Package ‘ibmdbR’

Version 1.36.7

Title IBM in-Database Analytics for R

Author IBM Corporation

Maintainer Michael Wurst <inza@de.ibm.com>

Depends R (>= 2.15.1), methods, RODBC, MASS

Description Functionality required to efficiently use R with IBM DB2(C) for Linux, Unix and Windows as well as IBM dashDB(C).
Many basic and complex R operations are pushed down into the database, which removes the main memory boundary of R and allows to make full use of parallel processing in the underlying database.

License GPL-3

LazyLoad Yes

NeedsCompilation no

Repository CRAN

Date/Publication 2015-03-08 07:18:11

R topics documented:

ibmdbR-package ................................. 2
as.ida.data.frame .................................. 2
ida.col.def methods ............................... 3
ida.data.frame methods ............................ 4
ida.data.frame, is.ida.data.frame ................. 4
ida.list ................................. 6
ida.list methods .............................. 7
idaConnect, idaClose ............................. 7
idaCreateView, idaDropView ....................... 8
idaDeleteTable, idaExistTable, idaGetValidTableName ................................. 9
idadf,idaSave,idaUpdate .......................... 10
idaDropModel .................................... 12
idaInit ........................................ 12
idaKMeans ...................................... 13
In-database analytics functions operate directly on data in a database, rather than requiring that the data first be extracted to working memory. This lets you analyze large amounts of data that would be impractical or impossible to extract. It also avoids security issues associated with extracting data, and ensures that the data being analyzed is as current as possible. Some functions additionally use lazy loading to load only those parts of the data that are actually required, to further increase efficiency.

This package also contains a data structure called an ida.list, which you can use to store R objects in the database. This simplifies the sharing of R objects among users. Each user is assigned two tables for R object storage: a private table, to which only that user has access, and a public table, which can be read by other users. Use a IDA list to generate a pointer to either of these tables, and use the pointer to list, store, or retrieve R objects.

---

**as.ida.data.frame**

*Convert an R object to a IDA data frame*

**Description**

This function creates a IDA data frame `ida.data.frame` from a local R `data.frame` by creating a table in the database.

**Usage**

```
as.ida.data.frame(x, table=NULL, clear.existing=FALSE, case.sensitive=TRUE)
```
Arguments

- **x**: The name of the input object that is to be converted to a IDA data frame.
- **table**: The name of the database table that is to be created to hold the contents of the IDA data frame. The specified name is folded to uppercase. If this parameter is not specified, a name is generated automatically.
- **clear.existing**: If the `table` parameter is specified and a table with the specified name already exists, this parameter specifies whether the existing table is to be dropped (TRUE) or whether the `as.ida.data.frame` statement is to be ignored and a warning message issued (FALSE).
- **case_sensitive**: If the `table` parameter is specified for an existing table, this parameter specifies whether the column names in that table name are to be treated as case-sensitive (TRUE) or not case-sensitive (FALSE).

Value

A IDA data frame that points to the newly created table.

See Also

`as.data.frame`

Examples

```r
## Not run:

#Add an ID column to iris
iris2 <- iris
iris2$ID <- 1:150

#Upload it and overwrite if already exists
idf <- as.ida.data.frame(iris2,"IRIS",clear.existing=T)
```

Description

`ida.col.def` objects are used to define new columns of a `ida.data.frame` based on existing ones. For details see the documentation of `ida.data.frame`. 

ida.data.frame methods

Available methods for ida.data.frame

Description

ida.data.frame objects provide many methods that will behave exactly like or very similar to methods defined on a regular data.frame. The following is a list of currently supported methods: as.data.frame, sd, max, mean, min, length, print, names, colnames, summary, NROW, NCOL, dim, var, head, hist, cor, cov. Furthermore, the $ and [ operators allow you to select columns and rows and the $<- operator will allow you to add columns. For details see the documentation of ida.data.frame.

ida.data.frame, is.ida.data.frame

Create a IDA data frame

Description

This function creates a IDA data frame (that is, an object of the class ida.data.frame). It does not store any data in local memory, but aggregates metadata used to determine the exact table subset (columns - SELECT clause; and/or rows - WHERE clause) and creates a pointer to a table located in the database.

