

# Package ‘BigSyn’

February 14, 2020

**Type** Package  
**Title** X  
**Version** 1.0  
**Date** 2020-02-14  
**Author** D. Bonnery  
**Maintainer** D. Bonnery <dbonnery@umd.edu>  
**Description** Data  
**Remotes** tidyverse/magrittr,  
DanielBonnery/StudyDataTools  
**Depends** devtools,  
ggplot2,  
sqldf,  
lattice,  
printr,  
knitr,  
reshape2,  
data.table,  
rlist,  
haven,  
sas7bdat,  
partykit  
**License** GPL (>= 2)  
**LazyLoad** yes  
**LazyData** true  
**RoxygenNote** 7.0.2

## R topics documented:

augmentT_f . . . . .	3
BigSyn . . . . .	6
compilefits . . . . .	6
compilesamlereports . . . . .	7

daniRules	8
drop_last	9
fitmodel.ctree	9
fitmodel.fn	10
fitmodel.rf	11
fitthemodel	11
GeneralReversetransposefunction	12
GeneralReversetransposefunctiondecoupe	13
Generaltransposefunction	14
Generaltransposefunctionsimple	14
getnodesfromrules	15
getpredictorsfromcaptureoutput	16
getpredictorsfromtree	16
get_cell	17
get_cellrn	17
get_cellXXgroup	18
get_cellXXmargincount	19
get_cellXXsplit	20
get_missingind	21
get_natural.predictors	21
get_origin	22
get_parent	23
get_presentind	24
get_var	25
get_XXpredecessoratmargin	25
get_XXpredecessorsatmargin	26
good.fit.parameters	27
good.sample.parameters	28
good.syn.parameters	28
NAt0	29
onlygoodargs	30
posixcttonumeric	30
predictor.matrix.default	31
predictor.matrix.rate	32
preparepredictorsfortreefit	32
reduceT_f	33
resampleT_f	34
runCompare	34
sample.ctree	35
sample.fn	35
sampladata	36
samplefrompool	37
SDPSYN2	37
sorttablewithingroup	39
Sparameters.default.f	40
Sparameters.variables.reorder.default	41
treedepth	43
treetopdf	43

augmentT\_f 3

Tsampledata . . . . .	44
TTsampledata . . . . .	44
%notin% . . . . .	45

**Index** 46

---

augmentT_f	<i>Convert cell totals to marginal ratios and create overall total.</i>
------------	---

---

**Description**

Convert cell totals to marginal ratios and create overall total.

**Usage**

```
augmentT_f(.data, variables, verbose = getOption("verbose"))
```

**Arguments**

.data	a dataframe
variables	a vector of character strings

**Details**

Assume one runs the program

augmentT\_f(dataBigSyn::STableA1,variables=c("AA.cont1","AA.cont1")). The program looks for all the "cell variables" corresponding to "AA.cont1", by using the function BigSyn::get\_var

The results is this:

```
AA.cont1_La_La_Lrn1, AA.cont1_La_Ld_Lrn1, AA.cont1_Lb_Lb_Lrn1, AA.cont1_Lc_La_Lrn1,
AA.cont1_Lc_Lb_Lrn1, AA.cont1_Lc_Ld_Lrn1, AA.cont1_La_Lb_Lrn1, AA.cont1_La_Lc_Lrn1,
AA.cont1_Lb_La_Lrn1, AA.cont1_Lb_Lc_Lrn1, AA.cont1_Lb_Ld_Lrn1, AA.cont1_Lc_Lc_Lrn1,
AA.cont1_La_La_Lrn2, AA.cont1_La_Ld_Lrn2, AA.cont1_Lb_Lb_Lrn2, AA.cont1_Lc_La_Lrn2,
AA.cont1_Lc_Lb_Lrn2, AA.cont1_Lc_Ld_Lrn2, AA.cont1_La_Lb_Lrn2, AA.cont1_La_Lc_Lrn2,
AA.cont1_Lb_La_Lrn2, AA.cont1_Lb_Lc_Lrn2, AA.cont1_Lb_Ld_Lrn2, AA.cont1_Lc_Lc_Lrn2,
AA.cont1_La_La_Lrn3, AA.cont1_La_Ld_Lrn3, AA.cont1_Lb_Lb_Lrn3, AA.cont1_Lc_La_Lrn3,
AA.cont1_Lc_Lb_Lrn3, AA.cont1_Lc_Ld_Lrn3, AA.cont1_La_Lb_Lrn3, AA.cont1_La_Lc_Lrn3,
AA.cont1_Lb_La_Lrn3, AA.cont1_Lb_Lc_Lrn3, AA.cont1_Lb_Ld_Lrn3, AA.cont1_Lc_Lc_Lrn3,
AA.cont1_La_La_Lrn4, AA.cont1_La_Ld_Lrn4, AA.cont1_Lb_Lb_Lrn4, AA.cont1_Lc_La_Lrn4,
AA.cont1_Lc_Lb_Lrn4, AA.cont1_Lc_Ld_Lrn4
```

The programs computes the number of marginal variables with the function looks for BigSyn::get\_cellXXmarginscount. Here it is 3

The program creates the following character matrix, named patterns:

```
"1" "La" ""
"1" "Lb" ""
"1" "Lc" ""
```

"2" "La\_La" "La"  
 "2" "La\_Ld" "La"  
 "2" "Lb\_Lb" "Lb"  
 "2" "Lc\_La" "Lc"  
 "2" "Lc\_Lb" "Lc"  
 "2" "Lc\_Ld" "Lc"  
 "2" "La\_Lb" "La"  
 "2" "La\_Lc" "La"  
 "2" "Lb\_La" "Lb"  
 "2" "Lb\_Lc" "Lb"  
 "2" "Lb\_Ld" "Lb"  
 "2" "Lc\_Lc" "Lc"  
 "3" "La\_La\_Lrn1" "La\_La"  
 "3" "La\_Ld\_Lrn1" "La\_Ld"  
 "3" "Lb\_Lb\_Lrn1" "Lb\_Lb"  
 "3" "Lc\_La\_Lrn1" "Lc\_La"  
 "3" "Lc\_Lb\_Lrn1" "Lc\_Lb"  
 "3" "Lc\_Ld\_Lrn1" "Lc\_Ld"  
 "3" "La\_Lb\_Lrn1" "La\_Lb"  
 "3" "La\_Lc\_Lrn1" "La\_Lc"  
 "3" "Lb\_La\_Lrn1" "Lb\_La"  
 "3" "Lb\_Lc\_Lrn1" "Lb\_Lc"  
 "3" "Lb\_Ld\_Lrn1" "Lb\_Ld"  
 "3" "Lc\_Lc\_Lrn1" "Lc\_Lc"  
 "3" "La\_La\_Lrn2" "La\_La"  
 "3" "La\_Ld\_Lrn2" "La\_Ld"  
 "3" "Lb\_Lb\_Lrn2" "Lb\_Lb"  
 "3" "Lc\_La\_Lrn2" "Lc\_La"  
 "3" "Lc\_Lb\_Lrn2" "Lc\_Lb"  
 "3" "Lc\_Ld\_Lrn2" "Lc\_Ld"  
 "3" "La\_Lb\_Lrn2" "La\_Lb"  
 "3" "La\_Lc\_Lrn2" "La\_Lc"  
 "3" "Lb\_La\_Lrn2" "Lb\_La"  
 "3" "Lb\_Lc\_Lrn2" "Lb\_Lc"  
 "3" "Lb\_Ld\_Lrn2" "Lb\_Ld"  
 "3" "Lc\_Lc\_Lrn2" "Lc\_Lc"  
 "3" "La\_La\_Lrn3" "La\_La"

```

"3" "La_Ld_Lrn3" "La_Ld"
"3" "Lb_Lb_Lrn3" "Lb_Lb"
"3" "Lc_La_Lrn3" "Lc_La"
"3" "Lc_Lb_Lrn3" "Lc_Lb"
"3" "Lc_Ld_Lrn3" "Lc_Ld"
"3" "La_Lb_Lrn3" "La_Lb"
"3" "La_Lc_Lrn3" "La_Lc"
"3" "Lb_La_Lrn3" "Lb_La"
"3" "Lb_Lc_Lrn3" "Lb_Lc"
"3" "Lb_Ld_Lrn3" "Lb_Ld"
"3" "Lc_Lc_Lrn3" "Lc_Lc"
"3" "La_La_Lrn4" "La_La"
"3" "La_Ld_Lrn4" "La_Ld"
"3" "Lb_Lb_Lrn4" "Lb_Lb"
"3" "Lc_La_Lrn4" "Lc_La"
"3" "Lc_Lb_Lrn4" "Lc_Lb"
"3" "Lc_Ld_Lrn4" "Lc_Ld"

