Package 'toaster'

August 29, 2016

Type Package

Title Big Data in-Database Analytics with Teradata Aster

Version 0.5.2

Description A consistent set of tools to perform in-database analytics on Teradata Aster Big Data Discovery Platform. toaster (a.k.a 'to Aster') embraces simple 2-step approach: compute in Aster - visualize and analyze in R. Its `compute` functions use combination of parallel SQL, SQL-MR and SQL-GR executing in Aster database - highly scalable parallel and distributed analytical platform. Then `create` functions visualize results with boxplots, scatterplots, histograms, heatmaps, word clouds, maps, networks, or slope graphs. Advanced options such as faceting, coloring, labeling, and others are supported with most visualizations.

SystemRequirements Teradata Aster Database 5.20 (AD5.20) or higher, Teradata Aster Analytical Foundation 5.20 (AA5.20) or higher.

Depends R (>= 3.0), RODBC (>= 1.3-12)

Imports plyr (>= 1.8.3), reshape2 (>= 1.4.1), ggplot2 (>= 2.1.0), ggthemes (>= 3.0.2), scales, RColorBrewer (>= 1.1-2), grid, wordcloud (>= 2.5), ggmap (>= 2.6.1), GGally (>= 1.0.1), memoise (>= 1.0.0), slam (>= 0.1-32), foreach (>= 1.4.3), stats, network (>= 1.13.0)

Suggests testthat (>= 0.9.1-1), tm, doParallel, ggnetwork

License GPL-2

URL https://github.com/teradata-aster-field/toaster

BugReports https://github.com/teradata-aster-field/toaster/issues

Collate 'toaster.R' 'misc.R' 'utils.R' 'tools.R' 'computeCorrelations.R' 'computeHistogram.R' 'computeHeatmap.R' 'maps.R' 'showData.R' 'computeAggregates.R' 'computeBarchart.R' 'computeSample.R' 'computePercentiles.R' 'computeLm.R' 'computeTfIdf.R' 'textParsers.R' 'computeKmeans.R' 'plotting.R' 'plottingKmeans.R' 'showGraph.R' 'computeGraph.R' 'computeGraphClusters.R' 'validateGraph.R'

RoxygenNote 5.0.1

NeedsCompilation no

Author Gregory Kanevsky [aut, cre]

Maintainer Gregory Kanevsky <gregory.kanevsky@teradata.com>

Repository CRAN

Date/Publication 2016-07-29 23:47:59

R topics documented:

`computeAggregates` *Compute aggregate values.*

Description

Compute aggregates using SQL `SELECT...GROUP BY` in Aster. Aggregates may be any valid SQL expressions (including SQL `WINDOW` functions) in context of group columns (parameter by). Neither SQL `ORDER BY` nor `LIMIT` clauses are supported (use `computeBarchart` when they are required).

Usage

```
computeAggregates(channel, tableName, aggregates = c("COUNT(*) cnt"),
  by = vector(), where = NULL, stringsAsFactors = FALSE, test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`tableName`	table name
`aggregates`	vector of SQL aggregates to compute. Aggregates may have optional aliases like in `"AVG(era) avg_era"`
`by`	character vecotr of column names and/or expressions on which grouping is performed (with SQL `GROUP BY ...`). Each can be a column or a valid SQL non-aggregate expression with otional alias separated by space (e.g. `"UPPER(car_make) make"`).

`where` specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

`stringsAsFactors` logical: should character vectors returned as part of results be converted to factors?

`test` logical: if TRUE show what would be done, only (similar to parameter `test` in RODBC functions like sqlQuery and sqlSave).

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# compute average team rank and attendance by decade
data = computeAggregates(channel = conn, tableName = "teams_enh",
               by = c("name || ', ' || park teamname", "lgid", "teamid", "decadeid"),
               aggregates = c("min(name) name", "min(park) park", "avg(rank) rank",
                              "avg(attendance) attendance"))

# compute total strike-outs for each team in decades starting with 1980
# and also percent (share) of team strikeouts within a decade
data = computeAggregates(channel = conn, "pitching_enh",
               by = c("teamid", "decadeid"),
               aggregates = c("sum(so) so",
                         "sum(so)/(sum(sum(so)) over (partition by decadeid)) percent"),
               where = "decadeid >= 1980")
}
```

`computeBarchart` *Compute one or more aggregates across single class.*

Description

Compute aggregates across category class represented by the table column. Values are one or more SQL aggregates that are valid expressions with `GROUP BY <class column>`. Class column usually is of character or other discrete type. Typical example is computing a bar chart for the column using SQL `COUNT(*) ... GROUP BY` - hence the name of the function. Result is a data frame to visualize as bar charts or heatmaps (see creating visualizations with `createHistogram` and `createHeatmap`).

Usage

```
computeBarchart(channel, tableName, category, aggregates = "COUNT(*) cnt",
  where = NULL, orderBy = NULL, top = NULL, by = NULL,
  withMelt = FALSE, stringsAsFactors = FALSE, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	table name
category	column name or expression associated with categories. Name may be valid SQL expression and can contain otional alias (e.g. "UPPER(car_make) make")
aggregates	SQL aggregates to compute. Each aggregate corresponds to category value. Aggregates may have optional aliases like in "AVG(era) era"
where	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside WHERE clause).
orderBy	list of column names, aliases, references or their combinations to use in SQL ORDER BY clause. Use in combination with top below to compute only limited number of results in certain order.
top	if specified indicates number of bars to include in bar plot. In combination with orderBy it works as computing first top results in certain order.
by	for optional grouping by one or more columns for faceting or alike (effectively these elements will be part of GROUP BY ...)
withMelt	logical if TRUE then uses **reshape2** melt to transform result data frame aggregate values into a molten data frame
stringsAsFactors	logical: should columns returned as character and not excluded by as.is and not converted to anything else be converted to factors?
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions like sqlQuery and sqlSave).

Value

Data frame to use for bar chart plots with createHistogram.

See Also

computeHistogram, createHistogram

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# Compute average team season era, walks, and hits for each decade starting with 1980
computeBarchart(channel=conn, "teams_enh", "teamid team",
               aggregates=c("avg(era) era", "avg(bb) bb", "avg(h) h"),
               where="yearid >=1980", by=c("decadeid"))

# multipe aggregates in the same bar chart (with melt)
bc = computeBarchart(channel=conn, tableName="pitching_enh", category="teamid",
                   aggregates=c("AVG(era) era", "AVG(whip) whip"), withMelt=TRUE,
```

```
                    where="yearid >= 2000 and lgid='AL'")

# adding facets by decadeid
bc = computeBarchart(channel=conn, tablelName="pitching_enh", category="teamid",
                     aggregates=c("AVG(era) era", "AVG(whip) whip", "AVG(ktobb) ktobb"),
                     where="yearid >= 1990 and lgid='AL'", by="decadeid", withMelt=TRUE)
}
```

`computeClusterSample` *Random sample of clustered data*

Description

Random sample of clustered data

Usage

```
computeClusterSample(channel, km, sampleFraction, sampleSize, scaled = FALSE,
  includeId = FALSE, test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`.
`km`	an object of class `"toakmeans"` obtained with `computeKmeans`.
`sampleFraction`	one or more sample fractions to use in the sampling of data. (multipe sampling fractions are not yet supported.)
`sampleSize`	total sample size (applies only when `sampleFraction` is missing).
`scaled`	logical: indicates if original (default) or scaled data returned.
`includeId`	logical indicates if sample should include the key uniquely identifying each data row.
`test`	logical: if TRUE show what would be done, only (similar to parameter `test` in **RODBC** functions: sqlQuery and sqlSave).

Value

`computeClusterSample` returns an object of class `"toakmeans"` (compatible with class `"kmeans"`).

See Also

`computeKmeans`

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
                aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                    id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                    include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                    centroidTableName='kmeans_test_centroids',
                    where="yearid > 2000")
km = computeClusterSample(conn, km, 0.01)
km
createClusterPairsPlot(km, title="Batters Clustered by G, H, R", ticks=FALSE)
}
```

`computeCorrelations` *Compute correlation between pairs of columns.*

Description

Compute global correlation between all pairs of numeric columns in table. Result includes all pairwise combinations of numeric columns in the table, with optionally limiting columns to those in the parameter `include` or/and excluding columns defined by parameter `except`. Limit computation on the table subset defined with `where`. Use `output='matrix'` to produce results in matrix format (compatible with function `cor`).

Usage

```
computeCorrelations(channel, tableName, tableInfo, include = NULL,
  except = NULL, where = NULL, by = NULL, output = c("data.frame",
  "matrix"), test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`tableName`	database table name
`tableInfo`	pre-built summary of data to use (must have with `test=TRUE`)
`include`	a vector of column names to include. Output never contains attributes other than in the list. When missing all columns from `tableInfo` included.
`except`	a vector of column names to exclude. Output never contains attributes from the list.
`where`	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

by	vector of column names to group results by one or more table columns for faceting or alike (optional).
output	Default output is a data frame of column pairs with correlation coefficient (melt format). To return correlation matrix compatible with function cor use 'matrix' .
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions like sqlQuery and sqlSave).

Value

data frame with columns:

- *corr* pair of 1st and 2d columns "column1:column2"
- *value* computed correlation value
- *metric1* name of 1st column
- *metric2* name of 2d column
- *sign* correlation value sign sign(value) (-1, 0, or 1)

Note that while number of correlations function computes is choose(N, 2), where N is number of table columns specified, resulting data frame contains twice as many rows by duplicating each correlation value with swaped column names (1st column to 2d and 2d to 1st positions). This makes resulting data frame symmetrical with respect to column order in pairs and is necessary to correctly visualize correlation matrix with createBubblechart.

See Also

createBubblechart and showData.

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

cormat = computeCorrelations(channel=conn, "pitching_enh", sqlColumns(conn, "pitching_enh"),
                         include = c('w','l','cg','sho','sv','ipouts','h','er','hr','bb',
                                     'so','baopp','era','whip','ktobb','fip'),
                              where = "decadeid = 2000", test=FALSE)
# remove duplicate correlation values (no symmetry)
cormat = cormat[cormat$metric1 < cormat$metric2, ]
createBubblechart(cormat, "metric1", "metric2", "value", label=NULL, fill="sign")

# Grouped by columns
cormatByLg = computeCorrelations(channel=conn, "pitching_enh",
                                 include=c('w','sv','h','er','hr','bb','so'),
                                 by=c('lgid','decadeid'),
                                 where = "decadeid >= 1990")

createBubblechart(cormatByLg, "metric1", "metric2", "value",
```

```
                  label=NULL, fill="sign", facet=c('decadeid','lgid'),
                  title="Correlations by Leagues and Decades",
                  defaultTheme = theme_wsj(), legendPosition = 'none')
}
```

`computeEgoGraph` *Find the vertices not farther than a given limit from another fixed vertex, and create egographs (subgraphs) with the given order parameter.*

Description

Find the vertices not farther than a given limit from another fixed vertex, and create egographs (subgraphs) with the given order parameter.

Usage

```
computeEgoGraph(channel, graph, ego, order = 1, mode = "all",
  createDistanceAttr = TRUE, distanceAttrname = "ego.distance",
  vertexWhere = graph$vertexWhere, edgeWhere = graph$edgeWhere,
  allTables = NULL, test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`graph`	an object of class `'toagraph'` referencing graph tables in Aster database.
`ego`	list of vertices for which the calculation of corresponding ego graphs is performed.
`order`	integer giving the order of the ego graph neighborhood.
`mode`	character constant, it specifies how to use the direction of the edges if a directed graph is analyzed. For `'out'` only the outgoing edges are followed, so all vertices reachable from the source vertex in at most order steps are counted. For `'in'` all vertices from which the source vertex is reachable in at most `order` steps are counted. `'all'` ignores the direction of the edges. This argument is ignored for undirected graphs.
`createDistanceAttr`	logical: indicates if vertices should receive attribute with the distance to ego graph centeral vertex.
`distanceAttrname`	name of the vertex distance attribute.
`vertexWhere`	SQL WHERE clause limiting data from the vertex table. This value when not null overrides corresponding value `vertexWhere` from `graph` (use SQL as if in WHERE clause but omit keyword WHERE).
`edgeWhere`	SQL WHERE clause limiting data from the edge table. This value when not null overrides corresponding value `edgeWhere` from `graph` (use SQL as if in WHERE clause but omit keyword WHERE).

allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).

Examples

```
if(interactive()) {
library(GGally)

policeGraphDi = toaGraph(vertices = "dallaspolice_officer_vertices",
                         edges = "dallaspolice_officer_edges_di",
                         directed = TRUE,
                         key = "officer", source = "officer1", target = "officer2",
                         vertexAttrnames = c("offense_count"),
                         edgeAttrnames = c("weight"))

# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

setVertexColor <- function(graph, vertex, color="red", default="grey") {
  graph %v% "color" =
  ifelse(get.vertex.attribute(graph, "vertex.names") == as.character(vertex),
         color, default)

  return(graph)
}

topPagerankPolice = computeGraphMetric(conn, policeGraphDi, type='pagerank', top=3)
egoCenters = as.list(as.character(topPagerankPolice$key))

egoGraphsTopPagerank = computeEgoGraph(conn, policeGraphDi, order = 1, ego = egoCenters)

egoGraph = setVertexColor(egoGraphsTopPagerank[[1]], egoCenters[[1]])
ggnet2(egoGraph, node.label="vertex.names",  node.size="offense_count",
       legend.position="none", color="color")

egoGraph = setVertexColor(egoGraphsTopPagerank[[2]], egoCenters[[2]])
ggnet2(egoGraph, node.label="vertex.names",  node.size="offense_count",
       legend.position="none", color="color")

egoGraph = setVertexColor(egoGraphsTopPagerank[[3]], egoCenters[[3]])
ggnet2(egoGraph, node.label="vertex.names",  node.size="offense_count",
       legend.position="none", color="color")
}
```

---

computeGraph *Materialize Aster graph as network object in R.*

Description

Results in network object representation of the graph stored in Aster tables. Usually in Aster database a graph is represented using a pair of vertice and edge tables.

Usage

```
computeGraph(channel, graph, v = NULL, vertexWhere = graph$vertexWhere,
  edgeWhere = graph$edgeWhere, allTables = NULL, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
graph	an object of class 'toagraph' referencing graph tables in Aster database.
v	a SQL SELECT that returns key values or a list of key values (corresponding to the vertex.names attribute) of the vertices to include in the graph. When not NULL this guarentees that no other vertices or edges between other vertices are included in the resulting network.
vertexWhere	optionally, a SQL WHERE clause to subset vertex table. When not NULL it overrides vertexWhere condition from the graph.
edgeWhere	optionally, a SQL WHERE clause to subset edge table. When not NULL it overrides edgeWhere condition from the graph.
allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).

Details

Use caution when computing network objects stored in Aster with this function as data may include considerable amount of vertices and edges which are too large to load into a memory.

Value

network class object materializing an Aster graph represented by toaGraph.

Examples

```
if(interactive()) {
library(GGally)

policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
                         directed = FALSE, key = "officer",
                         source = "officer1", target = "officer2",
                       vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")
```

```
# create network object and visualize with ggplot2
net1 = computeGraph(conn, policeGraphUn)
ggnet2(net1, node.label="vertex.names", node.size="offense_count",
       legend.position="none")

# network object with filters and color attribute
net2 = computeGraph(conn, policeGraphUn, vertexWhere = "officer ~ '[A-Z].*'",
                    edgeWhere = "weight > 0.36")
net2 %v% "color" = substr(get.vertex.attribute(net2, "vertex.names"), 1, 1)
ggnet2(net2, node.label="vertex.names", node.size="offense_count",
       size.cut=TRUE, node.color="color", legend.position="none",
       palette = "Set2")

# networ object for subgraph of top degree vertices
topDegree = computeGraphMetric(conn, policeGraphUn, type="degree", top=50)
net3 = computeGraph(conn, policeGraphUn, v=as.list(as.character(topDegree$key)))
net3 %v% "degree" = topDegree[match(get.vertex.attribute(net3, "vertex.names"),
                                    topDegree$key), "degree"]
ggnet2(net3, node.label="vertex.names", node.size="degree",
       legend.position="none")

}
```

`computeGraphClusters` *Pefrom graph clustering of various types.*

Description

Graph clustering (or decomposition) divides graph into set of subgraphs that span whole graph. Depending on the `type` argument the subgraphs coudl be either non-intersecting or overlapping. Available types of decomposition include finding connected componenets, modularity clustering.