Usage

ida.data.frame(table)
is.ida.data.frame(x)

Arguments

table Name of a table or view in the current database.
x An ida.data.frame object.

Details

The argument table must be a valid table or view name and the table/view must exist.

If schema or table are set in quotes, they will be treated case sensitive otherwise they are automatically converted to the default schema of the database. Columns are always treated case sensitive.

A subset of columns and/or rows may be specified using the indexing operator [] (which is translated to the SELECT clause for columns and/or the WHERE clause for rows). Note that columns are treated case sensitive.
One limitation is that rows cannot be selected using their numbers. Instead, you must specify value-based conditions, for example `d[d$ID > 10, ]` which means “all rows where the value of the first column is greater than 10”. The `$` operator may be also used to select a `ida.data.frame` column.

You can also add and alter columns in an `ida.data.frame`. Currently, a limited set of functions and operators is supported to define columns based on other columns. The following is supported:

- Arithmetic operators: `+,-,*,^`
- Mathematical functions: `abs,sqrt,log,log10,exp,floor,round,ceiling`
- Casting functions: `as.numeric, as.integer, as.character`
- Comparison and logical operators: `<,<=,>,>=,!=,==,!,&`, `|`
- Conditional functions: `ifelse`
- Special functions: `is.db.null` (checks whether column value is NULL in the table)

There are several rules for adding columns:

1. You can not combine columns from different tables or from `ida.data.frames` that have different `WHERE` conditions.
2. You cannot add a column to a `ida.data.frame` that was defined on columns from another `ida.data.frame`
3. You can only add columns that evaluate to non-logical, atomar values

The package does basic type checking to enforce these rules, however, it is still possible that the database will refuse a query that was not properly defined.

`is.ida.data.frame` checks if the given object’s class is `ida.data.frame`.

### Value

`ida.data.frame` returns a IDA data frame.

`is.ida.data.frame` returns a logical value that indicates whether the specified object is a IDA data frame.

### Examples

```r
# Not run:
idf <- ida.data.frame('IRIS')
is.ida.data.frame(idf)

# Select only certain rows or columns
# The following creates a ida.data.frame that only selects rows with
# Species=='setosa' and the first three columns of the table
idf2 <- idf[idf$Species=='setosa',1:3]

# Define new columns based on existing ones
idf$SepalLengthX <- idf$SepalLength+1
idf$SepalLengthY <- ifelse(idf$SepalLengthX>4.5,idf$ID,10)

# Take a look at the newly defined columns
head(idf)
```
ida.list  

Store and retrieve R objects in the database

Description

A user can elect to store R objects in a database table rather than storing them in a workstation file system. This makes it easier for users to share objects, and simplifies backup tasks.

Each user has two R object storage tables:

- A private table, for objects that other users are not to be able to access
- A public table, for objects that other users are to be able to read

Use the `ida.list` function to create a pointer to either of your own R object storage tables, or to the public R object storage table of another user. You can then use the pointer to store objects in or retrieve objects from the corresponding table. (If the table belongs to another user you can only retrieve objects from it, not store objects in it.)

Please note that whether public tables might not have effect on databases that do not allow to set permissions accordingly, for instance, in multi-tenant environments. To enable the sharing of objects, an administrator needs to first create a role names `R_USERS_PUBLIC` and assign it to all users who should be allowed to share objects.

Usage

`ida.list(type='public',user=NULL)`

Arguments

- `type`  
  The type (private or public) of the table. You can specify 'private' only if `user` is NULL or is set explicitly to your own user ID.

- `user`  
  The user ID of the owner of the R object storage table. If set to NULL, the user ID is that of the current user.

Value

A pointer to an R object storage table.
**Examples**

```r
## Not run:
# Create a pointer to the private R object storage table of the current user.
myPrivateObjects <- ida.list(type='private')

# Use the pointer created in the previous example to store a series of numbers in an object with
# the name 'series100' in the private R object storage table of the current user.
myPrivateObjects['series100'] <- 1:100

# Retrieve the object with the name 'series100' from the
# private R object storage table of the current user.
x <- myPrivateObjects['series100']

# Delete the object with name 'series100' from the
# private R object storage table of the current user.
myPrivateObjects['series100'] <- NULL

# List all objects in the private R object storage table of the current user.
names(myPrivateObjects)

# Return the number of objects in the private R object storage table of the current user.
length(myPrivateObjects)

# Create a pointer to the public R object storage table of the current user.
myPublicObjects <- ida.list(type="public")
```

## End(Not run)

**ida.list methods**

**Available methods for ida.list**

**Description**

**ida.list** objects provide methods that will behave exactly like or very similar to methods defined on a regular list. The following methods are currently supported: length, names, print.