```

Then for all i in 3:1 (starting with the maximum depth) list the different aggregations to the upper level to perform. So for i=3, aggregating to the second level will be done by computing the variables : AA.cont1\_La\_La, AA.cont1\_La\_Ld, AA.cont1\_Lb\_Lb, AA.cont1\_Lc\_La, AA.cont1\_Lc\_Lb, AA.cont1\_Lc\_Ld, AA.cont1\_La\_Lb, AA.cont1\_La\_Lc, AA.cont1\_Lb\_La, AA.cont1\_Lb\_Lc, AA.cont1\_Lb\_Ld, AA.cont1\_Lc\_Lc

For example `AA.cont1_La_La=rowSums(.data[,c("AA.cont1_La_La_Lrn1", "AA.cont1_La_La_Lrn2", "AA.cont1_La_La_Lrn3", "AA.cont1_La_La_Lrn4"),drop=FALSE])`

For i=2 aggregating to the upper level will be done by computing the variables : AA.cont1\_La, AA.cont1\_Lb, AA.cont1\_Lc AA.cont1\_La=rowSums(.data[,c("AA.cont1\_La\_La", "AA.cont1\_La\_Ld", "AA.cont1\_La\_Lb", "AA.cont1\_La\_Lc"),drop=FALSE])

For i=1 aggregating to the upper level will be done by computing the variable `AA.cont1_=rowSums(.data[,c("AA.cont1_La", "AA.cont1_Lb", "AA.cont1_Lc"),drop=FALSE])`

The computation of the marginal totals is done, the second step is the computation of the marginal ratios.

It is done by looping on the rows of the patterns matrix

Line j of pattern is a length 3 character vector. let call `patterns[j,2]` x and `patterns[j,3]` y The programs replaces the variable names `paste0("AA.cont1",x)` by the ratio of the variable `paste0("AA.cont1",x)` by the variable named `paste0("AA.cont1",y)`.

For example for the line "3" "La\_Ld\_Lrn3" "La\_Ld", the following replacement will be made:  
`AA.cont1_La_Ld_Lrn3=AA.cont1_La_Ld_Lrn3/AA.cont1_La_Ld`

The same is applied to all the elements of the input parameter variables.

## Value

a dataframe.

## Examples

```
.data=BigSyn::STtableA1
variable="AA.cont1"
variables=variable
STtableA1<-augmentT_f(.data,variables)
STtableA1$AA.cont1_[6]
STtableA1[6,names(STtableA1)[get_var(names(STtableA1))=="AA.cont1"]]
sum(STtableA1[6,names(STtableA1)[get_var(names(STtableA1))=="AA.cont1"]],na.rm=TRUE)
STtableA1[6,"AA.cont1_Lc_La_Lrn1"]
STtableA1[6,"AA.cont1_Lc_La_Lrn1"]
```

---

BigSyn

*BigSyn: Some non confidential R functions developped for the MLDSC synthetic data project*

---

## Description

The BigSyn package allows to synthesize big hierarchical databases by opposition to just a single small "rectangular" table. The general idea is to - provide tools to transpose the data and back transpose the synthetic version of the transposed data. - provide a synthetisation procedure that runs the modeling and the sampling separately - provide tools to operate a reasonable pre-selection of predictors. - provide tools to visualize the synthetisation.

## BigSyn functions

The main BigSyn functions are SDPSYN2 Generaltransposefunction GeneralReversetransposefunction

## General approach

NA

---

compilefits

*Save a pdf image of each regression tree grown in the modeling phase and discard useless information*

---

## Description

For each element of save parameters, look at the tree and produces the corresponding pdf. It also removes all the information that is stored in the ouptut of parykit::Ctree, e.g. the data. It only keeps the tree and the rules to get it.

**Usage**

```
compilefits(
  Sparameters,
  fitmodelsavepath,
  pdfpath = fitmodelsavepath,
  .progress = "text"
)
```

**Arguments**

pdfpath            where to save the pdfs

Sparameters:      a list, that has the same structure than the outputs of  
fitmodelsavepath:  
                    a file path where to store the pdf of the plot

.progress:        a string, name of the progress bar to use, see `plyr::create_progress_bar`

**Details**

Depends on `plyr`. Partykit output contain all the data that was used to grow the tree. this function removes the unwanted information.

**Examples**

```
y=iris$Species;x=iris[,-5]
partykitctree <- partykit::ctree(y ~ ., data=cbind(y=y,x))
```

---

compilesamplereports    *compilesamplereports*

---

**Description**

Sample reports are the output of the function `ReportonSample`

**Usage**

```
compilesamplereports(Sparameters, samplereportssavepath)
```

**Arguments**

Sparameters:      a list, that has the same structure than the outputs of  
samplereportssavepath:  
                    a file path where to store the sample reports

**Details**

depends on `plyr`

**See Also**

ReportonSample

**Examples**

```
y=iris$Species;x=iris[,-5]
partykitctree <- partykit::ctree(y ~ ., data=cbind(y=y,x))
```

---

daniRules

*daniRules*


---

**Description**

the rules for the tree. Add an extra rule. For example, if the left branch rule is X in (a,b,c) and the right branch rule is X in (e,f), the right branch rule is modified by X not in (a,b,c). The extreme right branch rule is replaced by the negation of any of the other branches, for each node.

