Package: dsBase (via r-universe)

May 18, 2026

Title 'DataSHIELD' Server Side Base Functions

Description Base 'DataSHIELD' functions for the server side. 'DataSHIELD' is a software package which allows you to do non-disclosive federated analysis on sensitive data. 'DataSHIELD' analytic functions have been designed to only share non disclosive summary statistics, with built in automated output checking based on statistical disclosure control. With data sites setting the threshold values for the automated output checks. For more details, see 'citation(``dsBase'')'.

Version 6.3.6.9000

License GPL-3

Depends R (>= 4.0.0)

Imports RANN, stringr, lme4, dplyr, tibble, purrr, tidyselect, reshape2, polycor (>= 0.8), splines, gamlss, gamlss.dist, mice, childsds

Suggests spelling, testthat

RoxygenNote 8.0.0

Encoding UTF-8

Language en-GB

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev libx11-dev zlib1g-dev

Repository https://stuartwheater.r-universe.dev

Date/Publication 2026-05-14 15:22:11 UTC

RemoteUrl https://github.com/stuartwheater/dsbase

RemoteRef HEAD

RemoteSha 69a7c1263ea5718c4ec9c2e4c97b28d271a73c0e

Contents

absDS *Computes the absolute values of the input variable*

Description

This function is similar to R function abs.

Usage

```
absDS(x)
```

Arguments

x	a string character, the name of a numeric or integer vector

Details

The function computes the absolute values of an input numeric or integer vector.

Value

the object specified by the newobj argument of ds.abs (or default name abs.newobj) which is written to the serverside. The output object is of class numeric or integer.

Author(s)

Demetris Avraam for DataSHIELD Development Team

asCharacterDS *Coerces an R object into class character*

Description

this function is based on the native R function as.character

Usage

```
asCharacterDS(x.name)
```

Arguments

x.name	the name of the input object to be coerced to class character. Must be specified in inverted commas. But this argument is usually specified directly by x.name argument of the clientside function ds.asCharacter

Details

See help for function as.character in native R

Value

the object specified by the newobj argument (or its default name "ascharacter.newobj") which is written to the serverside. For further details see help on the clientside function ds.asCharacter

Author(s)

Amadou Gaye, Paul Burton, Demetris Avraam for DataSHIELD Development Team

---

asDataMatrixDS *asDataMatrixDS a serverside assign function called by ds.asDataMatrix*

Description

Coerces an R object into a matrix maintaining original class for all columns in data.frames.

Usage

```
asDataMatrixDS(x.name)
```

Arguments

x.name	the name of the input object to be coerced to class data.matrix. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function ds.asDataMatrix

Details

This assign function is based on the native R function data.matrix If applied to a data.frame, the native R function as.matrix converts all columns into character class. In contrast, if applied to a data.frame the native R function data.matrix converts the data.frame to a matrix but maintains all data columns in their original class

Value

the object specified by the <newobj> argument (or its default name "asdatamatrix.newobj") which is written to the serverside. For further details see help on the clientside function ds.asDataMatrix

Author(s)

Paul Burton for DataSHIELD Development Team

asFactorDS1 *Determines the levels of the input variable in each single study*

Description

This function is an aggregate DataSHIELD function that returns the levels of the input variable from each single study to the client-side function.

Usage

```
asFactorDS1(input.var.name = NULL)
```

Arguments

`input.var.name` the name of the variable that is to be converted to a factor.

Details

The function encodes the input vector as factor and returns its levels in ascending order if the levels are numerical or in alphabetical order if the levels are of type character.

Value

the levels of the input variable.

asFactorDS2 *Converts a numeric vector into a factor*

Description

This function is an assign DataSHIELD function that converts a numeric vector into a factor type that presented as a vector or as a matrix with dummy variables.

Usage

```
asFactorDS2(
  input.var.name = NULL,
  all.unique.levels.transmit = NULL,
  fixed.dummy.vars = NULL,
  baseline.level = NULL
)
```

Arguments

`input.var.name`	the name of the variable that is to be converted to a factor.
`all.unique.levels.transmit`	the levels that the variable will be transmitted to.
`fixed.dummy.vars`	a boolean that determines whether the new object will be represented as a vector or as a matrix of dummy variables indicating the factor level of each data point. If this argument is set to FALSE (default) then the input variable is converted to a factor and assigned as a vector. If is set to TRUE then the input variable is converted to a factor but assigned as a matrix of dummy variables.
`baseline.level`	a number indicating the baseline level to be used in the creation of the matrix of dummy variables.

Details

The functions converts the input variable into a factor which is presented as a vector if the `fixed.dummy.vars` is set to FALSE or as a matrix with dummy variables if the `fixed.dummy.vars` is set to TRUE (see the help file of ds.asFactor.b for more details).

Value

an object of class factor

---

asFactorSimpleDS *Converts a numeric vector into a factor*

Description

This function is an assign DataSHIELD function that coerces a numeric or character vector into a factor

Usage

```
asFactorSimpleDS(input.var.name = NULL)
```

Arguments

`input.var.name`	the name of the variable that is to be converted to a factor.

Details

The functions converts the input variable into a factor. Unlike ds.asFactor and its serverside functions, ds.asFactorSimple does no more than coerce the class of a variable to factor in each study. It does not check for or enforce consistency of factor levels across sources or allow you to force an arbitrary set of levels unless those levels actually exist in the sources. In addition, it does not allow you to create an array of binary dummy variables that is equivalent to a factor. If you need to do any of these things you will have to use the ds.asFactor function.

Value

an object of class factor

`asIntegerDS` *Coerces an R object into class integer*

Description

This function is based on the native R function `as.integer`.

Usage

```
asIntegerDS(x.name)
```

Arguments

`x.name`	the name of the input object to be coerced to class integer. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function `ds.asInteger`.

Details

See help for function `as.integer` in native R, and details section in the help file of the clientside function `ds.asInteger`.

Value

the object specified by the <newobj> argument (or its default name "asinteger.newobj") which is written to the serverside. For further details see help on the clientside function `ds.asInteger`.

Author(s)

Amadou Gaye, Paul Burton, Demetris Avraam, for DataSHIELD Development Team

---

`asListDS` *asListDS a serverside aggregate function called by ds.asList*

Description

Coerces an R object into a list

Usage

```
asListDS(x.name, newobj)
```

Arguments

x.name	the name of the input object to be coerced to class data.matrix. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function ds.asList
newobj	is the object hard assigned '«-' to be the output of the function written to the serverside

Details

Unlike most other class coercing functions this is an aggregate function rather than an assign function. This is because the datashield.assign function in the data repository deals specially with a created object (newobj) if it is of class list. Reconfiguring the function as an aggregate function works around this problem. This aggregate function is based on the native R function as.list and so additional information can be found in the help for as.list

Value

the object specified by the <newobj> argument (or its default name <x.name>.mat) which is written to the serverside. In addition, two validity messages are returned. The first confirms an output object has been created, the second states its class. The way that as.list coerces objects to list depends on the class of the object, but in general the class of the output object should usually be 'list'

Author(s)

Amadou Gaye, Paul Burton for DataSHIELD Development Team

---

asLogicalDS *Coerces an R object into class numeric*

Description

this function is based on the native R function as.numeric

Usage

```
asLogicalDS(x.name)
```

Arguments

x.name	the name of the input object to be coerced to class numeric. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function ds.aslogical

Details

See help for function as.logical in native R

Value

the object specified by the <newobj> argument (or its default name <x.name>.logic) which is written to the serverside. For further details see help on the clientside function `ds.asLogical`

Author(s)

Amadou Gaye, Paul Burton for DataSHIELD Development Team

---

asMatrixDS *Coerces an R object into a matrix*

Description

this function is based on the native R function `as.matrix`

Usage

```
asMatrixDS(x.name)
```

Arguments

`x.name`	the name of the input object to be coerced to class matrix. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function `ds.asMatrix`

Details

See help for function `as.matrix` in native R

Value

the object specified by the <newobj> argument (or its default name <x.name>.mat) which is written to the serverside. For further details see help on the clientside function `ds.asMatrix`

Author(s)

Amadou Gaye, Paul Burton for DataSHIELD Development Team

asNumericDS *Coerces an R object into class numeric*

Description

This function is based on the native R function `as.numeric`.

Usage

```
asNumericDS(x.name)
```

Arguments

`x.name`	the name of the input object to be coerced to class numeric. Must be specified in inverted commas. But this argument is usually specified directly by <x.name> argument of the clientside function `ds.asNumeric`.

Details

See help for function `as.numeric` in native R, and details section in the help file of the clientside function `ds.asNumeric`.

Value

the object specified by the <newobj> argument (or its default name <x.name>.num) which is written to the serverside. For further details see help on the clientside function `ds.asNumeric`.

Author(s)

Amadou Gaye, Paul Burton, Demetris Avraam, for DataSHIELD Development Team

---

aucDS *aucDS an aggregate function called by ds.auc*

Description

This function calculates the C-statistic or AUC for logistic regression models.

Usage

```
aucDS(pred = pred, y = y)
```

Arguments

`pred`	the name of the vector of the predicted values
`y`	the name of the outcome variable. Note that this variable should include the complete cases that are used in the regression model.

Details

The AUC determines the discriminative ability of a model.

Value

returns the AUC and its standard error

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

blackBoxDS *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

The first key serverside function that sets up the V2BR for ranking in the client.

Usage

```
blackBoxDS(input.var.name = NULL, shared.seedval, synth.real.ratio, NA.manage)
```

Arguments

`input.var.name`	a character string specifying the name of V2BR. This argument is set by the argument with the same name in the clientside function ds.ranksSecure
`shared.seedval`	a pseudorandom number seed that ensures that the processes generating the order and parameterisation of the encryption algorithms are the same in each study. This argument is set by the argument <shared.seed.value> in the clientside function ds.ranksSecure. For more details, including future plans to share this starting seed in a more secure way, please see the associated document entitled "secure.global.ranking.docx" and the header file for ds.ranksSecure.
`synth.real.ratio`	an integer value representing the ratio of synthetic (pseudo-data) values to the real number of values in V2BR. This argument is set by the argument with the same name in the clientside function ds.ranksSecure. For more details, please see the associated document entitled "secure.global.ranking.docx" and the header file for ds.ranksSecure.
`NA.manage`	character string indicating how missing values (NAs) in V2BR should be managed. It takes three possible values: "NA.delete", "NA.low","NA.hi". This argument is set by the argument with the same name in the clientside function ds.ranksSecure. For more details, please see the associated document entitled "secure.global.ranking.docx" and the header file for ds.ranksSecure.

Details

Severside assign function called by ds.ranksSecure. Creates pseudo-data by using the real distribution of values in V2BR to create a large number of synthetic data with a similar distribution to the values in V2BR but with a slightly broader distribution at both ends to ensure that any extreme values in the "combined real+pseudo data vector" are all pseudo-data. Also ensures that the number of decimal places of the values in the V2BR is reflected by the number of decimal places in the pseudodata. Finally, takes the "combined real+pseudo data vector" through seven rounds of rank consistent encryption that involves algorithms themselves generated by a pseudorandom process that selects which transformation to apply and with what parameters. The encryption algorithms are the same in each study ensuring that ranks also remain consistent between studies. After encryption the encrypted "combined real+pseudo data vector" is written to the serverside as a dataframe also including other key component vectors from the first stage of the ranking procedure. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Value

writes a data frame object entitled blackbox.output.df to the serverside. In each study this contains the encrypted "combined real+pseudo data vector" and a range of other key components from the first stage of the ranking procedure. For more details see the associated document entitled "secure.global.ranking.docx"

Author(s)

Paul Burton 9th November, 2021

`blackBoxRanksDS` *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

The second key serverside function that prepares the global ranks of the the real data only generated in the first stage of the ranking procedure and encrypts them in preparation for generating global ranks that correspond 1 to 1 with only the real data in V2BR.

Usage

```
blackBoxRanksDS(input.var.name = NULL, shared.seedval)
```

Arguments

`input.var.name`	a character string specifying the name of the vector holding the global ranks. This argument is set automatically by the clientside function ds.ranksSecure

`shared.seedval` a pseudorandom number seed that ensures that the processes generating the order and parameterisation of the encryption algorithms are the same in each study. This argument is set by the argument <shared.seed.value> in the clientside function ds.ranksSecure. The seed value shared by all studies in setting up the encryption procedures in blackBoxRanksDS is arbitrarily changed from that used to set up the encryption procedures in blackBoxDS, so the the set of 7 encryption algorithms is deliberately different. For more details, including future plans to share this starting seed in a more secure way, please see the associated document entitled "secure.global.ranking.docx" and the header file for ds.ranksSecure.

Details

Severside assign function called by ds.ranksSecure. It takes the global ranks currently held in sR5.df which reflect the global ranks based on the "combined real+pseudo data vector" as encrypted by blackBoxDS but with all pseudo-data stripped out. It then uses these global ranks (of the real data) as if they were a new variable to be ranked. This is then equivalent to blackBoxDS with the primary difference that no pseudo-data are needed. This is because the global ranks are fundamentally non-disclosive and so can be transferred to the clientside with no risk of disclosure. However, in order to ensure that the client cannot compare the list of global.ranks in sR4.df (after initial global ranking based on ranking of real and pseudo-data combined) with the global.ranks to be generated by blackBoxRanksDS (based solely on the real data they are processed through seven more rounds of encryption as before in blackBoxDS. In consequence the client remains unable to determine which of the original global ranks corresponded to real data and which to pseudo-data. In addition, blackBoxRanksDS does not need to determine the number of decimal places in the data because it is only applied to ranks which are assumed to be integers. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure and the header file for blackBoxDS

Value

writes a data frame object entitled blackbox.ranks.df to the serverside. In each study this contains the encrypted global ranks and a range of other key components from the second stage (ranking of global ranks for real observations only) of the ranking procedure. For more details see the associated document entitled "secure.global.ranking.docx"

Author(s)

Paul Burton 9th November, 2021

`BooleDS` *BooleDS*

Description

Converts the individual elements of a vector or other object into Boolean indicators.

Usage

```
BooleDS(
  V1.name = NULL,
  V2.name = NULL,
  Boolean.operator.n = NULL,
  na.assign.text,
  numeric.output = TRUE
)
```

Arguments

V1.name	A character string specifying the name of the vector to which the Boolean operator is to be applied
V2.name	A character string specifying the name of the vector or scalar to which <V1> is to be compared.
Boolean.operator.n	An integer value (1 to 6) providing a numeric coding for the character string specifying one of six possible Boolean operators: '==', '!=', '>', '>=','<', '<=' that could legally be passed from client to server via DataSHIELD parser
na.assign.text	A character string taking values 'NA', '1' or '0'. If 'NA' then any NA values in the input vector remain as NAs in the output vector. If '1' or '0' NA values in the input vector are all converted to 1 or 0 respectively.
numeric.output	a TRUE/FALSE indicator defaulting to TRUE determining whether the final output variable should be of class numeric (1/0) or class logical (TRUE/FALSE).

Details

The function converts the input vector into Boolean indicators.

Value

the levels of the input variable.

Author(s)

DataSHIELD Development Team

---

boxPlotGG_data_Treatment_numericDS

Arrange vector to pass it to the boxplot function

Description

Arrange vector to pass it to the boxplot function

Usage

```
boxPlotGG_data_Treatment_numericDS(vector)
```

Arguments

vector	numeric vector Vector to arrange to be plotted later

Value

data frame with the following structure:

Column 'x': Names on the X axis of the boxplot, aka name of the vector (vector argument)
Column 'value': Values for that variable

---

boxPlotGG_data_TreatmentDS

Arrange data frame to pass it to the boxplot function

Description

Arrange data frame to pass it to the boxplot function

Usage

```
boxPlotGG_data_TreatmentDS(table, variables, group = NULL, group2 = NULL)
```

Arguments

table	data frame Table that holds the information to be plotted later
variables	character vector Name of the column(s) of the data frame to include on the boxplot
group	character (default NULL) Name of the first grouping variable.
group2	character (default NULL) Name of the second grouping variable.

Value

data frame with the following structure:

Column 'x': Names on the X axis of the boxplot, aka variables to plot
Column 'value': Values for that variable (raw data of columns rbinded)
Column 'group': (Optional) Values of the grouping variable
Column 'group2': (Optional) Values of the second grouping variable

boxPlotGGDS *Create the identity stats and necessary data to draw a plot on the client*

Description

In order to create a non disclosive box plot, the data that is passed to the client is purely geometrical aspects of the plot, as a ggplot object contains all the data inside, only the graphical parameters are passed. There are three different cases depending if there are grouping variables. The outliers are also removed from the graphical parameters.

Usage

```
boxPlotGGDS(data_table, group = NULL, group2 = NULL)
```

Arguments

data_table	data frame Table that holds the information to be plotted, arranged as: Column 'x': Names on the X axis of the boxplot, aka variables to plot Column 'value': Values for that variable (raw data of columns rbinded) Column 'group': (Optional) Values of the grouping variable Column 'group2': (Optional) Values of the second grouping variable
group	character (default NULL) Name of the first grouping variable.
group2	character (default NULL) Name of the second grouping variable.

Value

list with:
-data frame Geometrical parameters (identity stats of ggplot)
-character Type of plot (single_group, double_group or no_group)

bp_standardsDS *Calculates Blood pressure z-scores*

Description

The function calculates blood pressure z-scores in two steps: Step 1. Calculates z-score of height according to CDC growth chart (Not the WHO growth chart!). Step 2. Calculates z-score of BP according to the fourth report on BP management, USA

Usage

```
bp_standardsDS(
  sex = sex,
  age = age,
  height = height,
  bp = bp,
  systolic = systolic
)
```

Arguments

sex	the name of the sex variable. The variable should be coded as 1 for males and 2 for females. If it is coded differently (e.g. 0/1), then you can use the ds.recodeValues function to recode the categories to 1/2 before the use of ds.bp_standards
age	the name of the age variable in years.
height	the name of the height variable in cm
bp	the name of the blood pressure variable.
systolic	logical. If TRUE (default) the function assumes conversion of systolic blood pressure. If FALSE the function assumes conversion of diastolic blood pressure.

Value

assigns a new object on the server-side. The assigned object is a list with two elements: the 'Zbp' which is the zscores of the blood pressure and 'perc' which is the percentiles of the BP zscores.

Note

The z-scores of height based on CDC growth charts are calculated by the sds function from the childsds R package.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

cbindDS *cbindDS called by ds.cbind*

Description

serverside assign function that takes a sequence of vector, matrix or data-frame arguments and combines them by column to produce a data-frame.

Usage

```
cbindDS(x.names.transmit = NULL, colnames.transmit = NULL)
```

Arguments

`x.names.transmit`
: This is a vector of character strings representing the names of the elemental components to be combined converted into a transmittable format. This argument is fully specified by the `x` argument of the client-side `ds.cbind` function.

`colnames.transmit`
: This is a vector of character strings representing column names for the output object converted into a transmittable format.

Details

A sequence of vector, matrix or data-frame arguments is combined column by column to produce a data-frame which is written to the serverside. A critical requirement is that the length of all component variables, and the number of rows of the component data.frames or matrices must all be the same. The output data.frame will then have this same number of rows. For more details see help for `ds.cbind` and the native R function `cbind`.

Value

the object specified by the `newobj` argument of `ds.cbind` (or default name `cbind.newobj`) which is written to the serverside. The output object is of class data.frame.

Author(s)

Paul Burton and Demetris Avraam for DataSHIELD Development Team

---

cDS *Concatenates objects into a vector or list*

Description

This function is similar to the R base function 'c'.

Usage

```
cDS(objs)
```

Arguments

`objs` a list which contains the the objects to concatenate.

Details

Unlike the R base function 'c' on vector or list of certain length are allowed as output

Value

a vector or list

Author(s)

Gaye, A.

`changeRefGroupDS` *Changes a reference level of a factor*

Description

This function is similar to R function `relevel`,

Usage

```
changeRefGroupDS(xvect, ref = NULL, reorderByRef = NULL)
```

Arguments

`xvect`	a factor vector
`ref`	a character, the reference level
`reorderByRef`	a boolean that tells whether or not the new vector should be ordered by the reference group.

Details

In addition to what the R function does, this function allows for the user to re-order the vector, putting the reference group first. If the user chooses the re-order a warning is issued as this can introduce a mismatch of values if the vector is put back into a table that is not reordered in the same way. Such mismatch can render the results of operations on that table invalid.