For details see the documentation of **ida.list**.

**idaConnect, idaClose**  
*Open or closes a IDA database connection*

**Description**

These functions are used to open or close an existing IDA database connection.
Usage
idaConnect(dsn, uid = "", pwd = "", conType = "odbc")
idaClose(idaConn, conType = "odbc")

Arguments

dsn The DSN of the data base.
uid The user name.
pwd The password.
conType The connection type.
idaConn The connection object.

Details
Opens or closes a connection to a database. Currently, RODBC is used as underlying library, this
might change, however, in the future.

Examples
## Not run:
#Connect locally
con <- idaConnect('BLUBB','','')

#Close the connection
idaClose(con)
## End(Not run)

Description
Use these functions to create or drop a view that is based on a ida.data.frame.

Usage
idaCreateView(x)
idaDropView(v)

Arguments

x ida.data.frame for which a view is to be created.
v Name of the view to be dropped.
Details

The idaCreateView function creates a view from the specified IDA data frame. The idaDropView function drops the specified view.

Value

The idaCreateView function returns the view name. The idaDropView function does not return a value.

Examples

```r
## Not run:
idf <- ida.data.frame('IRIS')

# Create a view based on the IDA data frame
vname <- idaCreateView(idf)

# Drop the view
idaDropView(vname)

## End(Not run)
```

Miscellaneous tools

These functions simplify working with database tables.

Usage

`idaDeleteTable(table)`

`idaExistTable(tableName)`

`idaIsView(tableName)`

`idaGetValidTableName(prefix="DATA_FRAME_")`

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The name of a database table or a ida.data.frame.</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of a database table.</td>
</tr>
<tr>
<td>prefix</td>
<td>Keyword used to specify the prefix of a table name.</td>
</tr>
</tbody>
</table>
Details

Use the idaDeleteTable function to drop the specified table. The specified table must exist in the current database.

Use the idaExistTable function to determine whether the specified table exists in the database.

Use the idaGetValidTableName function to obtain a table name that is not yet in use. This name will be the specified or default prefix followed by a number, for example, data_frame_7.

Value

The idaDeleteTable function does not return a value.

The idaExistTable function returns a logical value (TRUE or FALSE) that indicates whether the specified table exists in the database.

The idaGetValidTableName function returns a string representing a table name.

Examples

```r
## Not run:

# Check whether a table with a given name exists
idaExistTable('IRIS')

# Create a pointer to the table
idf <- ida.data.frame('IRIS')

# Obtain a unique table name for a copy
copyTableName <- idaGetValidTableName(prefix = "COPY_")

# Create a copy of the original table
idfCopy <- as.ida.data.frame(as.data.frame(idf), copyTableName)

# Delete the copy again
idaDeleteTable(copyTableName)

## End(Not run)
```

Description

These functions allow to query, store and update data in the database. Usually, it is easier to use idaQuery.ida.data.frame and as.ida.data.frame instead of these methods.

They can be useful, however, if an explicit connection object is needed, e.g. if there are several connections to different databases.
**Usage**

idaDef(idaConn, query)
idaSave(idaConn, dfrm, tblName = "", rowName = "", conType = "odbc")
idaUpdate(db2Conn, updf, dfrm, idaIndex = "", conType = "odbc")

**Arguments**

- idaConn: The IDA connection object.
- db2Conn: The IDA connection object.
- query: A query.
- dfrm: A data.frame to store.
- tblName: Name of the table to which to store the data.
- rowName: Name of the row name column.
- updf: Name of the table to update.
- idaIndex: Name of the index column.
- conType: Type of the connection.

**Details**

idaDef, idaSave and idaUpdate are simple wrappers around the RODBC functions sqlQuery, sqlSave and sqlUpdate.