Usage

```
computeGraphClusters(channel, graph, type = "connected",
  createMembership = FALSE, includeMembership = FALSE, weight = FALSE,
  vertexWhere = graph$vertexWhere, edgeWhere = graph$edgeWhere,
  distanceTableName = NULL, membershipTableName = NULL, schema = NULL,
  allTables = NULL, test = FALSE, ...)
```

Arguments

`channel`	connection object as returned by `odbcConnect`.
`graph`	an object of class `'toagraph'` referencing graph tables in Aster database.
`type`	specifies type of clustering or community detection to perform.
`createMembership`	logical indicates if vertex cluster membership table should be created (see `membershipTableName`). Currently, you must set it to `TRUE` if cluster membership data (see `includeMembership`)

expected in the result. Also, required if operations that create graphs corresponding to some of the clusters to be performed later.

`includeMembership`
: logical indicates if result should contain vertex cluster membership information. Currently, only supported when `createMembership` is `TRUE`. WARNING: including cluster membership may result in very large data set returned from Aster into memory.

`weight`
: logical or character: if logical then `TRUE` indicates using `'weight'` edge attribute, otherwise no weight used. If character then use as a name for the edge weight attribute. The edge weight may apply with types `'clustering'`, `'shortestpath'` and centrality measures.

`vertexWhere`
: optionally, a `SQL WHERE` clause to subset vertex table. When not `NULL` it overrides `vertexWhere` condition from the `graph`.

`edgeWhere`
: optionally, a `SQL WHERE` clause to subset edge table. When not `NULL` it overrides `edgeWhere` condition from the `graph`.

`distanceTableName`
: this table will contain distances between vertices (or other corresponding metrics associated with community detection algorithm chosen). By default, random table name that begins with `toa_temp_graphcluster_distance` is generated.

`membershipTableName`
: when `createMembership` is `TRUE` then this table will contain vertex cluster membership information. By default, random table name that begins with `toa_temp_graphcluster_memb` is generated. This argument is ignored when `createMembership` is `FALSE`.

`schema`
: name of Aster schema for the table name arguments `distanceTableName` and `membershipTableName`. There are two distinct approaches to providing table names: one that uses explicity schema name using this argument and another when table names already contain schema followed by dot and table name. The latter method is not applicable when generating randon table name with schema.

`allTables`
: pre-built information about existing tables.

`test`
: logical: if TRUE show what would be done, only (similar to parameter `test` in **RODBC** functions: sqlQuery and sqlSave).

`...`
: other arguments passed on to Aster graph functions except for `EDGEWEIGHT` argument - use argument `weight` instead. Aster function areguments are not case-sensetive.

Value

`computeGraphClusters` returns an object of class `"toacommunities"` (compatible with both class `"communities"` and the value returned by `clusters` - all from the package **igraph**). It is a list with the following components:

`membership` numeric vector giving the cluster (component or community) id to which each vertex belongs.

`csize` **and** `sizes` numeric vector giving the sizes of the clusters.

`no` **and** `length` numeric constant, the number of clusters.

`algorithm` gives the name of the algorithm that was used to calculate the community structure.

`id` integer vector of cluster ids from 1 to number `no`.

`componentid` character vector of cluster names (or component ids) where names are derived from the cluster elements and naming convention differs for each community type.

`distance` numeric vector of average distances within clusters.

`diameter` numeric vector of the maximum distances with clusters.

`graph` original graph object that identifies a graph for which clusters are crated.

`weight` see argument `weight` above.

`vertexWhere` see argument `vertexWhere` above.

`edgeWhere` see argument `edgeWhere` above.

`distanceTableName` Aster table name containing graph distances (applies to connected components only).

`membershipTableName` (optional) Aster table name containing graph vertex to cluster memberships.

`time` An object of class `proc_time` with user, system, and total elapsed times for the `computeGraphClusters` function call.

Examples

```
if(interactive()) {

# undirected graph
policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
     directed = FALSE, key = "officer", source = "officer1", target = "officer2",
     vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

communities = computeGraphClusters(conn, policeGraphUn, type="connected",
                                   createMembership = TRUE, includeMembership = TRUE,
                                   distanceTableName = "public.shortestpathdistances",
                                   membershipTableName = "public.clustermembership")

# get first 5 largest connected components as graphs
cluster_graphs = computeGraphClustersAsGraphs(conn, communities = communities, ids = 1:5)

# visualize component 2
library(GGally)
ggnet2(cluster_graphs[[2]], node.label="vertex.names", node.size="offense_count",
       node.color="color", legend.position="none")

# compute connected components for certain type of subgraph that
# includes only verteics that start with the letters
communities2 = computeGraphClusters(conn, policeGraphUn, type="connected", membership = TRUE,
                                    distanceTableName = "public.shortestpathdistances",
                                    vertexWhere = "officer ~ '[A-Z].*'",
                                    edgeWhere = "weight > 0.36")
}
```

computeGraphClustersAsGraphs
Creates list of graphs for each specified component.

Description

Based on the decomposition specified by communities object (see computeGraphClusters) materiazlies produced clusters as graph objects from Aster graph tables.

Usage

```
computeGraphClustersAsGraphs(channel, communities, ids = NULL,
  componentids = NULL, allTables = NULL, test = FALSE, parallel = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect.
communities	community object returned by computeGraphClusters.
ids	integer vector with cluster integer ids (from 1 to N, where N is the number of clusters). At least one value for this or componentids must be specified.
componentids	character vector with cluster component ids assigned during community generation with computeGraphClusters. These component ids are derived from one of the vertex name (likely first vertex when ordered alphabetically). At least one value for this or ids must be specified.
allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).
parallel	logical: enable parallel calls to Aster database. This option requires parallel backend enabled and registered (see in examples). Parallel execution requires ODBC channel obtained without explicit password: either with odbcConnect(dsn) or odbcDriverConnect calls, but not with odbcConnect(dsn, user, password).

Value

list of network objects materializing specified clusters (communities) represented by communities object.

Examples

```
if(interactive()) {

# undirected graph
policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
     directed = FALSE, key = "officer", source = "officer1", target = "officer2",
     vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))
```

```
communities = computeGraphClusters(conn, policeGraphUn, type="connected",
                                   createMembership = TRUE, includeMembership = TRUE,
                                   distanceTableName = "public.shortestpathdistances",
                                   membershipTableName = "public.clustermembership")

# get first 5 largest connected components as graphs
cluster_graphs = computeGraphClustersAsGraphs(conn, communities = communities, ids = 1:5)

# visualize component 2
library(GGally)
ggnet2(cluster_graphs[[2]], node.label="vertex.names", node.size="offense_count",
       node.color="color", legend.position="none")

}
```

---

computeGraphHistogram *Compute various statistic distributions on graph edges and vertices.*

Description

Compute various statistic distributions on graph edges and vertices.

Usage

```
computeGraphHistogram(channel, graph, type = "degree", weight = FALSE,
  binMethod = "manual", numbins = NULL, binsize = NULL,
  startvalue = NULL, endvalue = NULL, vertexWhere = graph$vertexWhere,
  edgeWhere = graph$edgeWhere, allTables = NULL, test = FALSE, ...)
```

Arguments

channel	connection object as returned by odbcConnect
graph	an object of class 'toagraph' referencing graph tables in Aster database.
type	choose between graph measures to compute histogram distribution for: 'degree', 'clustering', 'sho
weight	logical or character: if logical then TRUE indicates using 'weight' edge attribute, otherwise no weight used. If character then use as a name for the edge weight attribute. The edge weight may apply with types 'clustering', 'shortestpath' and centrality measures.
binMethod	one of several methods to determine number and size of bins: 'manual' indicates to use paramters below, both 'Sturges' or 'Scott' will use corresponding methods of computing number of bins and width (see http://en.wikipedia.org/wiki/Histogram#Number_of_bins_and_width).
numbins	number of bins to use in histogram.
binsize	size (width) of discrete intervals defining histogram (all bins are equal).
startvalue	lower end (bound) of values to include in histogram.

endvalue	upper end (bound) of values to include in histogram.
vertexWhere	SQL WHERE clause limiting data from the vertex table. This value when not null overrides corresponding value vertexWhere from graph (use SQL as if in WHERE clause but omit keyword WHERE).
edgeWhere	SQL WHERE clause limiting data from the edge table. This value when not null overrides corresponding value edgeWhere from graph (use SQL as if in WHERE clause but omit keyword WHERE).
allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).
...	other arguments passed on to Aster graph functions except for EDGEWEIGHT argument - use argument weight instead. Aster function areguments are not casesensetive

Examples

```
if(interactive()) {

policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
                         directed = FALSE, key = "officer",
                         source = "officer1", target = "officer2",
                        vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

hdegreePolice = computeGraphHistogram(conn, policeGraphUn, type='degree', numbins=36)
createHistogram(hdegreePolice,
                title = "Dallas Police Graph Degree Distribution",
                xlab='Degree', ylab='Count')

hshortestpathPolice = computeGraphHistogram(conn, policeGraphUn, type='shortestpath',
                                numbins = 10)
createHistogram(hshortestpathPolice,
                title = "Dallas Police Shortest Path Distribution",
                xlab = "Distance", ylab = "Count")
}
```

---

computeGraphMetric *Compute top vertices by the metric values on a graph.*

Description

Compute top vertices by the metric values on a graph.

Usage

```
computeGraphMetric(channel, graph, type = "degree", top = 10,
  rankFunction = "rank", weight = FALSE, vertexWhere = graph$vertexWhere,
  edgeWhere = graph$edgeWhere, keyAsFactor = TRUE, allTables = NULL,
  test = FALSE, ...)
```

Arguments

channel	connection object as returned by odbcConnect
graph	an object of class 'toagraph' referencing graph tables in Aster database.
type	choose between graph metrics to compute: 'degree', 'in-degree','out-degree', 'clustering',
top	the number of vertices to return. If top >= 0 then top vertices sorted by the metric value are returned, otherwise all vertices are returned. Returned vertices are ordered by the computed graph metric only when top >= 0.
rankFunction	one of rownumber, rank, denserank, percentrank. Rank computed and returned for each vertex and each metric type. rankFunction determines which SQL window function computes vertex rank value (default rank corresponds to SQL RANK() window function). When threshold top is greater than 0 ranking function used to limit number of vertices returned (see details).
weight	logical or character: if logical then TRUE indicates using 'weight' edge attribute, otherwise no weight used. If character then use as a name for the edge weight attribute. The edge weight may apply with types 'clustering', 'shortestpath' and centrality measures.
vertexWhere	SQL WHERE clause limiting data from the vertex table. This value when not null overrides corresponding value vertexWhere from graph (use SQL as if in WHERE clause but omit keyword WHERE).
edgeWhere	SQL WHERE clause limiting data from the edge table. This value when not null overrides corresponding value edgeWhere from graph (use SQL as if in WHERE clause but omit keyword WHERE).
keyAsFactor	logical: should key column be converted to factor? If TRUE then conversion will always take place for any of integer, numeric, or character data types.
allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).
...	other arguments passed on to Aster graph functions except for EDGEWEIGHT argument - use argument weight instead. Aster function areguments are not case-sensetive.

Value

dataframe containing one vertice per row with key value, computed metric value, and its rank using rankFunction.

Examples

```
if(interactive()) {
library(ggplot2)

policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
                         directed = FALSE, key = "officer",
                         source = "officer1", target = "officer2",
                        vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

createTopMetricPlot <- function(data, metric, xlab='Officer', ylab='Degree', title) {
   p = ggplot(data) +
       geom_bar(aes_string("key", metric, fill="key"), stat='identity') +
       labs(x=xlab,y=ylab,title=title) +
       ggthemes::theme_tufte() +
       theme(legend.position='none',
             axis.text.x = element_text(size=16, angle = 315, vjust = 1),
             plot.title = element_text(size=20),
             axis.ticks = element_blank())

   return(p)
}

# top degree officers
topDegree = computeGraphMetric(conn, policeGraphUn, type="degree", top=30)
createTopMetricPlot(topDegree, 'degree', ylab='Degree', title='Top 30 Officers by Degree')

# top betweenness officers
topbetweenness = computeGraphMetric(conn, policeGraphUn, type='betweenness', top=25)
createTopMetricPlot(topbetweenness, 'betweenness', ylab='Betweenness',
                    title='Top 25 Officers (Betweenness)')

}
```

`computeHeatmap` *Compute 2-dimensional multi-layered matrix for heat map visualizations.*

Description

Compute aggregate value(s) across two category classes represented by the table columns `dimension1` and `dimension2`. Resulting data frame represents 2-dimensional multi-layered matrix where each layer comprises values from single aggregate. Category columns usually are of character, temporal, or discrete types. Values are aggregates computed across category columns utilizing SQL `GROUP BY <dimension1>, <dimension2>`. Aggregate formula may use any SQL expressions allowed with the `GROUP BY` as defined above. Results are usually fed into `createHeatmap` for heat

map visualizations. If defined, parameter by expands grouping columns to be used with heat maps with faceting.

Result represents 2-dimensional matrix with as many data layers as there were aggregates computed. Additionally more layers defined with parameter by support facets.

Usage

```
computeHeatmap(channel, tableName, dimension1, dimension2,
  aggregates = "COUNT(*) cnt", aggregateFun = NULL, aggregateAlias = NULL,
  dimAsFactor = TRUE, withMelt = FALSE, where = NULL, by = NULL,
  test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	table name
dimension1	name of the column for for heatmap x values. This value along with dimension2 are x and y scales of heatmap table.
dimension2	name of the column for for heatmap y values. This value along with dimension1 are x and y scales of heatmap table.
aggregates	vector with SQL aggregates to compute values for heat map. Aggregate may have optional aliases like in "AVG(era) avg_era". Subsequently, use in createHeatmap as color (fill), text, and threshold values for heat map cells.
aggregateFun	deprecated. Use aggregates instead.
aggregateAlias	deprecated. Use aggregates instead.
dimAsFactor	logical indicates if dimensions and optional facet columns should be converted to factors. This is almost always necessary for heat maps.
withMelt	logical if TRUE then uses **reshape2** melt to transform data frame with aggregate values in designated columns into a molten data frame.
where	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside WHERE clause).
by	vector of column names to group by one or more table columns for faceting or alike (optional).
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).

Value

Data frame representing 2-dimensional multi-layered matrix to use with createHeatmap. Matrix has as many layers as there are aggregates computed. If by defined, data frame contains multiple matrices for each value(s) from the column(s) in by (to support facets). When withMelt TRUE function melt applies transforming data frame and columns with aggregate values for easy casting: expands number of rows and replaces all aggregate columns with two: variable and value.

See Also

createHeatmap

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

hm = computeHeatmap(conn, "teams_enh", 'franchid', 'decadeid', 'avg(w) w',
                    where="decadeid >= 1950")
hm$decadeid = factor(hm$decadeid)
createHeatmap(hm, 'decadeid', 'franchid', 'w')

# with diverging color gradient
hm = computeHeatmap(conn, "teams_enh", 'franchid', 'decadeid', 'avg(w-l) wl',
                    where="decadeid >= 1950")
hm$decadeid = factor(hm$decadeid)
createHeatmap(hm, 'decadeid', 'franchid', 'wl', divergingColourGradient = TRUE)
}
```

---

computeHistogram *Compute histogram distribution of the column.*

Description

Compute histogram of the table column in Aster by mapping its value to bins based on parameters specified. When column is of numeric or temporal data type it uses map-reduce histogram function over continuous values. When column is categorical (character data types) it defers to computeBarchart that uses SQL aggregate `COUNT(*)` with `GROUP BY <column>`. Result is a data frame to visualize as bar charts (see creating visualizations with createHistogram).

Usage

```
computeHistogram(channel, tableName, columnName, tableInfo = NULL,
  columnFrequency = FALSE, binMethod = "manual", binsize = NULL,
  startvalue = NULL, endvalue = NULL, numbins = NULL, useIQR = TRUE,
  datepart = NULL, where = NULL, by = NULL, test = FALSE,
  oldStyle = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
columnName	table column name to compute histogram
tableInfo	pre-built summary of data to use (require when `test=TRUE`). See getTableSummary.

`columnFrequency`
: logical indicates to build histogram of frequencies of column

`binMethod`
: one of several methods to determine number and size of bins: `'manual'` indicates to use paramters below, both `'Sturges'` or `'Scott'` will use corresponding methods of computing number of bins and width (see http://en.wikipedia.org/wiki/Histogram#Number_of_bins_and_width).

`binsize`
: size (width) of discrete intervals defining histogram (all bins are equal)

`startvalue`
: lower end (bound) of values to include in histogram

`endvalue`
: upper end (bound) of values to include in histogram

`numbins`
: number of bins to use in histogram

`useIQR`
: logical indicates use of IQR interval to compute cutoff lower and upper bounds for values to be included in histogram: `[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR], IQR = Q3 - Q1`

`datepart`
: field to extract from timestamp/date/time column to build histogram on

`where`
: specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

`by`
: for optional grouping by one or more values for faceting or alike

`test`
: logical: if TRUE show what would be done, only (similar to parameter `test` in RODBC functions like sqlQuery and sqlSave).