Value

a factor of the same length as xvect

Author(s)

Isaeva, J., Gaye, A.

checkNegValueDS *Checks if a numeric variable has negative values*

Description

this function is only called by the client function `ds.glm`.

Usage

```
checkNegValueDS(weights)
```

Arguments

`weights`	a numeric vector

Details

if a user sets the parameter 'weights' on the client side function `ds.glm` this server side function is called to verify that the 'weights' vector does not have negative values because no negative are allowed in weights.

Value

a boolean; TRUE if the vector has one or more negative values and FALSE otherwise

Author(s)

Gaye, A.

checkPermissivePrivacyControlLevel *checkPermissivePrivacyControlLevel*

Description

This server-side function check that the server is running in "permissive" privacy control level.

Usage

```
checkPermissivePrivacyControlLevel(privacyControlLevels)
```

Arguments

`privacyControlLevels`	is a vector of strings which contains the privacy control level names which are permitted by the calling method.

Details

Tests whether the R option "datashield.privacyControlLevel" is set to "permissive", if it isn't will cause a call to stop() with the message "BLOCKED: The server is running in 'non-permissive' mode which has caused this method to be blocked".

Value

No return value, called for side effects

Author(s)

Wheater, Dr SM., DataSHIELD Development Team.

---

classDS *Returns the class of an object*

Description

This function is similar to R function `class`.

Usage

```
classDS(x)
```

Arguments

x	a string character, the name of an object

Details

The function returns the class of an object

Value

the class of the input object

Author(s)

Stuart Wheater, for DataSHIELD Development Team

colnamesDS *Returns the column names of a data frame or matrix*

Description

This function is similar to R function `colnames`.

Usage

```
colnamesDS(x)
```

Arguments

`x`	a string character, the name of a dataframe or matrix

Details

The function returns the column names of the input dataframe or matrix

Value

the column names of the input object

Author(s)

Demetris Avraam, for DataSHIELD Development Team

completeCasesDS *completeCasesDS: an assign function called by ds.completeCases*

Description

Identifies and strips out all rows of a data.frame, matrix or vector that contain NAs.

Usage

```
completeCasesDS(x1.transmit)
```

Arguments

`x1.transmit`	This argument determines the input data.frame, matrix or vector from which rows with NAs are to be stripped. The <x1.transmit> argument is fully specified by the <x1> argument of the `ds.completeCases` function.

Details

In the case of a data.frame or matrix, completeCasesDS identifies all rows containing one or more NAs and deletes those rows altogether. Any one variable with NA in a given row will lead to deletion of the whole row. In the case of a vector, completeCasesDS acts in an equivalent manner but there is no equivalent to a 'row' and so it simply strips out all observations recorded as NA. ds.completeCASES is analogous to the complete.cases function in native R. Limited additional information can therefore be found under help("complete.cases") in native R.

Value

a modified data.frame, matrix or vector from which all rows containing at least one NA have been deleted. This modified object is written to the serverside in each source. In addition, two validity messages are returned indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - ds.completeCases also returns any studysideMessages that can help explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the ds.message function. If you type ds.message("newobj") it will print out the relevant studysideMessage from any datasource in which there was an error in creating <newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message("newobj") will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

Paul Burton for DataSHIELD Development Team

---

corDS *Computes the sum of each variable and the sum of products for each pair of variables*

Description

This function computes the sum of each vector of variable and the sum of the products of each two variables (i.e. the scalar product of each two vectors).

Usage

```
corDS(x = NULL, y = NULL)
```

Arguments

x	a character, the name of a vector, matrix or dataframe of variables(s) for which the correlation(s) is (are) going to calculated for.
y	NULL (default) or the name of a vector, matrix or dataframe with compatible dimensions to x.

Details

computes the sum of each vector of variable and the sum of the products of each two variables

Value

a list that includes a matrix with elements the sum of products between each two variables, a matrix with elements the sum of the values of each variable, a matrix with elements the number of complete cases in each pair of variables, a list with the number of missing values in each variable separately (columnwise) and the number of missing values casewise, and a vector with elements the sum of squares of each variable. The first disclosure control checks that the number of variables is not bigger than a percentage of the individual-level records (the allowed percentage is pre-specified by the 'nfilter.glm'). The second disclosure control checks that none of them is dichotomous with a level having fewer counts than the pre-specified 'nfilter.tab' threshold.

Author(s)

Paul Burton, and Demetris Avraam for DataSHIELD Development Team

---

corTestDS *Tests for correlation between paired samples*

Description

This function is similar to R function `cor.test`.

Usage

```
corTestDS(x, y, method, exact, conf.level)
```

Arguments

`x`	a character string providing the name of a numerical vector.
`y`	a character string providing the name of a numerical vector.
`method`	a character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated.
`exact`	a logical indicating whether an exact p-value should be computed. Used for Kendall's tau and Spearman's rho.
`conf.level`	confidence level for the returned confidence interval. Currently only used for the Pearson product moment correlation coefficient if there are at least 4 complete pairs of observations.

Details

The function runs a two-sided correlation test

Value

the results of the correlation test.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

`covDS` *Computes the sum of each variable and the sum of products for each pair of variables*

Description

This function computes the sum of each vector of variable and the sum of the products of each two variables (i.e. the scalar product of each two vectors).

Usage

```
covDS(x = NULL, y = NULL, use = NULL)
```

Arguments

`x`	a character, the name of a vector, matrix or dataframe of variable(s) for which the covariance(s) and the correlation(s) is (are) going to calculated for.
`y`	NULL (default) or the name of a vector, matrix or dataframe with compatible dimensions to x.
`use`	a character string giving a method for computing covariances in the presence of missing values. This must be one of the strings "casewise.complete" or "pairwise.complete". If `use` is set to 'casewise.complete' then any rows with missing values are omitted from the vector, matrix or dataframe before the calculations of the sums. If `use` is set to 'pairwise.complete' (which is the default case set on the client-side), then the sums of products are computed for each two variables using only the complete pairs of observations on the two variables.

Details

computes the sum of each vector of variable and the sum of the products of each two variables

Value

a list that includes a matrix with elements the sum of products between each two variables, a matrix with elements the sum of the values of each variable, a matrix with elements the number of complete cases in each pair of variables, a list with the number of missing values in each variable separately (columnwise) and the number of missing values casewise or pairwise depending on the argument `use`, and an error message which indicates whether or not the input variables pass the disclosure controls. The first disclosure control checks that the number of variables is not bigger than a percentage of the individual-level records (the allowed percentage is pre-specified by the

'nfilter.glm'). The second disclosure control checks that none of them is dichotomous with a level having fewer counts than the pre-specified 'nfilter.tab' threshold. If any of the input variables do not pass the disclosure controls then all the output values are replaced with NAs.

Author(s)

Amadou Gaye, Paul Burton, and Demetris Avraam for DataSHIELD Development Team

---

`dataFrameDS` *dataFrameDS called by ds.dataFrame*

Description

The serverside function that creates a data frame from its elemental components. That is: pre-existing data frames; single variables; and/or matrices

Usage

```
dataFrameDS(
  vectors = NULL,
  r.names = NULL,
  ch.rows = FALSE,
  ch.names = TRUE,
  clnames = NULL,
  strAsFactors = TRUE,
  completeCases = FALSE
)
```

Arguments

`vectors`	a list which contains the elemental components to combine. These correspond to the vector of character strings specified in argument x of the clientside function ds.dataFrame()
`r.names`	NULL or a character vector specifying the names of the rows. Default NULL.
`ch.rows`	logical, if TRUE then the rows are checked for consistency of length and names. Default FALSE.
`ch.names`	logical, if TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated. Default TRUE. In fact, the clientside function ensures no duplicated names can be presented to dataFrameDS but this argument is kept to check for other forms of syntactic validity.
`clnames`	a list of characters, the column names of the output data frame. These are generated by the clientside function from the names of vectors, and the column names of data.frames and matrices being combined in producing the output data.frame
`strAsFactors`	logical, if TRUE determines whether character vectors should automatically be converted to factors? Default TRUE.
`completeCases`	logical. If TRUE indicates that only complete cases should be included: any rows with missing values in any component will be excluded. Default FALSE.

Details

A data frame is a list of variables all with the same number of rows with unique row names, which is of class 'data.frame'. ds.dataFrame will create a data frame by combining a series of elemental components which may be pre-existing data.frames, matrices or variables. A critical requirement is that the length of all component variables, and the number of rows of the component data.frames or matrices must all be the same. The output data.frame will then have this same number of rows. The serverside function dataFrameDS() calls the native R function data.frame() and several of its arguments are precisely the same as for data.frame(). In consequence, additional information can be sought from the help() for data.frame().

Value

a dataframe composed of the specified elemental components will be created on the serverside and named according to the <newobj> argument of the clientside function ds.dataFrame()

Author(s)

DataSHIELD Development Team

`dataFrameFillDS` *dataFrameFillDS*

Description

An assign function called by the clientside ds.dataFrameFill function.

Usage

```
dataFrameFillDS(
  df.name,
  allNames.transmit,
  class.vect.transmit,
  levels.vec.transmit
)
```

Arguments

`df.name`	a character string representing the name of the input data frame that will be filled with extra columns with missing values if a number of variables is missing from it compared to the data frames of the other studies used in the analysis.
`allNames.transmit`	unique names of all the variables that are included in the input data frames from all the used datasources.
`class.vect.transmit`	the classes of all the variables that are included in the vector `allNames.transmit`.
`levels.vec.transmit`	the levels of all factor variables. The classes supported are 'numeric', 'integer', 'character', 'factor' and 'logical'.

Details

This function checks if each study has all the variables compared to the other studies in the analysis. If a study does not have some of the variables, the function generates those variables as vectors of missing values and combines them as columns to the input data frame. Then, the "complete" in terms of the columns dataframe is saved in each server with a name specified by the argument `newobj` on the clientside.

Value

Nothing is returned to the client. The generated object is written to the serverside.

Author(s)

Demetris Avraam for DataSHIELD Development Team

`dataFrameSortDS` *Sorting and reordering data frames, vectors or matrices*

Description

Sorts a data frame using a specified alphanumeric or numeric sort key

Usage

```
dataFrameSortDS(
  df.name = NULL,
  sort.key.name = NULL,
  sort.descending,
  sort.method
)
```

Arguments

`df.name`	a character string providing the name for the serverside data.frame to be sorted. This parameter is fully specified by the equivalent argument in ds.dataFrameShort and further details can be found at help("ds.dataFrameSort").
`sort.key.name`	a character string providing the name for the sort key. This will be a serverside vector which may sit inside the data frame to be sorted or independently in the serverside analysis environment. But, if it sits outside the data frame it must then be the same length as the data frame. This parameter is fully specified by the equivalent argument in ds.dataFrameShort and further details can be found at help("ds.dataFrameSort").
`sort.descending`	logical, if TRUE the data.frame will be sorted by the sort key in descending order. Default = FALSE (sort order ascending). This parameter is fully specified by the equivalent argument in ds.dataFrameShort and further details can be found at help("ds.dataFrameSort").

`sort.method` A character string taking one of the values: "default", "d", "alphabetic", "a", "numeric", "n", or NULL. Default value is "default". This parameter is fully specified by the equivalent argument in ds.dataFrameShort and further details can be found at help("ds.dataFrameSort").

Details

Serverside assign function dataFrameSortDS is called by clientside function ds.dataFrameSort. A vector or a matrix can be added to, or coerced into, a data frame (using function [ds.dataFrame]) and this means that they too can be sorted/reordered using ds.dataFrameSort. Fundamentally, the function [ds.dataFrameSort] will sort a specified data frame on the serverside using a sort key also on the serverside. For more details see help for the clientside function: [ds.dataFrameShort]

Value

the appropriately re-sorted data.frame will be written to the serverside R environment as a data.frame named according to the <newobj> argument(or with default name 'dataframesort.newobj') if no name is specified

Author(s)

Paul Burton, with critical error identification by Leire Abarrategui-Martinez, for DataSHIELD Development Team, 2/4/2020

---

dataFrameSubsetDS1 *dataFrameSubsetDS1 an aggregate function called by ds.dataFrameSubset*

Description

First serverside function for subsetting a data frame by row or by column.

Usage

```
dataFrameSubsetDS1(
  df.name = NULL,
  V1.name = NULL,
  V2.name = NULL,
  Boolean.operator.n = NULL,
  keep.cols = NULL,
  rm.cols = NULL,
  keep.NAs = NULL
)
```

Arguments

df.name	a character string providing the name for the data.frame to be sorted. <df.name> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
V1.name	A character string specifying the name of a subsetting vector to which a Boolean operator will be applied to define the subset to be created. <V1.name> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
V2.name	A character string specifying the name of the vector or scalar to which the values in the vector specified by the argument <V1.name> is to be compared. <V2.name> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
Boolean.operator.n	A character string specifying one of six possible Boolean operators: '==', '!=', '>', '>=', '<', '<=' <Boolean.operator.n> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
keep.cols	a numeric vector specifying the numbers of the columns to be kept in the final subset when subsetting by column. For example: keep.cols=c(2:5,7,12) will keep columns 2,3,4,5,7 and 12. <keep.cols> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
rm.cols	a numeric vector specifying the numbers of the columns to be removed before creating the final subset when subsetting by column. For example: rm.cols=c(2:5,7,12) will remove columns 2,3,4,5,7 and 12. <rm.cols> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset
keep.NAs	logical, if TRUE any NAs in the vector holding the final Boolean vector indicating whether a given row should be included in the subset will be converted into 1s and so they will be included in the subset. Such NAs could be caused by NAs in either <V1.name> or <V2.name>. If FALSE or NULL NAs in the final Boolean vector will be converted to 0s and the corresponding row will therefore be excluded from the subset. <keep.NAs> argument generated and passed directly to dataFrameSubsetDS1 by ds.dataFrameSubset

Details

A data frame is a list of variables all with the same number of rows, which is of class 'data.frame'. For all details see the help header for ds.dataFrameSubset

Value

This first serverside function called by ds.dataFrameSubset provides first level traps for a comprehensive series of disclosure risks which can be returned directly to the clientside because dataFrameSubsetDS1 is an aggregate function. The second serverside function called by ds.dataFrameSubset (dataFrameSubsetDS2) carries out most of the same disclosure tests, but it is an assign function because it writes the subsetted data.frame to the serverside. In consequence, it records error messages as studysideMessages which can only be retrieved using ds.message

Author(s)

Paul Burton

dataFrameSubsetDS2 *dataFrameSubsetDS2 an assign function called by ds.dataFrameSubset*

Description

Second serverside function for subsetting a data frame by row or by column.

Usage

```
dataFrameSubsetDS2(
  df.name = NULL,
  V1.name = NULL,
  V2.name = NULL,
  Boolean.operator.n = NULL,
  keep.cols = NULL,
  rm.cols = NULL,
  keep.NAs = NULL
)
```

Arguments

df.name	a character string providing the name for the data.frame to be sorted. <df.name> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
V1.name	A character string specifying the name of a subsetting vector to which a Boolean operator will be applied to define the subset to be created. <V1.name> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
V2.name	A character string specifying the name of the vector or scalar to which the values in the vector specified by the argument <V1.name> is to be compared. <V2.name> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
Boolean.operator.n	A character string specifying one of six possible Boolean operators: '==', '!=', '>', '>=', '<', '<=' <Boolean.operator.n> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
keep.cols	a numeric vector specifying the numbers of the columns to be kept in the final subset when subsetting by column. For example: keep.cols=c(2:5,7,12) will keep columns 2,3,4,5,7 and 12. <keep.cols> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
rm.cols	a numeric vector specifying the numbers of the columns to be removed before creating the final subset when subsetting by column. For example: rm.cols=c(2:5,7,12) will remove columns 2,3,4,5,7 and 12. <rm.cols> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset
keep.NAs	logical, if TRUE any NAs in the vector holding the final Boolean vector indicating whether a given row should be included in the subset will be converted

into 1s and so they will be included in the subset. Such NAs could be caused by NAs in either <V1.name> or <V2.name>. If FALSE or NULL NAs in the final Boolean vector will be converted to 0s and the corresponding row will therefore be excluded from the subset. <keep.NAs> argument generated and passed directly to dataFrameSubsetDS2 by ds.dataFrameSubset

Details

A data frame is a list of variables all with the same number of rows, which is of class 'data.frame'. For all details see the help header for ds.dataFrameSubset

Value

the object specified by the <newobj> argument (or default name '<df.name>_subset') initially specified in calling ds.dataFrameSubset. The output object (the required subsetted data.frame called <newobj> is written to the serverside. In addition, two validity messages are returned via ds.dataFrameSubset indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - dataFrameSubsetDS2 (via ds.dataFrame()) also returns any studysideMessages that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the `ds.message` function. If you type ds.message("newobj") it will print out the relevant studysideMessage from any datasource in which there was an error in creating <newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message("newobj") will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

DataSHIELD Development Team

---

densityGridDS *Generates a density grid with or without a priori defined limits*

Description

Generates a density grid that can then be used for heatmap or contour plots.

Usage

```
densityGridDS(
  xvect,
  yvect,
  limits = FALSE,
  x.min = NULL,
  x.max = NULL,
  y.min = NULL,
```

```
  y.max = NULL,
  numints = 20
)
```

Arguments

xvect	a numerical vector
yvect	a numerical vector
limits	a logical expression for whether or not limits of the density grid are defined by a user. If limits is set to "FALSE", min and max of xvect and yvect are used as a range. If limits is set to "TRUE", limits defined by x.min, x.max, y.min and y.max are used.
x.min	a minimum value for the x axis of the grid density object, if needed
x.max	a maximum value for the x axis of the grid density object, if needed
y.min	a minimum value for the y axis of the grid density object, if needed
y.max	a maximum value for the y axis of the grid density object, if needed
numints	a number of intervals for the grid density object, by default is 20

Details

Invalid cells (cells with count < to the set filter value for the minimum allowed counts in table cells) are turn to 0.

Value

a grid density matrix

Author(s)

Julia Isaeva, Amadou Gaye, Demetris Avraam for DataSHIELD Development Team

---

dimDS *Returns the dimension of a data frame or matrix*

Description

This function is similar to R function dim.

Usage

```
dimDS(x)
```

Arguments

x	a string character, the name of a dataframe or matrix

Details

The function returns the dimension of the input dataframe or matrix

Value

the dimension of the input object

Author(s)

Demetris Avraam, for DataSHIELD Development Team

`dmtC2SDS` *Copy a clientside data.frame, matrix or tibble (DMT) to the serverside.*

Description

Creates a data.frame, matrix or tibble on the serverside that is equivalent to that same data.frame, matrix or tibble (DMT) on the clientside.

Usage

```
dmtC2SDS(
  dfdata.mat.transmit,
  inout.object.transmit,
  from,
  nrows.transmit,
  ncols.transmit,
  colnames.transmit,
  colclass.transmit,
  byrow
)
```

Arguments

`dfdata.mat.transmit`
: a character string in a format that can pass through the DataSHIELD R parser which specifies the name of the DMT to be copied from the clientside to the serverside. Value fully specified by <dfdata> argument of ds.dmtC2S.

`inout.object.transmit`
: a character string taking values "DF", "MAT" or "TBL". The value of this argument is automatically set by ds.dmtC2S depending on whether the clientside DMT is a data.frame, matrix or tibble. Correspondingly, its value determines whether the object created on the serverside is a data.frame, matrix or tibble. This is unlikely to always work (some class misspecifications may occur) but it works in all the test cases.

`from`
: a character string specifying the source of <dfdata>. Fixed by clientside function as "clientside.matdftbl".

`nrows.transmit` specifies the number of rows in the matrix to be created. Fixed by the clientside function as equal to the number of rows in the clientside DMT to be transferred.

`ncols.transmit` specifies the number of columns in the matrix to be created. Fixed by the clientside function as equal to the number of columns in the clientside DMT to be transferred.

`colnames.transmit`
a parser-transmissible vector specifying the name of each column in the DMT being transferred from clientside to serverside. Generated automatically by clientside function from colnames of clientside DMT.

`colclass.transmit`
a parser-transmissible vector specifying the class of the vector representing each individual column in the DMT to be transferred. Generated automatically by clientside function. This allows the transmission of DMTs containing columns with different classes.If something is going to go wrong with class misspecification (see inout.object.transmit) it is a DMT with a complex combination of data/column types that will most likely be the cause. This suggests that you always check the class of the serverside DMT and its individual columns (if the latter is important). If a situation arises where the class of the columns is crucial and the function cannot do what is needed please contact the DataSHIELD forum and we can try to remedy the problem.