Usually, it is easier to use idaQuery.ida.data.frame and as.ida.data.frame instead of these methods.

**See Also**

idaQuery.ida.data.frame, as.ida.data.frame, sqlQuery, sqlSave, sqlUpdate

**Examples**

```r
## Not run:
# create connection to DB
con <- idaConnect("BLUDB", ",", ",")

# create data.frame from table
df <- idaDef(con, "SELECT * FROM IRIS")

# close the connection again
idaClose(con)

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

*Drop a predictive model from the database*

**Description**

Use this function to drop from the database a model that was created using the `idaNaiveBayes` or `idaKMeans` function.

**Usage**

```r
idaDropModel(modelname)
```

**Arguments**

- `modelname` The name of the predictive model to be dropped.

**Examples**

```r
# Not run:

drop the model with the name MYTREEMODEL
idaDropModel("KMEANSMODEL");

# End(Not run)
```

### idaInit

*Initialize the In-Database Analytics functions*

**Description**

This function initializes the In-Database Analytics functions.

**Usage**

```r
idaInit(con, jobDescription=NULL)
```

**Arguments**

- `con` An open RODBC connection.
- `jobDescription` Optional argument that allows to assign a description to the jobs submitted from the R session.

**Details**

Use an existing RODBC connection to initialize the IDA in-database analytics functions. All commands are sent through this connection.
Value

No value is returned.

Examples

```r
## Not run:

#Initialize the IDA Analytics functions
con <- idaConnect('BLUDB', '', '')

#Initialize the in-database functionality
idaInit(con)

## End(Not run)
```

### idaKMeans

#### k-means clustering

**Description**

This function generates a k-means clustering model based on the contents of a IDA data frame (`ida.data.frame`).

**Usage**

```r
idaKMeans(
  data,
  id,
  k=3,
  maxiter=5,
  distance="euclidean",
  outtable=NULL,
  randseed=12345,
  statistics=NULL,
  modelname=NULL
)
```

```r
## S3 method for class 'idaKMeans'
print(x,...)
## S3 method for class 'idaKMeans'
predict(object, newdata, id,...)
```

**Arguments**

- **data**: A IDA data frame that contains the input data for the function. The input IDA data frame must include a column that contains a unique ID for each row.
The idaKMeans function calculates the squared Euclidean distance between rows, and groups them into clusters. Initial clusters are chosen randomly using a random seed, and the results are adjusted iteratively until either the maximum number of iterations is reached or until two iterations return identical results. Variables with missing values are set zero for distance calculation.

The parameter `modelname` can be used to decide under which name the model is stored in the database. This name can be used to retrieve or drop the model later (`idaRetrieveModel`, `idaDropModel`).

The output of the print function for a `idaKMeans` object is:

- A vector containing a list of centers
- A vector containing a list of cluster sizes
- A vector containing a list of the number of elements in each cluster
- A data frame or the name of the table containing the calculated cluster assignments
- The within-cluster sum of squares (which indicates cluster density)
- The names of the slots that are available in the `idaKMeans` object

The `idaKMeans` function returns an object of class `idaKMeans` and `kmeans`.

See Also

`idaRetrieveModel`, `idaDropModel`, `idaListModels`
**idaListModels and idaModelExists**

**List all predictive models in the database**

**Examples**

```r
## Not run:

# Create ida data frame
idf <- ida.data.frame("IRIS")

# Create a kmeans model stored in the database as KMEANSMODEL
km <- idaKMeans(idf, id="ID", modelname="KMEANSMODEL")

# Print the model
print(km)

# Predict the model
pred <- predict(km, idf, id="ID")

# Inspect the predictions
head(pred)

## End(Not run)
```

**Description**

Use these function to list all models in the schema of the current user that were created using the `idaNaiveBayes` or `idaKMeans` functions or check whether a model with a specific name exists.

**Usage**

```r
idaListModels()
idaModelExists(modelname)
```

**Arguments**

- `modelname` The name of a predictive model.

**Value**

- `idaListModels` returns a data frame that contains a list of the predictive models that are stored in the current schema and information about each one.
- `idaModelExists` returns a boolean value depending on whether the model exists or not.
Examples

```r
## Not run:
# Get a list of all models
q <- idaListModels();