**Usage**

```
daniRules(x, i = NULL, ...)
```

**Arguments**

x                    same format than output of partykit::ctree  
i                    (default: NULL)

**Details**

Adapted from partykit:::list.rules.party

**Value**

for each terminal node of the tree, give the definition. For example: node 11 corresponds to 'x>1 and y in ("a","b")'

**Examples**

```
x=partykit::ctree(Petal.Width~.,iris)
daniRules(x)
```



---

drop_last	<i>Drop last margin position (Trims all strings of a vector of strings after the last "_")</i>
-----------	--

---

**Description**

Drop last margin position (Trims all strings of a vector of strings after the last "\_")

**Usage**

```
drop_last(x)
```

**Arguments**

x                      a vector of character strings

**Details**

if x is "AA.char1\_La\_Ld\_Lrn1" returns "AA.char1\_La\_Ld", if x contains no "\_", returns empty string

**Value**

a vector of character strings

**Examples**

```
drop_last("AA.char1_La_Ld_Lrn1")
drop_last("iojoi")
drop_last("aa.iojoi")
```

---

fitmodel.ctree	<i>Function to fit a ctree model.</i>
----------------	---------------------------------------

---

**Description**

Function to fit a ctree model.

**Usage**

```
fitmodel.ctree(x, y, treeplotsavepath = NULL, ...)
```

**Arguments**

x                      a dataframe of predictors  
y                      a vector :dependent variable  
treeplotsavepath:      a path to save the graph as a pdf. if NULL, no pdf is saved

**Value**

a named list of 4 elements: "Rules" a data.frame with two variables: terminalnode (a integer vector) and condition a string that gives the rule for each terminal node. "y" the values of the predictor "terminalnodes" a vector: the terminal nodes for each element of \$y\$. "shortlist" a character string giving the names of the variables in x that were used for the classification

**Examples**

```
fitmodel.ctree(x=iris[,-5],y=iris$Species)
```

---

fitmodel.fn

*Fit a model with a specific function*


---

**Description**

Fit a model with a specific function

**Usage**

```
fitmodel.fn(method, x, y, treeplotsavepath = NULL, ...)
```

**Arguments**

method	a string. currently only method="ctree" or "rf" (random forest).
x	a predictors, a dataframe.
y	variable to predict, a vector
treeplotsavepath	a
...	synthetic parameters to pass to the right fit model function. the fit model function name is the concatenation of "fit.model" and method

**Value**

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

**Examples**

```
fitmodel.fn(method="ctree",x=iris[,-5],y=iris$Species,nbuckets=30,tutu="not a good argument")
```

---

fitmodel.rf	<i>Function to fit a ctree model.</i>
-------------	---------------------------------------

---

**Description**

Function to fit a ctree model.

**Usage**

```
fitmodel.rf(x, y, treeplotsavepath = NULL, ...)
```

**Arguments**

x	a dataframe of predictors
y	a vector :dependent variable
treeplotsavepath:	a path to save the graph as a pdf. if NULL, no pdf is saved

**Value**

a named list of 4 elements: "Rules" a data.frame with two variables: terminalnode (a integer vector) and condition a string that gives the rule for each terminal node. "y" the values of the predictor "terminalnodes" a vector: the terminal nodes for each element of \$y\$. "shortlist" a character string giving the names of the variables in x that were used for the classification

**Examples**

```
fitmodel.ctree(x=iris[,-5],y=iris$Species)
```

---

fitthemodel	<i>Function to fit the model.</i>
-------------	-----------------------------------

---

**Description**

Function to fit the model.

**Usage**

```
fitthemodel(
  Sparameters_i,
  fitmodelsavepath,
  TtableANAt0,
  redcomputationsevenifexists = FALSE,
  treeplotsavefolder = NULL
)
```

**Arguments**

`Sparameters_i` an element of the list output from `Sparameters.default.f`

`fitmodelsavepath`  
a folder path. Results will either be read from or stored in this folder. If the file exists, by default it is not replaced.

`TtableANato0` a table containing the predictors without NAs as well as the outcome

**Value**

a list.

**Examples**

```
#Load the data
data(TtableA,package="BigSyn")
#define parameters
Sparameters<-Sparameters.default.f(ref.table=TtableA)
Sparameters_i<-Sparameters[[53]];
fitmodelsavepath<-NULL;
TtableANato0<-Nato0(TtableA);
redocomputationsevenifexists<-FALSE
treeplotsavefolder=tempdir()
#fit the model:
fitthmodel(Sparameters_i,fitmodelsavepath = NULL,TtableANato0 = TtableANato0,
           treeplotsavefolder=tempdir())
Sparameters_i<-Sparameters[["AA.present_La_La_Lrn1"]];
treeplotsavefolder=tempdir()
fitthmodel(Sparameters_i,NULL,TtableANato0,treeplotsavefolder=tempdir())
```

---

GeneralReversetransposefunction

*General Reverse Transpose function*

---

**Description**

General Reverse Transpose function

**Usage**

`GeneralReversetransposefunction(TtableA, key)`

**Arguments**

`key` A list of variables (columns of the transposed table)

`table` A dataframe

**Value**

A list: first element of the list is a dataframe, the transposed version of the original table. Second element is a key to allow back transposition

**Examples**

```
data(tableA);data(TtableA);data(XKA);key<-XKA$key
RtableA=GeneralReversetransposefunction(TtableA,key)
ordertableA <-do.call(order,tableA[c(id1,id2)])
orderRtableA<-do.call(order,RtableA[c(id1,id2)])
identical(nrow(tableA),nrow(RtableA))
identical(lapply(tableA,class),lapply(RtableA,class))
identical(tableA[ordertableA,],RtableA[orderRtableA,])
identical(names(tableA),names(RtableA))
all (lapply(names(tableA),function(x){identical(tableA[ordertableA,x],RtableA[orderRtableA,x])}))
```

---

GeneralReversetransposefunctiondecoupe

*General Reverse Transpose function with split*

---

**Description**

General Reverse Transpose function with split

**Usage**

```
GeneralReversetransposefunctiondecoupe(.data, key, nrowmax = 10000)
```

**Arguments**

key	A list of variables (columns of the transposed table)
table	A dataframe

**Value**

A list: first element of the list is a dataframe, the transposed version of the original table. Second element is a key to allow back transposition

**Examples**

```
data(tableA);data(TtableA);data(XKA);key<-XKA$key
RtableA=GeneralReversetransposefunctiondecoupe(TtableA,key,10)
```

---

Generaltransposefunction
<i>General Transpose function</i>

---

**Description**

General Transpose function

**Usage**

```
Generaltransposefunction(  
  tableA,  
  id1,  
  id2,  
  origin = deparse(substitute(tableA))  
)
```

**Arguments**

id1	A list of variables (rows)
id2	A list of variables (columns of the transposed table), id2 can contain as a last element the strint "rn", if the variable rn is an index for the cells formed by the variables listed first in id2
table	A dataframe

**Value**

A list: first element of the list is a dataframe, the transposed version of the orioginal table. Second element is a key to allow back transposition

**Examples**

```
tableA<-sampledata(TRUE)  
id1=c("id1a","id1b")  
id2=c("id2a","id2b")  
TtableA<-Generaltransposefunction(tableA,id1,id2)
```

---

Generaltransposefunctionsimple
<i>Simple General Transpose function</i>

---

**Description**

Simple General Transpose function

**Usage**

```
Generaltransposefunctionsimple(tableA, id1, id2)
```

**Arguments**

tableA	A dataframe
id1	A list of variables (rows)
id2	A list of variables (columns of the transposed table)

**Value**

A data frame

**Examples**

```
tableA<-sampledata(TRUE)
id1=c("id1a","id1b")
id2=c("id2a","id2b")
TtableA<-Generaltransposefunctionsimple(tableA,id1,id2)
```

---

getnodesfromrules	<i>Function to get terminal node from a set of partitioning rules and new predictors</i>
-------------------	--

---

**Description**

Function to get terminal node from a set of partitioning rules and new predictors

**Usage**

```
getnodesfromrules(x, Rules)
```

**Arguments**

x	a dataframe of predictors
Rules	a data frame containing 2 character variables: "terminalnode" and "condition"

**Value**

a vector of length the number of rows of x indicating the terminal nodes.

