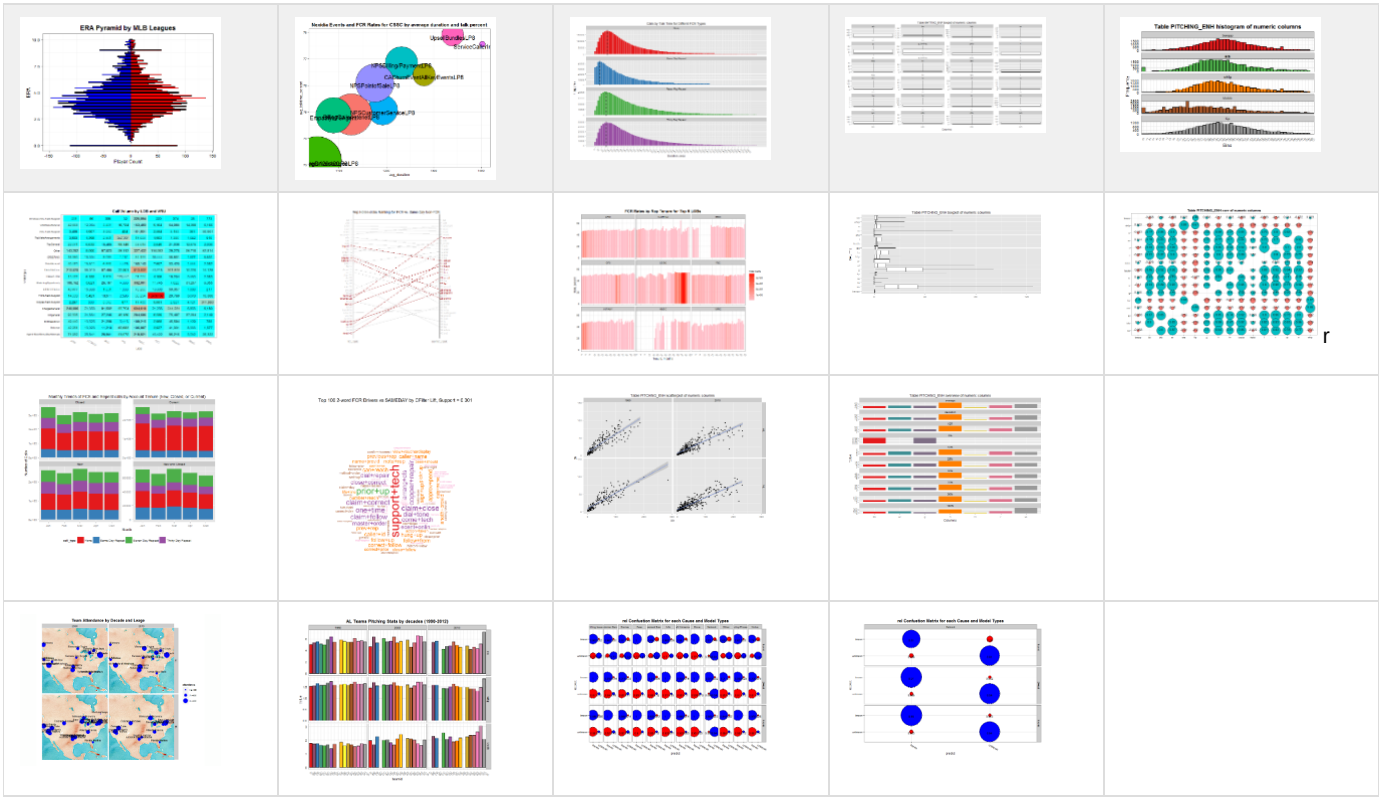
toAster (R package toaster)

Teasers



Overview

toaster is an R package for analyzing and visualizing Aster data sets.

toaster aims at combining power of Aster big data processing and analytics with versatility of R visualizations. **toaster** usually accomplishes each task in 2 steps:

1. compute results in Aster database
2. create visualization using computed results

When computing results **toaster** utilizes Aster SQL and Analytical Foundation functions, while for visualizations it relies on powerful **ggplot2** pack age, its extensions and sometimes other specialized visualization functions.

It is **toaster** convention that names of functions for computing in Aster begin with ***compute*** and functions for visualizations begin with ***create***. For example, function *computeHistogram* computes histogram results inside Aster using its histogram SQL/MR functions and function *createHistogra m* creates a histogram in R from the result of *computeHistogram*.

One exception from 2-step process is function *showData* that does everything. *showData* produces pre-defined set of visuals for initial data exploration in Aster and is a good starting tool to look at new data sets.