## End(Not run)
```

### idaLm

**Linear regression**

**Description**

This function performs a linear regression on the contents of a IDA data frame (`ida.data.frame`).

**Usage**

```r
idaLm(form, idadf, limit=25)
```

**Arguments**

- `form`: A formula object that specifies both the name of the column that contains the continuous target variable and either a list of columns separated by plus symbols or a single period (to specify that all other columns in the IDA data frame are to be used as predictors). The specified columns can contain continuous or categorical values. The specified formula cannot contain transformations.
- `idadf`: A IDA data frame that contains the input data for the function.
- `limit`: The maximum number of distinct values per categorical column. The default is 25.
- `x`: An object of the class `idaLm`.

**Details**

The `idaLm` function computes a linear regression model by extracting a covariance matrix and computing its inverse. This implementation is optimized for problems that involve a large number of samples and a relatively small number of predictors. The maximum number of columns is 87.

Missing values in the input table are ignored when calculating the covariance matrix. If this leads to undefined entries in the covariance matrix, the function fails. If the inverse of the covariance matrix cannot be computed (for example, due to correlated predictors), the Moore-Penrose generalized inverse is used instead.

The output of the `idaLm` function has the following attributes:

- `$coefficients`: is a vector with two values. The first value is the slope of the line that best fits the input data; the second value is its y-intercept.
**RSS** is the root sum square (that is, the square root of the sum of the squares).

*effects* is not used and can be ignored.

*rank* is the rank.

*df.residuals* is the number of degrees of freedom associated with the residuals.

*coefftab* is a vector with four values: - The slope and y-intercept of the line that best fits the input data - The standard error - The t-value - The p-value

*Loglike* is the log likelihood ratio.

*AIC* is the Akaike information criterion. This is a measure of the relative quality of the model.

*BIC* is the Bayesian information criterion. This is used for model selection.

### Value

The procedure returns a linear regression model in an object of class *ida.lm*.

### Examples

```r
## Not run:
# Create a pointer to table IRIS
idf <- ida.data.frame('IRIS')

# Calculate linear model in-db
lm1 <- ida.lm(SepalLength~SepalWidth, idf)
```

---

**idaMerge**

**Merge IDA data frames**

### Description

This function merges two IDA data frames (that is, two objects of the class *ida.data.frame*).

### Usage

```r
idaMerge(x, y, by=intersect(x@cols, y@cols), by.x=by, by.y=by,
         all=FALSE, all.x=all, all.y=all, sort=TRUE,
         suffixes=c("_x", "_y"), table=NULL)
```

### Arguments

- **x**: The first *ida.data.frame* object to be merged.
- **y**: The second *ida.data.frame* object to be merged.
- **by**: Specification of the common columns; see the *Details* section.
- **by.x**: Specification of the common columns; see the *Details* section.
- **by.y**: Specification of the common columns; see the *Details* section.
### Details

This function merges two IDA data frames on the columns that they have in common. The rows in the two data frames that match on the specified columns are extracted, and joined together. If there is more than one match, all possible matches contribute one row each. For the precise meaning of ‘match’.

If by or both by.x and by.y are of length 0 (a length zero vector or NULL), the result, r, is the Cartesian product of x and y, that is, a cross join.

If non-merged columns of the data frames have identical names and are to be included in the output, suffixes are appended to the names of the corresponding columns in the output to make their names.

Note that this function creates, in the current database, a view that corresponds to the output object. Within the current session, this view can be accessed using the same IDA data frame object. However, it is persistent and, after it is no longer needed, it must be dropped manually.

### Value

A `ida.data.frame` object.
See Also

ida.data.frame

Examples

## Not run:

idf <- ida.data.frame('IRIS')

# Perform a self-join
idf2 <- idaMerge(idf, idf, by="ID")

## End(Not run)

### idaNaiveBayes

#### Naive Bayes Classifier

**Description**

This function generates a Naive Bayes classification model based on the contents of a IDA data frame (ida.data.frame).

**Usage**

idaNaiveBayes(form, data, id="id", modelname=NULL)

## S3 method for class 'idaNaiveBayes'
predict(object, newdata, id,...)

## S3 method for class 'idaNaiveBayes'
print(x,...)

**Arguments**

- **form**: A formula object that describes the model to fit.
- **data**: A ida.data.frame object.
- **id**: The name of the column that contains unique IDs.
- **modelname**: Name for the model. Will be created automatically unless specified otherwise.
- **object**: An object of the class idaNaiveBayes to be predicted.
- **newdata**: A IDA data frame that contains the data to which to apply the model.
- **x**: An object of the class idaNaiveBayes to be printed.
- **...**: Additional parameters to pass to the predict method.
Details

idaNaiveBayes builds a Naive Bayes classification model, thus a model that assumes independence of input variables with respect to the target variable.

Continuous input variables are discretized using equal width discretization. Missing values are ignored on a record and attribute level when calculating the conditional probabilities.

Value

The function idaNaiveBayes returns an object of class "idaNaiveBayes" and "naiveBayes" compatible with Naive Bayes objects produced by the e1071 package.

The predict.idaNaiveBayes method applies the model to the data in a table and returns a IDA data frame that contains a list of tuples, each of which comprises one row ID and one prediction.

Examples