---

```
getpredictorsfromcaptureoutput
  getpredictorsfromcaptureoutput
```

---

**Description**

for each terminal node of the tree, give the elements of "predictors" who appear in the node definition

**Usage**

```
getpredictorsfromcaptureoutput(tree, predictors)
```

**Arguments**

tree	same format than output of partykit::ctree
predictors	list of variables

**See Also**

daniRules

---

```
getpredictorsfromtree  getpredictorsfromtree
```

---

**Description**

for each terminal node of the tree, give the elements of "predictors" who appear in the node definition

**Usage**

```
getpredictorsfromtree(tree, predictors)
```

**Arguments**

tree	same format than output of partykit::ctree
predictors	list of variables

**See Also**

daniRules

**Examples**

```
tree<-partykit::ctree(Petal.Width~.,iris)
getpredictorsfromtree(tree,names(iris))
```



---

get_cell	<i>get cell without the row number</i>
----------	--

---

**Description**

get cell without the row number

**Usage**

```
get_cell(x, iscellrn = FALSE, iscell = FALSE)
```

**Arguments**

x                      a vector of character strings

**Details**

if x is "aa.xoijj\_a\_1\_f\_1\_" returns "a\_1\_f"

**Value**

a vector of character strings

**Examples**

```
get_cell("aa.x_1_2_3_4")#default
get_cell("1_2_3", TRUE)
get_cell("1_2_3", FALSE, TRUE)
unique(Tsampledata(TRUE)$variables))
unique(get_cell(Tsampledata(FALSE)$variables))
```

---

get_cellrn	<i>Get cell and row number</i>
------------	--------------------------------

---

**Description**

Get cell and row number

**Usage**

```
get_cellrn(x)
```

**Arguments**

x                      a vector of character strings

**Details**

if x is "aa.x\_a\_1\_f\_1" returns "a\_1\_f\_1"

**Value**

a vector of character strings

**Examples**

```
get_cellrn("AA.char1_La_Ld_Lrn1")
data(TtableA);
unique(get_cellrn(names(TtableA)))
#Second example: no transposing variables
data(TtableB);data(XKB)
unique(get_cellrn(names(XKB)))
```

---

get_cellXXgroup	<i>Get cell group</i>
-----------------	-----------------------

---

**Description**

Get cell group

**Usage**

```
get_cellXXgroup(x, marginpos, iscellXX = TRUE)
```

**Arguments**

- x                    a vector of character strings
- marginpos           a vector of integer

**Details**

#' if x is "a\_1\_f\_2\_aa.xoijj",marginpos=2 returns "1"; if x is "a\_1\_f\_2\_aa.xoijj",marginpos=-2 returns "a\_f\_2"; if x is "a\_1\_f\_2\_aa.xoijj",marginpos=c(1:2) returns "a\_1"

**Value**

a vector of character strings

**Examples**

```

get_cellXXgroup(c("aa.x_1_2_3_4", "bb.x_1_2_3_4"), 2, iscellXX=FALSE)
get_cellXXgroup(c("1_2_3_4", "1_2_3_4"), 2:3, iscellXX=TRUE)
variables<-Tsampledata(TRUE)$variables
unique(get_cellXXgroup(variables, 2, iscellXX=FALSE))
unique(get_cellXXgroup(variables, -2, iscellXX=FALSE))
get_cellXXgroup(variables[50], 2, iscellXX=FALSE)
get_cellXXgroup(variables[50], -2, iscellXX=FALSE)

#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXgroup(TK$variable, 1, iscell=FALSE))

```

---

```
get_cellXXmarginscount
```

*Get the number of margins for a cell*

---

**Description**

Get the number of margins for a cell

**Usage**

```
get_cellXXmarginscount(x, iscellXX = FALSE)
```

**Arguments**

x	a vector of character strings
iscell	a boolean indicating if x is a variable name or a cell name.

**Details**

if x is "aa.xoijj\_a\_1\_f\_1", cell=FALSE returns 4"; if x is "a\_1\_f\_1", cell=TRUE returns 4"

**Value**

a vector of integers.

**Examples**

```

get_cellXXmarginscount("1_2_3_4", iscellXX=TRUE)
get_cellXXmarginscount("aa.x_1_2_3_4", iscellXX=FALSE)
data(TtableA)
unique(get_cellXXmarginscount(names(TtableA), iscellXX=FALSE))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXmarginscount(TK$variables))

```

---

get_cellXXsplit	<i>split a cell</i>
-----------------	---------------------

---

## Description

split a cell

## Usage

```
get_cellXXsplit(x, marginpos = NULL, iscellXX = FALSE)
```

## Arguments

x	a vector of character strings
iscell	x a boolean indicating if x is a cell

## Details

if x is "aa.xoijj\_a\_1\_f\_1" returns c("a","1","f","1")

## Value

a vector of character strings

## Examples

```
get_cellXXsplit("aa.x_1_2_3_4", iscellXX=FALSE)
get_cellXXsplit("1_2_3_4", iscellXX=TRUE)
get_cellXXsplit("1_2_3_4", 2:3, iscellXX=TRUE)
get_cellXXsplit("1_2_3_4", -(2:3), iscellXX=TRUE)
variables<-Tsampledata(TRUE)$variables
unique(get_cellXXsplit(variables, iscell=FALSE))
get_cellXXsplit(variables[50], iscell=FALSE)
get_cellXXsplit(variables[50], -(2:3), iscell=FALSE)
unique(get_cellXXsplit(variables, 2, iscell=FALSE))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXsplit(TK$variables, iscell=FALSE))
```

---

get_missingind	<i>Get missing indicator for a cell or variable</i>
----------------	---

---

**Description**

Get missing indicator for a cell or variable

**Usage**

```
get_missingind(x, variables)
```

**Arguments**

x                      a vector of character strings

**Details**

if x is "a\_1\_f\_1\_aa.xoijj" returns c("a","1","f","1")

**Value**

a vector of character strings

**Examples**

```
variables<-Tsampledata(TRUE)$variables
unlist(unique(get_missingind(variables,variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_missingind(variables,variables)))
```

---

get_natural.predictors	<i>get variable predecessors at margin</i>
------------------------	--

---

**Description**

get variable predecessors at margin

**Usage**

```
get_natural.predictors(x, variables = x, predictors = NULL)
```

Arguments

- x a vector of character strings
- variables a vector of character strings
- cells a vector of character strings containing the potential predecessors
- marginpos a vector of integers

Details

if x is "a\_1\_f\_1\_aa.xoiij" returns c("a","1","f","1")  
if x is "a\_1\_f\_1\_aa.xoiij" returns c("a","1","f","1")

