

# Package ‘BigSyn’

January 16, 2020

**Type** Package

**Title** X

**Version** 1.0

**Date** 2020-01-16

**Author** D. Bonnery

**Maintainer** D. Bonnery <dbonnery@umd.edu>

**Description** Data

**Remotes** tidyverse/magrittr

**Depends** devtools,

RODBC,

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:

BigSyn . . . . .	3
compilefits . . . . .	3
compilesamplereports . . . . .	4
daniRules . . . . .	4

drop_last	5
fitmodel.ctree	6
fitmodel.fn	6
fitmodel.rf	7
fitthmodel	8
GeneralReversetransposefunction	9
GeneralReversetransposefunctiondecoupe	9
Generaltransposefunction	10
Generaltransposefunctionsimple	11
getnodesfromrules	12
getpredictorsfromcaptureoutput	12
getpredictorsfromtree	13
get_cell	13
get_cellrn	14
get_cellXXgroup	15
get_cellXXmargincount	16
get_cellXXsplit	16
get_missingind	17
get_natural.predictors	18
get_origin	19
get_parent	19
get_presentind	20
get_var	21
get_XXpredecessoratmargin	21
get_XXpredecessorsatmargin	22
good.fit.parameters	23
good.sample.parameters	24
good.syn.parameters	24
NAt0	25
onlygoodargs	26
posixcttonumeric	26
predictor.matrix.default	27
predictor.matrix.rate	28
preparepredictorsfortreefit	28
runCompare	29
sample.ctree	30
sample.fn	30
sampladata	31
samplefrompool	31
SDPSYN2	32
sorttablewithingroup	33
Sparameters.default.f	34
Sparameters.variables.reorder.default	35
treedepth	36
treetopdf	37
Tsampladata	37
TTsampladata	38
%notin%	38

---

BigSyn	<i>BigSyn: Some non confidential functions developped for the MLDSCR synthetic data project</i>
--------	---

---

**Description**

The BigSyn package allows to synthesize big hierarchical databases by opposition to just a single small "rectangular" table.

**BigSyn functions**

The BigSyn functions are

---

compilefits	<i>compilefits</i>
-------------	--------------------

---

**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 output of partykit::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

**Examples**

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

---

compilesamplereports	<i>compilesamplereports</i>
----------------------	-----------------------------

---

### 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	<i>daniRules</i>
-----------	------------------

---

### 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("iojoij")
drop_last("aa.iojoij")
```

---

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	<i>Fit a model with a specific function</i>
-------------	---

---

### 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,
  TtableANato0,
  redocomputationsevenifexists = 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

```
data(TtableA, package="BigSyn")
Sparameters<-Sparameters.default.f(ref.table=TtableA)
Sparameters_i<-Sparameters[[53]]; fitmodelsavepath=NULL; TtableANato0<-Nato0(TtableA);redocomputationsevenifexists=FALSE;
treeplotsavefolder=tempdir()
fitthemodel(Sparameters_i, fitmodelsavepath = NULL, TtableANato0 = TtableANato0, treeplotsavefolder=tempdir())
Sparameters_i<-Sparameters[["AA.present_La_La_Lrn1"]];
treeplotsavefolder=tempdir()
fitthemodel(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 orioginal 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

*Simple General Transpose function*

---

**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	<i>getpredictorsfromcaptureoutput</i>
--------------------------------	---------------------------------------

---

**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  get cell without the row number
```

---

**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

*Get cell and row number*


---

**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.xoiij",marginpos=2 returns "1"; if x is "a\_1\_f\_2\_aa.xoiij",marginpos=-2 returns "a\_f\_2"; if x is "a\_1\_f\_2\_aa.xoiij",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
```

*split a cell*

---

### 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
```

*get variable predecessors at margin*

---

**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.xoijj" returns c("a","1","f","1")

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

**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_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)
```

---

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

*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**

```
NAto0(tableA)
```

**Arguments**

tableA            a dataframe

**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)
```

---

runCompare

*runCompare*


---

**Description**

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

**Usage**

