Package 'toaster'

April 9, 2014

Type Package

Title analytics and visualization with Aster Database

Version 0.2.5

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

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

Description toaster (a.k.a 'to Aster') is a set of tools to perform
in-database analytics with Teradata Aster Discovery Platform.
toaster embraces simple approach by dividing tasks into 2 steps:
compute in Aster - visualize and analyze in R. toaster `compute` functions
use distributed, highly scalable, parallel SQL and map-reduce
for processing of large data sets in Aster database. Then
`create` functions visualize results with boxplots, scatter-
plots,histograms, heatmaps, word clouds, maps, or slope graphs. Advanced options
such as faceting, coloring, labeling, and others are supported with most plots.

URL https://grigory@bitbucket.org/grigory/toaster.git

SystemRequirements Teradata Aster 5.1 or higher, Teradata Aster
Analytical Foundation 5.10 or higher (5.11 or higher is recommended)

Depends R (>= 2.14), RODBC (>= 1.3-9)

Suggests testthat (>= 0.2), memoise

Imports plyr (>= 1.8), reshape2 (>= 1.2.2), ggplot2 (>= 0.9.3.1),scales, RColorBrewer (>= 1.0-
5), grid, wordcloud (>= 2.4),ggmap (>= 2.3)

License GPL-2

Collate 'toaster.R' 'misc.R' 'utils.R' 'computeCorrelations.R'
'computeHistogram.R' 'computeHeatmap.R' 'plotting.R' 'maps.R'
'showData.R' 'computeAggregates.R' 'computeBarchart.R'
'computeSample.R' 'computePercentiles.R' 'computeLm.R'

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
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").
aggregates	vector of SQL aggregates to compute. Aggregates may have optional aliases like in "AVG(era) avg_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).
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

```
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"
by	for optional grouping by one or more columns for faceting or alike (effectively these elements will be part of GROUP BY ...)
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.
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

```
# 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)
```

`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`.

Usage

```
computeCorrelations(channel, tableName, tableInfo,
  include, except = NULL, where = NULL, 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.
`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).
`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

```
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, ]
```

`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.

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	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: sqlQuery and sqlSave).

Details

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

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

```
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

```
# 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')
```

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, expr, where = NULL,
  test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`tableName`	Aster table name
`expr`	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'.
`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

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

```
model1 = computeLm(channel=conn, tableName="batting_enh", expr= ba ~ rbi + bb + so)
```

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,
  percentiles = c(0, 5, 10, 25, 50, 75, 90, 95, 100),
  by = NULL, where = NULL, stringsAsFactors = FALSE,
  test = FALSE)
```

Arguments

`channel`	connection object as returned by `odbcConnect`
`tableName`	Aster table name
`columnName`	name of the column to compute percentiles on
`percentiles`	integer vector with percentiles to compute. Values `0, 25, 50, 75, 100` will always be added if omitted.
`by`	for optional grouping by one or more values for faceting or alike. If used with `createBoxplot` then use first name for x-axis and the rest for 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).
`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).

Examples

```
# 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")
```

computeSample *Randomly sample data from the table.*

Description

Draws a sample of rows from the table randomly. The function offers two sampling schemes: - a simple binomial (Bernoulli) sampling on a row-by-row basis with given sample rate(s) - sampling a given number of rows without replacement The sampling can be applied to the entire table or can be refined with conditions.

Usage

```
computeSample(channel, tableName, sampleFraction,
  sampleSize, 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 not yet supported.)
sampleSize	total sample size (applies only when sampleFraction is missing).
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).

Examples

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

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

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, useIQR = FALSE,
  facet = NULL, ncol = 1, facetScales = "fixed",
  paletteValues = NULL, palette = "Set1",
  title = paste("Boxplots", ifelse(is.null(x), NULL, paste("by", x))),
  xlab = x, ylab = NULL, legendPosition = "right",
  coordFlip = FALSE, baseSize = 12, baseFamily = "sans",
  defaultTheme = theme_bw(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
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.
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.

baseSize	theme base font size
baseFamily	theme base font family
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 to use: theme_bw (default), theme_grey, theme_classic or custom.
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'.

See Also

computePercentiles for computing boxplot quartiles

Examples

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

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

# boxplots by the league with facet yearid of pitching ipouts in 2010s
ipopLgYear = computePercentiles(conn, "pitching", "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", "ba", 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), xlab = x,
  ylab = y, labelSize = 5, labelFamily = "",
  labelFontface = "plain", labelColour = "black",
  labelVJust = 0.5, labelHJust = 0.5, labelAlpha = 1,
  labelAngle = 0, legendPosition = "right",
  defaultTheme = theme_bw(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)
baseSize	theme base font size
baseFamily	theme base font family
xlim	a vector specifying the data range for the x scale and the default order of their display in the x axis.

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.
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.
defaultTheme	plot theme to use: `theme_bw` (default), `theme_grey`, `theme_classic` or custom.
themeExtra	any additional `theme` settings that override default theme.

See Also

`computeAggregates` computes data for the bubble chart.

`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), xlab = x, ylab = y,
  legendPosition = "right",
  defaultTheme = theme_bw(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
lowGradient	colour for low end of gradient.
midGradient	colour for mid point.
highGradient	colour for high end of gradient.
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.)
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.
defaultTheme	plot theme to use: theme_bw (default), theme_grey, theme_classic or custom.
themeExtra	any additional theme settings that override default theme.

See Also

computeHeatmap for computing data for heat map

---

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), xlab = x,
  ylab = y, legendPosition = "right", coordFlip = FALSE,
  defaultTheme = theme_bw(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
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
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).
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
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 to use: theme_bw (default), theme_grey, theme_classic or custom.
themeExtra	any additional theme settings that override default theme.

See Also

computeHistogram and computeBarchart to compute data for histogram

Examples