`byrow` a logical value specifying whether the DMT created on the serverside should be filled row by row or column by column. This is fixed by the clientside function as FALSE (fill column by column).

Details

dmtC2SDS is a serverside assign function called by ds.dmtC2S. For more information about how it works see help for ds.dmtC2S

Value

the object specified by the <newobj> argument (or default name "matdftbl.copied.C2S") which is written as a data.frame, matrix or tibble to the serverside.

Author(s)

Paul Burton for DataSHIELD Development Team - 3rd June, 2021

---

elsplineDS *Basis for a piecewise linear spline with meaningful coefficients*

Description

This function is based on the native R function `elspline` from the `lspline` package. This function computes the basis of piecewise-linear spline such that, depending on the argument marginal, the coefficients can be interpreted as (1) slopes of consecutive spline segments, or (2) slope change at consecutive knots.

Usage

```
elsplineDS(x = x, n = n, marginal = FALSE, names = NULL)
```

Arguments

x	the name of the input numeric variable
n	integer greater than 2, knots are computed such that they cut n equally-spaced intervals along the range of x
marginal	logical, how to parametrize the spline, see Details
names	character, vector of names for constructed variables

Details

If marginal is FALSE (default) the coefficients of the spline correspond to slopes of the consecutive segments. If it is TRUE the first coefficient correspond to the slope of the first segment. The consecutive coefficients correspond to the change in slope as compared to the previous segment. Function elspline wraps lspline and computes the knot positions such that they cut the range of x into n equal-width intervals.

Value

an object of class "lspline" and "matrix", which its name is specified by the newobj argument (or its default name "elspline.newobj"), is assigned on the serverside.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

extractQuantilesDS1 *Secure ranking of "V2BR" (vector to be ranked) across all sources and use of these ranks to estimate global quantiles across all studies*

Description

identify the global values of V2BR (i.e. the values across all studies) that relate to a set of quantiles to be evaluated.

Usage

```
extractQuantilesDS1(extract.quantiles, extract.summary.output.ranks.df)
```

Arguments

`extract.quantiles`
: one of a restricted set of character strings that fix the set of quantile values for which the corresponding values across all studies are to be estimated. For more details see the associated document entitled "secure.global.ranking.docx", the header for ds.ranksSecure and ds.extractQuantiles functions. The value of this argument is set in choosing the value of the argument <quantiles.for.estimation> in ds.ranksSecure.

`extract.summary.output.ranks.df`
: character string specifying optional name for the data.frame written to the serverside on each data source that contains 5 of the key output variables from the ranking procedure pertaining to that particular data source. This data frame represents the key source of information - including global ranks - that determines the values of V2BR that are identified as corresponding to the particular set of quantiles to be estimated as specified by the <quantiles.for.estimation> argument of function ds.ranksSecure (and the <extract.quantiles> argument of ds.extractQuantiles).

Details

Severside aggregate function called by ds.extractQuantiles via ds.ranksSecure. As well as estimating the key values of V2BR that correspond to the selected quantiles, this function also implements a disclosure control trap. If the ratio of the total number of all observations across all studies divided by the number of quantile values to be estimated is less than or equal to nfilter.subset (which specifies the minimum size of a subset) the process stops and an error message is returned suggesting that you might try selecting a narrower range of quantiles with less quantile values to be estimated as specified by the argument <quantiles.for.estimation> of the function ds.ranksSecure. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx"

Value

as a first step in creating the vector of values of values of V2BR that correspond to each quantile value, extractQuantilesDS1 identifies the two closest quantile values across all studies that span each key quantile value. These are saved as the data frame "closest.bounds.df" on the clientside and then saved on the serverside by ds.dmtC2S into the data frame "global.bounds.df". Also if the number of observations across all studies is too small, and a disclosure risk exists if the final.quantile.vector is made available via the client, this function stops the processing and returns a warning/error message.

Author(s)

Paul Burton 11th November, 2021

`extractQuantilesDS2` *Secure ranking of "V2BR" (vector to be ranked) across all sources and use of these ranks to estimate global quantiles across all studies*

Description

identify the global values of V2BR (i.e. the values across all studies) that relate to a set of quantiles to be evaluated.

Usage

```
extractQuantilesDS2(extract.summary.output.ranks.df)
```

Arguments

`extract.summary.output.ranks.df`
: character string specifies an optional name for the data.frame written to the serverside on each data source that contains 5 of the key output variables from the ranking procedure pertaining to that particular data source. This data frame represents the key source of information - including global ranks - that determines the values of V2BR that are identified as corresponding to the particular set of quantiles to be estimated as specified by the <quantiles.for.estimation> argument of function ds.ranksSecure (and the <extract.quantiles> argument of ds.extractQuantiles).

Details

Severside aggregate function called by ds.extractQuantiles via ds.ranksSecure. This takes the "global.bounds.df" data frame saved on the serverside following construction by extractQuantilesDS1. This data frame includes the two quantile values that most closely span each quartile value to be estimated. If either of the values had been the correct value for a given quantile, both the bounding values would have taken that value in global.bounds.df. This is because the upper bound was defined as the lowest value that was equal to or greater than the true value for that quantile while the lower bound was defined as the highest value that was equal to or lower than the true value. Next, the function extractQuantileDS2 goes round study by study to identify the values of V2BR that actually correspond to each of the spanning values around each quantile. Then the function goes quantile by quantile and estimates the mean of the two values of V2BR that correspond to the the spanning quantiles. If these two values are the same it means that that value of V2BR is the "true" value and the mean of two (or potentially several) instances of that value is inevitably also equal to that true value. If the upper and lower bounding values of V2BR differ, neither can be the precisely correct single value of V2BR for that quantile (see above for explanation) and so the mean of the two is a reasonable interpolated summary.

Value

the single value of V2BR which best corresponds to each key quantile value to be estimated as specified by the argument <quantiles.for.estimation> A data frame (final.quantile.df)summarising

the results of this analysis is written to the clientside. This data frame consists of two vectors. The first is named "evaluation.quantiles". It lists the full set of quantiles you have requested for evaluation as specified by the argument "quantiles.for.estimation" The second vector which is called "final.quantile.vector" details the values of V2BR that correspond to the the key quantiles listed in vector 1.

Author(s)

Paul Burton 11th November, 2021

`fixClassDS` *Change Class of Target Variables in a Data Frame*

Description

Change Class of Target Variables in a Data Frame

Usage

```
fixClassDS(df.name, target_vars, target_class)
```

Arguments

`df.name`	A string representing the name of the data frame.
`target_vars`	A character vector specifying the columns to be modified.
`target_class`	A character vector specifying the new classes for each column (1 = factor, 2 = integer, 3 = numeric, 4 = character, 5 = logical).

Value

A modified data frame with the specified columns converted to the target classes.

`fixColsDS` *Add Missing Columns with NA Values*

Description

Add Missing Columns with NA Values

Usage

```
fixColsDS(.data, cols)
```

Arguments

.data	A string representing the name of the data frame.
cols	A character vector specifying the columns to be added if missing.

Value

A modified data frame with missing columns added and filled with NA.

---

fixLevelsDS *Set Factor Levels for Specific Columns in a Data Frame*

Description

Set Factor Levels for Specific Columns in a Data Frame

Usage

```
fixLevelsDS(df.name, vars, levels)
```

Arguments

df.name	A string representing the name of the data frame to modify.
vars	A character vector specifying the columns to be modified.
levels	A named list where each element contains the levels for the corresponding factor variable.

Value

A modified data frame with the specified columns converted to factors with the provided levels.

---

gamlssDS *gamlssDS an aggregate function called by ds.gamlss*

Description

This function calls the gamlssDS that is a wrapper function from the gamlss R package. The function returns an object of class "gamlss", which is a generalized additive model for location, scale and shape (GAMLSS). The function also saves the residuals as an object on the server-side with a name specified by the newobj argument. In addition, if the argument centiles is set to TRUE, the function calls the centiles function from the gamlss package and returns the sample percentages below each centile curve.

Usage

```
gamlssDS(
  formula = formula,
  sigma.formula = sigma.formula,
  nu.formula = nu.formula,
  tau.formula = tau.formula,
  family = family,
  data = data,
  method = method,
  mu.fix = mu.fix,
  sigma.fix = sigma.fix,
  nu.fix = nu.fix,
  tau.fix = tau.fix,
  control = control,
  i.control = i.control,
  centiles = centiles,
  xvar = xvar,
  newobj = newobj
)
```

Arguments

formula	a formula object, with the response on the left of an ~ operator, and the terms, separated by + operators, on the right. Nonparametric smoothing terms are indicated by pb() for penalised beta splines, cs for smoothing splines, lo for loess smooth terms and random or ra for random terms, e.g. y~cs(x,df=5)+x1+x2*x3.
sigma.formula	a formula object for fitting a model to the sigma parameter, as in the formula above, e.g. sigma.formula=~cs(x,df=5).
nu.formula	a formula object for fitting a model to the nu parameter, e.g. nu.formula=~x
tau.formula	a formula object for fitting a model to the tau parameter, e.g. tau.formula=~cs(x,df=2)
family	a gamlss.family object, which is used to define the distribution and the link functions of the various parameters. The distribution families supported by gamlss() can be found in gamlss.family. Functions such as BI() (binomial) produce a family object. Also can be given without the parentheses i.e. BI. Family functions can take arguments, as in BI(mu.link=probit).
data	a data frame containing the variables occurring in the formula. If this is missing, the variables should be on the parent environment.
method	a character indicating the algorithm for GAMLSS. Can be either 'RS', 'CG' or 'mixed'. If method='RS' the function will use the Rigby and Stasinopoulos algorithm, if method='CG' the function will use the Cole and Green algorithm, and if method='mixed' the function will use the RS algorithm twice before switching to the Cole and Green algorithm for up to 10 extra iterations.
mu.fix	logical, indicate whether the mu parameter should be kept fixed in the fitting processes.
sigma.fix	logical, indicate whether the sigma parameter should be kept fixed in the fitting processes.

`nu.fix` logical, indicate whether the nu parameter should be kept fixed in the fitting processes.

`tau.fix` logical, indicate whether the tau parameter should be kept fixed in the fitting processes.

`control` this sets the control parameters of the outer iterations algorithm using the gamlss.control function. This is a vector of 7 numeric values: (i) c.crit (the convergence criterion for the algorithm), (ii) n.cyc (the number of cycles of the algorithm), (iii) mu.step (the step length for the parameter mu), (iv) sigma.step (the step length for the parameter sigma), (v) nu.step (the step length for the parameter nu), (vi) tau.step (the step length for the parameter tau), (vii) gd.tol (global deviance tolerance level). The default values for these 7 parameters are set to c(0.001, 20, 1, 1, 1, 1, Inf).

`i.control` this sets the control parameters of the inner iterations of the RS algorithm using the glim.control function. This is a vector of 4 numeric values: (i) cc (the convergence criterion for the algorithm), (ii) cyc (the number of cycles of the algorithm), (iii) bf.cyc (the number of cycles of the backfitting algorithm), (iv) bf.tol (the convergence criterion (tolerance level) for the backfitting algorithm). The default values for these 4 parameters are set to c(0.001, 50, 30, 0.001).

`centiles` logical, indicating whether the function centiles() will be used to tabulate the sample percentages below each centile curve. Default is set to FALSE.

`xvar` the unique explanatory variable used in the centiles() function. This variable is used only if the centiles argument is set to TRUE. A restriction in the centiles function is that it applies to models with one explanatory variable only.

`newobj` a character string that provides the name for the output object that is stored on the data servers. Default `gamlss_residuals`.

Details

For additional details see the help header of gamlss and centiles functions in native R gamlss package.

Value

a gamlss object with all components as in the native R gamlss function. Individual-level information like the components y (the response response) and residuals (the normalised quantile residuals of the model) are not disclosed to the client-side.

Author(s)

Demetris Avraam for DataSHIELD Development Team

getAllLevelsDS *Retrieve Factor Levels for Specific Columns*

Description

Retrieve Factor Levels for Specific Columns

Usage

```
getAllLevelsDS(df.name, factor_vars)
```

Arguments

df.name	A string representing the name of the data frame.
factor_vars	A character vector specifying the factor columns.

Value

A list of factor levels for the specified columns.

getClassAllColsDS *Get the Class of All Columns in a Data Frame*

Description

Get the Class of All Columns in a Data Frame

Usage

```
getClassAllColsDS(df.name)
```

Arguments

df.name	A string representing the name of the data frame.

Value

A tibble with the class of each column in the data frame.

getWGSRDS *Computes the WHO Growth Reference z-scores of anthropometric data*

Description

Calculate WHO Growth Reference z-score for a given anthropometric measurement This function is similar to R function getWGSR from the zscorer package.

Usage

```
getWGSRDS(sex, firstPart, secondPart, index, standing = NA, thirdPart = NA)
```

Arguments

sex	the name of the binary variable that indicates the sex of the subject. This must be coded as 1 = male and 2 = female. If in your project the variable sex has different levels, you should recode the levels to 1 for males and 2 for females using the ds.recodeValues DataSHIELD function before the use of the ds.getWGSR.
firstPart	Name of variable specifying: Weight (kg) for BMI/A, W/A, W/H, or W/L Head circumference (cm) for HC/A Height (cm) for H/A Length (cm) for L/A MUAC (cm) for MUAC/A Sub-scapular skinfold (mm) for SSF/A Triceps skinfold (mm) for TSF/A Give a quoted variable name as in (e.g.) "weight". Be careful with units (weight in kg; height, length, head circumference, and MUAC in cm, skinfolds in mm).
secondPart	Name of variable specifying: Age (days) for H/A, HC/A, L/A, MUAC/A, SSF/A, or TSF/A Height (cm) for BMI/A, or W/H Length (cm) for W/L Give a quoted variable name as in (e.g.) "age". Be careful with units (age in days; height and length in cm).
index	The index to be calculated and added to data. One of: bfa BMI for age hca Head circumference for age hfa Height for age lfa Length for age mfa MUAC for age ssa Sub-scapular skinfold for age tsa Triceps skinfold for age wfa Weight for age wfh Weight for height wfl Weight for length Give a quoted index name as in (e.g.) "wfh".

standing	Variable specifying how stature was measured. If NA (default) then age (for "hfa" or "lfa") or height rules (for "wfh" or "wfl") will be applied. This must be coded as 1 = Standing; 2 = Supine; 3 = Unknown. Missing values will be recoded to 3 = Unknown. Give a single value (e.g."1"). If no value is specified then height and age rules will be applied.
thirdPart	Name of variable specifying age (in days) for BMI/A. Give a quoted variable name as in (e.g.) "age". Be careful with units (age in days).

Details

The function computes the WHO Growth Reference z-scores of anthropometric data for weight, height or length, MUAC, head circumference, sub-scapular skinfold and triceps skinfold. Note that the function might fail or return NAs when the variables are outside the ranges given in the WGS (WHO Child Growth Standards) reference (i.e. 45 to 120 cm for height and 0 to 60 months for age). It is up to the user to check the ranges and the units of their data.

Value

`ds.getWGSR` assigns a numeric vector that includes the z-scores for the specified index.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

glmDS1 *glmDS1 called by ds.glm*

Description

This is the first server-side aggregate function called by ds.glm

Usage

```
glmDS1(formula, family, weights, offset, data)
```

Arguments

formula	a glm() formula consistent with R syntax eg U~x+y+Z to regress variables U on x,y and Z
family	a glm() family consistent with R syntax eg "gaussian", "poisson", "binomial"
weights	an optional variable providing regression weights
offset	the offset
data	an optional character string specifying a data.frame object holding the data to be analysed under the specified model

Details

It is an aggregation function that sets up the model structure and creates the starting beta.vector that feeds, via ds.glm, into glmDS2 to enable iterative fitting of the generalized linear model that has been been specified. For more details please see the extensive header for ds.glm.

Value

List with values from GLM model.

Author(s)

Burton PR for DataSHIELD Development Team

`glmDS2` *glmDS2 called by ds.glm*

Description

This is the second server-side aggregate function called by ds.glm.

Usage

```
glmDS2(formula, family, beta.vect, offset, weights, dataName)
```

Arguments

`formula`	a glm() formula consistent with R syntax eg U~x+y+Z to regress variables U on x, y and Z
`family`	a glm() family consistent with R syntax eg "gaussian", "poisson", "binomial"
`beta.vect`	a numeric vector created by the clientside function specifying the vector of regression coefficients at the current iteration
`offset`	an optional variable providing a regression offset
`weights`	an optional variable providing regression weights
`dataName`	an optional character string specifying a data.frame object holding the data to be analysed under the specified model same

Details

It is an aggregate function that uses the model structure and starting beta.vector constructed by glmDS1 to iteratively fit the generalized linear model that has been specified. The function glmDS2 also carries out a series of disclosure checks and if the arguments or data fail any of those tests, model construction is blocked and an appropriate serverside error message is created and returned to ds.glm on the clientside. For more details please see the extensive header for ds.glm.

Value

List with values from GLM model

Author(s)

Paul Burton, for DataSHIELD Development Team

`glmerSLMADS.assign` *Fitting generalized linear mixed effect models - serverside function*

Description

glmerSLMADS.assign is the same as glmerSLMADS2 which fits a generalized linear mixed effects model (glme) per study and saves the outcomes in each study

Usage

```
glmerSLMADS.assign(
  formula,
  offset,
  weights,
  dataName,
  family,
  control_type = NULL,
  control_value.transmit = NULL,
  nAGQ = 1L,
  verbose = 0,
  theta = NULL,
  fixef = NULL
)
```

Arguments

`formula`	see help for ds.glmerSLMA
`offset`	see help for ds.glmerSLMA
`weights`	see help for ds.glmerSLMA
`dataName`	see help for ds.glmerSLMA
`family`	see help for ds.glmerSLMA
`control_type`	see help for ds.glmerSLMA
`control_value.transmit`	see help for argument <control_value> for function ds.glmerSLMA
`nAGQ`	integer scalar, defaulting to 1L. IN PRACTICE, IT MAY BE NECESSARY TO SET nAGQ TO 0L when the model appears to converge perfectly well (e.g. verbose=2 demonstrates good initial convergence of both the log-likelihood and regression coefficients) but formal convergence does not get declared - so no output is produced - despite running the model for many iterations. The nAGQ argument is set by the nAGQ argument for ds.glmerSLMA and further details can be found in help(ds.glmerSLMA) and in the native R help for glmer()

verbose	see help for ds.glmerSLMA
theta	see help for argument <start_theta> for function ds.glmerSLMA
fixef	see help for argument <start_fixef> for function ds.glmerSLMA

Details

glmerSLMADS.assign is a serverside function called by ds.glmerSLMA on the clientside. The analytic work engine is the glmer function in R which sits in the lme4 package. glmerSLMADS.assign fits a generalized linear mixed effects model (glme) - e.g. a logistic or Poisson regression model including both fixed and random effects - on data from each single data source and saves the regression outcomes on the serverside.