`oldStyle`
: logical indicates if old style histogram paramters are in use (before Aster AF 5.11)

See Also

computeBarchart and createHistogram

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# Histogram of team ERA distribution: Rangers vs. Yankees in 2000s
h2000s = computeHistogram(channel=conn, tableName='pitching_enh', columnName='era',
                          binsize=0.2, startvalue=0, endvalue=10, by='teamid',
                         where="yearID between 2000 and 2012 and teamid in ('NYA','TEX')")
createHistogram(h2000s, fill='teamid', facet='teamid',
                title='TEX vs. NYY 2000-2012', xlab='ERA', ylab='count',
                legendPosition='none')
}
```

computeKmeans *Perform k-means clustering on the table.*

Description

K-means clustering algorithm runs in-database, returns object compatible with kmeans and includes arbitrary aggregate metrics computed on resulting clusters.

Usage

```
computeKmeans(channel, tableName, centers, threshold = 0.0395, iterMax = 10,
  tableInfo, id, include = NULL, except = NULL,
  aggregates = "COUNT(*) cnt", scale = TRUE,
  idAlias = gsub("[^0-9a-zA-Z]+", "_", id), where = NULL,
  scaledTableName = NULL, centroidTableName = NULL, schema = NULL,
  test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect.
tableName	Aster table name.
centers	either the number of clusters, say k, or a matrix of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres. If a matrix then number of rows determines the number of clusters as each row determines initial center.
threshold	the convergence threshold. When the centroids move by less than this amount, the algorithm has converged.
iterMax	the maximum number of iterations the algorithm will run before quitting if the convergence threshold has not been met.
tableInfo	pre-built summary of data to use (require when test=TRUE). See getTableSummary.
id	column name or SQL expression containing unique table key.
include	a vector of column names with variables (must be numeric). Model never contains variables other than in the list.
except	a vector of column names to exclude from variables. Model never contains variables from the list.
aggregates	vector with SQL aggregates that define arbitrary aggreate metrics to be computed on each cluster after running k-means. Aggregates may have optional aliases like in "AVG(era) avg_era". Subsequently, used in createClusterPlot as cluster properties.
scale	logical if TRUE then scale each variable in-database before clustering. Scaling performed results in 0 mean and unit standard deviation for each of input variables.
idAlias	SQL alias for table id. This is required when SQL expression is given for id.

`where` specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

`scaledTableName` name of Aster table with results of scaling

`centroidTableName` name of Aster table with centroids found by kmeans

`schema` name of Aster schema tables `scaledTableName` and `centroidTableName` belong.

`test` logical: if TRUE show what would be done, only (similar to parameter `test` in **RODBC** functions: sqlQuery and sqlSave).

Details

The function fist scales not-null data (if `scale=TRUE`) or just eliminate nulls without scaling. After that the data given (table `tableName` with option of filering with `where`) are clustered by the k-means in Aster. Next, all standard metrics of k-means clusters plus additional aggregates provided with `aggregates` are calculated again in-database.

Value

`computeKmeans` returns an object of class `"toakmeans"` (compatible with class `"kmeans"`). It is a list with at least the following components:

`cluster` A vector of integers (from 0:K-1) indicating the cluster to which each point is allocated. `computeKmeans` leaves this component empty. Use function `computeClusterSample` to set this compoenent.

`centers` A matrix of cluster centres.

`totss` The total sum of squares.

`withinss` Vector of within-cluster sum of squares, one component per cluster.

`tot.withinss` Total within-cluster sum of squares, i.e. `sum(withinss)`.

`betweenss` The between-cluster sum of squares, i.e. `totss-tot.withinss`.

`size` The number of points in each cluster. These includes all points in the Aster table specified that satisfy optional `where` condition.

`iter` The number of (outer) iterations.

`ifault` integer: indicator of a possible algorithm problem (always 0).

`scale` logical: indicates if variable scaling was performed before clustering.

`aggregates` Vectors (dataframe) of aggregates computed on each cluster.

`tableName` Aster table name containing data for clustering.

`columns` Vector of column names with variables used for clustering.

`scaledTableName` Aster table name containing scaled data for clustering.

`centroidTableName` Aster table name containing cluster centroids.

`id` Column name or SQL expression containing unique table key.

`idAlias` SQL alias for table id.

`whereClause` SQL `WHERE` clause expression used (if any).

`time` An object of class `proc_time` with user, system, and total elapsed times for the `computeKmeans` function call.

See Also

computeClusterSample, computeSilhouette

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
                aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                    id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                    include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                    centroidTableName='kmeans_test_centroids',
                    where="yearid > 2000")
km
createCentroidPlot(km)
createClusterPlot(km)
}
```

---

computeLm *Fit Linear Model and return its coefficients.*

Description

Outputs coefficients of the linear model fitted to Aster table according to the formula expression containing column names. The zeroth coefficient corresponds to the slope intercept. R formula expression with column names for response and predictor variables is exactly as in lm function (though less features supported).

Usage

```
computeLm(channel, tableName, formula, tableInfo = NULL, categories = NULL,
  sampleSize = 1000, where = NULL, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under ‘Details‘.
tableInfo	pre-built table summary with data types

`categories` vector with column names containing categorical data. Optional if the column is of character type as it is automatically treated as categorical predictors. But if numerical column contains categorical data then then it has to be specified for a model to view it as categorical. Apply extra care not to have columns with too many values (approximaltely > 10) as categorical because each value results in dummy predictor variable added to the model.

`sampleSize` function always computes regression model coefficent on all data in the table. But it computes predictions and returns an object of `class` "lm" based on sample of data. The sample size is in an absolute value for number of rows in the sample. Be careful not overestimating the size as all results are loaded into memory. Special value `"all"` or `"ALL"` will include all data in computation.

`where` specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

`test` logical: if TRUE show what would be done, only (similar to parameter `test` in RODBC functions like sqlQuery and sqlSave).

Details

Models for `computeLm` are specified symbolically. A typical model has the form `response ~ terms` where response is the (numeric) column and terms is a series of column terms which specifies a linear predictor for response. A terms specification of the form `first + second` indicates all the terms in first together with all the terms in second with duplicates removed. A specification of the form `first:second` and `first*second` (interactions) are not supported yet.

Value

`computeLm` returns an object of `class` `"toalm", "lm"`.

The function `summary` .....

For backward compatibility Outputs data frame containing 3 columns:

coefficient_name name of predictor table column, zeroth coefficient name is "0"

coefficient_index index of predictor table column starting with 0

value coefficient value

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# batting average explained by rbi, bb, so
lm1 = computeLm(channel=conn, tableName="batting_enh", formula= ba ~ rbi + bb + so)
summary(lm1)

# with category predictor league and explicit sample size
lm2 = computeLm(channel=conn, tableName="batting_enh", formula= ba ~ rbi + bb + so + lgid,
                , sampleSize=10000, where="lgid in ('AL','NL') and ab > 30")
```

```
summary(lm2)
}
```

---

computePercentiles *Compute percentiles of column values.*

Description

Compute percentiles including boxplot quartiles across values of column columnName. Multiple sets of percentiles achieved with the parameter by. Vector by may contain arbitrary number of column names: the percentiles are computed for each combination of values from these columns. Remember that when using computed quartiles with function createBoxplot it can utilize up to 3 columns by displaying them along the x-axis and inside facets.

Usage

```
computePercentiles(channel, tableName, columnName = NULL,
  columns = columnName, temporal = FALSE, percentiles = c(ifelse(temporal,
  5, 0), 5, 10, 25, 50, 75, 90, 95, 100), by = NULL, where = NULL,
  nameInDataFrame = "column", stringsAsFactors = FALSE, test = FALSE,
  parallel = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
columnName	deprecated. Use vector columns instead.
columns	names of the columns to compute percentiles on
temporal	logical: TRUE indicates all columns are temporal, otherwsie numerical. Temporal percentiles have 2 values: character value representing temporal percentile (date, time, timestamp or datetime) and integer epoch value of the number of seconds since 1970-01-01 00:00:00-00 (can be negative) or for interval values includeing time, the total number of seconds in the interval.
percentiles	integer vector with percentiles to compute. Values 0, 25, 50, 75, 100 will always be added if omitted for numerical types, and 25, 50, 75, 100 for temporal. Percentile 0 (minimum) has to be included explicitly for temporals as its computation affects performance more than others.
by	for optional grouping by one or more values for faceting or alike. Used with createBoxplot in combination with column name for x-axis and wrap or grid faceting.
where	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside WHERE clause).

`nameInDataFrame`
: name of the column in returned data frame to store table column name(s) defined by parameter `columns`. `NULL` indicates omit this column from the data frame (not recommended when computing percentiles for multiple columns).

`stringsAsFactors`
: logical: should columns returned as character and not excluded by `as.is` and not converted to anything else be converted to factors?

`test`
: logical: if TRUE show what would be done, only (similar to parameter `test` in RODBC functions like sqlQuery and sqlSave).

`parallel`
: logical: enable parallel calls to Aster database. This option requires parallel backend enabled and registered (see in examples). Parallel execution requires ODBC `channel` obtained without explicit password: either with `odbcConnect(dsn)` or `odbcDriverConnect` calls, but not with `odbcConnect(dsn, user, password)`.

Value

For numeric data function returns a data frame with percentile values organized into following columns:

- *percentile* percentile to compute (from 0 to 100): will contain all valid values from `percentiles`
- *value* computed percentile
- *column* table column name. Override name `column` with parameter `nameInDataFrame` or omit this column all together if `NULL`.
- *by[1], by[2], ...* in presence of parameter `by`, contain values of the grouping columns for computed percentiles (optional).

For temporal data function returns a data frame with percentile values organized into following columns:

- *percentile* percentile to compute (from 0 to 100): will contain all valid values from `percentiles`
- *value* computed percentile value converted from temporal data type to its character representation.
- *epoch* corresponding to temporal percentile value epoch: for `date` and `timestamp` values, the number of seconds since 1970-01-01 00:00:00-00 (can be negative); for interval values include `time`, the total number of seconds in the interval.
- *column* table column name. Override name `column` with parameter `nameInDataFrame` or omit this column all together if `NULL`.
- *by[1], by[2], ...* in presence of parameter `by`, contain values of the grouping columns for computed percentiles (optional).

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# ipouts percentiles for pitching ipouts for AL in 2000s
```

```
ipop = computePercentiles(conn, "pitching", "ipouts",
                          where = "lgid = 'AL' and yearid >= 2000")

# ipouts percentiles by league
ipopLg = computePercentiles(conn, "pitching", "ipouts", by="lgid")

# percentiles on temporal columns
playerAllDates = computePercentiles(conn, "master_enh",
                    columns=c('debut','finalgame','birthdate','deathdate'),
                    temporal=TRUE, percentiles=c(0))
createBoxplot(playerAllDates, x='column', value='epoch', useIQR=TRUE,
              title="Boxplots for Date columns (epoch values)",
              legendPosition="none")

}
```

`computeSample` *Randomly sample data from the table.*

Description

Draws a sample of rows from the table randomly. The function offers two sampling approaches and three stratum strategies. Sampling approaches by

- *sample fraction*: a simple binomial (Bernoulli) sampling on a row-by-row basis with given sample rate(s) (see `sampleFraction`)
- *sample size*: sampling a given number of rows without replacement (see `sampleSize`)

Stratum strategies:

- *single stratum*: the whole table or its subset (defined using `where`).
- *by column values*: using `conditionColumn` and `conditionValues` arguments define stratum per value in the table column.
- *by SQL expression*: using `conditionStratum` and `conditionValues` arguments define stratum using SQL expression (with SQL `CASE` function but not necessarily) per value.

Usage

```
computeSample(channel, tableName, sampleFraction, sampleSize,
  conditionColumn = NULL, conditionStratum = NULL, conditionValues = NULL,
  include = NULL, except = NULL, where = NULL, as.is = FALSE,
  stringsAsFactors = FALSE, test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`tableName`	table name

`sampleFraction` one or more sample fractions to use in the sampling of data. Multipe sampling fractions are applicable only in combination with the arguments `conditionColumn` and `conditionValues` when present. In this case number of fractions in `sampleFraction` and number of values in `conditionValues` must be the same.

`sampleSize` total sample size (applies only when `sampleFraction` is missing). This may too be a vector of total values when used in combination with the arguments `conditionColumn` and `conditionValues`. In this case number of sizes in `sampleSize` and number of values in `conditionValues` must be the same.

`conditionColumn`
if you use this argument, you must also use the `conditionValues` argument. Either both are used, or neither is used. Values in a particular column `conditionColumn` are used as sampling conditions directly and its data type must be of a groupable type. Only those values listed in `conditionValues` are used for sampling with the rest ignored. Also, see `conditionStratum`.

`conditionStratum`
if you use this argument, you must also use the `conditionValues` argument. When defined it is used in place of `conditionColumn`. `conditionStratum` should define a SQL expression (usually using `CASE` function but not necessarily). Resulting sample data frame will contain a column named `stratum` just as if `conditionColumn = 'stratum'` was used. Arguments `conditionColumn` and `conditionStratum` are mutually exclusive: the former is ignored if both are defined.

`conditionValues`
see argument `conditionColumn` and `conditionStratum`.

`include` a vector of column names to include. Output never contains attributes other than in the list.

`except` a vector of column names to exclude. Output never contains attributes from the list.

`where` specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside `WHERE` clause).

`as.is` which (if any) columns returned as character should be converted to another type? Allowed values are as for `read.table`. See also `sqlQuery`.

`stringsAsFactors`
logical: should columns returned as character and not excluded by `as.is` and not converted to anything else be converted to factors?

`test` logical: if TRUE show what would be done, only (similar to parameter `test` in RODBC functions like sqlQuery and sqlSave).

Details

The sampling can be applied to the entire table or can be refined with either `conditionColumn` or `conditionStratum`. In each case a subset of the table defined with `where` argument may apply too. The resulting stratum models are:

- *Single Sample Fraction*: provide only one value in `sampleFraction`, this single fraction is targeted throughout the whole population or across all the strata defined by the sample conditions `conditionColumn` or `conditionStrata` in combination with `conditionValues`.

- *Variable Sample Fractions*: provide multiple values in sampleFraction, each of them is used for sampling a particular stratum defined by the conditionColumn or conditionStratum arguments in combination with conditionValues. Number of values in sampleFraction and conditionValues must be the same.
- *Total Sample Size*: provide only one value in sampleSize for the total sample size for the entire population. If in addition you specify the conditionColumn or conditionStratum arguments, the function proportionally generates sample units for each stratum defined by the conditionColumn or conditionStratum arguments in combination with conditionValues.
- *Variable Sample Sizes*: provide multiple sizes in sampleSize so that each size corresponds to a stratum defined by conditionColumn or conditionStratum arguments in combination with conditionValues. The sample function generates sample units directly for each stratum based on the supplied sizes. Number of values in sampleSize and conditionValues must be the same.

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

batters = computeSample(conn, "batting", sampleFraction=0.01)
dim(batters)

pitchersAL = computeSample(conn, "pitching", sampleSize=1000, where="lgid = 'AL'")
dim(ptichersAL)

battersByDecadesSingleSize = computeSample(conn, tableName = 'batting_enh',
                                           sampleSize=1000,
                                           conditionColumn = 'decadeid',
                                           conditionValues = c(1990,2000,2010))
dim(battersByDecadesSingleSize)

battersByDecades = computeSample(conn, tableName = 'batting_enh',
                                 sampleFraction = c(0.01,0.01,0.02),
                           conditionColumn = 'decadeid', conditionValues = c(1990,2000,2010))
dim(battersByDecades)

battersByOddEvenYears = computeSample(channel=NULL, tableName = 'batting_enh',
                                      sampleFraction = c(0.01,0.02),
                                      include = c('decadeid','g','ab','r','h'),
                                      conditionStratum = "yearid % 2",
                                      conditionValues = c('0','1'),
                                      where = "lgid = 'NL'")
dim(battersByOddEvenYears)

battersBeforeAfter1960 = computeSample(channel=NULL, tableName = 'batting_enh',
                                       sampleSize = c(200, 200),
                                 conditionStratum = "CASE WHEN yearid <- 1960 THEN 'before'
                                                          ELSE 'after'
                                                     END",
```

```
                                        conditionValues = c('before','after'))
dim(battersBeforeAfter1960)
}
```

---

computeSilhouette *Compute Silhouette (k-means clustering).*

Description

Compute Silhouette (k-means clustering).

Usage

```
computeSilhouette(channel, km, scaled = TRUE, silhouetteTableName = NULL,
  drop = TRUE, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect.
km	an object of class "toakmeans" obtained with computeKmeans.
scaled	logical: indicates if computation performed on original (default) or scaled values.
silhouetteTableName	name of the Aster table to hold silhouette scores. The table persists silhoutte scores for all clustered elements. Set parameter drop=F to keep the table.
drop	logical: indicates if the table silhouetteTableName
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).