Value

- a vector of character strings
- a vector of character strings

Examples

```
get_XXpredecessoratmargin(cellXXs="aa.x_1_2_3_4", refcellXXs=c("bb.x_1_2_2_4","aa.x_1_2_2_4","aa.x_1_1_3_4"),2,  
get_XXpredecessoratmargin(cellXXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=FALSE)  
data(XKA)  
cells<-unique(get_cellrn(XKA$variables))  
get_XXpredecessoratmargin(cells,marginpos=1,iscellXX=TRUE)  
get_XXpredecessoratmargin(cells[10],cells,1,iscellXX=TRUE)  
Get natural predictors  
  
TK<-TtableA  
get_natural.predictors(x=sample(names(TtableA),5),variables=names(TtableA))
```

---

get_origin	<i>Get origin table</i>
------------	-------------------------

---

Description

Get origin table

Usage

```
get_origin(x)
```

Arguments

- x a vector of character strings

Details

if x is "aa.xoiij\_a\_1\_f\_1\_" returns c("aa")

**Value**

a vector of character strings

**Examples**

```
get_origin("tableA.cont1_1_Lrn1")
variables<-Tsampledata(TRUE)$variables
unlist(unique(get_origin(variables,variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_origin(variables,variables)))
```

---

get_parent	<i>get parent if any</i>
------------	--------------------------

---

**Description**

get parent if any

**Usage**

```
get_parent(variables, variable_ref)
```

**Arguments**

- variables      a character strings
- variable\_ref    a vector of character strings

**Details**

if x is "aa.xoijj\_a\_1\_f\_1\_" returns "a\_1\_f"

**Value**

a vector of character strings

**Examples**

```
get_parent("aa.x_1_2_3_4","aa.x_1_2_3")#default
```

---

get_presentind	<i>get the present indicator for a cell</i>
----------------	---

---

## Description

get the present indicator for a cell

## Usage

```
get_presentind(
  variables,
  refvariables = variables,
  rns = unlist(unique(get_cellrn(refvariables)))
)
```

## Arguments

x                      a vector of character strings

## Details

if x is "a\_1\_f\_1\_aa.xoiij" returns c("a","l","f","l")

## Value

a vector of character strings

## Examples

```
get_presentind("AA.x_1_2_3_4", "AA.present_1_2_3_4")
get_presentind("AA.present_1_2_3_4", c("AA.present_1_2_3_3", "AA.present_1_2_3"))
get_presentind("AA.present_1_2_3_4", c("AA.present_1_2_3_3", "AA.present_1_2_3_4"))
variables<-Tsampledata(TRUE)$variables
variable<-"AA.present_La_La_Lrn1"
get_presentind(variable, variables)
unlist(unique(get_presentind(variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_presentind(variables, variables)))
```



---

get_var	<i>Get variable name</i>
---------	--------------------------

---

**Description**

Get variable name

**Usage**

```
get_var(x)
```

**Arguments**

x                      a vector of character strings

**Details**

if x is "aa.xoiij\_a\_1\_f\_1" returns "aa.xoiij"

**Value**

a vector of character strings

**Examples**

```
get_var("aa.x_1_2_3_4")
data(TtableA)
unique(get_var(names(TtableA)))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_var(TK$variables))
```

---

get_XXpredecessoratmargin	<i>get cell predecessors at margin</i>
---------------------------	--

---

**Description**

get cell predecessors at margin

**Usage**

```
get_XXpredecessoratmargin(
  XXs,
  refXXs = XXs,
  marginpos = NULL,
  iscellXX = FALSE
)
```

**Arguments**

XXs	a vector of character strings
refXXs	a vector of character strings containing the potential predecessors
marginpos	a vector of integers

**Details**

if XXs is "aa.xoiij\_a\_1\_f\_1" and refXXs contains "aa.xoiij\_a\_1\_e\_1" and marginpos=3 returns "aa.xoiij\_a\_1\_e\_1" if XXs is "aa.xoiij\_a\_1\_f\_2" and refXXs contains "aa.xoiij\_a\_1\_f\_1" and marginpos=NULL returns "aa.xoiij\_a\_1\_f\_1" if XXs is "id1" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "b\_1\_f\_1" and iscellXX=TRUE and refXXs contains "a\_1\_f\_1" returns "a\_1\_f\_1"

**Value**

a vector of character strings

**Examples**

```
get_XXpredecessoratsmargin(XXs="aa.x_1_2_3_4", refXXs=c("bb.x_1_2_2_4", "aa.x_1_2_2_4", "aa.x_1_1_3_4"), 2, iscellXX=FALSE)
get_XXpredecessoratsmargin(XXs=c("1_2_2_4", "1_2_2_4", "1_1_3_4", "1_1_3_3"), iscellXX=TRUE)
get_XXpredecessoratsmargin(XXs="1_1_3_4", refXXs=c("1_2_2_4", "1_2_2_4", "1_1_3_4", "1_1_3_3"), iscellXX=TRUE)
data(XKA)
cells<-unique(get_cellrn(XKA$variables))
get_XXpredecessoratsmargin(cells, marginpos=1, iscellXX=TRUE)
get_XXpredecessoratsmargin(cells[10], cells, 1, iscellXX=TRUE)
```

---

get\_XXpredecessorsatmargin  
*get cell predecessors at margin*

---

**Description**

get cell predecessors at margin

**Usage**

```
get_XXpredecessorsatmargin(
  XXs,
  marginpos,
  refXXs = XXs[order(get_cellXXgroup(XXs, marginpos))],
  iscellXX = FALSE,
  cellXXgroup = get_cellXXgroup(refXXs, marginpos2, iscellXX),
  CompcellXXgroup = get_cellXXgroup(refXXs, -marginpos2, iscellXX)
)
```

**Arguments**

XXs	a vector of character strings
marginpos	a vector of integers
refXXs	a vector of character strings containing the potential predecessors

**Details**

if XXs is "aa.xoijj\_a\_1\_f\_1" and refXXs contains "aa.xoijj\_a\_1\_e\_1" and marginpos=3 returns "aa.xoijj\_a\_1\_e\_1" if XXs is "aa.xoijj\_a\_1\_f\_2" and refXXs contains "aa.xoijj\_a\_1\_f\_1" and marginpos=NULL returns "aa.xoijj\_a\_1\_f\_1" if XXs is "id1" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "b\_1\_f\_1" and iscellXX=TRUE and refXXs contains "a\_1\_f\_1" returns "a\_1\_f\_1"

**Value**

a vector of character strings

**Examples**

```
get_XXpredecessoratmargin(XXs="aa.x_1_2_3_4", refXXs=c("bb.x_1_2_2_4", "aa.x_1_2_2_4", "aa.x_1_1_3_4"), 2, iscellXX=FALSE)
get_XXpredecessoratmargin(XXs=c("1_2_2_4", "1_2_2_4", "1_1_3_4", "1_1_3_3"), iscellXX=TRUE)
get_XXpredecessoratmargin(XXs="1_1_3_4", refXXs=c("1_2_2_4", "1_2_2_4", "1_1_3_4", "1_1_3_3"), iscellXX=TRUE)
data(XKA)
cells<-unique(get_cellrn(XKA$variables))
get_XXpredecessoratmargin(cells, marginpos=1, iscellXX=TRUE)
get_XXpredecessoratmargin(cells[10], cells, 1, iscellXX=TRUE)
```

---

good.fit.parameters      *Select only arguments for a specific fitting function*

---

**Description**

Select only arguments for a specific fitting function

**Usage**

```
good.fit.parameters(method, synparameters)
```

**Arguments**

method	a string. currently only method="ctree".
synparameters	a named list.