```r
## Not run:
# Create data frame
idf <- ida.data.frame("IRIS")

# Create a naive bayes model
nb <- idaNaiveBayes(Species~SepalLength,idf,"ID")

# Print the model
print(nb)

# Apply the model to data
idf2 <- predict(nb,idf,"ID")

# Inspect the results
head(idf2)

## End(Not run)
```

idaQuery, idaScalarQuery

Run an SQL query on the database

Description

Use these functions to run any SQL query on the database and put the results into a data frame.

Usage

idaQuery(..., as.is=TRUE)
idaScalarQuery(..., as.is=TRUE)
Arguments

... Any number of query parts which are passed to paste.

as.is Specifies whether the result columns are to be converted using RODBC type conversions (as.is=FALSE) or left unconverted (as.is=TRUE). For more information about RODBC type conversions, see the descriptions of the functions sqlGetResults and type.convert.

Details

All parts of the input query are concatenated with paste(..., sep="") and the result is passed to the database.

Value

The idaQuery function returns a data frame that contains the result of the specified query. The idaScalarQuery function returns the result of the specified query coerced to a single scalar value.

Examples

```r
# Not run:
#idaScalarQuery returns a single value
v <- idaScalarQuery("SELECT COUNT(*) FROM IRIS")

#idaQuery returns a data.frame
df <- idaScalarQuery("SELECT * FROM IRIS")

#idaQuery and idaScalarQuery automatically paste all arguments into a single query
#This is convenient if you use variables
tableName <- "IRIS"
df <- idaQuery("SELECT COUNT(*) FROM ", tableName)

# End(Not run)
```

idaRetrieveModel Retrieve a predictive model from the database

Description

Use this function to retrieve from the database a model that was created using the idaNaiveBayes or idaKMeans function.

Usage

idaRetrieveModel(modelname)
Arguments

modelname  The name of the predictive model to be retrieved.

Value

This function returns an R object that contains a representation of the retrieved model. The class of the returned object depends on the function that was used to create the model.

Examples

## Not run:

#Retrieve the model with name "MYKMEANSMODEL" from the database
trCopy <- idaRetrieveModel("KMEANSMODEL");

## End(Not run)

idaSample  Taking a random sample from a IDA data frame

Description

This function draws a random sample from a IDA data frame (that is, an object of the class ida.data.frame).

Usage

idaSample(bdf, n, stratCol=NULL, stratVals=NULL, stratProbs=NULL, 
           dbPreSamplePercentage=100, fetchFirst=F);

Arguments

bdf  The IDA data frame from which the sample is to be drawn.
n  The number of rows of sample data to be retrieved.
stratCol  For stratified sampling, the column that determines the strata.
stratVals  For stratified sampling, a vector of values that determine the subset of strata from which samples are to be drawn.
stratProbs  For stratified sampling, a vector of explicit sampling probabilities. Each value corresponds to a value of the vector specified for stratVals.
dbPreSamplePercentage  The percentage of the IDA data frame from which the sample is to be drawn (see details).
fetchFirst  Fetch first rows instead of using random sample.
Details

If stratCol is specified, a stratified sample based on the contents of the specified column is taken. Unless stratVals is also specified, each unique value in the column results in one stratum. If stratVals is also specified, only the values it specifies result in strata, and only rows that contain one of those values are included in the sample; other rows are ignored.

Unless stratProbs is also specified, the number of rows retrieved for each stratum is proportional to the size of that stratum relative to the overall sample.

To undersample or oversample data, use stratProbs to specify, for each value of stratVals, the fraction of the rows of the corresponding stratum that are to be included in the sample.

For each stratum, the calculated number of rows is rounded up to the next highest integer. This ensures that there is at least one sample for each stratum. Consequently, the number of samples that is returned might be higher than the value specified for n.

The value of dbPreSamplePercentage is a numeric value in the range 0-100 that represents the percentage of the IDA data frame that is to serve as the source of the sample data. When working with an especially large IDA data frame, specifying a value smaller than 100 improves performance, because less data must be processed. However, the proportionality of the pre-sampled data might vary from that of the complete data, and this would result in a biased sample. It can even happen that entire strata are excluded from the final sample.

When fetchFirst is set to TRUE, the sample values of each stratum are taken in the order in which they are returned from the database rather than randomly. This is usually much faster than random sampling, but can introduce bias.

Value

An object of class data.frame that contains the sample.

Examples