Value

writes glmerMod object summarising the fitted model to the serverside. For more detailed information see help for ds.glmerSLMA.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

glmerSLMADS2 *Fitting generalized linear mixed effect models - serverside function*

Description

glmerSLMADS2 fits a generalized linear mixed effects model (glme) - e.g. a logistic or Poisson regression model including both fixed and random effects - on data from one or multiple sources with pooling via SLMA (study level meta-analysis)

Usage

```
glmerSLMADS2(
  formula,
  offset,
  weights,
  dataName,
  family,
  control_type = NULL,
  control_value.transmit = NULL,
  nAGQ = 1L,
  verbose = 0,
  theta = NULL,
  fixef = NULL
)
```

Arguments

formula	see help for ds.glmerSLMA
offset	see help for ds.glmerSLMA
weights	see help for ds.glmerSLMA
dataName	see help for ds.glmerSLMA
family	see help for ds.glmerSLMA
control_type	see help for ds.glmerSLMA
control_value.transmit	see help for argument <control_value> for function ds.glmerSLMA
nAGQ	integer scalar, defaulting to 1L. IN PRACTICE, IT MAY BE NECESSARY TO SET nAGQ TO 0L when the model appears to converge perfectly well (e.g. verbose=2 demonstrates good initial convergence of both the log-likelihood and regression coefficients) but formal convergence does not get declared - so no output is produced - despite running the model for many iterations. The nAGQ argument is set by the nAGQ argument for ds.glmerSLMA and further details can be found in help(ds.glmerSLMA) and in the native R help for glmer()
verbose	see help for ds.glmerSLMA
theta	see help for argument <start_theta> for function ds.glmerSLMA
fixef	see help for argument <start_fixef> for function ds.glmerSLMA

Details

glmerSLMADS2 is a serverside function called by ds.glmerSLMA on the clientside. The analytic work engine is the glmer function in R which sits in the lme4 package. ds.glmerSLMA fits a generalized linear mixed effects model (glme) - e.g. a logistic or Poisson regression model including both fixed and random effects - on data from a single or multiple sources. When there are multiple data sources, the glme is fitted to convergence in each data source independently and the estimates and standard errors returned to the client thereby enabling cross-study pooling using study level meta-analysis (SLMA). By default the SLMA is undertaken using the metafor package, but as the SLMA occurs on the clientside which, as far as the user is concerned is just a standard R environment, the user can choose to use any approach to meta-analysis they choose. Additional information about fitting glmes using the glmer engine can be obtained using R help for glmer and the lme4 package

Value

all key model components see help for ds.glmerSLMA

Author(s)

Tom Bishop, with some additions by Paul Burton

`glmPredictDS.ag` *predict regression responses from a glm object*

Description

identify and return key components/summaries of a serverside glm_predict object that can safely be returned to the clientside without disclosure risk

Usage

```
glmPredictDS.ag(
  glmname.transmit,
  newdataname.transmit,
  output.type,
  se.fit,
  dispersion,
  terms.transmit,
  na.action
)
```

Arguments

`glmname.transmit`	a character string specifying the name of the glm object on the serverside that is to be used for prediction. Fully specified by glmname argument in ds.glmPredict
`newdataname.transmit`	a character string specifying an (optional) dataframe on the serverside in which to look for (potentially) new covariate values on which to base the predictions. Fully specified by newdataname argument in ds.glmPredict.
`output.type`	a character string taking the values 'response', 'link' or 'terms'. Fully specified by corresponding argument in ds.glmPredict.
`se.fit`	logical if standard errors for the fitted predictions are required. Fully specified by corresponding argument in ds.glmPredict.
`dispersion`	numeric value specifying the dispersion of the GLM fit to be assumed in computing the standard errors. Fully specified by corresponding argument in ds.glmPredict.
`terms.transmit`	a character vector specifying a subset of terms to return in the prediction. Fully specified by 'terms' argument in ds.glmPredict.
`na.action`	character string determining what should be done with missing values in the data.frame identified by <newdataname.transmit>. Fully specified by na.action argument in ds.glmPredict.

Details

Serverside aggregate function called by ds.glmPredict. It is called immediately after the assign function glmPredict.as has created a predict_glm object on the serverside by applying the equivalent

of predict.glm() in native R to a glm object on the serverside. The aggregate function, glmPredict.ag, then identifies and returns components of that predict_glm object that can safely be returned to the clientside without a risk of disclosure. For further details see DataSHIELD help for ds.glmPredict and glmPredict.as and help in native R for predict.glm

Value

components/summarising statistics of a serverside predict_glm object that can safely be transmitted to the clientside without a risk of disclosure. For further details see DataSHIELD help for ds.glmPredict and glmPredict.as and help in native R for predict.glm predict.glm in native R

Author(s)

Paul Burton for DataSHIELD Development Team (20/7/20)

`glmPredictDS.as` *predict regression responses from a glm object*

Description

create a predict_glm object on the serverside by applying the equivalent of predict.glm() in native R to a glm object on the serverside. Identify and return components of the predict_glm object that can safely be sent to the clientside without a risk of disclosure

Usage

```
glmPredictDS.as(
  glmname.transmit,
  newdataname.transmit,
  output.type,
  se.fit,
  dispersion,
  terms.transmit,
  na.action
)
```

Arguments

`glmname.transmit`
: a character string specifying the name of the glm object on the serverside that is to be used for prediction. Fully specified by glmname argument in ds.glmPredict

`newdataname.transmit`
: a character string specifying an (optional) dataframe on the serverside in which to look for (potentially) new covariate values on which to base the predictions. Fully specified by newdataname argument in ds.glmPredict.

`output.type`
: a character string taking the values 'response', 'link' or 'terms'. Fully specified by corresponding argument in ds.glmPredict.

se.fit	logical if standard errors for the fitted predictions are required. Fully specified by corresponding argument in ds.glmPredict.
dispersion	numeric value specifying the dispersion of the GLM fit to be assumed in computing the standard errors. Fully specified by corresponding argument in ds.glmPredict.
terms.transmit	a character vector specifying a subset of terms to return in the prediction. Fully specified by 'terms' argument in ds.glmPredict.
na.action	character string determining what should be done with missing values in the data.frame identified by <newdataname.transmit>. Fully specified by na.action argument in ds.glmPredict.

Details

Serverside assign function called by ds.glmPredict makes predictions of regression responses based on a serverside glm object that has already been created on the serverside by ds.glmSLMA and and writes the predict_glm object to the serverside. For further details see help for ds.glmPredict and help in native R for predict.glm

Value

glmPredict.as writes a new object to the serverside containing output precisely equivalent to the output from predict.glm in native R. For more details see DataSHIELD help for ds.glmPredict and help for predict.glm in native R

Author(s)

Paul Burton for DataSHIELD Development Team (20/7/20)

---

glmSLMADS.assign — *Fit a Generalized Linear Model (GLM) with pooling via Study Level Meta-Analysis (SLMA)*

Description

Fits a generalized linear model (GLM) on data from single or multiple sources with pooled co-analysis across studies being based on SLMA (Study Level Meta Analysis).

Usage

```
glmSLMADS.assign(formula, family, offsetName, weightsName, dataName)
```

Arguments

formula	a glm formula, specified in call to ds.glmSLMA
family	a glm family, specified in call to ds.glmSLMA
offsetName	a character string specifying a variable to be used as an offset. Specified in call to ds.glmSLMA.

weightsName	a character string specifying a variable to be used as regression weights. Specified in call to ds.glmSLMA. Specified in call to ds.glmSLMA.
dataName	a character string specifying the name of a data.frame holding the data for the model. Specified in call to ds.glmSLMA.

Details

glmSLMADS.assign is an assign function called by clientside function ds.glmSLMA. ds.glmSLMA also calls two aggregate functions glmSLMADS1 and glmSLMADS2. For more detailed information see help for ds.glmSLMA.

Value

writes glm object summarising the fitted model to the serverside. For more detailed information see help for ds.glmSLMA.

Author(s)

Paul Burton for DataSHIELD Development Team (14/7/20)

---

glmSLMADS1 *Fit a Generalized Linear Model (GLM) with pooling via Study Level Meta-Analysis (SLMA)*

Description

Fits a generalized linear model (GLM) on data from single or multiple sources with pooled co-analysis across studies being based on SLMA (Study Level Meta Analysis).

Usage

```
glmSLMADS1(formula, family, weights, offset, data)
```

Arguments

formula	a glm formula, specified in call to ds.glmSLMA
family	a glm family, specified in call to ds.glmSLMA
weights	a character string specifying a variable to be used as regression weights. Specified in call to ds.glmSLMA. Specified in call to ds.glmSLMA.
offset	a character string specifying a variable to be used as an offset. Specified in call to ds.glmSLMA.
data	a character string specifying the name of a data.frame holding the data for the model. Specified as dataName in call to ds.glmSLMA.

Details

glmSLMADS.assign is an aggregate function called by clientside function ds.glmSLMA. ds.glmSLMA also calls another aggregate function glmSLMADS2 and an assign function glmSLMADS.assign For more detailed information see help for ds.glmSLMA.

Value

assesses and returns information about failure to pass disclosure traps such as test of model complexity (saturation). For more detailed information see help for ds.glmSLMA.

Author(s)

Paul Burton for DataSHIELD Development Team (14/7/20)

`glmSLMADS2` *Fit a Generalized Linear Model (GLM) with pooling via Study Level Meta-Analysis (SLMA)*

Description

Fits a generalized linear model (GLM) on data from single or multiple sources with pooled co-analysis across studies being based on SLMA (Study Level Meta Analysis).

Usage

```
glmSLMADS2(formula, family, offset, weights, newobj, dataName)
```

Arguments

`formula`	a glm formula, specified in call to ds.glmSLMA
`family`	a glm family, specified in call to ds.glmSLMA
`offset`	a character string specifying a variable to be used as an offset. Specified in call to ds.glmSLMA.
`weights`	a character string specifying a variable to be used as regression weights. Specified in call to ds.glmSLMA. Specified in call to ds.glmSLMA.
`newobj`	a character string specifying the name of the glm object written to the server-side by glmSLMADS.assign. This is either the name specified by the newobj argument in ds.glmSLMA or if newobj was unspecified or NULL it is called new.glm.obj.
`dataName`	a character string specifying the name of a data.frame holding the data for the model. Specified in call to ds.glmSLMA.

Details

glmSLMADS.assign is an aggregate function called by clientside function ds.glmSLMA. ds.glmSLMA also calls another aggregate function glmSLMADS2 and an assign function glmSLMADS.assign For more detailed information see help for ds.glmSLMA.

Value

All quantitative, Boolean, and character objects required to enable the SLMA pooling of the separate glm models fitted to each study - in particular including the study-specific regression coefficients and their corresponding standard errors.

Author(s)

Paul Burton for DataSHIELD Development Team (14/7/20)

`glmSummaryDS.ag` *summarize a glm object on the serverside*

Description

returns the non-disclosive elements to the clientside of a glm object and the corresponding object holding the output of summary(glm object) on the serverside.

Usage

```
glmSummaryDS.ag(x.transmit)
```

Arguments

`x.transmit`	a character string specifying the name of the glm object on the serverside that is to be summarised. This is specified by x.name argument in ds.glmSummary

Details

Serverside aggregate function called by ds.glmSummary. ds.glmSummary first calls glmSummaryDS.ag to create a glm_summary object on the serverside based on applying native R's summary.glm() to a serverside glm object previously created by ds.glmSLMA. Then it calls glmSummaryDS.ag to return to the clientside all of the non-disclosive elements (and only the non-disclosive elements) of the serverside glm and its corresponding summary_glm object.

Value

returns to the clientside all of the non-disclosive elements (and only the non-disclosive elements) of a specified serverside glm and its corresponding summary_glm object.

Author(s)

Paul Burton for DataSHIELD Development Team (20/7/20)

glmSummaryDS.as *summarize a glm object on the serverside*

Description

summarize a glm object on the serverside to create a summary_glm object. Also identify and return components of both the glm object and the summary_glm object that can safely be sent to the clientside without a risk of disclosure

Usage

```
glmSummaryDS.as(x.transmit)
```

Arguments

x.transmit	a character string specifying the name of the glm object on the serverside that is to be summarised. This is specified by x.name argument in ds.glmSummary

Details

Serverside assign function called by ds.glmSummary summarises a glm object that has already been created on the serverside by fitting ds.glmSLMA and writes the summary_glm object to the serverside. For further details see help for ds.glmSLMA and help in native R for glm() and summary.glm

Value

writes object to serverside which is precisely equivalent to summary(glm object) in native R

Author(s)

Paul Burton for DataSHIELD Development Team (20/7/20)

heatmapPlotDS *Calculates the coordinates of the centroid of each n nearest neighbours*

Description

This function calculates the coordinates of the centroids for each n nearest neighbours.

Usage

```
heatmapPlotDS(x, y, k, noise, method.indicator)
```

Arguments

x	the name of a numeric vector, the x-variable.
y	the name of a numeric vector, the y-variable.
k	the number of the nearest neighbours for which their centroid is calculated if the method.indicator is equal to 1 (i.e. deterministic method).
noise	the percentage of the initial variance that is used as the variance of the embedded noise if the method.indicator is equal to 2 (i.e. probabilistic method).
method.indicator	a number equal to either 1 or 2. If the value is equal to 1 then the 'deterministic' method is used. If the value is set to 2 the 'probabilistic' method is used.

Details

The function finds the n-1 nearest neighbours of each data point in a 2-dimensional space. The nearest neighbours are the data points with the minimum Euclidean distances from the point of interest. Each point of interest and its n-1 nearest neighbours are then used for the calculation of the coordinates of the centroid of those n points. Centroid here is referred to the centre of mass, i.e. the x-coordinate of the centroid is the average value of the x-coordinates of the n nearest neighbours and the y-coordinate of the centroid is the average of the y-coordinates of the n nearest neighbours. The coordinates of the centroids return to the client side function and can be used for the plot of non-disclosive graphs (e.g. scatter plots, heatmap plots, contour plots, etc).

Value

a list with the x and y coordinates of the centroids if the deterministic method is used or the x and y coordinated of the noisy data if the probabilistic method is used.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

hetcorDS *Heterogeneous Correlation Matrix*

Description

This function is based on the hetcor function from the R package polycor.

Usage

```
hetcorDS(data, ML, std.err, bins, pd, use)
```

Arguments

data	the name of a data frame consisting of factors, ordered factors, logical variables, character variables, and/or numeric variables, or the first of several variables.
ML	if TRUE, compute maximum-likelihood estimates; if FALSE (default), compute quick two-step estimates.
std.err	if TRUE (default), compute standard errors.
bins	number of bins to use for continuous variables in testing bivariate normality; the default is 4.
pd	if TRUE (default) and if the correlation matrix is not positive-definite, an attempt will be made to adjust it to a positive-definite matrix, using the nearPD function in the Matrix package. Note that default arguments to nearPD are used (except corr=TRUE); for more control call nearPD directly.
use	if "complete.obs", remove observations with any missing data; if "pairwise.complete.obs", compute each correlation using all observations with valid data for that pair of variables.

Details

Computes a heterogeneous correlation matrix, consisting of Pearson product-moment correlations between numeric variables, polyserial correlations between numeric and ordinal variables, and polychoric correlations between ordinal variables.

Value

Returns an object of class "hetcor" with the following components: the correlation matrix; the type of each correlation: "Pearson", "Polychoric", or "Polyserial"; the standard errors of the correlations, if requested; the number (or numbers) of observations on which the correlations are based; p-values for tests of bivariate normality for each pair of variables; the method by which any missing data were handled: "complete.obs" or "pairwise.complete.obs"; TRUE for ML estimates, FALSE for two-step estimates.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

histogramDS1 *returns the minimum and the maximum of the input numeric vector*

Description

this function returns the minimum and maximum of the input numeric vector which depends on the argument `method.indicator`. If the method.indicator is set to 1 (i.e. the 'smallCellsRule' is used) the computed minimum and maximum values are multiplied by a very small random number. If the method.indicator is set to 2 (i.e. the 'deterministic' method is used) the function returns the minimum and maximum values of the vector with the scaled centroids. If the method.indicator is set to 3 (i.e. the 'probabilistic' method is used) the function returns the minimum and maximum values of the generated 'noisy' vector.

Usage

```
histogramDS1(xvect, method.indicator, k, noise)
```

Arguments

xvect	the numeric vector for which the histogram is desired.
method.indicator	a number equal to either 1, 2 or 3 indicating the method of disclosure control that is used for the generation of the histogram. If the value is equal to 1 then the 'smallCellsRule' is used. If the value is equal to 2 then the 'deterministic' method is used. If the value is set to 3 then the 'probabilistic' method is used.
k	the number of the nearest neighbours for which their centroid is calculated if the method.indicator is equal to 2 (i.e. deterministic method).
noise	the percentage of the initial variance that is used as the variance of the embedded noise if the method.indicator is equal to 3 (i.e. probabilistic method).

Value

a numeric vector which contains the minimum and the maximum values of the vector

Author(s)

Amadou Gaye, Demetris Avraam for DataSHIELD Development Team

---

histogramDS2 *Computes a histogram of the input variable without plotting.*

Description

This function produces the information required to plot a histogram. This is done without allowing for bins (cells) with number of counts less than the pre-specified disclosure control set for the minimum cell size of a table. If a bin has less counts than this threshold then their counts and its density are replaced by a 0 value.

Usage

```
histogramDS2(xvect, num.breaks, min, max, method.indicator, k, noise)
```

Arguments

xvect	the numeric vector for which the histogram is desired.
num.breaks	the number of breaks that the range of the variable is divided.
min	a numeric, the lower limit of the distribution.
max	a numeric, the upper limit of the distribution.

`method.indicator`
: a number equal to either 1, 2 or 3 indicating the method of disclosure control that is used for the generation of the histogram. If the value is equal to 1 then the 'smallCellsRule' is used. If the value is equal to 2 then the 'deterministic' method is used. If the value is set to 3 then the 'probabilistic' method is used.

`k`
: the number of the nearest neighbours for which their centroid is calculated if the `method.indicator` is equal to 2 (i.e. deterministic method).

`noise`
: the percentage of the initial variance that is used as the variance of the embedded noise if the `method.indicator` is equal to 3 (i.e. probabilistic method).

Details

Please find more details in the documentation of the clientside ds.histogram function.

Value

a list with an object of class `histogram` and the number of invalid cells

Author(s)

Amadou Gaye, Demetris Avraam for DataSHIELD Development Team

---

igb_standardsDS *Converts birth measurements to intergrowth z-scores/centiles*

Description

Converts birth measurements to INTERGROWTH z-scores/centiles (generic)

Usage

```
igb_standardsDS(
  gagebrth = gagebrth,
  z = z,
  p = p,
  val = val,
  var = var,
  sex = sex,
  fun = fun
)
```

Arguments

`gagebrth`
: the name of the "gestational age at birth in days" variable.

`z`
: z-score(s) to convert (must be between 0 and 1). Default value is 0. This value is used only if `fun` is set to "igb_zscore2value".

p	centile(s) to convert (must be between 0 and 100). Default value is p=50. This value is used only if fun is set to "igb_centile2value".
val	the name of the anthropometric variable to convert.
var	the name of the measurement to convert ("lencm", "wtkg", "hcircm", "wlr")
sex	the name of the sex factor variable. The variable should be coded as Male/Female. If it is coded differently (e.g. 0/1), then you can use the ds.recodeValues function to recode the categories to Male/Female before the use of ds.igb_standards
fun	the name of the function to be used. This can be one of: "igb_centile2value", "igb_zscore2value", "igb_value2zscore" (default), "igb_value2centile".

Value

assigns the converted measurement as a new object on the server-side

Note

For gestational ages between 24 and 33 weeks, the INTERGROWTH very early preterm standard is used.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

isNaDS *Checks if a vector is empty*

Description

this function is similar to R function `is.na` but instead of a vector of booleans it returns just one boolean to tell if all the element are missing values.

Usage

```
isNaDS(xvect)
```

Arguments

xvect	a numerical or character vector

Value

the integer '1' if the vector contains on NAs and '0' otherwise

Author(s)

Gaye, A.

isValidDS *Checks if an input is valid*

Description

Tells if an object on the server side is valid.

Usage

```
isValidDS(obj)
```

Arguments

obj	a vector (numeric, integer, factor, character), data.frame or matrix

Details

This function checks if an object is valid.

Value

a boolean, TRUE if input is valid or FALSE if not.

Author(s)

Gaye, A.

kurtosisDS1 *Calculates the kurtosis of a numeric variable*

Description

This function calculates the kurtosis of a numeric variable for each study separately.

Usage

```
kurtosisDS1(x, method)
```

Arguments

x	a string character, the name of a numeric variable.
method	an integer between 1 and 3 selecting one of the algorithms for computing kurtosis detailed in the headers of the client-side ds.kurtosis function.

Details

The function calculates the kurtosis of an input variable x with three different methods. The method is specified by the argument `method` in the client-side `ds.kurtosis` function.

Value

a list including the kurtosis of the input numeric variable, the number of valid observations and the study-side validity message.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

---

kurtosisDS2 *Calculates the kurtosis of a numeric variable*

Description

This function calculates summary statistics that are returned to the client-side and used for the estimation of the combined kurtosis of a numeric variable across all studies.

Usage

```
kurtosisDS2(x, global.mean)
```

Arguments

`x`	a string character, the name of a numeric variable.
`global.mean`	a numeric, the combined mean of the input variable across all studies.

Details

The function calculates the sum of squared differences between the values of x and the global mean of x across all studies, the sum of quatric differences between the values of x and the global mean of x across all studies and the number of valid observations of the input variable x.

Value

a list including the sum of quartic differences between the values of x and the global mean of x across all studies, the sum of squared differences between the values of x and the global mean of x across all studies, the number of valid observations (i.e. the length of x after excluding missing values), and a validity message indicating indicating a valid analysis if the number of valid observations are above the protection filter nfilter.tab or invalid analysis otherwise.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

lengthDS *Returns the length of a vector or list*

Description

This function is similar to R function `length`.

