3=c("val1","val2","val3","val3","val4","val4"),
                       stringsAsFactors=FALSE))
```

```
#create dummies of training set
(dummies_train <- dummy(x=traindata))
#create dummies of new set
(dummies_new <- dummy(x=newdata))

#how many new dummy variables should not have been created?
sum(! colnames(dummies_new) %in% colnames(dummies_train))

#create dummies of new set using categories found in training set
(dummies_new <- dummy(x=newdata,object=categories(traindata,p="all")))

#how many new dummy variables should not have be created?
sum(! colnames(dummies_new) %in% colnames(dummies_train))


#create dummies of training set,
#using the top 2 categories of all variables found in the training data
dummy(x=traindata,p=2)

#create dummies of training set,
#using respectively the top 2,3 and 1 categories of the three
#variables found in training data
dummy(x=traindata,p=c(2,3,1))

#create all dummies of training data
dummy(x=traindata)

## Not run:
########################
#example ref parameter

#ref=TRUE, example 1
(DT = data.table(a=c("a","b"),b=c("c","c")))
dummy(DT,ref=TRUE)
DT[] #DT has changed

#ref=TRUE, example 2
#uses exactly same amount of memory as example 1
(DT = data.table(a=c("a","b"),b=c("c","c")))
d1 <- dummy(DT,ref=TRUE)
DT[] #DT has changed
d1[] #d1 is a reference (not a copy) to DT

#ref=FALSE, example 3
#example 1 and 2 are more memory efficient than example 3
(DT = data.table(a=c("a","b"),b=c("c","c")))
d2 <- dummy(DT, ref=FALSE)
DT[] #DT has not changed
d[]
# deleting DT after dummy finishes would result in the same final
# memory footprint as example 1 and 2, except that in example 3
# memory usage is higher when dummy is being executed, and this may be
# problematic when DT is large.
```

```
## End(Not run)
```

Index