```r
## Not run:
idf<-.ida.data.frame('IRIS')

#Simple random sampling
df <- idaSample(idf,10)

#Stratified sample
df <- idaSample(idf,10,'Species')

## End(Not run)
```

Description

Returns a data frame that contains the names of the tables contained in the current database.
Usage

idaShowTables(showAll=FALSE)

Arguments

showAll List all tables that are listed in the catalog of the current database (TRUE) or only those tables that are in the current schema (FALSE).

Value

A data frame with the columns Schema, Name, Owner, and Type.

Examples

### Not run:

#Get a list of all tables in the current schema
tabs <- idaShowTables()

### End(Not run)

idaTable  

**In-Database Cross Tabulation and Table Creation**

Description

Function used to build a contingency table of the counts at each combination of factor levels based on the contents of a IDA data frame (ida.data.frame).

Usage

idaTable(idadf,max.entries=1000)

Arguments

idadf A IDA data frame that contains the input data for the function.

max.entries The maximum number of entries. If the cross product of all columns exceeds this number, an error will be thrown.

Details

idaTable uses the cross-classifying factors to build a contingency table of the counts at each combination of categorical values in all categorical columns of the ida.data.frame passed as input.

Value

The idaTable function returns a contingency table, an object of class "table".
Examples

## Not run:

```r
#Create a pointer to the table IRIS
idf <- ida.data.frame('IRIS')

#Add a column
idf$SepalLength4 <- ifelse(idf$SepalLength>4,'t','f')