Usage

```
lengthDS(x)
```

Arguments

x	a string character, the name of a vector or list

Details

The function returns the length of the input vector or list.

Value

a numeric, the number of elements of the input vector or list.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

levelsDS *Returns the levels of a factor vector*

Description

This function is similar to R function `levels`.

Usage

```
levelsDS(x)
```

Arguments

x	a factor vector

Details

The function returns the levels of the input vector or list.

Value

a list, the factor levels present in the vector

Author(s)

Alex Westerberg, for DataSHIELD Development Team

`lexisDS1` *lexisDS1*

Description

The first server-side function called by ds.lexis.

Usage

```
lexisDS1(exitCol = NULL)
```

Arguments

`exitCol`	a character string specifying the variable holding the time that each individual is censored or fails

Details

This is an aggregate function. For more details see the extensive header for ds.lexis.

Value

List with ‘max.time‘

Author(s)

Burton PR

lexisDS2 *lexisDS2*

Description

The second serverside function called by ds.lexis.

Usage

```
lexisDS2(
  datatext = NULL,
  intervalWidth,
  maxmaxtime,
  idCol,
  entryCol,
  exitCol,
  statusCol,
  vartext = NULL
)
```

Arguments

datatext	a clientside provided character string specifying the data.frame holding the data set to be expanded
intervalWidth	a clientside generated character string specifying the width of the survival epochs in the expanded data
maxmaxtime	a clientside generated object specifying the maximum follow up time in any of the sources
idCol	a clientside generated character string specifying the variable holding the IDs of individuals in the data set to be expanded
entryCol	a clientside specified character string identifying the variable holding the time that each individual starts follow up
exitCol	a clientside specified character string identifying the variable holding the time that each individual ends follow up (is censored or fails)
statusCol	a clientside specified character string identifying the variable holding the final censoring status (failed/censored)
vartext	is a clientside provided vector of character strings denoting the column names of additional variables to include in the final expanded table. If the 'variables' argument is not set (is null) but the 'data' argument is set the full data.frame will be expanded and carried forward

Details

This is the assign function which actually creates the expanded dataframe containing survival data for a piecewise exponential regression. lexisDS2 also carries out a series of disclosure checks and if the arguments or data fail any of those tests, creation of the expanded dataframe is blocked and an appropriate serverside error message is stored. For more details see the extensive header for ds.lexis.

Value

List with ‘expanded.table‘

Author(s)

Burton PR

`lexisDS3` *@title lexisDS3*

Description

The third serverside function called by ds.lexis.

Usage

```
lexisDS3()
```

Details

This is an assign function that simplifies the returned output from ds.lexis. Specifically, without lexisDS3 the output consists of a table within a list, but lexisDS3 converts this directly into a dataframe. For more details see the extensive header for ds.lexis.

Value

Data frame with ‘messageobj‘ object

listDisclosureSettingsDS *listDisclosureSettingsDS*

Description

This serverside function is an aggregate function that is called by the ds.listDisclosureSettings

Usage

```
listDisclosureSettingsDS()
```

Details

For more details see the extensive header for ds.listDisclosureSettings

Value

List with DataSHIELD disclosure settings

Author(s)

Paul Burton, Demetris Avraam for DataSHIELD Development Team

listDS *Coerce objects into a list*

Description

this function is similar to R function 'list'

Usage

```
listDS(input = NULL, eltnames = NULL)
```

Arguments

input	a list of objects to coerce into a list
eltnames	a character list, the names of the elements in the list.

Details

Unlike the R function 'list' it takes also a vector of characters, the names of the elements in the output list.

Value

a list

Author(s)

Gaye, A.

`lmerSLMADS.assign` *Fitting linear mixed effect models - serverside function*

Description

lmerSLMADS.assign is the same as lmerSLMADS2 which fits a linear mixed effects model (lme) per study and saves the outcomes in each study

Usage

```
lmerSLMADS.assign(
  formula,
  offset,
  weights,
  dataName,
  REML = TRUE,
  control_type,
  control_value.transmit,
  optimizer,
  verbose = 0
)
```

Arguments

`formula`	see help for ds.lmerSLMA
`offset`	see help for ds.lmerSLMA
`weights`	see help for ds.lmerSLMA
`dataName`	see help for ds.lmerSLMA
`REML`	see help for ds.lmerSLMA
`control_type`	see help for ds.lmerSLMA
`control_value.transmit`	see help for argument <control_value> for function ds.lmerSLMA
`optimizer`	see help for ds.lmerSLMA
`verbose`	see help for ds.lmerSLMA

Details

lmerSLMADS.assign is a serverside function called by ds.lmerSLMA on the clientside. The analytic work engine is the lmer function in R which sits in the lme4 package. lmerSLMADS.assign fits a linear mixed effects model (lme) including both fixed and random effects - on data from each single data source and saves the regression outcomes on the serverside.

Value

writes lmerMod object summarising the fitted model to the serverside. For more detailed information see help for ds.lmerSLMA.

Author(s)

TDemetris Avraam for DataSHIELD Development Team

`lmerSLMADS2` *Fitting linear mixed effect models - serverside function*

Description

lmerSLMADS2 is a serverside function which fits a linear mixed effects model (lme) - i.e. can include both fixed and random effects - on data from one or multiple sources with pooling via SLMA (study level meta-analysis)

Usage

```
lmerSLMADS2(
  formula,
  offset,
  weights,
  dataName,
  REML = TRUE,
  control_type,
  control_value.transmit,
  optimizer,
  verbose = 0
)
```

Arguments

`formula`	see help for ds.lmerSLMA
`offset`	see help for ds.lmerSLMA
`weights`	see help for ds.lmerSLMA
`dataName`	see help for ds.lmerSLMA
`REML`	see help for ds.lmerSLMA

control_type	see help for ds.lmerSLMA
control_value.transmit	see help for argument <control_value> for function ds.lmerSLMA
optimizer	see help for ds.lmerSLMA
verbose	see help for ds.lmerSLMA

Details

lmerSLMADS2 is a serverside function called by ds.lmerSLMA on the clientside. The analytic work engine is the lmer function in R which sits in the lme4 package. ds.lmerSLMA fits a linear mixed effects model (lme) - can include both fixed and random effects - on data from a single or multiple sources. When there are multiple data sources, the lme is fitted to convergence in each data source independently and the estimates and standard errors returned to the client thereby enabling cross-study pooling using study level meta-analysis (SLMA). By default the SLMA is undertaken using the metafor package, but as the SLMA occurs on the clientside which, as far as the user is concerned is just a standard R environment, the user can choose to use any approach to meta-analysis they choose. For more detailed help about any aspect of lmerSLMDS2 please see the extensive help for ds.lmerSLMA. Additional information about fitting lmes using the lmer engine can be obtained using R help for lmer and the lme4 package

Value

all key model components see help for ds.lmerSLMA

Author(s)

Tom Bishop, with some additions by Paul Burton

---

lsDS *lists all objects on a serverside environment*

Description

creates a list of the names of all of the objects in a specified serverside environment

Usage

```
lsDS(search.filter = NULL, env.to.search)
```

Arguments

search.filter	either NULL or a character string (potentially including '*' wildcards) specifying required search criteria. This argument is fully specified by its corresponding argument in the clientside function.
env.to.search	integer (e.g. in a format such as '2' or '5L' format) specifying the position in the search path of the environment to be explored. This argument is fully specified by its corresponding argument in the clientside function.

Details

Serverside aggregate function lsDS called by clientside function ds.ls. When running analyses one may want to know the objects already generated. This request is not disclosive as it only returns the names of the objects and not their contents. By default, objects in the current 'active analytic environment' (".GlobalEnv") will be displayed. This is the environment that contains all of the objects that serverside DataSHIELD is using for the main analysis or has written out to the serverside during the process of managing or undertaking the analysis (variables, scalars, matrices, data.frames etc). For further details see help for ds.ls function and for native R function ls

Value

a list containing: (1) the name/details of the serverside R environment which ds.ls has searched; (2) a vector of character strings giving the names of all objects meeting the naming criteria specified by the argument <search.filter> in this specified R serverside environment; (3) the nature of the search filter string as it was actually applied

Author(s)

Gaye, A (2015). Updated and extended by Paul Burton (2020).

---

lsplineDS *Basis for a piecewise linear spline with meaningful coefficients*

Description

This function is based on the native R function lspline from the lspline package. This function computes the basis of piecewise-linear spline such that, depending on the argument marginal, the coefficients can be interpreted as (1) slopes of consecutive spline segments, or (2) slope change at consecutive knots.

Usage

```
lsplineDS(x = x, knots = NULL, marginal = FALSE, names = NULL)
```

Arguments

x	the name of the input numeric variable
knots	numeric vector of knot positions
marginal	logical, how to parametrize the spline, see Details
names	character, vector of names for constructed variables

Details

If marginal is FALSE (default) the coefficients of the spline correspond to slopes of the consecutive segments. If it is TRUE the first coefficient correspond to the slope of the first segment. The consecutive coefficients correspond to the change in slope as compared to the previous segment.

Value

an object of class "lspline" and "matrix", which its name is specified by the newobj argument (or its default name "lspline.newobj"), is assigned on the serverside.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

matrixDetDS1 *matrixDetDS aggregate function called by ds.matrixDet.report*

Description

Calculates the determinant of a square matrix A and returns the output to the clientside

Usage

```
matrixDetDS1(M1.name = NULL, logarithm)
```

Arguments

M1.name	A character string specifying the name of the matrix for which determinant to be calculated
logarithm	logical. Default is FALSE, which returns the determinant itself, TRUE returns the logarithm of the modulus of the determinant.

Details

Calculates the determinant of a square matrix (for additional information see help for det function in native R). This operation is only possible if the number of columns and rows of A are the same.

Value

Output is the determinant of the matrix identified by argument <M1> which is returned to the clientside. For more details see help for ds.matrixDet

Author(s)

Paul Burton for DataSHIELD Development Team

matrixDetDS2 *matrixDetDS assign function called by ds.matrixDet*

Description

Calculates the determinant of a square matrix A and writes the output to the serverside

Usage

```
matrixDetDS2(M1.name = NULL, logarithm)
```

Arguments

M1.name	A character string specifying the name of the matrix for which determinant to be calculated
logarithm	logical. Default is FALSE, which returns the determinant itself, TRUE returns the logarithm of the modulus of the determinant.

Details

Calculates the determinant of a square matrix (for additional information see help for det function in native R). This operation is only possible if the number of columns and rows of A are the same.

Value

Output is the determinant of the matrix identified by argument <M1> which is written to the serverside. For more details see help for ds.matrixDet

Author(s)

Paul Burton for DataSHIELD Development Team

matrixDiagDS *matrixDiagDS assign function called by ds.matrixDiag*

Description

Extracts the diagonal vector from a square matrix A or creates a diagonal matrix A based on a vector or a scalar value and writes the output to the serverside

Usage

```
matrixDiagDS(x1.transmit, aim, nrows.transmit)
```

Arguments

x1.transmit	identifies the input matrix or vector. Fully specified by <x1> argument of ds.matrixDiag. For more details see help for ds.matrixDiag.
aim	a character string specifying what behaviour is required of the function. Fully specified by <aim> argument of ds.matrixDiag. For more details see help for ds.matrixDiag.
nrows.transmit	a scalar value forcing the number of rows and columns in an output matrix.Fully specified by <nrows.scalar> argument of ds.matrixDiag. For more details see help for ds.matrixDiag.

Details

For details see help for function ds.matrixDiag.

Value

Output is the matrix or vector specified by the <newobj> argument (or default name diag_<x1>) which is written to the serverside. For more details see help for ds.matrixDiag.

Author(s)

Paul Burton for DataSHIELD Development Team

---

matrixDimnamesDS *matrixDimnamesDS assign function called by ds.matrixDimnames*

Description

Adds dimnames (row names, column names or both) to a matrix on the serverside.

Usage

```
matrixDimnamesDS(M1.name = NULL, dimnames)
```

Arguments

M1.name	Specifies the name of the serverside matrix to which dimnames are to be added. Fully specified by <M1> argument of function ds.matrixDimnames. For more details see help for ds.matrixDimnames.
dimnames	A dimnames attribute for the matrix: NULL or a list of length 2 giving the row and column names respectively. Fully specified by <dimnames> argument of function ds.matrixDimnames. For more details see help for ds.matrixDimnames.

Details

Adds dimnames (row names, column names or both) to a matrix on the serverside. Similar to the dimnames function in native R. For more details see help for function ds.matrixDimnames

Value

Output is the serverside matrix specified by the <newobj> argument (or default name diag_<x1>) with specified dimnames (row and column names) which is written to the serverside.

Author(s)

Paul Burton for DataSHIELD Development Team

`matrixDS` *matrixDS assign function called by ds.matrix*

Description

Creates a matrix A on the serverside

Usage

```
matrixDS(mdata.transmit, from, nrows.transmit, ncols.transmit, byrow, dimnames)
```

Arguments

`mdata.transmit`	specifies the elements of the matrix to be created. Fully specified by <mdata> argument of ds.matrix
`from`	a character string specifying the source and nature of <mdata>. Fully specified by <from> argument of ds.matrix
`nrows.transmit`	specifies the number of rows in the matrix to be created. Fully specified by <nrows.scalar> argument of ds.matrix
`ncols.transmit`	specifies the number of columns in the matrix to be created. Fully specified by <ncols.scalar> argument of ds.matrix
`byrow`	a logical value specifying whether, when <mdata> is a vector, the matrix created should be filled row by row or column by column. Fully specified by <byrow> argument of ds.matrix
`dimnames`	A dimnames attribute for the matrix: NULL or a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names only. Fully specified by <dimnames> argument of ds.matrix

Details

Similar to the `matrix()` function in native R. Creates a matrix with dimensions specified by <nrows.scalar> and <ncols.scalar> arguments and assigns the values of all its elements based on the <mdata> argument

Value

Output is the matrix A written to the serverside. For more details see help for ds.matrix

Author(s)

Paul Burton for DataSHIELD Development Team

`matrixInvertDS` *matrixInvertDS serverside assign function called by ds.matrixInvert*

Description

Inverts a square matrix A and writes the output to the serverside

Usage

```
matrixInvertDS(M1.name = NULL)
```

Arguments

`M1.name`	A character string specifying the name of the matrix to be inverted

Details

Undertakes standard matrix inversion. This operation is only possible if the number of columns and rows of A are the same and the matrix is non-singular - positive definite (eg there is no row or column that is all zeros)

Value

Output is the matrix representing the inverse of A which is written to the serverside. For more details see help for ds.matrixInvert

Author(s)

Paul Burton for DataSHIELD Development Team

`matrixMultDS` *matrixMultDS serverside assign function called by ds.matrixMult*

Description

Calculates the matrix product of two matrices and writes output to serverside

Usage

```
matrixMultDS(M1.name = NULL, M2.name = NULL)
```

Arguments

M1.name	A character string specifying the name of the first matrix (M1) argument specified by the M1 argument in the original call to ds.matrixMult
M2.name	A character string specifying the name of the second matrix (M2) argument specified by the M1 argument in the original call to ds.matrixMult

Details

Undertakes standard matrix multiplication where with input matrices A and B with dimensions A: mxn and B: nxp the output C has dimensions mxp and each element C[i,j] has value equal to the dot product of row i of A and column j of B where the dot product is obtained as sum(A[i,1]*B[1,j] + A[i,2]*B[2,j] + ... + A[i,n]*B[n,j]). This calculation is only valid if the number of columns of A is the same as the number of rows of B

Value

Output is the matrix representing the product of M1 and M2 which is written to the serverside. For more details see help for ds.matrixMult

Author(s)

Paul Burton for DataSHIELD Development Team

---

matrixTransposeDS *matrixTransposeDS serverside assign function called by ds.matrixTranspose*

Description

Transposes a matrix A and writes the output to the serverside

Usage

```
matrixTransposeDS(M1.name = NULL)
```

Arguments

M1.name	A character string specifying the name of the matrix to be transposed

Details

Undertakes standard matrix transposition. This operation converts matrix A to matrix C where element C[i,j] of matrix C equals element A[j,i] of matrix A. Matrix A therefore has the same number of rows as matrix C has columns and vice versa.

Value

Output is the matrix representing the transpose of A which is written to the serverside. For more details see help for ds.matrixTranspose

Author(s)

Paul Burton for DataSHIELD Development Team

`mdPatternDS` *Missing data pattern with disclosure control*

Description

This function is a serverside aggregate function that computes the missing data pattern using mice::md.pattern and applies disclosure control to prevent revealing small cell counts.

Usage

```
mdPatternDS(x)
```

Arguments

`x`	a character string specifying the name of a data frame or matrix containing the data to analyze for missing patterns.

Details

This function calls the mice::md.pattern function to generate a matrix showing the missing data patterns in the input data. To ensure disclosure control, any pattern counts that are below the threshold (nfilter.tab, default=3) are suppressed.

Suppression Method:

When a pattern count is below threshold: - Row name is changed to "suppressed(<N>)" where N is the threshold - All pattern values in that row are set to NA - Summary row is also set to NA (prevents back-calculation)

Output Matrix Structure:

- Rows represent different missing data patterns (plus a summary row at the bottom) - Row names contain pattern counts (or "suppressed(<N>)" for invalid patterns) - Columns show 1 if variable is observed, 0 if missing - Last column shows total number of missing values per pattern - Last row shows total number of missing values per variable

Note for Pooling:

When this function is called from ds.mdPattern with type='combine', suppressed patterns are excluded from pooling to prevent disclosure through subtraction. This means pooled counts may underestimate the true total when patterns are suppressed in some studies.

Value

A list containing:

pattern	The missing data pattern matrix with disclosure control applied
valid	Logical indicating if all patterns meet disclosure requirements
message	A message describing the validity status

Author(s)

Xavier Escribà montagut for DataSHIELD Development Team

---

meanDS *Computes statistical mean of a vector*

Description

Calculates the mean value.

Usage

```
meanDS(xvect)
```

Arguments

xvect	a vector

Details

if the length of input vector is less than the set filter a missing value is returned.

Value

a numeric, the statistical mean

Author(s)

Gaye A, Burton PR

meanSdGpDS *MeanSdGpDS*

Description

Server-side function called by ds.meanSdGp

Usage

```
meanSdGpDS(X, INDEX)
```

Arguments

X	a client-side supplied character string identifying the variable for which means/SDs are to be calculated
INDEX	a client-side supplied character string identifying the factor across which means/SDs are to be calculated

Details

Computes the mean and standard deviation across groups defined by one factor

Value

List with results from the group statistics

Author(s)

Burton PR

mergeDS *mergeDS (assign function) called by ds.merge*

Description

merges (links) two data.frames together based on common values in defined vectors in each data.frame

Usage

```
mergeDS(
  x.name,
  y.name,
  by.x.names.transmit,
  by.y.names.transmit,
  all.x,
  all.y,
  sort,
  suffixes.transmit,
  no.dups,
  incomparables
)
```

Arguments

x.name	the name of the first data.frame to be merged specified in inverted commas. Specified via argument <x.name> of ds.merge function
y.name	the name of the second data.frame to be merged specified in inverted commas. Specified via argument <y.name> of ds.merge function
by.x.names.transmit	the name of a single variable or a vector of names of multiple variables (in transmittable form) containing the IDs or other data on which data.frame x is to be merged/linked to data.frame y. Specified via argument <by.x.names> of ds.merge function
by.y.names.transmit	the name of a single variable or a vector of names of multiple variables (in transmittable form) containing the IDs or other data on which data.frame y is to be merged/linked to data.frame x. Specified via argument <by.y.names> of ds.merge function
all.x	logical, if TRUE, then extra rows will be added to the output, one for each row in x that has no matching row in y. Specified via argument <all.x> of ds.merge function. Default = FALSE.
all.y	logical, if TRUE, then extra rows will be added to the output, one for each row in y that has no matching row in x. Specified via argument <all.y> of ds.merge function. Default = FALSE.
sort	logical, if TRUE the merged result should be sorted on elements in the by.x.names and by.y.names columns. Specified via argument <sort> of ds.merge function. Default = TRUE.
suffixes.transmit	a character vector of length 2 (in transmittable form) specifying the suffixes to be used for making unique common column names in the two input data.frames when they both appear in the merged data.frame. Specified via argument <suffixes> of ds.merge function. Default '.x' and '.y'.
no.dups	logical, when TRUE suffixes are appended in more cases to rigorously avoid duplicated column names in the merged data.frame. Specified via argument

<no.dups> of ds.merge function. Default TRUE but was apparently implicitly FALSE before R version 3.5.0.

incomparables values intended for merging on one column which cannot be matched. See 'match' in help for Native R merge function. Specified via argument <incomparables> of ds.merge

Details

For further information see details of the native R function merge and the DataSHIELD clientside function ds.merge.