Value

computeSilhouette returns an object of class "toakmeans" (compatible with class "kmeans"). It adds a named list sil the km containing couple of elements: average value of silhouette value and silhouette profile (distribution of silhouette values on each cluster) profile

See Also

computeKmeans

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
```

```
                aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                    id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                    include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                    centroidTableName='kmeans_test_centroids',
                    where="yearid > 2000")
km = computeSilhouette(conn, km)
km$sil
createSilhouetteProfile(km, title="Cluster Silhouette Histograms (Profiles)")
}
```

---

computeTf *Compute term frequencies on a corpus.*

Description

Compute term frequencies on a corpus.

Usage

```
computeTf(channel, tableName, docId, textColumns, parser,
  weighting = "normal", top = NULL, rankFunction = "rank", where = NULL,
  idSep = "-", idNull = "(null)", stopwords = NULL, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
docId	vector with one or more column names comprising unique document id. Values are concatenated with idSep. Database NULLs are replaced with idNull string.
textColumns	one or more names of columns with text. Multiple coumn are concatenated into single text field first.
parser	type of parser to use on text. For example, ngram(2) parser generates 2-grams (ngrams of length 2), token(2) parser generates 2-word combinations of terms within documents.
weighting	term frequency formula to compute the tf value. One of following: 'raw', 'bool', 'binary', 'log', 'augment', and 'normal' (default).
top	specifies threshold to cut off terms ranked below top value. If value is greater than 0 then included top ranking terms only, otherwise all terms returned (also see paramter rankFunction). Terms are always ordered by their term frequency (tf) within each document. Filtered out terms have their rank ariphmetically greater than threshold top (see details): term is more important the smaller value of its rank.
rankFunction	one of rownumber, rank, denserank, percentrank. Rank computed and returned for each term within each document. function determines which SQL window function computes term rank value (default rank corresponds to SQL RANK() window function). When threshold top is greater than 0 ranking function used to limit number of terms returned (see details).

where	specifies criteria to satisfy by the table rows before applying computation. The criteria are expressed in the form of SQL predicates (inside WHERE clause).
idSep	separator when concatenating 2 or more document id columns (see docId).
idNull	string to replace NULL value in document id columns.
stopwords	character vector with stop words. Removing stop words takes place in R after results are computed and returned from Aster.
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions sqlQuery and sqlSave).

Details

By default function computes and returns all terms. When large number of terms is expected then use parameters top to limit number of terms returned by filtering top ranked terms for each document. Thus if set top=1000 and there is 100 documents then at least 100,000 terms (rows) will be returned. Result size could exceed this number when other than rownumber rankFunction used:

- rownumber applies a sequential row number, starting at 1, to each term in a document. The tie-breaker behavior is as follows: Rows that compare as equal in the sort order will be sorted arbitrarily within the scope of the tie, and all terms will be given unique row numbers.
- rank function assigns the current row-count number as the terms's rank, provided the term does not sort as equal (tie) with another term. The tie-breaker behavior is as follows: terms that compare as equal in the sort order are sorted arbitrarily within the scope of the tie, and the sorted-as-equal terms get the same rank number.
- denserank behaves like the rank function, except that it never places gaps in the rank sequence. The tie-breaker behavior is the same as that of RANK(), in that the sorted-as-equal terms receive the same rank. With denserank, however, the next term after the set of equally ranked terms gets a rank 1 higher than preceding tied terms.
- percentrank assigns a relative rank to each term, using the formula: (rank - 1) / (total rows - 1). The tie-breaker behavior is as follows: Terms that compare as equal are sorted arbitrarily within the scope of the tie, and the sorted-as-equal rows get the same percent rank number.

The ordering of the rows is always by their tf value within each document.

See Also

computeTfIdf, nGram, token

Examples

```
if(interactive()){
# initialize connection to Dallas database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# compute term-document-matrix of all 2-word Ngrams of Dallas police open crime reports
tdm1 = computeTf(channel=conn, tableName="public.dallaspoliceall", docId="offensestatus",
                 textColumns=c("offensedescription", "offensenarrative"),
                 parser=nGram(2),
```

```
                where="offensestatus NOT IN ('System.Xml.XmlElement', 'C')")

# compute term-document-matrix of all 2-word combinations of Dallas police crime reports
# by time of day (4 documents corresponding to 4 parts of day)
tdm2 = computeTf(channel=conn, tableName="public.dallaspoliceall",
                docId="(extract('hour' from offensestarttime)/6)::int%4",
                textColumns=c("offensedescription", "offensenarrative"),
                parser=token(2, punctuation="[-.,?\\!:;~()]+", stopWords=TRUE),
                where="offensenarrative IS NOT NULL")

# include only top 100 ranked 2-word ngrams for each offense status
# into resulting term-document-matrix using dense rank function
tdm3 = computeTf(channel=NULL, tableName="public.dallaspoliceall", docId="offensestatus",
                textColumns=c("offensedescription", "offensenarrative"),
                parser=nGram(2), top=100, rankFunction="denserank",
                where="offensestatus NOT IN ('System.Xml.XmlElement', 'C')")

}
```

---

computeTfIdf *Compute Term Frequency - Inverse Document Frequency on a corpus.*

Description

Compute Term Frequency - Inverse Document Frequency on a corpus.

Usage

```
computeTfIdf(channel, tableName, docId, textColumns, parser, top = NULL,
  rankFunction = "rank", idSep = "-", idNull = "(null)",
  adjustDocumentCount = FALSE, where = NULL, stopwords = NULL,
  test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
docId	vector with one or more column names comprising unique document id. Values are concatenated with idSep. Database NULLs are replaced with idNull string.
textColumns	one or more names of columns with text. Multiple coumn are concatenated into single text field first.
parser	type of parser to use on text. For example, ngram(2) parser generates 2-grams (ngrams of length 2), token(2) parser generates 2-word combinations of terms within documents.

top	specifies threshold to cut off terms ranked below top value. If value is greater than 0 then included top ranking terms only, otherwise all terms returned (also see paramter rankFunction). Terms are always ordered by their term frequency - inverse document frequency (tf-idf) within each document. Filtered out terms have their rank ariphmetically greater than threshold top (see details): term is more important the smaller value of its rank.
rankFunction	one of rownumber, rank, denserank, percentrank. Rank computed and returned for each term within each document. function determines which SQL window function computes term rank value (default rank corresponds to SQL RANK() window function). When threshold top is greater than 0 ranking function used to limit number of terms returned (see details).
idSep	separator when concatenating 2 or more document id columns (see docId).
idNull	string to replace NULL value in document id columns.
adjustDocumentCount	logical: if TRUE then number of documents 2 will be increased by 1.
where	specifies criteria to satisfy by the table rows before applying computation. The criteria are expressed in the form of SQL predicates (inside WHERE clause).
stopwords	character vector with stop words. Removing stop words takes place in R after results are computed and returned from Aster.
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions sqlQuery and sqlSave).

Details

By default function computes and returns all terms. When large number of terms is expected then use parameters top to limit number of terms returned by filtering top ranked terms for each document. Thus if set top=1000 and there is 100 documents then at least 100,000 terms (rows) will be returned. Result size could exceed this number when other than rownumber rankFunction used:

- rownumber applies a sequential row number, starting at 1, to each term in a document. The tie-breaker behavior is as follows: Rows that compare as equal in the sort order will be sorted arbitrarily within the scope of the tie, and all terms will be given unique row numbers.
- rank function assigns the current row-count number as the terms's rank, provided the term does not sort as equal (tie) with another term. The tie-breaker behavior is as follows: terms that compare as equal in the sort order are sorted arbitrarily within the scope of the tie, and the sorted-as-equal terms get the same rank number.
- denserank behaves like the rank function, except that it never places gaps in the rank sequence. The tie-breaker behavior is the same as that of RANK(), in that the sorted-as-equal terms receive the same rank. With denserank, however, the next term after the set of equally ranked terms gets a rank 1 higher than preceding tied terms.
- percentrank assigns a relative rank to each term, using the formula: (rank - 1) / (total rows - 1). The tie-breaker behavior is as follows: Terms that compare as equal are sorted arbitrarily within the scope of the tie, and the sorted-as-equal rows get the same percent rank number.

The ordering of the rows is always by their tf-idf value within each document.

See Also

computeTf, nGram, token

Examples

```
if(interactive()){
# initialize connection to Dallas database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# compute term-document-matrix of all 2-word Ngrams of Dallas police crime reports
# for each 4-digit zip
tdm1 = computeTfIdf(channel=conn, tableName="public.dallaspoliceall",
                    docId="substr(offensezip, 1, 4)",
                    textColumns=c("offensedescription", "offensenarrative"),
                    parser=nGram(2, ignoreCase=TRUE,
                                punctuation="[-.,?\\!:;~()]+"))

# compute term-document-matrix of all 2-word combinations of Dallas police crime reports
# for each type of offense status
tdm2 = computeTfIdf(channel=NULL, tableName="public.dallaspoliceall", docId="offensestatus",
                textColumns=c("offensedescription", "offensenarrative", "offenseweather"),
                    parser=token(2),
                    where="offensestatus NOT IN ('System.Xml.XmlElement', 'C')")

# include only top 100 ranked 2-word ngrams for each 4-digit zip into resulting
# term-document-matrix using rank function
tdm3 = computeTfIdf(channel=NULL, tableName="public.dallaspoliceall",
                    docId="substr(offensezip, 1, 4)",
                    textColumns=c("offensedescription", "offensenarrative"),
                    parser=nGram(2), top=100)

# same but get top 10% ranked terms using percent rank function
tdm4 = computeTfIdf(channel=NULL, tableName="public.dallaspoliceall",
                    docId="substr(offensezip, 1, 4)",
                    textColumns=c("offensedescription", "offensenarrative"),
                    parser=nGram(1), top=0.10, rankFunction="percentrank")

}
```

---

createBoxplot *Create box plot.*

Description

Create box plot visualization using quartiles calculated with computePercentiles. The simplest case without x value displays single boxplot from the single set of percentiles. To plot multiple box plots and multiple or single box plots with facets use parameters x and/or facet.

Usage

```
createBoxplot(data, x = NULL, fill = x, value = "value", useIQR = FALSE,
  facet = NULL, ncol = 1, facetScales = "fixed", paletteValues = NULL,
  palette = "Set1", title = paste("Boxplots", ifelse(is.null(x), NULL,
  paste("by", x))), subtitle = NULL, xlab = x, ylab = NULL,
  legendPosition = "right", fillGuide = "legend", coordFlip = FALSE,
  baseSize = 12, baseFamily = "sans", defaultTheme = theme_tufte(base_size
  = baseSize, base_family = baseFamily), themeExtra = NULL)
```

Arguments

data	quartiles precomputed with computePercentiles
x	column name of primary variance. Multiple boxplots are placed along the x-axis. Each value of x must have corresponding percentiles calculated.
fill	name of a column with values to colour box plots
value	column name with percentile value. Usually default 'value' with exception of temporal percentiles that should use 'epoch' value.
useIQR	logical indicates use of IQR interval to compute cutoff lower and upper bounds: [Q1 - 1.5 * IQR, Q3 + 1.5 * IQR], IQR = Q3 - Q1, if FALSE then use maximum and minimum bounds (all values).
facet	vector of 1 or 2 column names to split up data to plot the subsets as facets. If single name then subset plots are placed next to each other, wrapping with ncol number of columns (uses facet_wrap). When two names then subset plots vary on both horizontal and vertical directions (grid) based on the column values (uses facet_grid).
ncol	number of facet columns (applies when single facet column supplied only - see parameter facet).
facetScales	Are scales shared across all subset plots (facets): "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis, default), "free" - both rows and columns (see in facet_wrap parameter scales)
paletteValues	actual palette colours for use with scale_fill_manual (if specified then parameter palette is ignored)
palette	Brewer palette name - see display.brewer.all in RColorBrewer package for names
title	plot title.
subtitle	plot subtitle.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
fillGuide	Name of guide object, or object itself for the fill (when present). Typically "legend" name or object guide_legend.

coordFlip	logical flipped cartesian coordinates so that horizontal becomes vertical, and vertical horizontal (see coord_flip).
baseSize	theme base font size
baseFamily	theme base font family
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Details

Multiple box plots: x is a name of variable where each value corresponds to a set of percentiles. The boxplots will be placed along the x-axis. Simply use computePercentiles with parameter by="name to be passed in x variable".

Facets: facet vector contains one or two names of vairables where each combination of values corresponds to a set of percentiles. The boxplot(s) will be placed inside separate sections of the plot (facets). Both single boxplot (without variable x and with one) are supported.

Usually, with multiple percentile sets varying along single value use parameter x and add facets on top. The exception is when scale of percentile values differs between each boxplot. Then omit parameter x and use facet with facetScales='free_y'.

Value

ggplot object

See Also

computePercentiles for computing boxplot quartiles

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# boxplot of pitching ipouts for AL in 2000s
ipop = computePercentiles(conn, "pitching", columns="ipouts")
createBoxplot(ipop)

# boxplots by the league of pitching ipouts
ipopLg = computePercentiles(conn, "pitching", columns="ipouts", by="lgid")
createBoxplot(ipopLg, x="lgid")

# boxplots by the league with facet yearid of pitching ipouts in 2010s
ipopLgYear = computePercentiles(conn, "pitching", columns="ipouts", by=c("lgid", "yearid"),
                                where = "yearid >= 2010")
createBoxplot(ipopLgYear, x="lgid", facet="yearid", ncol=3)

# boxplot with facets only
```

```
bapLgDec = computePercentiles(conn, "pitching_enh", columns="era", by=c("lgid", "decadeid"),
                              where = "lgid in ('AL','NL')")
createBoxplot(bapLgDec, facet=c("lgid", "decadeid"))
}
```

`createBubblechart` *Create Bubble Chart type of plot.*

Description

Create a bubble chart that utilizes three dimensions of data. It is a variation of the scatter plot with data points replaced with shapes ("bubbles"): x and y are bubble location and z is its size. It can optionally assign data points labels and fill shapes with colors.

Usage

```
createBubblechart(data, x, y, z, label = z, fill = NULL, facet = NULL,
  ncol = 1, facetScales = "fixed", xlim = NULL, baseSize = 12,
  baseFamily = "sans", shape = 21, shapeColour = "black",
  scaleSize = TRUE, shapeSizeRange = c(3, 10), shapeMaxSize = 100,
  paletteValues = NULL, palette = "Set1", title = paste("Bubble Chart by",
  fill), subtitle = NULL, xlab = x, ylab = y, labelSize = 5,
  labelFamily = "", labelFontface = "plain", labelColour = "black",
  labelVJust = 0.5, labelHJust = 0.5, labelAlpha = 1, labelAngle = 0,
  legendPosition = "right", sizeGuide = FALSE, fillGuide = "colorbar",
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily),
  themeExtra = NULL)
```

Arguments

`data`	data frame contains data computed for bubblechart
`x`	name of a column containing x variable values
`y`	name of a column containing y variable values
`z`	name of a column containing bubble size value
`label`	name of a column containing bubble label
`fill`	name of a column with values to use for bubble colours
`facet`	vector of 1 or 2 column names to split up data to plot the subsets as facets. If single name then subset plots are placed next to each other, wrapping with `ncol` number of columns (uses `facet_wrap`). When two names then subset plots vary on both horizontal and vertical directions (grid) based on the column values (uses `facet_grid`).
`ncol`	number of facet columns (applies when single facet column supplied only - see parameter `facet`).

facetScales	Are scales shared across all subset plots (facets): "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis, default), "free" - both rows and columns (see in facet_wrap parameter scales)
xlim	a vector specifying the data range for the x scale and the default order of their display in the x axis.
baseSize	theme base font size
baseFamily	theme base font family
shape	bubble shape
shapeColour	colour of shapes
scaleSize	logical if TRUE then scale the size of shape to be proportional to the value, if FALSE then scale the area.
shapeSizeRange	bubble size range (applies only when scaleSize = TRUE)
shapeMaxSize	size of largest shape (applies only when scaleSize = FALSE)
paletteValues	actual palette colours for use with scale_fill_manual (if specified then parameter palette is ignored)
palette	Brewer palette name - see display.brewer.all in RColorBrewer package for names
title	plot title.
subtitle	plot subtitle.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
labelSize	size of labels
labelFamily	label font name or family name
labelFontface	label font face (c("plain","bold","italic","bold.italic"))
labelColour	color of labels
labelVJust	position of the anchor (0=bottom edge, 1=top edge), can go below 0 or above 1
labelHJust	position of the label anchor (0=left edge, 1=right edge), can go below 0 or above 1
labelAlpha	the transparency of the text label
labelAngle	the angle at which to draw the text label
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
sizeGuide	Name of guide object, or object itself for the z (bubble size). Typically "legend" name or object guide_legend.
fillGuide	Name of guide object, or object itself for the fill. Typically "colorbar" name or object guide_colourbar.
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

See Also

computeAggregates computes data for the bubble chart.