**Details**

Currently only works with method="ctree" Only selects the arguments that match the function par-tykit::ctree\_control

**Value**

a sublist of synparameters, which names are possible arguments of partykit::ctree\_control if method="ctree".

**Examples**

```
good.fit.parameters(method="ctree",list(teststat=30,tutu=3))
```

---

```
good.sample.parameters
```

*Select only arguments for sampling.*

---

**Description**

Select only arguments for sampling.

**Usage**

```
good.sample.parameters(method, synparameters)
```

**Arguments**

method            a string. currently only method="ctree".  
synparameters    a named list.

**Details**

In prevision of future developments. returns NULL for the moment

**Examples**

```
good.sample.parameters(method="ctree",list(teststat=30,tutu=3))
```

---

```
good.syn.parameters    Select only arguments for a specific fitting function This function is not  
                         used anymore
```

---

**Description**

Select only arguments for a specific fitting function This function is not used anymore

**Usage**

```
good.syn.parameters(method, synparameters)
```

**Arguments**

method            a string. currently only method="ctree".  
 synparameters    a named list.

**Details**

Currently only works with method="ctree" Only selects the arguments that match the function synthpop::syn.ctree

**Value**

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

**Examples**

```
good.syn.parameters(method="ctree",list(y=c(1:2),smoothing=TRUE,tutu="not in synthpop::syn.ctree arguments"))
```

---

NAto0	<i>Recoding of NAs to 0 or "NA"</i>
-------	-------------------------------------

---

**Description**

Recoding of NAs to 0 or "NA"

**Usage**

```
NAto0(tableA)
```

**Arguments**

tableA            a dataframe

**Details**

for synthetisation to run, missing values are treated as a special factor level for factor variables, or as 0 for continuous variables. To avoid issues, for continuous variables, a missing indicator is also created.

**Value**

a dataframe

**Examples**

```
toto<-cars
toto$speed[sample(nrow(cars),3)]<-NA
NAto0(toto)
```

---

onlygoodargs	<i>Generic function: remove all the elements of a named list which names are not arguments of a specific function.</i>
--------------	--

---

### Description

Generic function: remove all the elements of a named list which names are not arguments of a specific function.

### Usage

```
onlygoodargs(fun, L)
```

### Arguments

fun	a function
L	a list

### Details

remove all the elements of a list which names are not arguments of a specific function.

### Value

a list.

### Examples

```
onlygoodargs(lm,list(data=cars,formula=speed~dist,tutu="not an arg from lm"))
```

---

posixcttonumeric	<i>Converts all posixct variables of a dataframe into a numeric variable</i>
------------------	--

---

### Description

Converts all posixct variables of a dataframe into a numeric variable

### Usage

```
posixcttonumeric(tableA)
```

### Arguments

tableA	a dataframe
--------	-------------

### Value

a list.

### Examples

```
toto<-cars
toto$now<-Sys.time()
posixcttonumeric(toto)
```

---

```
predictor.matrix.default
```

*Define a default predictor matrix*

---

### Description

Define a default predictor matrix

### Usage

```
predictor.matrix.default(variables)
```

### Arguments

`variables`      a vector of character strings

### Details

Returns the lower diagonal matrix with ones.

### Value

a matrix

### Examples

```
variables<-Tsampledata(TRUE)$variables
predictor.matrix.default(TK$variables)
```

---

`predictor.matrix.rate` *predictor.matrix.rate*

---

### Description

`predictor.matrix.rate`

### Usage

```
predictor.matrix.rate(
  variables,
  nopredictor = character(0),
  allpredictor = character(0),
  marginposs = integer(0)
)
```

### Arguments

`x` a vector of character strings

### Details

if `x` is "aa.xoijj\_a\_1\_f\_1\_" returns `c("a","l","f","l")`

### Value

a vector of character strings

---

`preparepredictorsfortreefit`  
*Prepare predictors for ctree fit*

---

### Description

Prepare predictors for ctree fit

### Usage

```
preparepredictorsfortreefit(x, keep = NULL)
```

### Arguments

`x` a predictors, a dataframe.  
`method` a string. currently only `method="ctree"`.  
`y` variable to predict, a vector  
`...` synthetic parameters to pass to the right fit model function. the fit model function name is the concatenation of "fit.model" and method



**Value**

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

**Examples**

```
preparepredictorsfortreefit(x,
                             keep=NULL)
```

---

reduceT_f	<i>Reverse augmentT_f: Function that will convert cell and marginal ratios and overall total to cell values</i>
-----------	---

---

**Description**

Reverse augmentT\_f: Function that will convert cell and marginal ratios and overall total to cell values

**Usage**

```
reduceT_f(.data, variables, verbose = FALSE, hack = TRUE)
```

**Arguments**

.data	data frame to "reduce"
variables	list of variable names roots
verbose	(default FALSE) if verbose, the formulae to compute the new variables is printed.
hack	(default TRUE)

**Details**

this functions looks for the Augmentation parameters in the package object Augmentparameters[[tablename]]\$percent. For each variable listed in Augmentparameters[[tablename]]\$percent, it looks for the corresponding variable in .data and computes cell values from cell and marginal ratios and overall total

---

resampleT_f	<i>resample when synthetisation created incoherent high and low level aggregates</i>
-------------	--

---

**Description**

resample when synthetisation created incoherent high and low level aggregates

**Usage**

```
resampleT_f(.data, variables, verbose = FALSE)
```

**Arguments**

.data	data frame to "reduce"
variables	list of variable names roots
verbose	(default FALSE) if verbose, the formulae to compute the new variables is printed.

---



---

runCompare	<i>runCompare</i>
------------	-------------------

---

**Description**

Shiny App to visualize regression trees and compare synthetic vs non synthetic data

**Usage**

```
runCompare(
  data1 = NULL,
  data2 = NULL,
  listofpackage1 = installed.packages()[, "Package"],
  listofpackage2 = installed.packages()[, "Package"],
  package1 = if (is.element("BigSyn", listofpackage1)) { "BigSyn" } else {
    listofpackage1[1] },
  package2 = if (is.element("BigSyn", listofpackage2)) { "BigSyn" } else {
    listofpackage2[1] }
)
```

**Arguments**

data1	a dataframe
data2	a dataframe
listofpackage1	a vector of character strings
listofpackage2	a vector of character strings

package1        a character string  
 package2        a character string  
 Sparameters

### Examples

```
package1<-NULL
package2<-NULL
runCompare()
```

---

sample.ctree	<i>Function to sample from a ctree fitted model</i>
--------------	---

---

### Description

Function to sample from a ctree fitted model

### Usage

```
sample.ctree(xp, fit.model, smoothing = "none", ...)
```

### Arguments

y                a vector of values to pull from  
 terminalnodes    a vector of terminal nodes  
 newterminalnodes:  
                  a path to save the graph

### Value

a vector of the same size than terminalnodes, obtained by sampling between the values of y such for the same terminal node.