Value

the merged data.frame specified by the <newobj> argument of ds.merge (or by default 'x.name_y.name' if the <newobj> argument is NULL) which is written to the serverside. In addition, two validity messages are returned to the clientside indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study there may be a studysideMessage that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the ds.message function. If you type ds.message(<newobj>) it will print out the relevant studysideMessage from any datasource in which there was an error in creating <newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message(<newobj>) will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

Paul Burton, Demetris Avraam, for DataSHIELD Development Team

---

messageDS *messageDS*

Description

This function allows for error messages arising from the running of a server-side assign function to be returned to the client-side

Usage

```
messageDS(message.object.name)
```

Arguments

message.object.name
is a character string, containing the name of the list containing the message. See the header of the client-side function ds.message for more details.

Details

Errors arising from aggregate server-side functions can be returned directly to the client-side. But this is not possible for server-side assign functions because they are designed specifically to write objects to the server-side and to return no meaningful information to the client-side. Otherwise, users may be able to use assign functions to return disclosive output to the client-side. ds.message calls messageDS which looks specifically for an object called $serversideMessage in a designated list on the server-side. Server-side functions from which error messages are to be made available, are designed to be able to write the designated error message to the $serversideMessage object into the list that is saved on the server-side as the primary output of that function. So only valid server-side functions of DataSHIELD can write a $studysideMessage, and as additional protection against unexpected ways that someone may try to get round this limitation, a $studysideMessage is a string that cannot exceed a length of nfilter.string a default of 80 characters.

Value

a list object from each study, containing whatever message has been written by DataSHIELD into $studysideMessage.

Author(s)

Burton PR

---

metadataDS *Returns the metadata, if any, about the specified variable*

Description

This function returns metadata, if any, about specified variable.

Usage

```
metadataDS(x)
```

Arguments

x	a string character, containing the name of the specified variable

Details

The function returns the metadata, obtained from attributes function.

Value

a list containing the metadata. The elements of the list will depend on the metadata available.

Author(s)

Stuart Wheater, for DataSHIELD Development Team

`miceDS` *Aggregate function called by ds.mice*

Description

This function is a wrapper function of the mice from the mice R package. The function creates multiple imputations (replacement values) for multivariate missing data. The method is based on Fully Conditional Specification, where each incomplete variable is imputed by a separate model. The MICE algorithm can impute mixes of continuous, binary, unordered categorical and ordered categorical data. In addition, MICE can impute continuous two-level data, and maintain consistency between imputations by means of passive imputation.

Usage

```
miceDS(
  data = data,
  m = m,
  maxit = maxit,
  method = method,
  post = post,
  seed = seed,
  predictorMatrix = predictorMatrix,
  ncol.pred.mat = ncol.pred.mat,
  newobj_mids = newobj_mids,
  newobj_df = newobj_df
)
```

Arguments

`data`	a data frame or a matrix containing the incomplete data.
`m`	Number of multiple imputations. The default is m=5. The maximum allowed number in DataSHIELD is m=20.
`maxit`	A scalar giving the number of iterations. The default is 5. The maximum allowed number in DataSHIELD is maxit=30.
`method`	Can be either a single string, or a vector of strings with length ncol(data), specifying the imputation method to be used for each column in data. If specified as a single string, the same method will be used for all blocks. The default imputation method (when no argument is specified) depends on the measurement level of the target column, as regulated by the defaultMethod argument in native R mice function. Columns that need not be imputed have the empty method "".
`post`	A vector of strings with length ncol(data) specifying expressions as strings. Each string is parsed and executed within the sampler() function to post-process imputed values during the iterations. The default is a vector of empty strings, indicating no post-processing. Multivariate (block) imputation methods ignore the post parameter.

seed	either NA (default) or "fixed". If seed is set to "fixed" then a fixed seed random number generator which is study-specific is used.
predictorMatrix	A numeric matrix of ncol(data) rows and ncol(data) columns, containing 0/1 data specifying the set of predictors to be used for each target column. Each row corresponds to a variable to be imputed. A value of 1 means that the column variable is used as a predictor for the target variables (in the rows). By default, the predictorMatrix is a square matrix of ncol(data) rows and columns with all 1's, except for the diagonal.
ncol.pred.mat	the number of columns of the predictorMatrix.
newobj_mids	a character string that provides the name for the output mids object that is stored on the data servers. Default mids_object.
newobj_df	a character string that provides the name for the output dataframes that are stored on the data servers. Default imputationSet. For example, if m=5, and newobj_df="imputationSet", then five imputed dataframes are saved on the servers with names imputationSet.1, imputationSet.2, imputationSet.3, imputationSet.4, imputationSet.5.

Details

For additional details see the help header of mice function in native R mice package.

Value

a list with three elements: the method, the predictorMatrix and the post. The function also saves in each server the mids object and all completed datasets as dataframes.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

minMaxRandDS *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

Creates a minimum value that is more negative, and less positive than any real value in V2BR and a maximum value that is more positive and less negative than any value of V2BR.

Usage

```
minMaxRandDS(input.var.name)
```

Arguments

input.var.name	a character string specifying the name of V2BR. This argument is set by the argument with the same name in the clientside function ds.ranksSecure

Details

Severside aggregate function called by ds.ranksSecure. The minimum and maximum values it creates are used to replace missing values (NAs) in V2BR if the argument <NA.manag>e is set to "NA.low" or "NA.hi" respectively. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Value

the data frame objects containing the global ranks and quantiles. For more details see the associated document entitled "secure.global.ranking.docx"

Author(s)

Paul Burton 9th November, 2021

---

namesDS *Return the names of a list object*

Description

Returns the names of a designated server-side list

Usage

```
namesDS(xname.transmit)
```

Arguments

`xname.transmit`	a character string specifying the name of the list.

Details

namesDS is an aggregate function called by ds.names. This function is similar to the native R function `names` but it does not subsume all functionality, for example, it only works to extract names that already exist, not to create new names for objects. The function is restricted to objects of type list, but this includes objects that have a primary class other than list but which return TRUE to the native R function `is.list`. As an example, this includes the multi-component object created by fitting a generalized linear model using ds.glmSLMA. The resultant object saved on each separate server is formally of double class "glm" and "ls" but responds TRUE to is.list(),

Value

`namesDS` returns to the client-side the names of a list object stored on the server-side.

Author(s)

Amadou Gaye, updated by Paul Burton 25/06/2020

nsDS *Generate a Basis Matrix for Natural Cubic Splines*

Description

This function is based on the native R function ns from the splines package. This function generate the B-spline basis matrix for a natural cubic spline.

Usage

```
nsDS(x, df, knots, intercept, Boundary.knots)
```

Arguments

x	the predictor variable. Missing values are allowed.
df	degrees of freedom. One can supply df rather than knots; ns() then chooses df - 1 - intercept knots at suitably chosen quantiles of x (which will ignore missing values). The default, df = NULL, sets the number of inner knots as length(knots).
knots	breakpoints that define the spline. The default is no knots; together with the natural boundary conditions this results in a basis for linear regression on x. Typical values are the mean or median for one knot, quantiles for more knots. See also Boundary.knots.
intercept	if TRUE, an intercept is included in the basis; default is FALSE.
Boundary.knots	boundary points at which to impose the natural boundary conditions and anchor the B-spline basis (default the range of the data). If both knots and Boundary.knots are supplied, the basis parameters do not depend on x. Data can extend beyond Boundary.knots

Details

ns is native R is based on the function splineDesign. It generates a basis matrix for representing the family of piecewise-cubic splines with the specified sequence of interior knots, and the natural boundary conditions. These enforce the constraint that the function is linear beyond the boundary knots, which can either be supplied or default to the extremes of the data. A primary use is in modeling formula to directly specify a natural spline term in a model.

Value

A matrix of dimension length(x) * df where either df was supplied or if knots were supplied, df = length(knots) + 1 + intercept. Attributes are returned that correspond to the arguments to ns, and explicitly give the knots, Boundary.knots etc for use by predict.ns(). The object is assigned at each serverside.

Author(s)

Demetris Avraam for DataSHIELD Development Team

numNaDS *Counts the number of missing values*

Description

this function just counts the number of missing entries in a vector.

Usage

```
numNaDS(xvect)
```

Arguments

xvect	a vector

Value

an integer, the number of missing values

Author(s)

Gaye, A.

qlsplineDS *Basis for a piecewise linear spline with meaningful coefficients*

Description

This function is based on the native R function `qlspline` from the `lspline` package. This function computes the basis of piecewise-linear spline such that, depending on the argument marginal, the coefficients can be interpreted as (1) slopes of consecutive spline segments, or (2) slope change at consecutive knots.

Usage

```
qlsplineDS(x = x, q = q, na.rm = TRUE, marginal = FALSE, names = NULL)
```

Arguments

x	the name of the input numeric variable
q	numeric, a single scalar greater or equal to 2 for a number of equal-frequency intervals along x or a vector of numbers in (0; 1) specifying the quantiles explicitly.
na.rm	logical, whether NA should be removed when calculating quantiles, passed to na.rm of quantile. Default set to TRUE.
marginal	logical, how to parametrize the spline, see Details
names	character, vector of names for constructed variables

Details

If marginal is FALSE (default) the coefficients of the spline correspond to slopes of the consecutive segments. If it is TRUE the first coefficient correspond to the slope of the first segment. The consecutive coefficients correspond to the change in slope as compared to the previous segment. Function qlspline wraps lspline and calculates the knot positions to be at quantiles of x. If q is a numerical scalar greater or equal to 2, the quantiles are computed at seq(0, 1, length.out = q + 1)[-c(1, q+1)], i.e. knots are at q-tiles of the distribution of x. Alternatively, q can be a vector of values in [0; 1] specifying the quantile probabilities directly (the vector is passed to argument probs of quantile).

Value

an object of class "lspline" and "matrix", which its name is specified by the `newobj` argument (or its default name "qlspline.newobj"), is assigned on the serverside.

Author(s)

Demetris Avraam for DataSHIELD Development Team

`quantileMeanDS` *Generates quantiles and mean information without maximum and minimum*

Description

the probabilities 5 are used to compute the corresponding quantiles.

Usage

```
quantileMeanDS(xvect)
```

Arguments

`xvect`	a numerical vector

Value

a numeric vector that represents the sample quantiles

Author(s)

Burton, P.; Gaye, A.

rangeDS *returns the minimum and maximum of a numeric vector*

Description

this function is similar to R function range but instead to not return the real minimum and maximum, the computed values are multiplied by a very small random number.

Usage

```
rangeDS(xvect)
```

Arguments

xvect	a numerical

Value

a numeric vector which contains the minimum and the maximum values of the vector

Author(s)

Amadou Gaye, Demetris Avraam for DataSHIELD Development Team

ranksSecureDS1 *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

takes key non-disclosive components of the serverside data frame blackbox.output.df over to the clientside to enable global ranking.

Usage

```
ranksSecureDS1()
```

Details

Severside aggregate function called by ds.ranksSecure. The non-disclosive components of blackbox.output.df that are transmitted to the clientside are: (1) final values of the "combined real+pseudo data vector" after all seven rounds of encryption have been completed; (2) a set of sequential IDs allocated after sorting the "combined real+pseudo data vector" by value (in ascending order). This allows later re-linkage of values back on the serverside and confirmation that that linkage is correct. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Value

the non-disclosive elements of blackbox.output.df (see details) on the serverside as a data frame object (called blackbox.output) on the clientside. After processing to create the global ranks across all studies, this is returned to the serverside as the data frame sR4.df using the clientside function ds.dmtC2S

Author(s)

Paul Burton 9th November, 2021

---

ranksSecureDS2 *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

Checks that the data frame produced in creating the initial global ranks (ranks based on real and pseudo-data after the running of blackBoxDS)has the correct dimensions and order as the serverside data frames to which it will now be appended. If either the number of rows or the order of the rows are inconsistent with the pre-existing data frames on the serverside an error message is returned and the processing stops. Then strips out the pseudo-data leaving solely the global ranks based just on the real data

Usage

```
ranksSecureDS2()
```

Details

Severside assign function called by ds.ranksSecure. It works on the on the output created by serverside function ranksSecureDS1 and saved on the serverside in data frame sR4.df by ds.dmtC2S. Having checked QA it strips out all rows corresponding to pseudo-data. The resultant data frame contains the following vectors: (1) the fully encrypted V2BR (after application of blackBoxDS);(2) "ID.by.val" the sequential ID associated with the "combined real+pseudo data vector" sorted by value (ascending); (3) "studyid", a vector consisting solely of value n in the nth study; (4) "global.rank" the vector containing global ranks created by the clientside code in ds.ranksSecure after ranksSecureDS1 is called and up to the point where ds.dmtC2S sends sR4.df to the serverside. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Value

creates a new data frame sR5.df on the serverside containing solely the real data and including key elements needed for next stage of the ranking process. Most crucially these include "global.rank" and "ID.by.val" sorted in ascending order of the magnitude of V2BR

Author(s)

Paul Burton 9th November, 2021

`ranksSecureDS3` *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

takes key non-disclosive components of the serverside data frame blackbox.ranks.df over to the clientside to enable global re-ranking of the global ranks just applying to the real data (not the pseudo-data).

Usage

```
ranksSecureDS3()
```

Details

Severside aggregate function called by ds.ranksSecure. The non-disclosive components of blackbox.ranks.df that are transmitted to the clientside are: (1) final values of the encrypted global ranks vector after all seven rounds of encryption have been completed; (2) a set of sequential IDs allocated to the global ranks vector in each study in their current order based on increasing value of V2BR. This allows later re-linkage of values back on the serverside and confirmation that that linkage is correct. (3) a studyid vector with all values n in the nth study. This facilitates data management on the serverside during the global ranking of global ranks. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Value

the non-disclosive elements of blackbox.output.df (see details) on the serverside as a data frame object (called sR6.df) on the clientside. After processing within ds.ranksSecure to create the global ranks and global quantiles (of real data only) across all studies, this is returned to the serverside as data frame "global.ranks.quantiles.df" using the clientside function ds.dmtC2S. To illustrate the difference between ranks and quantiles, if there are a total of 1000 original real observations across all studies and one particular observation has the rank 250, it will have quantile value 0.25 (i.e. 25 increasing value). Both ranks and quantiles can have ties. For more details about the cluster of functions that collectively enable secure global ranking and estimation of global quantiles see the associated document entitled "secure.global.ranking.docx". Also see the header file for ds.ranksSecure

Author(s)

Paul Burton 9th November, 2021

`ranksSecureDS4` *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

Creates a data frame "sR8.df" by cbinding the data frame "blackBox.ranks.df" with the global ranks and global quantiles vectors in "global.ranks.quantiles.df". Performs QA on this matrix and orders the sR8.df data frame according to the argument <ranks.sort.by> in ds.ranksSecure

Usage

```
ranksSecureDS4(ranks.sort.by)
```

Arguments

`ranks.sort.by`	a character string taking two possible values. These are "ID.orig" and "vals.orig". These define the order in which the output.ranks.df and summary.output.ranks.df data frames are presented. This argument is set by the argument with the same name in ds.ranksSecure. Default value is "ID.orig".

Details

Severside assign function called by ds.ranksSecure. Creates a data frame "sR8.df" by cbinding the data frame "blackBox.ranks.df" with the global ranks and global quantiles vectors in "global.ranks.quantiles.df". Checks that all components of sR8.df have the correct dimensions and are consistent in their ordering. If either the number of rows or the order of the rows are inconsistent with those in "blackBox.ranks.df" an error message is returned and the processing stops. If sR8.df passes all QA tests it is written to the serverside as a data frame with its name identified by the argument <output.ranks.df> in ds.ranksSecure. If that argument is NULL or unspecified the data frame is called "main.ranks.df". The ranksSecureDS4 function also orders the combined data frame (<output.ranks.df>) in one of two ways: if the argument <ranks.sort.by> in ds.ranksSecure is set to "ID.orig" the combined data frame is ordered in the same way as the original V2BR vector; if the argument <ranks.sort.by> is set to vals.orig" the combined data frame is ordered by the magnitude of the values of V2BR (ascending). Having created the data frame (<output.ranks.df>) in this manner it can now be directly cbinded to either the V2BR vector itself or to a data frame, tibble or matrix containing V2BR (assuming they are also in the order corresponding to the argument <ranks.sort.by>) and this combined object can be used as the basis of analysis based on the global ranks or quantiles including a range of types of non-parametric analysis.

Value

Creates the data frame identified by the name given by the argument (<output.ranks.df>) of the ds.ranksSecure function and writes it to the serverside. If the argument <output.ranks.df> is NULL or unspecified the output data frame is called "main.ranks.df". The data frame is ordered according to the argument <ranks.sort.by> in ds.ranksSecure.

Author(s)

Paul Burton 9th November, 2021

`ranksSecureDS5` *Secure ranking of "V2BR" (vector to be ranked) across all sources*

Description

Summarises the serverside data frame written by ranksSecureDS4 which is identified by the name given by the argument (<output.ranks.df>) of the ds.ranksSecure function to produce a new output data frame containing only 5 key variables.

Usage

```
ranksSecureDS5(output.ranks.df)
```

Arguments

`output.ranks.df`
: a character string which specifies an optional name for the data.frame written to the serverside on each data source that contains 11 of the key output variables from the ranking procedure pertaining to that particular data source. This argument is set by the argument with the same name in ds.ranksSecure.

Details

Serverside assign function called by clientside function ds.ranksSecure. Takes the serverside data frame written by ranksSecureDS4 which is identified by the name given by the argument (<output.ranks.df>) of the ds.ranksSecure function. This holds 11 vectors including the final global ranks across all studies and final global quantiles. The data frame is ordered according to the argument <ranks.sort.by> in ds.ranksSecure. The ranksSecureDS5 function then extracts 5 key vectors from the larger data frame to produce a summary data frame that is given a name specified by the argument (<summary.output.ranks.df>) the ds.ranksSecure function. This data frame includes the following components: (1) The values of a sequential ID variable (ID.seq.real.orig) created to lie alongside the original V2BR vector in the same order as that vector was itself ordered. These ID values therefore reflect which row in the original data corresponds to a given row in the output. If the argument <ranks.sort.by> in ds.ranksSecure is set to "ID.orig" the values of the ID.seq.real.orig vector in the output data frame simply run sequentially from 1 to N where N is the number of individuals in the corresponding study. If <ranks.sort.by> is set to "vals.orig" the values of the ID.seq.real.orig vector will be determined by the magnitude of the corresponding V2BR value and will appear to be ordered in a haphazard manner; (2) the original values of V2BR; (3) the global ranks corresponding to the original values in V2BR, with ties reflected appropriately; (4) the global quantiles corresponding to the original values in V2BR, with ties reflected appropriately; (5) a studyid vector in which all elements take the value n in the nth study.

Value

extracts 5 key vectors from the larger data frame created by ranksSecureDS4 to produce a summary data frame that is written to the serverside. It is given a name specified by the argument.

Author(s)

Paul Burton 9th November, 2021

`rbindDS` *rbindDS called by ds.rbind*

Description

serverside assign function that takes a sequence of vector, matrix or data-frame arguments and combines them by row to produce a matrix.

Usage

```
rbindDS(x.names.transmit = NULL, colnames.transmit = NULL)
```

Arguments

`x.names.transmit`
: This is a vector of character strings representing the names of the elemental components to be combined converted into a transmittable format. This argument is fully specified by the <x> argument of `ds.rbind`

`colnames.transmit`
: This is NULL or a vector of character strings representing forced column names for the output object converted into a transmittable format. This argument is fully specified by the <force.colnames> argument of `ds.cbind`.

Details

A sequence of vector, matrix or data-frame arguments is combined row by row to produce a matrix which is written to the serverside. For more details see help for `ds.rbind` and the native R function `rbind`.