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

cormat = computeCorrelations(channel=conn, "pitching_enh", sqlColumns(conn, "pitching_enh"),
                         include = c('w','l','cg','sho','sv','ipouts','h','er','hr','bb',
                                     'so','baopp','era','whip','ktobb','fip'),
                              where = "decadeid = 2000", test=FALSE)
# remove duplicate correlation values (no symmetry)
cormat = cormat[cormat$metric1 < cormat$metric2, ]
createBubblechart(cormat, "metric1", "metric2", "value", label=NULL, fill="sign")
}
```

---

createCentroidPlot *Create plot of cluster centroids.*

Description

Visualize centroids produced by clustering function like k-means. Plots available are line plot, bar plot, or heatmap. Parameter `format` specifies which one to create.

Usage

```
createCentroidPlot(km, format = "line", groupByCluster = TRUE,
  baseSize = 12, baseFamily = "serif", title = paste("Cluster Centroids",
  format, "Plot"), xlab, ylab = ifelse(format == "heatmap", "cluster",
  ifelse(km$scale, "scaled value", "value")), legendPosition = ifelse(format
  == "bar", "none", "right"), coordFlip = FALSE, ticks = FALSE,
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily,
  ticks = ticks), themeExtra = NULL)
```

Arguments

km	an object of class "toakmeans" returned by computeKmeans.
format	type of plot to use: "line", "bar", "bar_dodge", "bar_facet" (same as "bar") or "heatmap".
groupByCluster	logical: indicates if centroids are grouped by clusters or variables. groupByCluster has no effect when format="heatmap".

baseSize	theme base font size.
baseFamily	theme base font family.
title	plot title.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
coordFlip	logical flipped cartesian coordinates so that horizontal becomes vertical, and vertical horizontal (see coord_flip).
ticks	logical Show axis ticks using default theme settings (see defaultTheme)?
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
                aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                    id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                    include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                    centroidTableName='kmeans_test_centroids',
                    where="yearid > 2000")
createCentroidPlot(km)
createCentroidPlot(km, format="bar_dodge")
createCentroidPlot(km, format="heatmap", coordFlip=TRUE)
}
```

---

createClusterPairsPlot

Create cluster variable plot.

Description

Create cluster variable plot.

Usage

```
createClusterPairsPlot(km, baseSize = 12, baseFamily = "serif",
  title = "Cluster Variable Pairs", ticks = FALSE,
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily,
  ticks = ticks), themeExtra = theme(), ...)
```

Arguments

km	an object of class "toakmeans" returned by computeKmeans.
baseSize	theme base font size.
baseFamily	theme base font family.
title	plot title.
ticks	logical Show axis ticks using default theme settings (see defaultTheme)?
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.
...	other parameters being suplied to geom's aes.

Value

ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
               aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                   id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                   include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                   centroidTableName='kmeans_test_centroids',
                   where="yearid > 2000")
km = computeClusterSample(conn, km, 0.01)
createClusterPairsPlot(km, title="Batters Clustered by G, H, R", ticks=FALSE)
}
```

---

createClusterPlot *Create clusters' properties plot.*

Description

Create clusters' properties plot.

Usage

```
createClusterPlot(km, baseSize = 12, baseFamily = "serif",
  title = paste("Cluster Properties Plot"), xlab = "cluster",
  ylab = "value", border = TRUE, colorByCluster = TRUE, ticks = FALSE,
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily,
  ticks = ticks), themeExtra = NULL)
```

Arguments

km	an object of class "toakmeans" returned by computeKmeans.
baseSize	theme base font size.
baseFamily	theme base font family.
title	plot title.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
border	boolean indicates to use border around plotting area. In case of facets border is around each facet.
colorByCluster	logical: color corresponds to clusters or properties.
ticks	logical Show axis ticks using default theme settings (see defaultTheme)?
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

See Also

computeKmeans

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
               aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                   id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                   include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                   centroidTableName='kmeans_test_centroids',
                   where="yearid > 2000")
createClusterPlot(km)
}
```

createHeatmap *Create Heat Map type of plot.*

Description

Create heat map visualization of 2D matrix from the data frame `data` pre-computed with `computeHeatmap`.

Usage

```
createHeatmap(data, x, y, fill, facet = NULL, ncol = 1, baseSize = 12,
  baseFamily = "sans", thresholdValue = NULL, thresholdName = fill,
  text = FALSE, textFill = fill, percent = FALSE,
  digits = ifelse(percent, 2, 4), divergingColourGradient = FALSE,
  lowGradient = ifelse(divergingColourGradient, muted("red"), "#56B1F7"),
  midGradient = "white", highGradient = ifelse(divergingColourGradient,
  muted("blue"), "#132B43"), title = paste("Heatmap by", fill),
  subtitle = NULL, xlab = x, ylab = y, legendPosition = "right",
  fillGuide = "colorbar", defaultTheme = theme_tufte(base_size = baseSize,
  base_family = baseFamily), themeExtra = NULL)
```

Arguments

`data`	data frame contains data computed for heatmap
`x`	name of a column containing x variable values (1st or horizontal dimenstion) in 2D matrix
`y`	name of a column containing y variable values (2d or vertical dimension) in 2D matrix
`fill`	name of a column with values to map to heatmap gradient colors (`lowGradient`, `highGradient`, and optionally `midGradient`).
`facet`	vector of 1 or 2 column names to split up data to plot the subsets as facets. If single name then subset plots are placed next to each other, wrapping with `ncol` number of columns (uses `facet_wrap`). When two names then subset plots vary on both horizontal and vertical directions (grid) based on the column values (uses `facet_grid`).
`ncol`	number of facet columns (applies when single facet column supplied only - see parameter `facet`).
`baseSize`	`theme` base font size
`baseFamily`	`theme` base font family
`thresholdValue`	threshold to use to display data in heatmap (if NULL then do not use threshold)
`thresholdName`	name of data attribute from `data` to use (by defult use `fill`)
`text`	if TRUE then display values in heatmap table (default: FALSE)
`textFill`	text to display (applies only when `text` is TRUE), by defaul use `fill` values
`percent`	format text as percent

digits	number of digits to use in text
divergingColourGradient	logical diverging colour gradient places emphasize on both low and high leaving middle neutral. Use when both end grandient colours represent critical values such as negative and positive extremes (e.g. temprature, outliers, etc.).
lowGradient	colour for low end of gradient.
midGradient	colour for mid point.
highGradient	colour for high end of gradient.
title	plot title.
subtitle	plot subtitle.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
fillGuide	Name of guide object, or object itself for the `fill`. Typically `"colorbar"` name or object `guide_colourbar`.
defaultTheme	plot theme settings with default value `theme_tufte`. More themes are available here: `ggtheme` (by ggplot2) and `ggthemes`.
themeExtra	any additional `theme` settings that override default theme.

Value

ggplot object

See Also

`computeHeatmap` for computing data for heat map

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

hm = computeHeatmap(conn, "teams_enh", 'franchid', 'decadeid', 'avg(w) w',
                    where="decadeid >= 1950")
hm$decadeid = factor(hm$decadeid)
createHeatmap(hm, 'decadeid', 'franchid', 'w')

# with diverging color gradient
hm = computeHeatmap(conn, "teams_enh", 'franchid', 'decadeid', 'avg(w-l) wl',
                    where="decadeid >= 1950")
hm$decadeid = factor(hm$decadeid)
createHeatmap(hm, 'decadeid', 'franchid', 'wl', divergingColourGradient = TRUE)
}
```

createHistogram *Create histogram type of plot.*

Description

Create histogram plot from the pre-computed distribution of data. Parameter data is a data frame containing intervals (bins) and counts obtained using computeHistogram or computeBarchart).

Usage

```
createHistogram(data, x = "bin_start", y = "bin_count", fill = NULL,
  position = "dodge", facet = NULL, ncol = 1, facetScales = "free_y",
  baseSize = 12, baseFamily = "", xlim = NULL, breaks = NULL,
  text = FALSE, percent = FALSE, digits = 0, textVJust = -2,
  mainColour = "black", fillColour = "grey", scaleGradient = NULL,
  paletteValues = NULL, palette = "Set1", trend = FALSE,
  trendLinetype = "solid", trendLinesize = 1, trendLinecolour = "black",
  title = paste("Histgoram by", fill), subtitle = NULL, xlab = x,
  ylab = y, legendPosition = "right", coordFlip = FALSE,
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily),
  themeExtra = NULL)
```

Arguments

data	data frame contains computed histogram
x	name of a column containing bin labels or interval values
y	name of a column containing bin values or counts (bin size)
fill	name of a column with values to colour bars
position	histogram position parameter to use for overlapping bars: stack, dodge (defult), fill, identity
facet	vector of 1 or 2 column names to split up data to plot the subsets as facets. If single name then subset plots are placed next to each other, wrapping with ncol number of columns (uses facet_wrap). When two names then subset plots vary on both horizontal and vertical directions (grid) based on the column values (uses facet_grid).
ncol	number of facet columns (applies when single facet column supplied only - see parameter facet).
facetScales	Are scales shared across all subset plots (facets): "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis, default), "free" - both rows and columns (see in facet_wrap parameter scales)
baseSize	theme base font size
baseFamily	theme base font family

xlim	a character vector specifying the data range for the x scale and the default order of their display in the x axis.
breaks	a character vector giving the breaks as they should appear on the x axis.
text	if TRUE then display values above bars (default: FALSE) (this feature is in development)
percent	format text as percent
digits	number of digits to use in text
textVJust	vertical justificaiton of text labels (relative to the top of bar).
mainColour	Perimeter color of histogram bars
fillColour	Fill color of histogram bars (applies only when fill is NULL)
scaleGradient	control ggplot2 scale fill gradient manually, e.g use scale_colour_gradient (if specified then parameter palette is ignored)
paletteValues	actual palette colours for use with scale_fill_manual (if specified then parameter palette is ignored)
palette	Brewer palette name - see display.brewer.all in RColorBrewer package for names
trend	logical indicates if trend line is shown.
trendLinetype	trend line type.
trendLinesize	size of trend line.
trendLinecolour	color of trend line.
title	plot title.
subtitle	plot subtitle.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
coordFlip	logical flipped cartesian coordinates so that horizontal becomes vertical, and vertical horizontal (see coord_flip).
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

See Also

computeHistogram and computeBarchart to compute data for histogram

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# AL teams pitching stats by decade
bc = computeBarchart(channel=conn, tableName="pitching_enh", category="teamid",
                     aggregates=c("AVG(era) era", "AVG(whip) whip", "AVG(ktobb) ktobb"),
                     where="yearid >= 1990 and lgid='AL'", by="decadeid", withMelt=TRUE)

createHistogram(bc, "teamid", "value", fill="teamid",
                facet=c("variable", "decadeid"),
                legendPosition="bottom",
                title = "AL Teams Pitching Stats by decades (1990-2012)",
                themeExtra = guides(fill=guide_legend(nrow=2)))

# AL Teams Average Win-Loss Difference by Decade
franchwl = computeBarchart(conn, "teams_enh", "franchid",
                           aggregates=c("AVG(w) w", "AVG(l) l", "AVG(w-l) wl"),
                           by="decadeid",
                           where="yearid >=1960 and lgid = 'AL'")

createHistogram(franchwl, "decadeid", "wl", fill="franchid",
                facet="franchid", ncol=5, facetScales="fixed",
                legendPosition="none",
                trend=TRUE,
                title="Average W-L difference by decade per team (AL)",
                ylab="Average W-L")

# Histogram of team ERA distribution: Rangers vs. Yankees in 2000s
h2000s = computeHistogram(channel=conn, tableName='pitching_enh', columnName='era',
                          binsize=0.2, startvalue=0, endvalue=10, by='teamid',
                         where="yearID between 2000 and 2012 and teamid in ('NYA','TEX')")
createHistogram(h2000s, fill='teamid', facet='teamid',
                title='TEX vs. NYY 2000-2012', xlab='ERA', ylab='count',
                legendPosition='none')

}
```

---

createMap *Locate map, geocode data, then plot both.*

Description

createMap is a smart function that places data artifact on the map. If necessary it geocodes the data, locates map that fits all data artifacts, and plots the map with the data shapes sized and colored using metrics.

Usage

```
createMap(data, maptype = "terrain", mapColor = c("color", "bw"),
  source = c("google", "osm", "stamen", "cloudmade"), location = NULL,
  locator = "center", boxBorderMargin = 10, zoom = NULL,
  locationName = NULL, lonName = "LONGITUDE", latName = "LATITUDE",
  metricName = NULL, metrics = metricName, labelName = NULL,
  scaleRange = c(1, 6), shape = 19, shapeColour = "red",
  shapeAlpha = 0.5, shapeStroke = 0.5, scaleSize = TRUE,
  textColour = "black", textFamily = "mono", textFace = "plain",
  textSize = 4, facet = NULL, ncol = 1, facetScales = "fixed",
  geocodeFun = memoise::memoise(geocode), getmapFun = get_map,
  urlonly = FALSE, api_key = NULL, baseSize = 12, baseFamily = "sans",
  title = NULL, legendPosition = "right", metricGuides = c("legend",
  "colorbar"), defaultTheme = theme_bw(base_size = baseSize),
  themeExtra = NULL)
```

Arguments

`data`	data frame with artifacts and their locations and metric(s) to be placed on the map. If location name is provided (with `locationName`) then it is used to gecode artifacts first. If not location then longitude and latitude must be provided. It is caller's responsibility adjust locations with value of `zoom` parameter to fit artifacts on the map.
`maptype`	map theme as defined in `get_map`. options available are 'terrain', 'satellite', 'roadmap', and 'hybrid'
`mapColor`	color (`'color'`) or black-and-white (`'bw'`)
`source`	Google Maps ('google'), OpenStreetMap ('osm'), Stamen Maps ('stamen'), or CloudMade maps ('cloudmade')
`location`	location of the map: longitude/latitude pair (in that order), or left/bottom/right/top bounding box: 'center' uses 2 value vector for the center of the map, while 'box' uses 4 value vector as left/bottom/right/top. If missing then function will derive map location using parameter `locator` and the `data`.
`locator`	in absence of `location` specifies how to use data to determine map location: when 'center' then function averages out data point longitude and latitude values to get approximate cneter for the map; when 'box' it will use min/max of longitude and latitude values to determine bounding box: left/bottom/right/top. If parameter `locationName` is specified then function will geocode values from this column first. If paramter `locationName` is missing then it assumes that data is already geocoded and stored in the columns with the names `lonName` and `latName`.
`boxBorderMargin`	margin size in percent of box sizes to increase box when computed from data locations.
`zoom`	map zoom as defined in `get_map`: an integer from 3 (continent) to 21 (building), default value 10 (city). Properly setting `zoom` for each map is responsibility of a caller. Zoom is optional when using bounding box location specification.

locationName	vector of the column names with address or name to geocode its location (find latitude and longitude) using geocode (see package **ggmap**). When locationName is specified then parameters lonName and latName are ignored. Multiple column names are used in order of appearance: geocoding tries 1st column's values first, then, for the data points that didn't get resolved, it tries the 2d column's values, and so on.
lonName	name of the column with longitude value. This value (in combination with value from column latName) is used to place each data point on the map. This parameter is ignored if locationName is defined.
latName	name of the column with latitude value. This value (in combination with value from column lonName) is used to place each data point on the map. This parameter is ignored if locationName is defined.
metricName	(deprecated) Use parameter metrics instead.
metrics	character vector of column names with metric values to scale shapes placed on map. First metric corresponds to the size (or area depending on scaleSize), second to the fill gradient. See also scaleSize and shapeStroke.
labelName	name of the column to use for the artifact label text when displaying data.
scaleRange	a numeric vector of lenght 2 that specifies the minimum and maximum size of the plotting symbol after transformation (see parameter range of scale_size).
shape	type of shape to use.
shapeColour	color of metric artifacts placed on map.
shapeAlpha	transparency of an artifact shape expressed as a fraction between 0 (complete transparency) and 1 (complete opacity).
shapeStroke	border width of an artifact shape. Remember, that in ggplot2 size and stroke are additive so a point with size = 5 and stroke = 5 will have a diameter of 10mm. createMap maps metrics[[1]] to shape size.
scaleSize	logical if TRUE then scale artifact shapes by size (radius), otherwise scale shape's area (artifact shapes scaling always uses metrics[[1]] values).
textColour	color of artifact labels on map.
textFamily	font family (when available) to use for artfiact labels.
textFace	font style to apply to artifact labels: 'plain' (default), 'bold', 'italic', or 'bold.italic'.
textSize	font size of artifact labels.
facet	name of a column to divide plot into facets for specificed parameter (defualt is NULL - no facets). If facet is single value then facet wrap applied (see facet_wrap), otherwise facet grid (see facet_grid with 1st 2 values of the vector.
ncol	number of facet columns (applies when single facet column supplied only - see parameter facet).
facetScales	Are scales shared across all facets: "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis) (default), "free" - both rows and columns (see in facet_wrap parameter scales).
geocodeFun	geocode function. Default is geocode but due to Google API restrictions use memoised version, e.g. memoise(geocode), instead (see package **memoise**).

getmapFun	get map function. Defayult is get_map but due to map APIs restrictions use memoised version, e.g. memose(get_map), instead (see package **memoise**).
urlonly	return url only.
api_key	an api key for cloudmade maps.
baseSize	base font size.
baseFamily	base font family.
title	plot title.
legendPosition	the position of metric guide ("left", "right", "bottom", "top", or two-element numeric vector; "none" is no legend).
metricGuides	list or vector with names of guide objects, or objects themselves, for up to 2 metrics. Typical guides are "legend" or "colorbar" names and guide_legend or guide_colorbar objects.
defaultTheme	plot theme to use, default is theme_bw.
themeExtra	any additional ggplot2 theme attributes to add.