#Calculate the cross-table between Species and SepalLength4
idaTable(idf[,c('Species','SepalLength4')])
```

## End(Not run)
Index

! [, ida.col.def-method (ida.col.def methods), 3
! [, ida.list-method (ida.list methods), 7
[, ida.data.frame, ANY, ANY, ANY-method
 (ida.data.frame methods), 4
[, ida.data.frame-method
 (ida.data.frame methods), 4
[, ida.list, ANY, ANY, ANY-method
 (ida.list methods), 7
[, ida.list-method (ida.list methods), 7
<-, [, ida.list, ANY, ANY, ANY-method
 (ida.list methods), 7
<-, [, ida.list-method (ida.list methods), 7
$, [, ida.data.frame-method
 (ida.data.frame methods), 4
$, [, ida.list-method (ida.list methods), 7
$<-, [, ida.data.frame-method
 (ida.data.frame methods), 4
$<-, [, ida.list-method (ida.list methods), 7

as.character, [, ida.col.def-method
 (ida.col.def methods), 3
as.data.frame, 3
as.data.frame, ida.data.frame-method
 (ida.data.frame methods), 4
as.ida.data.frame, 2, 10, 11
as.integer, [, ida.col.def-method
 (ida.col.def methods), 3
as.numeric, [, ida.col.def-method
 (ida.col.def methods), 3
as.vector, [, ida.col.def-method
 (ida.col.def methods), 3
as.vector, [, ida.col.def-method
 (ida.col.def methods), 3

colnames, [, ida.data.frame-method
 (ida.data.frame methods), 4
cor, [, ida.data.frame-method
 (ida.data.frame methods), 4
cov, ANY, ANY-method (ida.data.frame methods), 4
cov, ANY, ida.data.frame-method
 (ida.data.frame methods), 4
cov, ida.data.frame, ANY-method
 (ida.data.frame methods), 4
cov, ida.data.frame, ida.data.frame-method
 (ida.data.frame methods), 4
db.is.null (ida.col.def methods), 3
dim, [, ida.data.frame-method
 (ida.data.frame methods), 4
head, [, ida.data.frame-method
 (ida.data.frame methods), 4
hist, [, ida.data.frame-method
 (ida.data.frame methods), 4
ibmdbR (ibmdbR-package), 2
ibmdbR-package, 2
ida.col.def-methods, 3
ida.data.frame, 2–4, 8, 10, 11, 13, 16, 17,
  19, 22, 24
ida.data.frame (ida.data.frame
  is.ida.data.frame), 4
ida.data.frame-methods, 4
ida.data.frame, is.ida.data.frame, 4
ida.list, 2, 6, 7
ida.list-methods, 7
idaClose (idaConnect, idaClose), 7
idaConnect (idaConnect, idaClose), 7
idaConnect, idaClose, 7
idaCreateView (idaCreateView,
  idaDropView), 8
idaCreateView, idaDropView, 8
idaDeleteTable (idaDeleteTable,
  idaExistTable,
  idaGetValidTableName), 9

26
Index

idaDeleteTable, idaExistTable, idaGetValidTableName, 9
idaDropModel, 12, 14
idaDropView (idaCreateView, idaDropView), 8
idaExistTable (idaDeleteTable, idaExistTable, idaGetValidTableName), 9
idaGetValidTableName (idaDeleteTable, idaExistTable, idaGetValidTableName), 9
idaInit, 12
idaIsView (idaDeleteTable, idaExistTable, idaGetValidTableName), 9
idaKMeans, 12, 13, 15, 21
idaListModels, 14
idaListModels (idaListModels and idaModelExists), 15
idaListModels and idaModelExists, 15
idaLm, 16
idaMerge, 17
idaModelExists (idaListModels and idaModelExists), 15
idaNaiveBayes, 12, 15, 19, 21
idaQuery, 10, 11
idaQuery (idaQuery, idaScalarQuery), 20
idaQuery, idaScalarQuery, 20
idaRetrieveModel, 14, 21
idaSample, 22
idaSave (idaDF, idaSave, idaUpdate), 10
idaScalarQuery (idaQuery, idaScalarQuery), 20
idaShowTables, 23
idaTable, 24
idaUpdate (idaDF, idaSave, idaUpdate), 10
ifelse, ida.col.def-method (ida.col.def methods), 3
is.ida.data.frame (ida.data.frame, is.ida.data.frame), 4
is.ida.list (ida.list), 6
length, ida.data.frame-method (ida.data.frame methods), 4
length, ida.list-method (ida.list methods), 7
max, ida.data.frame-method (ida.data.frame methods), 4
mean, ida.data.frame-method (ida.data.frame methods), 4
min, ida.data.frame-method (ida.data.frame methods), 4
names, ida.data.frame-method (ida.data.frame methods), 4
names, ida.list-method (ida.list methods), 7
NCOL, ida.data.frame-method (ida.data.frame methods), 4
NROW, ida.data.frame-method (ida.data.frame methods), 4
predict.idaKMeans (idaKMeans), 13
predict.idaNaiveBayes (idaNaiveBayes), 19
print, ida.col.def-method (ida.col.def methods), 3
print, ida.data.frame-method (ida.data.frame methods), 4
print, ida.list-method (ida.list methods), 7
print.idaKMeans (idaKMeans), 13
print.idaLm (idaLm), 16
print.idaNaiveBayes (idaNaiveBayes), 19
sd, ida.data.frame-method (ida.data.frame methods), 4
sqlQuery, 11
sqlSave, 11
sqlUpdate, 11
summary, ida.data.frame-method (ida.data.frame methods), 4
var, ida.data.frame-method (ida.data.frame methods), 4
length, ida.data.frame-method (ida.data.frame methods), 4
length, ida.list-method (ida.list methods), 7