Value

the object specified by the <newobj> argument of `ds.rbind`(or default name <rbind.out>) which is written to the serverside. As well as writing the output object as <newobj> on the serverside, two validity messages are returned indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - ds.cbind() also returns any studysideMessages that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the `ds.message` function. If you type ds.message("<newobj>") it will print

out the relevant studysideMessage from any datasource in which there was an error in creating <newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message("<newobj>") will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

Paul Burton for DataSHIELD Development Team

`rBinomDS` *rBinomDS serverside assign function*

Description

primary serverside assign function called by ds.rBinom

Usage

```
rBinomDS(n, size = 1, prob = 0.5)
```

Arguments

`n`	length of the pseudorandom number vector to be generated as specified by the argument <samp.size> in the function ds.rBinom
`size`	a scalar that must be a positive integer. Value set directly by <size> argument of ds.rBinom - for details see help for ds.rBinom. May be a scalar or a vector allowing the size to vary from observation to observation.
`prob`	a numeric scalar in range 0 > prob > 1 which specifies the probability of a positive response. Value set directly by <prob> argument of ds.rBinom - for details see help for ds.rBinom May be a scalar or a vector allowing the size to vary from observation to observation.

Details

Generates the vector of pseudorandom numbers from a binomial distribution in each data source as specified by the arguments of ds.rBinom. This serverside function is effectively the same as the function rbinom() in native R and its arguments are the same.

Value

Writes the pseudorandom number vector with the characteristics specified in the function call as a new serverside vector on the data source on which it has been called. Also returns key information to the clientside: the random seed as specified by you in each source + (if requested) the full 626 length random seed vector this generated in each source (see info for the argument <return.full.seed.as.set>). It also returns a vector reporting the length of the pseudorandom vector created in each source.

Author(s)

Paul Burton for DataSHIELD Development Team

`recodeLevelsDS` *Recodes the levels of a categorical variables*

Description

The functions uses the input factor and generates a new factor with new levels.

Usage

```
recodeLevelsDS(x = NULL, classes = NULL)
```

Arguments

`x`	a factor vector
`classes`	a character vector the levels of the newt factor vector

Value

a factor vector with the new levels

Author(s)

Gaye, A.

`recodeValuesDS` *recodeValuesDS an assign function called by ds.recodeValues*

Description

This function recodes specified values of elements in a vector into a matched set of alternative specified values.

Usage

```
recodeValuesDS(
  var.name.text = NULL,
  values2replace.text = NULL,
  new.values.text = NULL,
  missing = NULL
)
```

Arguments

var.name.text	a character string providing the name for the vector representing the variable to be recoded. <var.name.text> argument generated and passed directly to recodeValuesDS by ds.recodeValues
values2replace.text	a character string specifying the values in the vector specified by the argument <var.name.text> that are to be replaced by new values as specified in the new.values.vector. The <values2replace.text> argument is generated and passed directly to recodeValuesDS by ds.recodeValues. In effect, the <values2replace.vector> argument of the ds.recodeValues function is converted to a character string format that is acceptable to the DataSHIELD R parser in the data repository and so can be accepted by recodeValuesDS
new.values.text	a character string specifying the new values to which the specified values in the vector <var.name> are to be converted. The <new.values.text> argument is generated and passed directly to recodeValuesDS by ds.recodeValues. In effect, the <new.values.vector> argument of the ds.recodeValues function is converted to a character string format that is acceptable to the DataSHIELD R parser in the data repository and so can be used in the call to recodeValuesDS.
missing	if supplied, any missing values in the variable referred to by var.name.text will be replaced by this value.

Details

For all details see the help header for ds.recodeValues

Value

the object specified by the <newobj> argument (or default name '<var.name>_recoded') initially specified in calling ds.recodeValues. The output object (the required recoded variable called <newobj> is written to the serverside.

Author(s)

Paul Burton, Demetris Avraam for DataSHIELD Development Team

---

repDS *repDS called by ds.rep*

Description

An assign function which creates a repetitive sequence by repeating an identified scalar, or specified elements of a vector or list. This is analogous to the rep function in native R. The sequence is written as a new object to the serverside

Usage

```
repDS(
  x1.transmit,
  times.transmit,
  length.out.transmit,
  each.transmit,
  x1.includes.characters,
  source.x1,
  source.times,
  source.length.out,
  source.each
)
```

Arguments

x1.transmit	This argument determines the input scalar, vector or list. for behaviour see help for ds.rep and "details from native R help for <rep>" (see above). This parameter is usually fully defined by the argument <x1> in the call to ds.rep that itself calls repDS.
times.transmit	This argument determines the number of replications and the pattern of these replications of the input scalar/vector to construct the output repetitive sequence. For behaviour see help for ds.rep and "details from native R help for <rep>" (see above). This parameter is usually fully defined by the argument <times> in the call to ds.rep that itself calls repDS.
length.out.transmit	This argument fixes the length of the output repetitive sequence vector For behaviour see help for ds.rep and "details from native R help for <rep>" (see above). This parameter is usually fully defined by the argument <length.out> in the call to ds.rep that itself calls repDS.
each.transmit	This argument specifies the number of replications of individual elements rather than replications of the full sequence. For behaviour see help for ds.rep and "details from native R help for <rep>" (see above). This parameter is usually fully defined by the argument <each> in the call to ds.rep that itself calls repDS.
x1.includes.characters	Boolean parameter determining whether to coerce the final output sequence to numeric. Defaults to FALSE and output is coerced to numeric. For detailed behaviour see help for ds.rep. This parameter is usually fully defined by the argument <x1.includes.characters> in the call to ds.rep that itself calls repDS.
source.x1	This defines the source of the scalar or vector defined by the <x1> argument. Four character strings are allowed: "clientside" or "c" and serverside or "s". For behaviour see help for ds.rep and "details from native R help for <rep>" (see above). This parameter is usually fully defined by the argument <source.x1> in the call to ds.rep that itself calls repDS.
source.times	see "param source.x1" This parameter is usually fully defined by the argument <source.times> in the call to ds.rep that itself calls repDS.

`source.length.out`
: see "param source.x1" This parameter is usually fully defined by the argument <source.length.out> in the call to `ds.rep` that itself calls `repDS`.

`source.each`
: see "param source.x1" This parameter is usually fully defined by the argument <source.each> in the call to `ds.rep` that itself calls `repDS`.

Details

Further details can be found in the help details for on ds.rep and the following aspects of the help for the function `rep` in native R also apply (as explained in more detail with exceptions identified in help for `ds.rep`):

In addition a Details from R help for <rep>:

The default behaviour is as if the call was rep(x, times = 1, length.out = NA, each = 1) Normally just one of the additional arguments is specified, but if 'each' is specified with either of the other two, its replication is performed first, and then that is followed by the replication implied by times or length.out.

If times consists of a single integer, the result consists of the whole input repeated this many times. If times is a vector of the same length as x (after replication by each), the result consists of x[1] repeated times[1] times, x[2] repeated times[2] times and so on. ***Note exception 1 above.

length.out may be given in place of times, in which case x is repeated as many times as is necessary to create a vector of this length. If both are given, length.out takes priority and times is ignored. ***Note exception 3 above.

Non-integer values of times will be truncated towards zero. If times is a computed quantity it is prudent to add a small fuzz or use round. And analogously for each.

Value

the vector containing the specified repetitive sequence and write to the output object defined by the <newobj> argument (or default name seq.vect) which is written to the serverside in each source. In addition, two validity messages are returned indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - ds.matrixDiag also returns any studysideMessages that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the `ds.message` function. If you type ds.message("newobj") it will print out the relevant studysideMessage from any datasource in which there was an error in creating <newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message("newobj") will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

Paul Burton for DataSHIELD Development Team, 14/10/2019

replaceNaDS *Replaces the missing values in a vector*

Description

This function identifies missing values and replaces them by a value or values specified by the analyst.

Usage

```
replaceNaDS(xvect, replacements)
```

Arguments

xvect	a character, the name of the vector to process.
replacements	a vector which contains the replacement value(s), a vector one or more values for each study.

Details

This function is used when the analyst prefer or requires complete vectors. It is then possible the specify one value for each missing value by first returning the number of missing values using the function numNaDS but in most cases it might be more sensible to replace all missing values by one specific value e.g. replace all missing values in a vector by the mean or median value. Once the missing values have been replaced a new vector is created.

Value

a new vector without missing values

Author(s)

Amadou Gaye, Demetris Avraam for DataSHIELD Development Team

reShapeDS *reShapeDS (assign function) called by ds.reShape*

Description

Reshapes a data frame containing longitudinal or otherwise grouped data from 'wide' to 'long' format or vice-versa

Usage

```
reShapeDS(
  data.name,
  varying.transmit,
  v.names.transmit,
  timevar.name,
  idvar.name,
  drop.transmit,
  direction,
  sep
)
```

Arguments

data.name	the name of the data.frame to be reshaped. Specified via argument <data.name> of ds.reShape function
varying.transmit	names of sets of variables in the wide format that correspond to single variables in long format (typically what may be called 'time-varying' or 'time-dependent' variables). Specified via argument <varying> of ds.reShape function.
v.names.transmit	the names of variables in the long format that correspond to multiple variables in the wide format - for example, sbp7, sbp11, sbp15 (measured systolic blood pressure at ages 7, 11 and 15 years). Specified via argument <v.names> of ds.reShape function
timevar.name	the variable in long format that differentiates multiple records from the same group or individual. Specified via argument <timevar.name> of ds.reShape function
idvar.name	names of one or more variables in long format that identify multiple records from the same group/individual. This/these variable(s) may also be present in wide format. Specified via argument <idvar.name> of ds.reShape function
drop.transmit	a vector of names of variables to drop before reshaping. Specified via argument <drop> of ds.reShape function
direction	a character string, partially matched to either "wide" to reshape from long to wide format, or "long" to reshape from wide to long format. Specified via argument <direction> of ds.reShape function
sep	a character vector of length 1, indicating a separating character in the variable names in the wide format. Specified via argument <sep> of ds.reShape function

Details

This function is based on the native R function reshape. It reshapes a data frame containing longitudinal or otherwise grouped data between 'wide' format with repeated measurements in separate columns of the same record and 'long' format with the repeated measurements in separate records. The reshaping can be in either direction

Value

a reshaped data.frame converted from long to wide format or from wide to long format which is written to the serverside and given the name provided as the <newobj> argument of `ds.reShape` or 'newObject' if no name is specified. In addition, two validity messages are returned to the clientside indicating whether <newobj> has been created in each data source and if so whether it is in a valid form (see header for `ds.reShape`.

Author(s)

Demetris Avraam, Paul Burton for DataSHIELD Development Team

`rmDS` *rmDS an aggregate function called by ds.rm*

Description

deletes an R object on the serverside

Usage

```
rmDS(x.names.transmit)
```

Arguments

`x.names.transmit`
: the names of the objects to be deleted converted into transmissible form, a comma separated list of character string. The argument is specified via the <x.names> argument of ds.rm

Details

this is a serverside function based on the rm() function in native R. It is an aggregate function which may be surprising because it modifies an object on the serverside, and would therefore be expected to be an assign function. However, as an assign function the last step in running it would be to write the modified object as newobj. But this would fail because the effect of the function is to delete the object and so it would be impossible to write it anywhere.

Value

the specified object is deleted from the serverside. If this is successful the message "Object <x.names> successfully deleted" is returned to the clientside (where x.names are the names of the object to be deleted). If the objects to be deleted is already absent on a given source, that source will return the message: "Object to be deleted, i.e. <x.names>, does not exist so does not need deleting".

Author(s)

Paul Burton for DataSHIELD Development Team

rNormDS *rNormDS serverside assign function*

Description

primary serverside assign function called by ds.rNorm

Usage

```
rNormDS(n, mean = 0, sd = 1, force.output.to.k.decimal.places = 9)
```

Arguments

n	length of the pseudorandom number vector to be generated as specified by the argument <samp.size> in the function ds.rNorm
mean	this specifies the mean of the pseudorandom number vector to be generated as specified by the argument <mean> in the function ds.rNorm. May be a scalar or a vector allowing the mean to vary from observation to observation.
sd	this specifies the standard deviation of the pseudorandom number vector to be generated as specified by the argument <sd> in the function ds.rNorm May be a scalar or a vector allowing the sd to vary from observation to observation.
force.output.to.k.decimal.places	scalar integer. Forces the output random number vector to have k decimal places. If 0 rounds it coerces decimal random number output to integer, a k in range 1-8 forces output to have k decimal places. If k = 9, no rounding occurs of native output. Default=9. Value specified by <force.output.to.k.decimal.places> argument in ds.rNorm

Details

Generates the vector of pseudorandom numbers from a normal distribution in each data source as specified by the arguments of ds.rNorm. This serverside function is effectively the same as the function rnorm() in native R and its arguments are the same.

Value

Writes the pseudorandom number vector with the characteristics specified in the function call as a new serverside vector on the data source on which it has been called. Also returns key information to the clientside: the random seed as specified by you in each source + (if requested) the full 626 length random seed vector this generated in each source (see info for the argument <return.full.seed.as.set>). It also returns a vector reporting the length of the pseudorandom vector created in each source.

Author(s)

Paul Burton for DataSHIELD Development Team

rowColCalcDS *Computes sums and means of rows or columns of numeric arrays*

Description

The function is similar to R base functions 'rowSums', 'colSums', 'rowMeans' and 'colMeans'.

Usage

```
rowColCalcDS(dataset, operation)
```

Arguments

dataset	an array of two or more dimensions.
operation	an integer that indicates the operation to carry out: 1 for 'rowSums', 2 for 'colSums', 3 for 'rowMeans' or 4 for 'colMeans'

Details

the output is returned to the user only the number of entries in the output vector is greater or equal to the allowed size.

Value

a numeric vector

Author(s)

Gaye, A.

rPoisDS *rPoisDS serverside assign function*

Description

primary serverside assign function called by ds.rPois

Usage

```
rPoisDS(n, lambda = 1)
```

Arguments

n	length of the pseudorandom number vector to be generated as specified by the argument <samp.size> in the function ds.rPois
lambda	a numeric scalar specifying the expected count of the Poisson distribution used to generate the random counts. Specified directly by the lambda argument in ds.rPois. May be a scalar or a vector allowing lambda to vary from observation to observation.

Details

Generates the vector of pseudorandom numbers (non-negative integers) from a Poisson distribution in each data source as specified by the arguments of ds.rPois. This serverside function is effectively the same as the function rpois() in native R and its arguments are the same.

Value

Writes the pseudorandom number vector with the characteristics specified in the function call as a new serverside vector on the data source on which it has been called. Also returns key information to the clientside: the random seed as specified by you in each source + (if requested) the full 626 length random seed vector this generated in each source (see info for the argument <return.full.seed.as.set>). It also returns a vector reporting the length of the pseudorandom vector created in each source.

Author(s)

Paul Burton for DataSHIELD Development Team

---

rUnifDS *rUnifDS serverside assign function*

Description

primary serverside assign function called by ds.rUnif

Usage

```
rUnifDS(n, min = 0, max = 1, force.output.to.k.decimal.places = 9)
```

Arguments

n	length of the pseudorandom number vector to be generated as specified by the argument <samp.size> in the function ds.rUnif
min	a numeric scalar specifying the minimum of the range across which the random numbers will be generated in each source. Specified directly by the min argument in ds.rUnif. May be a scalar or a vector allowing the min to vary from observation to observation.

`max` a numeric scalar specifying the maximum of the range across which the random numbers will be generated in each source. Specified directly by the max argument in ds.rUnif. May be a scalar or a vector allowing the min to vary from observation to observation.

`force.output.to.k.decimal.places`
scalar integer. Forces the output random number vector to have k decimal places. If 0 rounds it coerces decimal random number output to integer, a k in range 1-8 forces output to have k decimal places. If k = 9, no rounding occurs of native output. Default=9. Value specified by <force.output.to.k.decimal.places> argument in ds.rUnif

Details

Generates the vector of pseudorandom numbers from a uniform distribution in each data source as specified by the arguments of ds.rUnif. This serverside function is effectively the same as the function runif() in native R and its arguments are the same.

Value

Writes the pseudorandom number vector with the characteristics specified in the function call as a new serverside vector on the data source on which it has been called. Also returns key information to the clientside: the random seed as specified by you in each source + (if requested) the full 626 length random seed vector this generated in each source (see info for the argument <return.full.seed.as.set>). It also returns a vector reporting the length of the pseudorandom vector created in each source.

Author(s)

Paul Burton for DataSHIELD Development Team

---

sampleDS *random sampling and permuting of vectors, dataframes and matrices*

Description

draws a pseudorandom sample from a vector, dataframe or matrix on the serverside or - as a special case - randomly permutes a vector, dataframe or matrix.

Usage

```
sampleDS(
  x.transmit,
  size.transmit,
  replace.transmit = NULL,
  prob.transmit = NULL
)
```

Arguments

x.transmit	Either a character string providing the name for the serverside vector, matrix or data.frame to be sampled or permuted, or an integer/numeric scalar (e.g. 923) indicating that one should create a new vector on the serverside that is a randomly permuted sample of the vector 1:923. x.transmit is fully specified by the [x] argument of ds.sample. For further details see help for ds.sample and native R help for sample().
size.transmit	a numeric/integer scalar indicating the size of the sample to be drawn. size.transmit is fully specified by the [size] argument of ds.sample. For further details see help for ds.sample and native R help for sample().
replace.transmit	a Boolean indicator (TRUE or FALSE) specifying whether the sample should be drawn with or without replacement. Default is FALSE so the sample is drawn without replacement. replace.transmit is fully specified by the [replace] argument of ds.sample. For further details see help for ds.sample and native R help for sample().
prob.transmit	a character string containing the name of a numeric vector of probability weights on the serverside that is associated with each of the elements of the vector to be sampled enabling the drawing of a sample with some elements given higher probability of being drawn than others. prob.transmit is fully specified by the [prob] argument of ds.sample. For further details see help for ds.sample and native R help for sample().

Details

Serverside assign function sampleDS called by clientside function ds.sample. Based on the native R function `sample()` but deals slightly differently with data.frames and matrices. For further details see help for ds.sample and native R help for sample().

Value

the object specified by the <newobj> argument (or default name 'newobj.sample') which is written to the serverside. For further details see help for ds.sample and native R help for sample().

Author(s)

Paul Burton, for DataSHIELD Development Team, 15/4/2020

---

scatterPlotDS *Calculates the coordinates of the data to be plot*

Description

This function uses two disclosure control methods to generate non-disclosive coordinates that are returned to the client that generates the non-disclosive scatter plots.

Usage

```
scatterPlotDS(x, y, method.indicator, k, noise)
```

Arguments

x	the name of a numeric vector, the x-variable.
y	the name of a numeric vector, the y-variable.
method.indicator	an integer either 1 or 2. If the user selects the deterministic method in the client side function the method.indicator is set to 1 while if the user selects the probabilistic method this argument is set to 2.
k	the number of the nearest neighbours for which their centroid is calculated if the deterministic method is selected.
noise	the percentage of the initial variance that is used as the variance of the embedded noise if the probabilistic method is selected.

Details

If the user chooses the deterministic approach, the function finds the k-1 nearest neighbours of each data point in a 2-dimensional space. The nearest neighbours are the data points with the minimum Euclidean distances from the point of interest. Each point of interest and its k-1 nearest neighbours are then used for the calculation of the coordinates of the centroid of those k points. Centroid here is referred to the centre of mass, i.e. the x-coordinate of the centroid is the average value of the x-coordinates of the k nearest neighbours and the y-coordinate of the centroid is the average of the y-coordinates of the k nearest neighbours. If the user chooses the probabilistic approach, the function adds random noise to x and y separately. Each random noise follows a normal distribution with zero mean and variance equal to 10 disclosure we fix the random number generator in a value that is specified by the input variables. Thus the function returns always the same noisy data for a given pair of variables.

Value

a list with the x and y coordinates of the data to be plot

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

seqDS *seqDS a serverside assign function called by ds.seq*

Description

assign function seqDS called by ds.seq

Usage

```
seqDS(
  FROM.value.char,
  TO.value.char,
  BY.value.char,
  LENGTH.OUT.value.char,
  ALONG.WITH.name
)
```

Arguments

`FROM.value.char`
: the starting value for the sequence expressed as an integer or real number with a decimal point but in character form. Fully specified by <FROM.value.char> argument of `ds.seq`.

`TO.value.char`
: the terminal value for the sequence expressed as an integer or real number with a decimal point but in character form. Fully specified by <TO.value.char> argument of `ds.seq`.

`BY.value.char`
: the value to increment each step in the sequence expressed as an integer or real number with a decimal point but in character form. Fully specified by <BY.value.char> argument of `ds.seq`.

`LENGTH.OUT.value.char`
: length of the sequence at which point its extension should be stopped, expressed as an integer or real number with a decimal point but in character form. Fully specified by <LENGTH.OUT.value.char> argument of `ds.seq`.