---

sample.fn	<i>Sample a model with a specific function</i>
-----------	--

---

### Description

Sample a model with a specific function

### Usage

```
sample.fn(method, xp, fit.model, smoothing, ...)
```

**Arguments**

method	a string. currently only method="ctree".
...	synthetic parameters to pass to the right fit model function. the fit model function name is the concatenation of "fit.model" and method
x	a predictors, a dataframe.
y	variable to predict, a vector

**Value**

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

**Examples**

```
sample.fn(method="ctree",
          xp=iris[, -5],
          fit.model=fitmodel.fn(method="ctree", x=iris[, -5], y=iris$Species, nbuckets=30),
          smoothing=FALSE)
```

---

sampledata

---

*Sample data for transposition*


---

**Description**

Sample data for transposition

**Usage**

```
sampledata(transposingvariables = TRUE)
```

**Arguments**

transposingvariables	a boolean. If TRUE, transposing id variables are created.
----------------------	---

**Value**

a data frame with id variables, numeric, factor and character variables.

---

samplefrompool	<i>Function to sample from a set of partitioning rules</i>
----------------	--

---

**Description**

Function to sample from a set of partitioning rules

**Usage**

```
samplefrompool(y, terminalnodes, newterminalnodes)
```

**Arguments**

y                    a vector of values to pull from  
terminalnodes    a vector of terminal nodes  
newterminalnodes:  
                  a path to save the graph

**Value**

a vector of the same size than terminalnodes, obtained by sampling between the values of y such for the same terminal node.

**Examples**

```
y=iris$Species;x=iris[,-5];fit.mod<-fitmodel.ctree(x,y);terminalnodes<-getnodesfromrules(x,fit.mod$Rules);  
newterminalnodes<-sample(unique(terminalnodes),10,replace=TRUE);  
samplefrompool(y,terminalnodes,newterminalnodes)  
y<-y[terminalnodes!=7]  
terminalnodes<-terminalnodes[terminalnodes!=7]  
samplefrompool(y,terminalnodes,newterminalnodes)
```

---

SDPSYN2	<i>General SDP function.</i>
---------	------------------------------

---

**Description**

General SDP function.

**Usage**

```
SDPSYN2(
  TtableA,
  asis = NULL,
  notpredictor = asis,
  nrep = 1,
  synparameters = NULL,
  Sparameters = Sparameters.default.f(ref.table = TtableA, asis = asis, notpredictor =
    notpredictor, preferredmethod = "ctree", defaultsynparameters =
    c(as.list(synparameters),
      eval(formals(Sparameters.default.f)$defaultsynparameters)[setdiff(names(formals(Sparameters.default.f),
        c("", names(synparameters))))])),
  STtableA = if (is.null(asis)) { data.frame(.n = rep(nrep, each = nrow(TtableA))) }
  else { plyr::ddply(data.frame(.n = nrep), ~.n, function(d) { TtableA[asis]
    }) },
  fitmodelsavepath = NULL,
  treeplotsavefolder = NULL,
  samplereportsavepath = NULL,
  stepbystepsavepath = NULL,
  doparallel = TRUE,
  recode = NULL,
  saveeach = 200,
  randomfitorder = TRUE,
  fitonly = FALSE
)
```

**Arguments**

TtableA	a dataframe to synthesize
asis	list of variable names from TtableA to keep as is (e.g. not to synthesize)
notpredictor	list of variable names which should not be used as predictors
nrep	number of synthetic replicates wanted
synparameters	general synthetisation paramters
Sparameters	a list, Specific (variable by variable) synthetisation parameters, splits ...
STtableA	a dataframe
fitmodelsavepath	a path where to save the fitted models
treeplotsavefolder	a path where to save the tree plots
samplereportsavepath	a path where to save the sampling report
stepbystepsavepath	a path where to backup the synthetised in case of a crash
doparallel	a boolean indicating whether sampling should be done in parallel for each replicate

- recode : a vector of character strings or NULL, list of variables to be recoded
- saveeach an integer, indicating every how many variables a backup is done
- randomfitorder a boolean : fitting for each variable can be done in the order of appearance of each variables or at random
- fitonly a boolean, if TRUE, no sampling is done.

Details

This function is doing both the fitting and the sampling.

Examples

```
data(TtableA,package="BigSyn")
TtableA$AA.cont1_La_La<-rowSums(TtableA[grep("AA.cont1_La_La",names(TtableA))])
asis=NULL;notpredictor=asis;nrep=1;synparameters=NULL;
Sparameters=Sparameters.default.f(ref.table=TtableA,asis=asis,notpredictor=notpredictor,preferredmethod="ctree")
defaultsynparameters=c(as.list(synparameters),
eval(formals(Sparameters.default.f)$defaultsynparameters)[setdiff(names(formals(Sparameters.default.f)$default
STtableA=plyr::rdply(nrep,TtableA[asis]);samplereportsavepath=NULL;stepbystepsavepath=NULL;doparallel=FALSE;re
fitmodelsavepath=tempdir()
treeplotsavefolder=tempdir()
sapply(list.files(tempdir(),full.names = TRUE ),file.remove)
STtableA<-SDPSYN2(TtableA,asis=NULL,synparameters = defaultsynparameters,fitmodelsavepath = fitmodelsavepath,tre
todisplay<-grep("La_La_Lrn1",names(STtableA[[1]]),value=T);
STtableA[[1]][1:3,todisplay];TtableA[1:3,todisplay]
```

---

sorttablewithingroup	<i>sort table within group without changing the group position</i>
----------------------	--

---

Description

sort table within group without changing the group position

Usage

```
sorttablewithingroup(.data, groupvar, sortvar, decreasing = FALSE)
```

Arguments

- .data a dataframe
- groupvar a vector of character strings that are names of variables from .data
- sortvar a vector of character strings that are names of variables from .data
- decreasing a boolean if TRUE, decreasing order is used.

**Details**

Groups are defined by unique values of .data[groupvar]. Within each group, data is sorted according to sortvar.

**Value**

a list.

**Examples**

```
set.seed(1)
N=10
.data=data.frame(
  .group=sample(letters[1:2],N,replace=TRUE),
  y=runif(N),
  origorder=1:N)
groupvar=".group"
sortvar="y"
.data2=plyr::ddply(.data,".group",function(d){d$intraorder=order(d[[sortvar]]);d$neworder=d$origorder[order(ori
.data4=.data2[.data2[[groupvar]]=="a",];.data4[.data4$intraorder,]
cbind(.data,"|",.data2)
cbind(.data,"|",.data2[order(.data2$origorder),])
cbind(.data[order(order(.data2$origorder)),],"|",.data2)
.data3=cbind(.data,"|",.data2[order(.data2$neworder),]);.data3[.data[[groupvar]]=="a",];.data3
.data3=cbind(.data,"|",.data[order(order(.data2$origorder)),][order(.data2$neworder),]);.data3[.data[[groupvar]]
.data3=cbind(.data,"|",.data[order(order(.data2$origorder))[order(.data2$neworder),]);.data3[.data[[groupvar]]
cbind(.data,I="|",sorttablewithingroup(.data,groupvar,sortvar),I="|",sorttablewithingroup(.data,groupvar,sortvar))
```

---

Sparameters.default.f *Default synthetisation parameters based on variable names*

---

**Description**

Default synthetisation parameters based on variable names

**Usage**

```
Sparameters.default.f(
  ref.table,
  asis = NULL,
  notpredictor = NULL,
  variables = Sparameters.variables.reorder.default(names(ref.table)),
  predictors.matrix = predictor.matrix.default(variables)[!is.element(variables, asis),
    !is.element(variables, notpredictor)],
  splittingvar = NULL,
  moresplits = NULL,
  preferredmethod = "ctree",
  splithreshold = 100,
```



```

    defaultsynparameters = list(smoothing = "none", importance = TRUE, keep.forest = TRUE,
                                minbucket = 30)
  )