```
runCompare(
  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**

listofpackage1 a vector of character strings

listofpackage2 a vector of character strings

package1 a character string

package2 a character string

**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

*Function to sample from a set of partitioning rules***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

*General SDP function.***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
)
```



## 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)$defaultsynparameters),names(synparameters))])
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,treeplotsavefolder = treeplotsavefolder)
todisplay<-grep("La_La_Lrn1",names(STtableA[[1]]),value=T);
STtableA[[1]][1:3,todisplay];TtableA[1:3,todisplay]
```

---

sorttablewithingroup    *sort table within group without changing the group position*

---

## 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,sortv

```

---

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" or "ctree"
defaultparameters	a list indicating default parameters for synthpop synthesis functions, for example ntree=5, smoothing="none"

## Details

creates default synthetisation parameters

## 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

**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

---

*Compute depth of a "party" tree*


---

**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

*Ctree to pdf graph*


---

### 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")
```

---

Tsampladata

*Transposed sample data.*


---

### Description

Transposed sample data.

### Usage

```
Tsampladata(transposingvariables = TRUE)
```

### Arguments

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

### Details

Tsampladata(x) is Generaltransposefunction(Tsampladata(x))

### Value

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

TTsampledata	Transposed sample data.
<b>Description</b>	
Transposed sample data.	
<b>Usage</b>	
TTsampledata(transposingvariables = TRUE)	
<b>Arguments</b>	
transposingvariables	a boolean. If TRUE, stransposing id variables are created.
<b>Details</b>	
Tsampledata(x) is Generaltransposefunction(Tsampledata(x))	
<b>Value</b>	
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%](#), [38](#)

[BigSyn](#), [3](#)

[compilefits](#), [3](#)

[compilesamplerereports](#), [4](#)

[daniRules](#), [4](#)

[drop\\_last](#), [5](#)

[fitmodel.ctree](#), [6](#)

[fitmodel.fn](#), [6](#)

[fitmodel.rf](#), [7](#)

[fitthemodel](#), [8](#)

[GeneralReversetransposefunction](#), [9](#)

[GeneralReversetransposefunctiondecoupe](#),  
[9](#)

[Generaltransposefunction](#), [10](#)

[Generaltransposefunctionsimple](#), [11](#)

[get\\_cell](#), [13](#)

[get\\_cellrn](#), [14](#)

[get\\_cellXXgroup](#), [15](#)

[get\\_cellXXmargincount](#), [16](#)

[get\\_cellXXsplit](#), [16](#)

[get\\_missingind](#), [17](#)

[get\\_natural.predictors](#), [18](#)

[get\\_origin](#), [19](#)

[get\\_parent](#), [19](#)

[get\\_presentind](#), [20](#)

[get\\_var](#), [21](#)

[get\\_XXpredecessorathmargin](#), [21](#)

[get\\_XXpredecessorsathmargin](#), [22](#)

[getnodesfromrules](#), [12](#)

[getpredictorsfromcaptureoutput](#), [12](#)

[getpredictorsfromtree](#), [13](#)

[good.fit.parameters](#), [23](#)

[good.sample.parameters](#), [24](#)

[good.syn.parameters](#), [24](#)

[Nato0](#), [25](#)

[onlygoodargs](#), [26](#)

[posixcttonumeric](#), [26](#)

[predictor.matrix.default](#), [27](#)

[predictor.matrix.rate](#), [28](#)

[preparepredictorsfortreefit](#), [28](#)

[runCompare](#), [29](#)

[sample.ctree](#), [30](#)

[sample.fn](#), [30](#)

[sampledata](#), [31](#)

[samplefrompool](#), [31](#)

[SDPSYN2](#), [32](#)

[sorttablewithingroup](#), [33](#)

[Sparameters.default.f](#), [34](#)

[Sparameters.variables.reorder.default](#),  
[35](#)

[treedepth](#), [36](#)

[treetopdf](#), [37](#)

[Tsamledata](#), [37](#)

[TTsamledata](#), [38](#)