Details

Geocoding: If parameter locationName is missing then no geocoding is possible. In that case parameters lonName and latName must contain names of columns with longitude and latitude information assigned to each data artifact (data point). If parameter locationName is defined then geocoding attempts to use values from the column with this name. Function geocodeFun specifies geocoding function (with default geocode from ggmap package). To speed up processing and avoid hitting global limit on Google Map API use memoised version of this function: memoise(geocode) (see memoise).

Map Locating: Function operates in 2 modes: explicit map location mode and implicit mode. In explicit mode value location locates the map using one of two supported formats. If it is a 2-value vector then it contains a center of the map. If it is 4-value vector then it contains bounding box coordinates: left/bottom/right/top. In implicit mode, when location is missing, fuction uses parameters locator and data to locate the map. If locator is equal to 'center' then it centers map by averaging longitude and latitude values of all data artifacts. If locator is equal to 'box' then it determines min/max values of longitutude and latitude of all data artifacts and locates the map by corresponding bounding box. Note that both modes support require explicit parameter zoom if applicable.

Map Types: variety of map avaiable are from several public sources: google, OpenStreetMap, Stamen, and CloudMade maps. The options and terms for each are different. For example, not all sources support both color and black-and-white options, or map types terrain, satellite, roadmap or hybrid. Note that in most cases by using Google source you are agreeing to the Google Maps API Terms of Service at https://developers.google.com/maps/terms.

Shapes: data artifacts are shapes placed over the map. Their size and fill are scaled using values in metrics columns and their location is determined either by geocoding values from locationName column or with longitude and latitude values stored in lonName and latName columns.

Labels: If labelName is specified then column with such name contains text labels to place on the map (using the same locations as for the shapes).

Value

a ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

data = computeAggregates(channel = conn, "teams_enh",
               aggregates = c("min(name) name", "min(park) park", "avg(rank) rank",
                              "avg(attendance) attendance"),
               by = c("name || ', ' || park teamname", "lgid", "teamid", "decadeid"))

geocodeFun = memoise::memoise(ggmap::geocode)
getMapFun = memoise::memoise(ggmap::get_map)

createMap(data=data[data$decadeid>=2000,],
          source = "stamen", maptype = "watercolor", zoom=4,
          facet=c("lgid", "decadeid"),
          locationName=c('teamname','name'),
          metrics=c('rank', 'attendance'), shape = 21,
         labelName='name', shapeColour="blue", scaleRange = c(2,12), textColour="black",
          title='Game Attendance by Decade and League (yearly, 2000-2012)',
          geocodeFun=geocodeFun, getmapFun = getMapFun)
}
```

`createPopPyramid` *Create Population Pyramid type of histogram plot.*

Description

Create population pyramid type of histogram plot: two back-to-back bar graphs on the same category class (e.g. age) placed on Y-axis and distribution (population) placed on the X-axis. Bar graphs correspond to two distinct groups, e.g. sex (male and female), baseball leagues (AL and NL), or customer types (new customers and established customers).

Usage

```
createPopPyramid(data, bin = "bin_start", count = "bin_count", divideBy,
  values = NULL, fillColours = c("blue", "red"), mainColour = "black",
  facet = NULL, ncol = 1, facetScales = "fixed", baseSize = 12,
  baseFamily = "sans", title = paste("Population Pyramid Histogram by",
  divideBy), subtitle = NULL, xlab = bin, ylab = count,
  legendPosition = "right", fillGuide = "legend",
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily),
  themeExtra = NULL)
```

Arguments

data	data frame contains 2 histograms for the same bins. Bins are divided into 2 sets with parameter divideBy.
bin	name of a column containing bin labels or interval values
count	name of a column containing bin values or counts (bin size)
divideBy	name of the column to divide data into two histograms
values	two-valued vector containing values in divideBy (optional). If missing then it uses 1st 2 values from column divideBy (sorted with default order).
fillColours	2-value vector with colours for left and right histograms.
mainColour	histogram bar colour.
facet	vector of 1 or 2 column names to split up data to plot the subsets as facets. If single name then subset plots are placed next to each other, wrapping with ncol number of columns (uses facet_wrap). When two names then subset plots vary on both horizontal and vertical directions (grid) based on the column values (uses facet_grid).
ncol	number of facet columns (applies when single facet column supplied only - see parameter facet).
facetScales	Are scales shared across all subset plots (facets): "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis, default), "free" - both rows and columns (see in facet_wrap parameter scales)
baseSize	theme base font size
baseFamily	theme base font family
title	plot title.
subtitle	plot subtitle.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
legendPosition	the position of legends. ("left", "right", "bottom", "top", or two-element numeric vector). "none" is no legend.
fillGuide	Name of guide object, or object itself for divideBy. Typically "legend" name or object guide_legend.
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

pitchingInfo = getTableSummary(asterConn, tableName='pitching',
                               where='yearid between 2000 and 2013')
battingInfo = getTableSummary(asterConn, tableName='batting',
                              where='yearid between 2000 and 2013')

salaryHistAll = computeHistogram(asterConn, tableName='public.salaries', columnName='salary',
                                 binsize=200000, startvalue=0,
                                 by='lgid', where='yearID between 2000 and 2013')
createPopPyramid(data=salaryHistAll, bin='bin_start', count='bin_count', divideBy='lgid',
                 values=c('NL','AL'),
                 title="Salary Pyramid by MLB Leagues",
                 xlab='Salary', ylab='Player Count')

salaryHist5Mil = computeHistogram(asterConn, tableName='salaries', columnName='salary',
                                  binsize=100000, startvalue=0, endvalue=5000000,
                                  by='lgid', where='yearID between 2000 and 2013')
createPopPyramid(data=salaryHist5Mil, divideBy='lgid', values=c('NL','AL'),
                 title="Salary Pyramid by MLB Leagues (less 5M only)",
                 xlab='Salary', ylab='Player Count')

eraHist = computeHistogram(asterConn, tableName='pitching', columnName='era',
                           binsize=.1, startvalue=0, endvalue=10,
                           by='lgid', where='yearid between 2000 and 2013')
createPopPyramid(data=eraHist, divideBy='lgid', values=c('NL','AL'),
                 title="ERA Pyramid by MLB Leagues", xlab='ERA', ylab='Player Count')

# Log ERA
eraLogHist = computeHistogram(asterConn, tableName='pitching', columnName='era_log',
                              binsize=.02, startvalue=-0.42021640338318984325,
                              endvalue=2.2764618041732441,
                             by='lgid', where='yearid between 2000 and 2013 and era > 0')
createPopPyramid(data=eraLogHist, divideBy='lgid', values=c('NL','AL'),
                 title="log(ERA) Pyramid by MLB Leagues",
                 xlab='log(ERA)', ylab='Player Count')

# Batting (BA)
battingHist = computeHistogram(asterConn, tableName='batting_enh', columnName='ba',
                               binsize=.01, startvalue=0.01, endvalue=0.51,
                               by='lgid', where='yearid between 2000 and 2013')
createPopPyramid(data=battingHist, divideBy='lgid', values=c('NL','AL'),
                title="Batting BA Pyramid by MLB Leages", xlab='BA', ylab='Player Count')
}
```

createSilhouetteProfile *Create cluster silhouette profile plot.*

Description

Create cluster silhouette profile plot.

Usage

```
createSilhouetteProfile(km, baseSize = 12, baseFamily = "serif",
  title = "Cluster Silhouette Profile (Histogram)",
  xlab = "Silhouette Value", ylab = "Count", coordFlip = TRUE,
  ticks = FALSE, defaultTheme = theme_tufte(base_size = baseSize,
  base_family = baseFamily, ticks = ticks), themeExtra = NULL)
```

Arguments

km	an object of class "toakmeans" returned by computeKmeans.
baseSize	theme base font size.
baseFamily	theme base font family.
title	plot title.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
coordFlip	logical flipped cartesian coordinates so that horizontal becomes vertical, and vertical horizontal (see coord_flip).
ticks	logical Show axis ticks using default theme settings (see defaultTheme)?
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

km = computeKmeans(conn, "batting", centers=5, iterMax = 25,
                aggregates = c("COUNT(*) cnt", "AVG(g) avg_g", "AVG(r) avg_r", "AVG(h) avg_h"),
                    id="playerid || '-' || stint || '-' || teamid || '-' || yearid",
                    include=c('g','r','h'), scaledTableName='kmeans_test_scaled',
                    centroidTableName='kmeans_test_centroids',
```

```
                    where="yearid > 2000")
km = computeSilhouette(conn, km)
createSilhouetteProfile(km, title="Cluster Silhouette Histograms (Profiles)")
}
```

---

createSlopegraph *Create plot with Slope Graph visualization.*

Description

Create plot with Slope Graph visualization.

Usage

```
createSlopegraph(data, id, rankFrom, rankTo, reverse = TRUE, na.rm = FALSE,
  scaleFactor = 1, fromLabel = rankFrom, toLabel = rankTo,
  title = paste("Slopegraph by", rankTo), subtitle = NULL, baseSize = 12,
  baseFamily = "sans", classLabels = c(rankFrom, rankTo),
  classTextSize = 12, colour = "#999999", upColour = "#D55E00",
  downColour = "#009E73", highlights = integer(0), lineSize = 0.15,
  textSize = 3.75, panelGridColour = "black", panelGridSize = 0.1,
  defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily),
  themeExtra = NULL)
```

Arguments

data	data frame contains data computed for slopegraph
id	name of column identifying each graph element having from (before) and to (after) pair of values
rankFrom	name of column with from (before) value
rankTo	name of column with to (after) value
reverse	logical reverse values if TRUE (smaller is better)
na.rm	logical value indicating whether NA values should be stripped before the visualization proceeds.
scaleFactor	scale factor applied to all values (-1 can be used instead of reverse TRUE).
fromLabel	label for left values (from or before).
toLabel	label for right values (to or after).
title	plot title.
subtitle	plot subtitle.
baseSize	base font size.
baseFamily	base font family.
classLabels	pair of labels for to and from columns (or classes).
classTextSize	size of text for class labels.

colour	default colour.
upColour	colour of up slope.
downColour	colour of down slope.
highlights	vector with indexes of highlighted points.
lineSize	size of slope lines.
textSize	size of text.
panelGridColour	background panel grid colour.
panelGridSize	background panel grid size.
defaultTheme	plot theme settings with default value theme_tufte. More themes are available here: ggtheme (by ggplot2) and ggthemes.
themeExtra	any additional theme settings that override default theme.

Value

ggplot object

---

createWordcloud *Create Word Cloud Visualization.*

Description

Wrapper around wordcloud function that optionally saves graphics to the file of one of supported formats.

Usage

```
createWordcloud(words, freq, title = "Wordcloud", scale = c(8, 0.2),
  minFreq = 10, maxWords = 40, filename, format = c("png", "bmp", "jpeg",
  "tiff", "pdf"), width = 480, height = 480, units = "px",
  palette = brewer.pal(8, "Dark2"), titleFactor = 1)
```

Arguments

words	the words
freq	their frequencies
title	plot title
scale	a vector indicating the range of the size of the words (default c(4,.5))
minFreq	words with frequency below minFreq will not be displayed
maxWords	Maximum number of words to be plotted (least frequent terms dropped).
filename	file name to use where to save graphics
format	format of graphics device to save wordcloud image
width	the width of the output graphics device

height	the height of the output graphics device
units	the units in which height and width are given. Cab be px (pixels, the default), in (inches), cm or mm.
palette	color words from least to most frequent
titleFactor	numeric title character expansion factor; multiplied by par("cex") yields the final title character size. NULL and NA are equivalent to a factor of 1.

Details

Uses base graphics and worldcloud package to create a word cloud (tag cloud) visual reprsentation of for text data. Function uses 2 vectors of equal lengths: one contains list of words and the other has their frequencies.

Resulting graphics is saved in file in one of available graphical formats (png, bmp, jpeg, tiff, or pdf).

Word Cloud visuals apply to any concept that satisfies following conditions: * each data point (artifact) can be expressed with distinct word or compact text in distinct and self-explanatory fashion and * it assigns each artifact scalar non-negative metric. Given these two conditions we can use Word Clouds to visualize top, bottom or all artifacts in single word cloud visual.

Value

nothing

See Also

wordcloud

Examples

```
if(interactive()){
# initialize connection to Dallas database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

stopwords = c("a", "an", "the", "with")

# 2-gram tf-idf on offense table
daypart_tfidf_2gram = computeTfIdf(conn, "public.dallaspoliceall",
                                   docId="extract('hour' from offensestarttime)::int/6",
                                  textColumns=c('offensedescription','offensenarrative'),
                                   parser=nGram(2, delimiter='[  \\t\\b\\f\\r:\"]+'),
                                   stopwords=stopwords)

toRace <- function(ch) {
  switch(as.character(ch),
         "M" = "Male",
         "F" = "Female",
         "0" = "Night",
         "1" = "Morning",
         "2" = "Day",
         "3" = "Evening",
```

```
          "C" = "C",
          "Unknown")
}

createDallasWordcloud <- function(tf_df, metric, slice, n, maxWords=25, size=750) {
  words=with(tf_df$rs, tf_df$rs[docid==slice,])

  ## palette
  pal = rev(brewer.pal(8, "Set1"))[c(-3,-1)]

  createWordcloud(words$term, words[, metric], maxWords=maxWords, scale=c(4, 0.5), palette=pal,
                  title=paste("Top ", metric, "Offense", n, "- grams for", toRace(race)),
                file=paste0('wordclouds/',metric,'_offense_',n,'gram_',toRace(slice),'.png'),
                   width=size, height=size)
}

createDallasWordcloud(daypart_tfidf_2gram, 'tf_idf', 0, n=2, maxWords=200, size=1300)

}
```

---

getArbitraryPrecisionTypes *List Aster arbitrary precision number data types.*

Description

List Aster arbitrary precision number data types.

Usage

```
getArbitraryPrecisionTypes()
```

Value

character vector with names of Aster arbitrary precision numeric data types

See Also

getNumericTypes, getFloatingPointTypes, getIntegerTypes

Examples

```
getArbitraryPrecisionTypes()
```

getCharacterColumns *Filter character columns.*

Description

Selects character columns (names or rows) from table info data frame.

Usage

```
getCharacterColumns(tableInfo, names.only = TRUE, include = NULL,
  except = NULL)
```

Arguments

tableInfo	data frame obtained by calling getTableSummary.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.
include	a vector of column names to include. Output is restricted to this list.
except	a vector of column names to exclude. Output never contains names from this list.

See Also

getNumericColumns, getTemporalColumns, getTableSummary

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

pitchingInfo = getTableSummary(channel=conn, 'pitching_enh')
getCharacterColumns(pitchingInfo)
char_cols_df = getCharacterColumns(pitchingInfo, names.only=FALSE)
}
```

getCharacterTypes *List Aster character data types.*

Description

List Aster character data types.