```

### Arguments

ref.table	a dataframe
asis	a vector of character strings, indicating which variables to keep as is.
notpredictor	a vector of character strings, indicating which variables are not supposed to be used as predictors.
variables	a vector of character strings, indicating the variables to synthesize. Order is important.
predictors.matrix	a predictor matrix. Number of rows is the number of variables to synthesize, number of columns is all the variables from ref.table
moresplits	an object of class moresplit (not defined yet)
preferredmethod:	"rf" for random forest or "ctree" for classification tree
defaultparameters	a list indicating default parameters for synthpop synthesis functions, for example ntree=5, smoothing="none"

### Details

creates default synthetisation parameters Some rules: parents variable are potential predictors of their children, synthetisation is conditional to missing indicators, synthetisation is conditional to presence in cell

### Examples

```

data(TtableA)
ref.table<-TtableA
Sparameters.default.f(ref.table=TtableA)

```

---

Sparameters.variables.reorder.default

*General Default ordering of variables for synthetisation based on name of the variable.*

---

### Description

General Default ordering of variables for synthetisation based on name of the variable.

**Usage**

```
Sparameters.variables.reorder.default(
  variables,
  orderwithinorigin = NULL,
  id = NULL,
  extrasort = NULL
)
```

**Arguments**

`variables`            vector of character strings, indicating names of variables

`orderwithinorigin`    a list, see example

`id`                    a vector of character strings

**Details**

After transposition, variable names follow this format: origin.variablename\_margin1\_margin2....lastmargin  
Some rules have to be followed:

- Missing indicators have to be synthesised before the corresponding variables, for example AA.factor1missingind\_L1\_L2\_L1 needs to be synthesised before AA.factor1missingind\_L1\_L2\_L1
- Cell indicators must be synthesised before the corresponding variables. For example AA.present\_L1\_L2\_L1 must be synthesised before AA.factor1\_L1\_L2\_L1 and before AA.cont1\_L1\_L2\_L1
- Parent variables (aggregated) must be synthesised before their children: For example AA.present\_L1 must be synthesised before AA.present\_L1\_L2, AA.cont2\_L1\_L2 must be synthesised before AA.cont2\_L1\_L2\_L3 AA.present\_L1 must be synthesised before AA.present\_L1\_L3 AA.cont2missingind\_L1 must be synthesised before AA.cont2missingind\_L1\_L3
- if for examples variable AA.cont1 in each cell has to be synthesised before AA.cont2, this can be specified with the orderwithinorigin argument
- for the use of the argument extrasort, refer to sorttablewithingroup

**Value**

a list.

**Examples**

```
TK<-Tsampledata(TRUE)$TtableA
Sparameters.variables.reorder.default(names(TK$TtableA))
#Second example: no transposing variables
TtableA<-Tsampledata(TRUE)$TtableA
orderwithinorigin=c("AA.factor1","AA.factor2")
variables<-names(TtableA)
Sparameters.variables.reorder.default(variables,orderwithinorigin)
```

---

treedepth	<i>Compute depth of a "party" tree</i>
-----------	--

---

**Description**

this function takes the output of the partykit::ctree function and prints the tree into a pdf file, located in the specified folder. It computes the depth and width and tries to create a pdf with the right dimensions.

**Usage**

```
treedepth(x)
```

**Arguments**

x	a tree
---	--------

**Details**

recursive function

**Examples**

```
y=iris$Species;x=iris[,-5]
partyctree <- party::ctree(y ~ ., data=cbind(y=y,x))
treedepth(partyctree@tree)
partyctree <- party::ctree(y ~ ., data=cbind(y=y,x))
treedepth(partykit::ctree(y ~ ., data=cbind(y=y,x)))
```

---

treetopdf	<i>Ctree to pdf graph</i>
-----------	---------------------------

---

**Description**

this function takes the output of the partykit::ctree function and prints the tree into a pdf file, located in the specified folder. It computes the depth and width and tries to create a pdf with the right dimensions.

**Usage**

```
treetopdf(partykitctree, savepath)
```

**Arguments**

partykitctree:	an output of partykit::ctree
savepath:	a file path where to store the pdf of the plot

Examples

```
y=iris$Species;x=iris[,-5]
partykitctree <- partykit::ctree(y ~ ., data=cbind(y=y,x))
treetopdf(partykitctree,"./x.pdf")
```

---

Tsampledata	<i>Transposed sample data.</i>
-------------	--------------------------------

---

Description

Transposed sample data.

Usage

```
Tsampledata(transposingvariables = TRUE)
```

Arguments

transposingvariables  
a boolean. If TRUE, stransposing id variables are created.

Details

Tsampledata(x) is Generaltransposefunction(Tsampledata(x))

Value

a data frame with id variables, numeric, factor and character variables.

---

TTsampledata	<i>Transposed sample data.</i>
--------------	--------------------------------

---

Description

Transposed sample data.

Usage

```
TTsampledata(transposingvariables = TRUE)
```

Arguments

transposingvariables  
a boolean. If TRUE, stransposing id variables are created.

**Details**

Tsampledata(x) is Generaltransposefunction(Tsampledata(x))

**Value**

a data frame with id variables, numeric, factor and character variables.

---

%notin%	‘ ... %notin% NA negation de ‘ 1%notin%2:3 1%notin%1:3
---------	--

---

# Index

`%notin%`, 45

`augmentT_f`, 3

`BigSyn`, 6

`compilefits`, 6

`compilesamplereports`, 7

`daniRules`, 8

`drop_last`, 9

`fitmodel.ctree`, 9

`fitmodel.fn`, 10

`fitmodel.rf`, 11

`fitthemodel`, 11

`GeneralReversetransposefunction`, 12

`GeneralReversetransposefunctiondecoupe`, 13

`Generaltransposefunction`, 14

`Generaltransposefunctionsimple`, 14

`get_cell`, 17

`get_cellrn`, 17

`get_cellXXgroup`, 18

`get_cellXXmarginscount`, 19

`get_cellXXsplit`, 20

`get_missingind`, 21

`get_natural.predictors`, 21

`get_origin`, 22

`get_parent`, 23

`get_presentind`, 24

`get_var`, 25

`get_XXpredecessoratmargin`, 25

`get_XXpredecessorsatmargin`, 26

`getnodesfromrules`, 15

`getpredictorsfromcaptureoutput`, 16

`getpredictorsfromtree`, 16

`good.fit.parameters`, 27

`good.sample.parameters`, 28

`good.syn.parameters`, 28

`NAto0`, 29

`onlygoodargs`, 30

`posixcttonumeric`, 30

`predictor.matrix.default`, 31

`predictor.matrix.rate`, 32

`preparepredictorsfortreefit`, 32

`reduceT_f`, 33

`resampleT_f`, 34

`runCompare`, 34

`sample.ctree`, 35

`sample.fn`, 35

`sampladata`, 36

`samplefrompool`, 37

`SDPSYN2`, 37

`sorttablewithingroup`, 39

`Sparameters.default.f`, 40

`Sparameters.variables.reorder.default`, 41

`treedepth`, 43

`treetopdf`, 43

`Tsampladata`, 44

`TTsampladata`, 44