```
# 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, labelName = NULL,
    scaleRange = c(1, 6), shapeColour = "red",
    textColour = "black", textFamily = "mono",
    textFace = "plain", textSize = 4, facet = NULL,
    ncol = 1, facetScales = "fixed",
    geocodeFun = memoise(geocode), getmapFun = get_map,
    urlonly = FALSE, api_key = NULL, baseSize = 12,
    baseFamily = "sans", title = NULL,
    legendPosition = "right",
    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	name of the column with strings to be geocoded to determine longitude and latitude for each data point. If this value is specified then parameters lonName and latName are ignored.
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	name of the column to use for the artifact metric 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).
labelName	name of the column to use for the artifact label text when displaying data.
shapeColour	color of of artifact placed on map.
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 legends. ("left", "right", "bottom", "top", or two-element numeric vector. "none" is no legend.)
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 is scaled using values in `metricName` column 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).

Examples

```
data = computeAggregates(asterConn, "pitching",
             columns = c("name || ', ' || park teamname", "lgid", "teamid", "decadeid"),
              aggregates = c("min(name) name", "min(park) park", "avg(rank) rank",
                             "avg(attendance) attendance")
              )

geocodeMem = memoise(geocode)

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

`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),
    xlab = bin, ylab = count, legendPosition = "right",
    defaultTheme = theme_bw(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.
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.
defaultTheme	plot theme to use: theme_bw (default), theme_grey, theme_classic or custom.
themeExtra	any additional theme settings that override default theme.

Examples

```
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')
```

`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), 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_classic(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)
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
title	plot title
baseSize	base font size
baseFamily	base font family
defaultTheme	plot theme to use: theme_bw, theme_grey, theme_classic (default) or custom.
themeExtra	any additional theme settings that override default theme.

---

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

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.
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.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.

See Also

getTableSummary

Examples

```
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

Examples

```
getCharacterTypes()
```

getDateTimeColumns *Filter Date and Time Table Columns.*

Description

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

Usage

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

Arguments

tableInfo	data frame obtained by calling getTableSummary.
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.
names.only	logical: if TRUE returns column names only, otherwise full rows of tableInfo.

See Also

getTableSummary

Examples

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

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

---

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

getTableSummary

Examples

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

getNumericTypes *List Aster numeric data types.*

Description

List Aster numeric data types.

Usage

```
getNumericTypes()
```

Value

character vector with names of Aster numeric data types

Examples

```
getNumericTypes()
```

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(0, 5, 10, 25, 50, 75, 90, 95, 100),
  where = NULL, mock = 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.
`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.

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

```
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'))
```

getTemporalTypes *List Aster temporal data types.*

Description

List Aster temporal data types.

Usage

```
getTemporalTypes()
```

Value

character vector with names of Aster temporal data types

Examples

```
getTemporalTypes()
```

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 = FALSE, ncol = 4,
 scales = ifelse(facet & format %in% c("boxplot", "overview"), "free", "fixed"),
  coordFlip = FALSE, paletteName = "Set1", baseSize = 12,
  baseFamily = "sans", legendPosition = "none",
  defaultTheme = theme_bw(base_size = baseSize),
  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 Use one or more of the following with 'character' type: distinct_count,not_null_count. By default all measures above are used per respeictive type.
title	plot title
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'.
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 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.
ncol	Number of columns in facet wrap.

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.)
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 visual object.

Examples

```
# 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')
```

`theme_empty` *Creates empty theme.*

Description

Good to use with slopegraphs.

Usage

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

Arguments

`baseSize`	base font size
`baseFamily`	base font family

See Also

`createHistogram` and other visualization functions that start with create.

`toaster` *toaster: analytical and visualization toolbox for Teradata Aster Discovery and Big Data Analytics Platform.*

Description

toaster: analytical and visualization toolbox for Teradata Aster Discovery and Big Data Analytics Platform.

`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

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

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

Index