Usage

```
getCharacterTypes()
```

Value

character vector with names of Aster character data types

See Also

getNumericTypes, getTemporalTypes, getTableSummary

Examples

```
getCharacterTypes()
```

`getDiscretePaletteFactory` *Generate discrete palette maker*

Description

Generate discrete palette maker

Usage

```
getDiscretePaletteFactory(paletteName = "Set1")
```

Arguments

`paletteName`	name of palette from `brewer.pal.info` in `RColorBrewer` package

Value

function (factory) that creates discrete palette with for number of colors

See Also

getGradientPaletteFactory, colorRampPalette

Examples

```
paletteMaker = getDiscretePaletteFactory("PuOr")
myPalette = paletteMaker(25)
```

getFloatingPointTypes *List Aster floating piont numeric data types.*

Description

List Aster floating piont numeric data types.

Usage

```
getFloatingPointTypes()
```

Value

character vector with names of Aster floating point numeric data types

See Also

getNumericTypes, getIntegerTypes, getArbitraryPrecisionTypes

Examples

```
getFloatingPointTypes()
```

getGradientPaletteFactory *Generate gradient palette maker*

Description

inspired by http://stackoverflow.com/questions/13353213/gradient-of-n-colors-ranging-from-color-1-and-color-2

Usage

```
getGradientPaletteFactory(colors = c("black", "white"))
```

Arguments

colors	pair of colors for gradient range (min, max): default is c('black','white')

Value

function (factory) that creates linear gradient palette for given number of colors

See Also

getDiscretePaletteFactory, colorRampPalette

Examples

```
paletteMaker = getGradientPaletteFactory(c("yellow","red"))
myPalette = paletteMaker(10)
```

---

getIntegerTypes *List Aster integer data types.*

Description

List Aster integer data types.

Usage

```
getIntegerTypes()
```

Value

character vector with names of Aster integer data types

See Also

getNumericTypes, getFloatingPointTypes, getArbitraryPrecisionTypes

Examples

```
getIntegerTypes()
```

---

getMatchingColumns *Filter columns by pattern.*

Description

Selects columns with names matching regular expression pattern.

Usage

```
getMatchingColumns(pattern, channel, tableName, tableInfo, names.only = TRUE,
  ignore.case = TRUE, invert = FALSE)
```

Arguments

pattern	character string containing a regular expression to be matched in the given table info.
channel	connection object as returned by odbcConnect. Only used in combination with tableName.
tableName	Aster table name to use. If missing then tableInfo will be used instead.
tableInfo	data frame obtained by calling getTableSummary or sqlColumns.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.
ignore.case	if TRUE case is ignored during matching, otherwise matching is case sensitive.
invert	logical. if TRUE return columns that do not match.

See Also

grep, getTableSummary

---

getNullCounts *Counts nulls per column in the table.*

Description

Counts nulls per column in the table.

Usage

```
getNullCounts(channel, tableName, tableInfo = NULL, include = NULL,
  except = NULL, output = "long", percent = FALSE, schema = NULL,
  where = NULL, stringsAsFactors = FALSE, test = FALSE)
```

Arguments

channel	object as returned by odbcConnect.
tableName	name of the table in Aster.
tableInfo	pre-built summary of columns to use (require when test=TRUE). See sqlColumns or getTableSummary.
include	a vector of column names to include. Output never contains attributes other than in the list.
except	a vector of column names to exclude. Output never contains attributes from the list.
output	Default output is a data frame in 'long' format. Other options include 'wide' format and 'matrix'.
percent	logical: if TRUE then percent of NULL values instead of absolute count returned. To avoid division by zero small error is introduced by incrementing by 1 total count of rows in the table.

schema	NULL or character: optional schema to restric table search to signle schema. In general, table search performed across whole database. Including schema restricts it to this single schema only.
where	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside WHERE clause).
stringsAsFactors	logical: should data frame returned have column with variables (table column names) as factor? Applies only when the output is in long format.
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions like sqlQuery and sqlSave).

Examples

```
if (interactive()) {
# initialize connection to Dallas database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

null_counts = getNullCounts(conn, "baseball.batting",
                            include=c('g','ab','r','h','so','bb','cs'),
                            where='yearid > 2000')

}
```

`getNumericColumns` *Filter numeric columns.*

Description

Select numeric columns (names or rows) from table info data frame.

Usage

```
getNumericColumns(tableInfo, names.only = TRUE, include = NULL,
  except = NULL)
```

Arguments

tableInfo	data frame obtained by calling getTableSummary.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.
include	a vector of column names to include. Output is restricted to this list.
except	a vector of column names to exclude. Output never contains names from this list.

See Also

getCharacterColumns, getTemporalColumns, getTableSummary

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

pitchingInfo = getTableSummary(channel=conn, 'pitching_enh')
getNumericColumns(pitchingInfo)
num_cols_df = getNumericColumns(pitchingInfo, names.only=FALSE)
}
```

`getNumericTypes` *List Aster all numeric data types.*

Description

List Aster all numeric data types.

Usage

```
getNumericTypes()
```

Value

character vector with names of Aster numeric data types

See Also

`getCharacterTypes`, `getTemporalTypes`, `getTableSummary`

Examples

```
getNumericTypes()
```

`getTableCounts` *Counts number of rows and columns in the database tables.*

Description

Counts number of rows and columns in the database tables.

Usage

```
getTableCounts(channel, schema = NULL, tableType = "TABLE",
  pattern = NULL, columns = FALSE, where = NULL, tables = NULL,
  test = FALSE, parallel = FALSE)
```

Arguments

channel	object as returned by odbcConnect.
schema	character vector with schemas to restric tables to one or more schemas. If NULL table search performed across whole database. Including schema restricts it to the specified schemas only.
tableType	can specify zero or more types in separate elements of a character vector (one or more of "TABLE", "VIEW", "SYSTEM TABLE", "ALIAS", "SYNONYM").
pattern	character string containing regular expression to match table names (without schema).
columns	logical directs to include a column count for each table in the result.
where	specifies criteria to satisfy by the table rows before applying computation. The creteria are expressed in the form of SQL predicates (inside WHERE clause).
tables	optional pre-built list of tables (data frame returned by sqlTables).
test	logical: if TRUE show what would be done, only (similar to parameter test in RODBC functions like sqlQuery and sqlSave).
parallel	logical: enable parallel calls to Aster database. This option requires parallel backend enabled and registered (see in examples). Parallel execution requires ODBC channel obtained without explicit password: either with odbcConnect(dsn) or odbcDriverConnect calls, but not with odbcConnect(dsn, user, password).

Value

a data frame returned by sqlTables augmented with rowcount (number of rows) and optinal colcount (number of columns) columns for each table.

Examples

```
if (interactive()) {

# initialize connection to Dallas database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

table_counts = getTableCounts(conn, 'public')

library(reshape2)
library(ggplot2)
library(ggthemes)

data = melt(table_counts, id.vars='TABLE_NAME', measure.vars=c('rowcount','colcount'))
ggplot(data) +
  geom_bar(aes(TABLE_NAME, rowcount, fill=TABLE_NAME), stat='identity') +
  facet_wrap(~variable, scales = "free_y", ncol=1) +
  theme_tufte(ticks=FALSE) +
  theme(axis.text.x=element_text(size=12, angle=315, hjust=0),
        legend.position="none")
}
```

getTableSummary *Compute columnwise statistics on Aster table.*

Description

For table compute column statistics in Aster and augment data frame structure obtained with sqlColumns with columns containing computed statistics.

Usage

```
getTableSummary(channel, tableName, include = NULL, except = NULL,
  modeValue = FALSE, percentiles = c(5, 10, 25, 50, 75, 90, 95, 100),
  where = NULL, mock = FALSE, parallel = FALSE)
```

Arguments

channel	object as returned by odbcConnect.
tableName	name of the table in Aster.
include	a vector of column names to include. Output never contains attributes other than in the list.
except	a vector of column names to exclude. Output never contains attributes from the list.
modeValue	logical indicates if mode values should be computed. Default is FALSE.
percentiles	list of percentiles (integers between 0 and 100) to collect (always collects 25th and 75th for IQR calculation). There is no penalty in specifying more percentiles as they get calculated in a single call for each column - no matter how many different values are requested. When FALSE then percentiles calculations are skipped and result wil have no percentile and IQR columns.
where	SQL WHERE clause limiting data from the table (use SQL as if in WHERE clause but omit keyword WHERE).
mock	logical: if TRUE returns pre-computed table statistics for tables pitching or batting, only.
parallel	logical: enable parallel calls to Aster database. This option requires parallel backend enabled and registered (see in examples). Parallel execution requires ODBC channel obtained without explicit password: either with odbcConnect(dsn) or odbcDriverConnect calls, but not with odbcConnect(dsn, user, password).

Details

Computes columns statistics for all or specified table columns and adds them to the data frame with basic ODBC table metadata obtained with sqlColumns. Computed statistics include counts of all, non-null, distinct values; statistical summaries of maximum, minimum, mean, standard deviation, median (50th percentile), mode (optional), interquartile range, and desired percentiles. Each computed statistic adds a column to ODBC metadata data frame.

Value

data frame returned by `sqlColumns` with additional columns:

total_count total row count - the same for each table column

distinct_count distinct values count

not_null_count not null count

minimum minimum value (numerical data types only)

maximum maximum value (numerical data types only)

average mean (numerical data types only)

deviation standard deviation (numerical data types only)

percentiles defaults: 0,5,10,25,50,75,90,95,100. Always adds percentiles 25, 50 (median), 75

IQR interquartile range is the 1st Quartile subtracted from the 3rd Quartile

minimum_str minimum string value (character data types only)

maximum_str maximum string value (character data types only)

mode mode value (optional)

mode_count mode count (optional)

See Also

sqlColumns

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

pitchingInfo = getTableSummary(channel=conn, 'pitching_enh')
# list all table columns
pitchingInfo$COLUMN_NAME

# compute statistics on subset of baseball data after 1999
battingInfo = getTableSummary(channel=conn, 'batting_enh',
                              where='yearid between 2000 and 2013')

# compute statistics for certain columns including each percentile from 1 to 99
pitchingInfo = getTableSummary(channel=conn, 'pitching_enh',
                              include=c('h', 'er', 'hr', 'bb', 'so'),
                              percentiles=seq(1,99))
# list data frame column names to see all computed statistics
names(pitchingInfo)

# compute statitics on all numeric columns except certain columns
teamInfo = getTableSummary(channel=conn, 'teams_enh',
                  include=getNumericColumns(sqlColumns(conn, 'teams_enh')),
                  except=c('lgid', 'teamid', 'playerid', 'yearid', 'decadeid'))
}
```

getTemporalColumns *Filter Date and Time Table Columns.*

Description

Selects date and time columns (names or rows) from table info data frame.

Usage

```
getTemporalColumns(tableInfo, names.only = TRUE, include = NULL,
  except = NULL)
```

Arguments

tableInfo	data frame obtained by calling getTableSummary.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.
include	a vector of column names to include. Output is restricted to this list.
except	a vector of column names to exclude. Output never contains names from this list.

See Also

getCharacterColumns, getNumericColumns, getTableSummary

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

masterInfo = getTableSummary(channel=conn, 'master')
getTemporalColumns(masterInfo)
date_cols_df = getTemporalColumns(masterInfo, names.only=FALSE)
}
```

getTemporalTypes *List Aster temporal data types.*

Description

List Aster temporal data types.

Usage

```
getTemporalTypes()
```

Value

character vector with names of Aster temporal data types

See Also

getCharacterTypes, getCharacterTypes, getTableSummary

Examples

```
getTemporalTypes()
```

---

getWindowFunction *Determine window function to use*

Description

Determine window function to use

Usage

```
getWindowFunction(rankFunction)
```

Arguments

rankFunction	one of rank function codes to map to one of SQL window functions for ranking.

---

isTable *Test if table present in the database.*

Description

Tests each element if a table or a view exists in one of database schemas. Names could be just table name of schema followed by table name via '.'. ;

Usage

```
isTable(channel, tables, withNames = FALSE, allTables = NULL)
```

Arguments

channel	object as returned by odbcConnect.
tables	vector of table names. Name may contain schema name followed by dot and table name. All visible schemas are checked when table specified without a schema.
withNames	logical indicates if table components included in the results.
allTables	a data frame containing table-like objects accessible from Aster. Usually obtained with sqlTables.

Value

logical vector indicating if corresponding name is table in Aster database. Special case when name contains a SQL query triggers NA value instead. This behavior is handy when checking if a table name corresponds to a table (or a view) or a SQL query instead. For example, to test if all names are either existing tables or views or SQL queries use something like this: `result = isTable(...); if(all(result | is.na(`

See Also

getTableSummary

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

isTable(conn, "pitching")        # TRUE
isTable(conn, "pitch%")          # TRUE
isTable(conn, "public.pitching") # FALSE
}
```

---

makeFromClause *Make SQL FROM clause*

Description

Make SQL FROM clause

Usage

```
makeFromClause(name, flag, alias = "t")
```

Arguments

name	table or view name or a SQL query.
flag	logical indicates if a table or a query is visible. Special value NA indicates that name is a SQL query.
alias	query alias to use. Ignored if name is a table or a view.

makeTempTableName *Make Aster temporary table name.*

Description

Make Aster temporary table name.

Usage

```
makeTempTableName(prefix = NULL, n = 20, schema = NULL)
```

Arguments

prefix	Table name will always start with toa_temp_ followed by prefix (if exists).
n	non-negative integer giving number of random characters to include in the name.
schema	Aster database schema table should belong to.
	Table name generated will always begin with 'toa_temp_' followed by prefix (if not NULL) and n random alpha-numeric characters. Beware that total length can not exceed than 63 (Aster limit on table name length).

Value

character string suitable for Aster temporary table name

See Also

getTableSummary

Examples

```
tempTableName = makeTempTableName("centroids", 20)
```

nGram *Tokenize (or split) text and emit multi-grams.*

Description

Tokenize (or split) text and emit multi-grams.

Usage

```
nGram(n, ignoreCase = FALSE, delimiter = "[ \\t\\b\\f\\r]+",
  punctuation = NULL, overlapping = TRUE, reset = NULL, sep = " ",
  minLength = 1)
```

Arguments

n	length, in words, of each n-gram
ignoreCase	logical: if FALSE, the n-gram matching is case sensitive and if TRUE, case is ignored during matching.
delimiter	character or string that divides one word from the next. You can use a regular expression as the delimiter value.
punctuation	a regular expression that specifies the punctuation characters parser will remove before it evaluates the input text.
overlapping	logical: true value allows for overlapping n-grams.
reset	a regular expression listing one or more punctuation characters or strings, any of which the nGram parser will recognize as the end of a sentence of text. The end of each sentence resets the search for n-grams, meaning that nGram discards any partial n-grams and proceeds to the next sentence to search for the next n-gram. In other words, no n-gram can span two sentences.
sep	a character string to separate multiple text columns.
minLength	minimum length of words in ngram. Ngrams that contains words below shorter than the limit are omitted. Current implementation is not complete: it filters out ngrams where each word is below the minimum length, i.e. total length of ngram is below n*minLength + (n-1).

Value

pluggable n-gram parser

---

showData *Plot table level statistics, histograms, correlations and scatterplots in one go.*

Description

showData is the basic plotting function in the toaster package, designed to produce set of standard visualizations (see parameter format) in a single call. Depending on the format it is a wrapper to other functions or simple plotting function. It does all work in a single call by combining database round-trip (if necessary) and plotting functionality.