`ALONG.WITH.name`
: For convenience, rather than specifying a value for LENGTH.OUT it can often be better to specify a variable name as the <ALONG.WITH.name> argument. Fully specified by <ALONG.WITH.name> argument of `ds.seq`.

Details

An assign function that uses the native R function seq() to create any one of a flexible range of sequence vectors that can then be used to help manage and analyse data. As it is an assign function the resultant vector is written as a new object into all of the specified data source servers. Please see "details" for `ds.seq` for more information about allowable combinations of arguments etc.

Value

the object specified by the <newobj> argument of `ds.seq` (or its default name newObj) which is written to the serverside. As well as writing the output object as <newobj> on the serverside, two validity messages are returned indicating whether <newobj> has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - `ds.seq()` also returns any studysideMessages that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the `ds.message` function. If you type ds.message("<newobj>") it will print out the relevant studysideMessage from any datasource in which there was an error in creating

<newobj> and a studysideMessage was saved. If there was no error and <newobj> was created without problems no studysideMessage will have been saved and ds.message("<newobj>") will return the message: "ALL OK: there are no studysideMessage(s) on this datasource".

Author(s)

Paul Burton for DataSHIELD Development Team, 17/9/2019

`setSeedDS` *setSeedDs called by ds.setSeed, ds.rNorm, ds.rUnif, ds.rPois and ds.rBinom*

Description

An aggregate serverside function that primes the pseudorandom number generator in a data source

Usage

```
setSeedDS(seedtext = NULL, kind = NULL, normal.kind = NULL)
```

Arguments

`seedtext`	this is simply the value of the <seed.as.integer> argument of ds.setSeed, ds.rNorm, ds.rUnif, ds.rPois of ds.rBinom coerced into character format. This is done by the clientside functions themselves and does not require the DataSHIELD user to do anything. Please see the help for these clientside functions, and in particular, the information for the argument <seed.as.integer> for more details.
`kind`	see help for set.seed() function in native R
`normal.kind`	see help for set.seed() function in native R

Details

setSeedDS is effectively equivalent to the native R function set.seed() and so the help for that function can provide many additional details. The only very minor difference is that the first argument of setSeedDS, <seedtext> takes the integer priming seed in character format. However, for the user that integer is still specified directly as an integer as the <seed.as.integer> argument of one of the clientside functions ds.setSeed, ds.rNorm ..... Each of these clientside functions coerces the integer to character format calls setSeedDS and the first active line of code in setSeedDS converts the character string back to an integer and treats it as the first argument <seed> of the native R function set.seed(). The two other arguments of set.seed() in native R, <kind> and <normal.kind> are both defaulted by specifying them as NULL. This defaulting is hard wired into the setSeedDS function and as this cannot be changed by the analyst it means that setSeedDS is much less flexible than native R's set.seed() function. If any DataSHIELD user requires some aspect of this flexibility returned the development team can be approached, but unless you are actually doing theoretical work with random number generators it is likely that the

Value

Sets the values of the vector of integers of length 626 known as .Random.seed on each data source that is the true current state of the random seed in each source.

Author(s)

Paul Burton for DataSHIELD Development Team

---

skewnessDS1 *Calculates the skewness of a numeric variable*

Description

This function calculates the skewness of a numeric variable for each study separately.

Usage

```
skewnessDS1(x, method)
```

Arguments

x	a string character, the name of a numeric variable.
method	an integer between 1 and 3 selecting one of the algorithms for computing skewness detailed in the headers of the client-side ds.skewness function.

Details

The function calculates the skewness of an input variable x with three different methods. The method is specified by the argument method in the client-side ds.skewness function.

Value

a list including the skewness of the input numeric variable, the number of valid observations and the study-side validity message.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

skewnessDS2 *Calculates the skewness of a numeric variable*

Description

This function calculates summary statistics that are returned to the client-side and used for the estimation of the combined skewness of a numeric variable across all studies.

Usage

```
skewnessDS2(x, global.mean)
```

Arguments

x	a string character, the name of a numeric variable.
global.mean	a numeric, the combined mean of the input variable across all studies.

Details

The function calculates the sum of squared differences between the values of x and the global mean of x across all studies, the sum of cubed differences between the values of x and the global mean of x across all studies and the number of valid observations of the input variable x.

Value

a list including the sum of cubed differences between the values of x and the global mean of x across all studies, the sum of squared differences between the values of x and the global mean of x across all studies, the number of valid observations (i.e. the length of x after excluding missing values), and a validity message indicating indicating a valid analysis if the number of valid observations are above the protection filter nfilter.tab or invalid analysis otherwise.

Author(s)

Demetris Avraam, for DataSHIELD Development Team

sqrtDS *Computes the square root values of the input variable*

Description

This function is similar to R function sqrt.

Usage

```
sqrtDS(x)
```

Arguments

x	a string character, the name of a numeric or integer vector

Details

The function computes the square root values of an input numeric or integer vector.

Value

the object specified by the newobj argument of ds.sqrt (or default name sqrt.newobj) which is written to the server-side. The output object is of class numeric or integer.

Author(s)

Demetris Avraam for DataSHIELD Development Team

---

subsetByClassDS *Breaks down a dataframe or a factor into its sub-classes*

Description

The function takes a categorical vector or dataframe as input and generates subset(s) vectors or dataframes for each category. Subsets are considered invalid if they hold between 1 and 4 observations.

Usage

```
subsetByClassDS(data = NULL, variables = NULL)
```

Arguments

data	a string character, the name of the dataframe or the factor vector
variables	a vector of string characters, the names of the the variables to subset on.

Details

If the input data object is a dataframe it is possible to specify the variables to subset on. If a subset is not 'valid' all its the values are reported as missing (i.e. NA), the name of the subsets is labelled as '_INVALID'. If no variables are specified to subset on, the dataframe will be subset on each of its factor variables. And if none of the columns holds a factor variable a message is issued as output. A message is also issued as output if the input vector is not of type factor.

Value

a list which contains the subsetted datasets

Author(s)

Gaye, A.

subsetDS *Generates a valid subset of a table or a vector*

Description

The function uses the R classical subsetting with squared brackets '[]' and allows also to subset using a logical operator and a threshold. The object to subset from must be a vector (factor, numeric or character) or a table (data.frame or matrix).

Usage

```
subsetDS(
  dt = NULL,
  complt = NULL,
  rs = NULL,
  cs = NULL,
  lg = NULL,
  th = NULL,
  varname = NULL
)
```

Arguments

dt	a string character, the name of the dataframe or the factor vector and the range of the subset.
complt	a boolean that tells if the subset to subset should include only complete cases
rs	a vector of two integers that give the range of rows de extract.
cs	a vector of two integers or one or more characters; the indices of the columns to extract or the names of the columns (i.e. names of the variables to extract).
lg	a character, the logical parameter to use if the user wishes to subset a vector using a logical operator. This parameter is ignored if the input data is not a vector.
th	a numeric, the threshold to use in conjunction with the logical parameter. This parameter is ignored if the input data is not a vector.
varname	a character, if the input data is a table, if this parameter is provided along with the 'logical' and 'threshold' parameters, a subtable is based the threshold applied to the specified variable. This parameter is however ignored if the parameter 'rows' and/or 'cols' are provided.

Details

If the input data is a table: The user specifies the rows and/or columns to include in the subset if the input object is a table; the columns can be referred to by their names. The name of a vector (i.e. a variable) can also be provided with a logical operator and a threshold (see example 3). If the input data is a vector: when the parameters 'rows', 'logical' and 'threshold' are all provided the last two

are ignored ('rows' has precedence over the other two parameters then). If the requested subset is not valid (i.e. contains less than the allowed number of observations), the subset is not generated, rather a table or a vector of missing values is generated to allow for any subsequent process using the output of the function to proceed after informing the user via a message.

Value

a subset of the vector, matrix or dataframe as specified is stored on the server side

Author(s)

Gaye, A.

`table1DDS` *Creates 1-dimensional contingency tables*

Description

This function generates a 1-dimensional table where potentially disclosive cells. (based on the set threshold) are replaced by a missing value ('NA').

Usage

```
table1DDS(xvect)
```

Arguments

`xvect`	a numerical vector with discrete values - usually a factor.

Details

It generates a 1-dimensional tables where valid (non-disclosive) 1-dimensional tables are defined as data from sources where no table cells have counts between 1 and the set threshold. When the output table is invalid all cells but the total count are replaced by missing values. Only the total count is visible on the table returned to the client side. A message is also returned with the 1-dimensional; the message says "invalid table - invalid counts present" if the table is invalid and 'valid table' otherwise.

Value

a list which contains two elements: 'table', the 1-dimensional table and 'message' a message which informs about the validity of the table.

Author(s)

Gaye A.

table2DDS *table2DDS (aggregate function) called by ds.table2D*

Description

This function generates a 2-dimensional contingency table where potentially disclosive cells (based on a set threshold) are replaced by a missing value ('NA').

Usage

```
table2DDS(xvect, yvect)
```

Arguments

xvect	a numerical vector with discrete values - usually a factor.
yvect	a numerical vector with discrete values - usually a factor.

Details

It generates 2-dimensional contingency tables where valid (non-disclosive) tables are defined as those where none of their cells have counts between 1 and the set threshold "nfilter.tab". When the output table is invalid all cells except the total counts are replaced by missing values. Only the total counts are visible on the table returned to the client side. A message is also returned with the 2-dimensional table; the message says "invalid table - invalid counts present" if the table is invalid and 'valid table' otherwise.

Value

a list which contains two elements: 'table', the 2-dimensional table and 'message' a message which informs about the validity of the table.

Author(s)

Amadou Gaye, Paul Burton, Demetris Avraam for DataSHIELD Development Team

tableDS *tableDS is the first of two serverside aggregate functions called by ds.table*

Description

creates 1-dimensional, 2-dimensional and 3-dimensional tables using the table function in native R.

Usage

```
tableDS(
  rvar.transmit,
  cvar.transmit,
  stvar.transmit,
  rvar.all.unique.levels.transmit,
  cvar.all.unique.levels.transmit,
  stvar.all.unique.levels.transmit,
  exclude.transmit,
  useNA.transmit,
  force.nfilter.transmit
)
```

Arguments

`rvar.transmit`	is a character string (in inverted commas) specifying the name of the variable defining the rows in all of the 2 dimensional tables that form the output. Fully specified by <rvar> argument in `ds.table`. For more information see help for `ds.table`
`cvar.transmit`	is a character string specifying the name of the variable defining the columns in all of the 2 dimensional tables that form the output. Fully specified by <cvar> argument in `ds.table`. For more information see help for `ds.table`
`stvar.transmit`	is a character string specifying the name of the variable that indexes the separate two dimensional tables in the output if the call specifies a 3 dimensional table. Fully specified by <stvar> argument in `ds.table`. For more information see help for `ds.table`
`rvar.all.unique.levels.transmit`	is a character string containing all unique level in rvar, across the studies, separated by ','.
`cvar.all.unique.levels.transmit`	is a character string containing all unique level in cvar, across the studies, separated by ','.
`stvar.all.unique.levels.transmit`	is a character string containing all unique level in stvar, across the studies, separated by ','.
`exclude.transmit`	for information see help on <exclude> argument of `ds.table`. Fully specified by <exclude> argument of `ds.table`
`useNA.transmit`	for information see help on <useNA> argument of `ds.table`. Fully specified by <useNA> argument of `ds.table`
`force.nfilter.transmit`	for information see help on <force.nfilter> argument of `ds.table`. Fully specified by <force.nfilter> argument of `ds.table`

Details

this serverside function is the workhorse of ds.table - creating the table requested in the format specified by ds.table. For more information see help for ds.table in DataSHIELD and the table function in native R.

Value

For information see help for ds.table

Author(s)

Paul Burton for DataSHIELD Development Team, 13/11/2019

---

tableDS.assign *tableDS.assign is the serverside assign function called by ds.table*

Description

helps creates 1-dimensional, 2-dimensional and 3-dimensional tables using the table function in native R.

Usage

```
tableDS.assign(
  rvar.transmit,
  cvar.transmit,
  stvar.transmit,
  rvar.all.unique.levels.transmit,
  cvar.all.unique.levels.transmit,
  stvar.all.unique.levels.transmit,
  exclude.transmit,
  useNA.transmit
)
```

Arguments

rvar.transmit	is a character string (in inverted commas) specifying the name of the variable defining the rows in all of the 2 dimensional tables that form the output. Fully specified by <rvar> argument in ds.table. For more information see help for ds.table
cvar.transmit	is a character string specifying the name of the variable defining the columns in all of the 2 dimensional tables that form the output. Fully specified by <cvar> argument in ds.table. For more information see help for ds.table
stvar.transmit	is a character string specifying the name of the variable that indexes the separate two dimensional tables in the output if the call specifies a 3 dimensional table. Fully specified by <stvar> argument in ds.table. For more information see help for ds.table

`rvar.all.unique.levels.transmit`
: is a character string containing all unique level in rvar, across the studies, separated by ','.

`cvar.all.unique.levels.transmit`
: is a character string containing all unique level in cvar, across the studies, separated by ','.

`stvar.all.unique.levels.transmit`
: is a character string containing all unique level in stvar, across the studies, separated by ','.

`exclude.transmit`
: for information see help on <exclude> argument of `ds.table`. Fully specified by <exclude> argument of `ds.table`

`useNA.transmit`
: for information see help on <useNA> argument of `ds.table`. Fully specified by <useNA> argument of `ds.table`

Details

If the <table.assign> argument of `ds.table` is set to TRUE, this assign function writes the the table requested in the format specified by `ds.table` function as an object named by the <newobj> argument of `ds.table`. For more information see help for `ds.table` in DataSHIELD and the `table` function in native R.

Value

For information see help for `ds.table`

Author(s)

Paul Burton for DataSHIELD Development Team, 13/11/2019

---

tableDS2 *tableDS is the second of two serverside aggregate functions called by ds.table*

Description

Helps creates 1-dimensional, 2-dimensional and 3-dimensional tables using the `table` function in native R.

Usage

```
tableDS2(newobj, rvar.transmit, cvar.transmit, stvar.transmit)
```

Arguments

newobj	this a character string providing a name for the output table object to be written to the serverside if <table.assign> is TRUE. If no explicit name for the table object is specified, but <table.assign> is nevertheless TRUE, the name for the serverside table object defaults to 'newObj'. Fully specified by <newobj> argument in ds.table. For more information see help for ds.table
rvar.transmit	is a character string (in inverted commas) specifying the name of the variable defining the rows in all of the 2 dimensional tables that form the output. Fully specified by <rvar> argument in ds.table. For more information see help for ds.table
cvar.transmit	is a character string specifying the name of the variable defining the columns in all of the 2 dimensional tables that form the output. Fully specified by <cvar> argument in ds.table. For more information see help for ds.table
stvar.transmit	is a character string specifying the name of the variable that indexes the separate two dimensional tables in the output if the call specifies a 3 dimensional table. Fully specified by <stvar> argument in ds.table. For more information see help for ds.table

Details

If the <table.assign> argument of ds.table is set to TRUE, this aggregate function returns non-disclosive information about the table object written to the serverside by tableDS.assign. For more information see help for ds.table, tableDS.assign and tableDS in DataSHIELD and the table function in native R.

Value

For information see help for ds.table

Author(s)

Paul Burton for DataSHIELD Development Team, 13/11/2019

---

tapplyDS *tapplyDS called by ds.tapply*

Description

Apply one of a selected range of functions to summarize an outcome variable over one or more indexing factors and write the resultant summary to the clientside

Usage

```
tapplyDS(X.name, INDEX.names.transmit, FUN.name)
```

Arguments

X.name	the name of the variable to be summarized. Specified via argument <X.name> of ds.tapply function
INDEX.names.transmit	the name of a single factor or a vector of names of factors to index the variable to be summarized. Specified via argument <INDEX.names> of ds.tapply function
FUN.name	the name of one of the allowable summarizing functions to be applied. Specified via argument <FUN.name> of ds.tapply function.

Details

see details for ds.tapply function

Value

an array of the summarized values created by the tapplyDS function. This array is returned to the clientside. It has the same number of dimensions as INDEX.

Author(s)

Paul Burton, Demetris Avraam for DataSHIELD Development Team

---

tapplyDS.assign *tapplyDS.assign called by ds.tapply.assign*

Description

Apply one of a selected range of functions to summarize an outcome variable over one or more indexing factors and write the resultant summary as a newobj on the serverside

Usage

```
tapplyDS.assign(X.name, INDEX.names.transmit, FUN.name)
```

Arguments

X.name	the name of the variable to be summarized. Specified via argument <X.name> of ds.tapply.assign function
INDEX.names.transmit	the name of a single factor or a vector of names of factors to index the variable to be summarized. Specified via argument <INDEX.names> of ds.tapply.assign function
FUN.name	the name of one of the allowable summarizing functions to be applied. Specified via argument <FUN.name> of ds.tapply.assign function.

Details

see details for ds.tapply.assign function

Value

an array of the summarized values created by the tapplyDS.assign function. This array is written as a newobj on the serverside. It has the same number of dimensions as INDEX.

Author(s)

Paul Burton, Demetris Avraam for DataSHIELD Development Team

---

testObjExistsDS *testObjExistsDS*

Description

The server-side function called by ds.testObjExists

Usage

```
testObjExistsDS(test.obj.name = NULL)
```

Arguments

test.obj.name	a client-side provided character string specifying the variable whose presence is to be tested in each data source

Details

Tests whether a given object exists in all sources. It is called at the end of all recently written assign functions to check the new (assigned) object has been created in all sources

Value

List with ‘test.obj.exists‘ and ‘test.obj.class‘

Author(s)

Burton PR

uniqueDS *Applies the* unique *method to a server-side variable.*

Description

This function is similar to R function unique.

Usage

```
uniqueDS(x.name.transmit = NULL)
```

Arguments

x.name.transmit
: is the name of the variable upon which unique method will be applied

Details

The function computes the uniques values of a variable.

Value

the object specified by the newobj argument which is written to the server-side.

Author(s)

Stuart Wheater for DataSHIELD Development Team

unListDS unListDS *a serverside assign function called by* ds.unList

Description

this function is based on the native R function unlist which coerces an object of list class back to the class it was when it was coerced into a list

Usage

```
unListDS(x.name)
```

Arguments

x.name
: the name of the input object to be unlisted. It must be specified in inverted commas e.g. x.name="input.object.name". Fully specified by the x.name argument of ds.unList

Details

See details of the native R function unlist. This function represents a substantive restructuring of an earlier version created by Amadou Gaye. For further details of its working please see 'details' in the help for ds.unList.

Value

the object specified by the newobj argument of the ds.unList function (or by default "unlist.newobj" if the newobj argument is NULL). This is written to the serverside. As well as writing the output object as newobj on the serverside, two validity messages are returned indicating whether newobj has been created in each data source and if so whether it is in a valid form. If its form is not valid in at least one study - e.g. because a disclosure trap was tripped and creation of the full output object was blocked - ds.seq also returns any studysideMessages that can explain the error in creating the full output object. As well as appearing on the screen at run time,if you wish to see the relevant studysideMessages at a later date you can use the ds.message function. If you type ds.message("<newobj>") it will print out the relevant studysideMessage from any datasource in which there was an error in creating newobj and a studysideMessage was saved. Because the outcome object from ds.unList is typically a list object with no names, if there are no errors in creating it the message returned from ds.message("<newobj>") in each study will read "Outcome object is a list without names. So a studysideMessage may be hidden. Please check output is OK". This suggests that - in the case of this specific function - one should check as far as one can the nature of the output from a call to ds.unList - e.g. ds.class, ds.length etc

Author(s)

Amadou Gaye (2016), Paul Burton (19/09/2019) for DataSHIELD Development Team

---

varDS *Computes the variance of vector*

Description

Calculates the variance.

Usage

```
varDS(xvect)
```

Arguments

xvect	a vector

Details

if the length of input vector is less than the set filter a missing value is returned.

Value

a list, with the sum of the input variable, the sum of squares of the input variable, the number of missing values, the number of valid values, the number of total length of the variable, and a study message indicating whether the number of valid is less than the disclosure threshold

Author(s)

Amadou Gaye, Demetris Avraam, for DataSHIELD Development Team

---

vectorDS *Creates a vector on the server-side.*

Description

This function is similar to R function `c`.

Usage

```
vectorDS(...)
```

Arguments

...	parameter to be used to form the vector.

Details

The function computes the vectors values.

Value

the object specified by the `newobj` argument which is written to the server-side.

Author(s)

Stuart Wheater for DataSHIELD Development Team

Index