Usage

```
showData(channel = NULL, tableName = NULL, tableInfo = NULL,
  include = NULL, except = NULL, type = "numeric", format = "histgoram",
  measures = NULL, title = paste("Table", toupper(tableName), format, "of",
  type, "columns"), numBins = 30, useIQR = FALSE, extraPoints = NULL,
  extraPointShape = 15, sampleFraction = NULL, sampleSize = NULL,
  pointColour = NULL, facetName = NULL, regressionLine = FALSE,
  corrLabel = "none", digits = 2, shape = 21, shapeSizeRange = c(1, 10),
```

```
facet = ifelse(format == "overview", TRUE, FALSE), scales = ifelse(facet &
format == "boxplot", "free", ifelse(facet & format == "overview", "free_y",
"fixed")), ncol = 4, coordFlip = FALSE, paletteName = "Set1",
baseSize = 12, baseFamily = "sans", legendPosition = "none",
defaultTheme = theme_tufte(base_size = baseSize, base_family = baseFamily),
themeExtra = NULL, where = NULL, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
tableName	Aster table name
tableInfo	pre-built summary of data to use (parameters channel, tableName, where may not apply depending on format). See getTableSummary.
include	a vector of column names to include. Output never contains attributes other than in the list.
except	a vector of column names to exclude. Output never contains attributes from the list.
type	what type of data to visualize: numerical ("numeric"), character ("character" or date/time ("temporal")
format	type of plot to use: 'overview', 'histogram', 'boxplot', 'corr' for correlation matrix or 'scatterplot'
measures	applies to format 'overview' only. Use one or more of the following with 'numieric' type: maximum,minimum,average,deviation,0 type: distinct_count,not_null_count. By default all measures above are used per respeictive type.
title	plot title
numBins	number of bins to use in histogram(s)
useIQR	logical indicates use of IQR interval to compute cutoff lower and upper bounds for values to be included in boxplot or histogram: [Q1 - 1.5 * IQR, Q3 + 1.5 * IQR], IQR = Q3 - Q1, if FALSE then maximum and minimum are bounds (all values)
extraPoints	vector contains names of extra points to add to boxplot lines.
extraPointShape	extra point shape (see 'Shape examples' in aes_linetype_size_shape).
sampleFraction	sample fraction to use in the sampling of data for 'scatterplot'
sampleSize	if sampleFraction is not specified then size of sample must be specified for 'scatterplot'.
pointColour	name of column with values to colour points in 'scatterplot'.
facetName	name(s) of the column(s) to use for faceting when format is 'scatterplot'. When single name then facet wrap kind of faceting is used. When two names then facet grid kind of faceting is used. It overrides facet value in case of 'scatterplot'. Must be part of column list (e.g. include).
regressionLine	logical if TRUE then adds regression line to scatterplot.
corrLabel	column name to use to label correlation table: 'value', 'pair', or 'none' (default)

digits	number of digits to use in correlation table text (when displaying correlation coefficient value)
shape	shape of correlation figure (default is 21)
shapeSizeRange	correlation figure size range
facet	Logical - if TRUE then divide plot into facets for each COLUMN (defualt is FALSE - no facets). When set to TRUE and format is 'boxplot' scales defalut changes from 'fixed' to 'free'. Has no effect when format is 'corr'.
scales	Are scales shared across all facets: "fixed" - all are the same, "free_x" - vary across rows (x axis), "free_y" - vary across columns (Y axis) (default), "free" - both rows and columns (see in facet_wrap parameter scales. Also see parameter facet for details on default values.)
ncol	Number of columns in facet wrap.
coordFlip	logical flipped cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal (see coord_flip).
paletteName	palette name to use (run display.brewer.all to see available palettes).
baseSize	base font size.
baseFamily	base font family.
legendPosition	legend position.
defaultTheme	plot theme to use, default is theme_bw.
themeExtra	any additional ggplot2 theme attributes to add.
where	SQL WHERE clause limiting data from the table (use SQL as if in WHERE clause but omit keyword WHERE).
test	logical: when applicable if TRUE show what would be done, only (similar to parameter test in RODBC functions like sqlQuery and sqlSave). Doesn't apply when no sql expected to run, e.g. format is 'boxplot'.

Details

All formats support parameters include and except to include and exclude table columns respectively. The include list guarantees that no columns outside of the list will be included in the results. The excpet list guarantees that its columns will not be included in the results.

Format overview: produce set of histograms - one for each statistic measure - across table columns. Thus, it allows to compare averages, IQR, etc. across all or selected columns.

Format boxplot: produce boxplots for table columns. Boxplots can belong to the same plot or can be placed inside facet each (see logical parameter facet).

Format histogram: produce histograms - one for each column - in a single plot or in facets (see logical parameter facet).

Format corr: produce correlation matrix of numeric columns.

Format scatterplot: produce scatterplots of sampled data.

Value

a ggplot object

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                      server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# get summaries to save time
pitchingInfo = getTableSummary(conn, 'pitching_enh')
battingInfo = getTableSummary(conn, 'batting_enh')

# Boxplots
# all numerical attributes
showData(conn, tableInfo=pitchingInfo, format='boxplot',
         title='Boxplots of numeric columns')
# select certain attributes only
showData(conn, tableInfo=pitchingInfo, format='boxplot',
         include=c('wp','whip', 'w', 'sv', 'sho', 'l', 'ktobb', 'ibb', 'hbp', 'fip',
                   'era', 'cg', 'bk', 'baopp'),
         useIQR=TRUE, title='Boxplots of Pitching Stats')
# exclude certain attributes
showData(conn, tableInfo=pitchingInfo, format='boxplot',
       except=c('item_id','ingredient_item_id','facility_id','rownum','decadeid','yearid',
                'bfp','ipouts'),
         useIQR=TRUE, title='Boxplots of Pitching Stats')
# flip coordinates
showData(conn, tableInfo=pitchingInfo, format='boxplot',
       except=c('item_id','ingredient_item_id','facility_id','rownum','decadeid','yearid',
                'bfp','ipouts'),
         useIQR=TRUE, coordFlip=TRUE, title='Boxplots of Pitching Stats')

# boxplot with facet (facet_wrap)
showData(conn, tableInfo=pitchingInfo, format='boxplot',
         include=c('bfp','er','h','ipouts','r','so'), facet=TRUE, scales='free',
         useIQR=TRUE, title='Boxplots Pitching Stats: bfp, er, h, ipouts, r, so')

# Correlation matrix
# on all numerical attributes
showData(conn, tableName='pitching_enh', tableInfo=pitchingInfo,
         format='corr')

# correlation matrix on selected attributes
# with labeling by attribute pair name and
# controlling size of correlation bubbles
showData(conn, tableName='pitching', tableInfo=pitchingInfo,
         include=c('era','h','hr','gs','g','sv'),
         format='corr', corrLabel='pair', shapeSizeRange=c(5,25))

# Histogram on all numeric attributes
showData(conn, tableName='pitching', tableInfo=pitchingInfo, include=c('hr'),
         format='histogram')

# Overview is a histogram of statistical measures across attributes
```

```
showData(conn, tableName='pitching', tableInfo=pitchingInfo,
         format='overview', type='numeric', scales="free_y")

# Scatterplots
# Scatterplot on pair of numerical attributes
# sample by size with 1d facet (see \code{\link{facet_wrap}})
showData(conn, 'pitching_enh', format='scatterplot',
         include=c('so', 'er'), facetName="lgid", pointColour="lgid",
         sampleSize=10000, regressionLine=TRUE,
         title="SO vs ER by League 1980-2000",
         where='yearid between 1980 and 2000')

# sample by fraction with 2d facet (see \code{\link{facet_grid}})
showData(conn, 'pitching_enh', format='scatterplot',
         include=c('so','er'), facetName=c('lgid','decadeid'), pointColour="lgid",
         sampleFraction=0.1, regressionLine=TRUE,
         title="SO vs ER by League by Decade 1980 - 2012",
         where='yearid between 1980 and 2012')
}
```

---

showGraph *Plot an Aster graph object comprised of the vertice and edge tables.*

Description

Function for obtaining and plotting graph (network) objects using ggplot2 and ggnet2 functions.

Usage

```
showGraph(channel, graph, v = NULL, vertexWhere = graph$vertexWhere,
  edgeWhere = graph$edgeWhere, ..., allTables = NULL, test = FALSE)
```

Arguments

channel	connection object as returned by odbcConnect
graph	an object of class 'toagraph' referencing graph tables in Aster database.
v	a SQL SELECT that returns key values or a list of key values (corresponding to the vertex.names attribute) of the vertices to include in the graph. When not NULL this guarentees that no other vertices or edges between other vertices are included in the resulting network.
vertexWhere	optionally, a SQL WHERE clause to subset vertex table. When not NULL it overrides vertexWhere condition from the graph.
edgeWhere	optionally, a SQL WHERE clause to subset edge table. When not NULL it overrides edgeWhere condition from the graph.
...	Other arguments passed on to ggnet2 visualization function.
allTables	pre-built information about existing tables.
test	logical: if TRUE show what would be done, only (similar to parameter test in **RODBC** functions: sqlQuery and sqlSave).

Value

a ggplot object

Examples

```
if(interactive()) {
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                        server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
                          directed = FALSE, key = "officer",
                          source = "officer1", target = "officer2",
                        vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

# visualize the whole graph
showGraph(conn, policeGraphUn, node.label = "vertex.names", node.size="offense_count")

# visualize subgraph using both vertex and edge filters
showGraph(conn, policeGraphUn,
          vertexWhere = "officer ~ '[A-Z ].*'", edgeWhere = "weight > 0.10",
          node.label = "vertex.names", node.size="offense_count")

}
```

---

theme_empty *Creates empty theme.*

Description

Deprecated. Use theme_void instead. Good to use with slopegraphs.

Usage

```
theme_empty(baseSize = 12, baseFamily = "")
```

Arguments

baseSize	base font size
baseFamily	base font family

See Also

theme_void createHistogram and other visualization functions that start with create.

toaGraph *Define an object corresponding to a graph in Aster database.*

Description

In Aster Database, to process graphs using SQL-GR, it is recommended to represent a graph using two tables:

1. Vertices table
2. edges table

Vertices table must contain a unique key so that each row represents a vertex. Edges table must contain a pair of source and target keys (from vertices table) so that each row represents an edge. Both vertices and edges tables may contain additional columns representing optional attributes. For example if edges table has column 'weight' it can correspond a graph with edge weights.

Usage

```
toaGraph(vertices, edges, directed = FALSE, key = "id", source = "source",
  target = "target", vertexAttrnames = NULL, edgeAttrnames = NULL,
  vertexWhere = NULL, edgeWhere = NULL)
```

Arguments

vertices	A table, view, or query of a collection of vertices in the graph.
edges	A table, view, or query of a collection of edges of the graph. The collection must contain at least two lists of columns, one list that represents the source vertex key and another list that represents the target vertex key.
directed	logical: should edges be interpreted as directed?
key	name of the column with vertex unique id (in the table vertices).
source	name of the column with the from vertex (in the table edges).
target	name of the column with the to vertex (in the tbale edges).
vertexAttrnames	optionally, a list of columns containing vertex attribute names.
edgeAttrnames	optionally, a list of columns containing edge attribute names.
vertexWhere	optionally, a SQL WHERE clause to subset vertex table (use SQL as if in WHERE clause but omit the keyword WHERE).
edgeWhere	optionally, a SQL WHERE clause to subset edge table (use SQL as if in WHERE clause but omit the keyword WHERE).

Examples

```
# undirected graph
policeGraphUn = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_un",
     directed = FALSE, key = "officer", source = "officer1", target = "officer2",
     vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"))

# directed graph with the vertex filter
policeGraphDi = toaGraph("dallaspolice_officer_vertices", "dallaspolice_officer_edges_di",
     directed = TRUE, key = "officer", source = "officer1", target = "officer2",
     vertexAttrnames = c("offense_count"), edgeAttrnames = c("weight"),
     vertexWhere = "officer ~ '[A-Z].*'")
```

---

toaster *toaster: analytical and visualization toolbox for Teradata Aster*

Description

toaster provides simple 2-step approach: compute in Aster - visualize and analyze in R. Its ‘compute‘ functions use powerful in-database SQL statements and SQL/MR functions running inside Aster’s highly scalable parallel and distributed analytical platform. Then ‘create‘ functions visualize results with boxplots, scatterplots, histograms, heatmaps, word clouds, maps, or slope graphs.

---

toa_dep *Give a deprecation error, warning, or messsage, depending on version number.*

Description

Version numbers have the format <major>.<minor>.<subminor>, like 0.9.2. This function compares the current version number of toaster against the specified `version`, which is the most recent version before the function (or other object) was deprecated.

Usage

```
toa_dep(version, msg)
```

Arguments

`version`	The last version of toaster where this function was good (in other words, the last version where it was not deprecated).
`msg`	The message to print.

Details

toa_dep will give an error, warning, or message, depending on the difference between the current toaster version and the specified version.

If the current major number is greater than version's major number, or if the current minor number is more than 1 greater than version's minor number, give an error.

If the current minor number differs from version's minor number by one, give a warning.

If the current subminor number differs from version's subminor number, print a message.

See Also

gg_dep

---

token *Tokenize (or split) text and emit n-word combinations from a document.*

Description

When n=1 simply tokenize text and emit words with counts. When n>1 tokenized words are combined into permutations of length n within each document.

Usage

```
token(n, tokenSep = "+", ignoreCase = FALSE,
  delimiter = "[ \\t\\b\\f\\r]+", punctuation = NULL,
  stemming = FALSE, stopWords = FALSE, sep = " ", minLength = 1)
```

Arguments

n	number of words
tokenSep	a character string to separate the tokens when n > 1
ignoreCase	logical: treat text as-is (FALSE) or convert to all lowercase (true); Default is TRUE. Note that if the stemming is set to TRUE, tokens will always be converted to lowercase, so this option will be ignored.
delimiter	character or string that divides one word from the next. You can use a regular expression as the delimiter value.
punctuation	a regular expression that specifies the punctuation characters parser will remove before it evaluates the input text.
stemming	logical: If true, apply Porter2 Stemming to each token to reduce it to its root form. Default is FALSE.
stopWords	logical or string with the name of the file that contains stop words. If TRUE then that should be ignored when parsing text. Each stop word is specified on a separate line.
sep	a character string to separate multiple text columns.
minLength	exclude tokens shorter than minLength characters.

Value

pluggable token parser

`validateGraph` *Validate graph data in Aster for consistency.*

Description

Aster uses the following pair of tables to store graph data: - vertices table with unique key and optional attributes - Edges table with source, target and optional attributes Also see `toaGraph` Graph validation checks for the following:

- Both tables exist (error returned).
- Edge sources exist in the vertices table.
- Edge targets exist in the vertices table.
- No duplicate vertices defined.
- No duplicate edges defined.
- No loops (self-directed edges) defined.

Usage

```
validateGraph(channel, graph, weight = NULL,
  vertexWhere = graph$vertexWhere, edgeWhere = graph$edgeWhere,
  allTables = NULL, test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`graph`	an object of class `'toagraph'` referencing graph tables in Aster database.
`weight`	logical or character: if logical then `TRUE` indicates using `'weight'` edge attribute, otherwise no weight used. If character then use as a name for the edge weight attribute. The edge weight may apply with types `'clustering'`, `'shortestpath'` and centrality measures.
`vertexWhere`	SQL WHERE clause limiting data from the vertex table. This value when not null overrides corresponding value `vertexWhere` from `graph` (use SQL as if in WHERE clause but omit keyword WHERE).
`edgeWhere`	SQL WHERE clause limiting data from the edge table. This value when not null overrides corresponding value `edgeWhere` from `graph` (use SQL as if in WHERE clause but omit keyword WHERE).
`allTables`	pre-built information about existing tables.
`test`	logical: if TRUE show what would be done, only (similar to parameter `test` in **RODBC** functions: sqlQuery and sqlSave).

Examples

```
if(interactive()) {

# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

# undirected graph
policeGraphUn = toaGraph(vertices="dallaspolice_officer_vertices",
                         edges="dallaspolice_officer_edges_un",
                         directed=FALSE, key="officer",
                         source="officer1", target="officer2",
                         vertexAttrnames = c("offense_count"),
                         edgeAttrnames = c("weight"))
validateGraph(conn, policeGraphUn)

# directed graph
policeGraphDi = toaGraph(edges="dallaspolice_officer_vertices",
                         vertices="dallaspolice_officer_edges_di",
                         directed=TRUE, key="officer",
                         source="officer1", target="officer2",
                         vertexAttrnames = c("offense_count"),
                         edgeAttrnames = c("weight"))
validateGraph(conn, policeGraphDi, weight=TRUE)

}
```

`viewTableSummary` *Invoke a Data Viewer on table statistics.*

Description

view computed column statistics in a spreadsheet-style viewer in R.

Usage

```
viewTableSummary(tableInfo, types = NULL, include = NULL, except = NULL,
  basic = FALSE, percentiles = FALSE)
```

Arguments

`tableInfo`	data frame with columns statistics to display.
`types`	vector with types of columns to include: numerical (`"numeric"`), character (`"character"` or date/time (`"temporal"`)
`include`	a vector of column names to include. Output never contains attributes other than in the list.
`except`	a vector of column names to exclude. Output never contains attributes from the list.

basic	logical: if TRUE display minimum, maximum, average, deviation and mode (if present)
percentiles	logical: if TRUE display percentiles

Details

When both parameters basic and percentiles are FALSE view displays *all* statistics.

See Also

getTableSummary

Examples

```
if(interactive()){
# initialize connection to Lahman baseball database in Aster
conn = odbcDriverConnect(connection="driver={Aster ODBC Driver};
                       server=<dbhost>;port=2406;database=<dbname>;uid=<user>;pwd=<pw>")

pitchingInfo = getTableSummary(channel=conn, 'pitching_enh')
viewTableSummary(pitchingInfo, percentiles=TRUE)

viewTableSummary(pitchingInfo, types=c("numeric", "temporal"))
}
```

Index