As of version 0.2 **toaster** offers scatter plot, histogram, bar chart, heatmap, bubble plot, word cloud, slope chart, population pyramid, and map visualizations.

Dependencies: R environment (see below), R packages: *RODBC*, *plyr*, *reshape2*, *ggplot2*, *scales*, *RColorBrewer*, *wordcloud*, *map*, Teradata Aster ODBC driver.

How To Install

R environment

toaster is R package which means it runs as part of R free software language and environment for statistical computing and graphics. To install R please download the installer and follow instructions from here (Windows) or here (Mac) (always use the latest available versions of R and packages unless specified otherwise).

I also recommend using RStudio Desktop (for enhanced R development experience).

Package Dependencies

The following packages are required for toaster: **RODBC, plyr, reshape2, ggplot2, scales, RColorBrewer, wordcloud, ggmap**. You can not simply install **toaster** without installing packages it depends on so install them first.
See here on how to install package in R; the following are instructions for RStudio (as of version 0.97.449):

- *Tools -> Install Packages..*
- Start typing name of the package in *Packages* box (it should complete name of the package automatically)
- *Install* (keep all defaults)

Repeat this for each package from the list (if upon loading **toaster** R still complaints about missing packages it means they haven't been installed yet - install them and reload **toaster** with *library* or *require* commands again).

Alternatively, run the following command in R (add/remove package names as necessary):

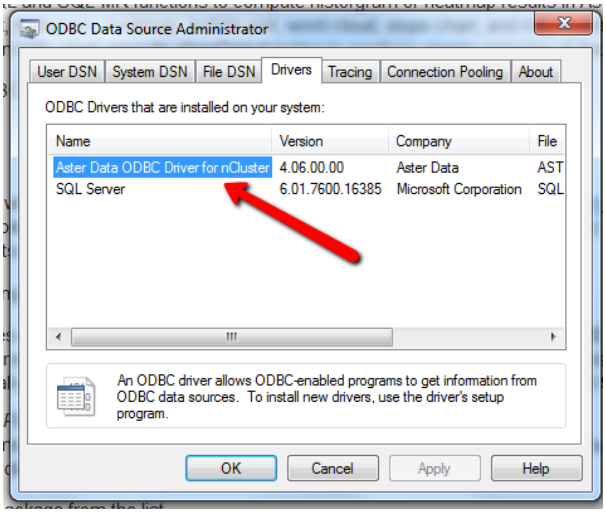
Installing Dependencies

```
install.packages(c('RODBC','plyr','reshape2','ggplot2','scales',
                   'RColorBrewer','wordcloud','ggmap'))
```

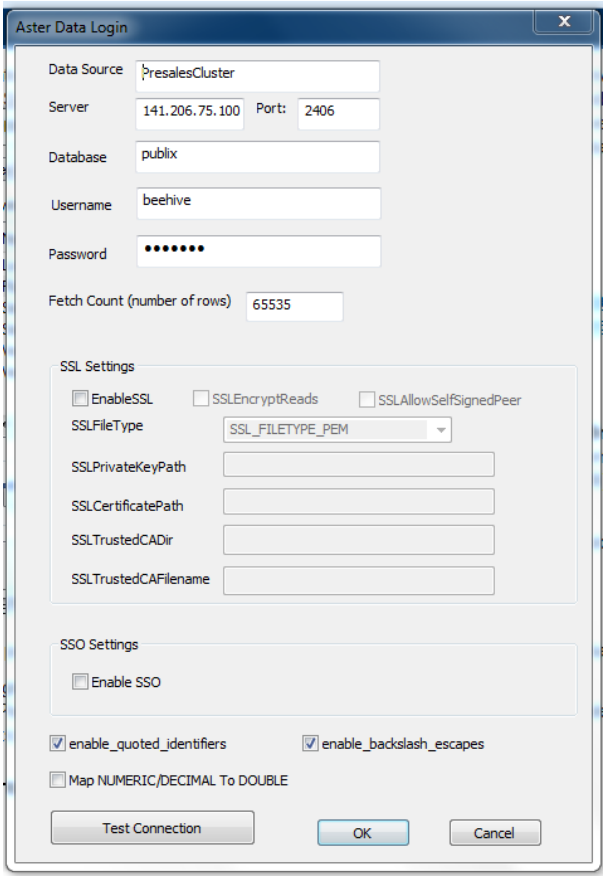
ODBC

toaster requires access to ODBC data source to establish connection to Aster database. First, obtain latest version of Teradata Aster ODBC driver for your environment and follow instructions to install it (download Windows version for 5.x here).

ODBC Driver for Aster installed in Windows:



For each database define a separate ODBC data source should you need access to it. Aster ODBC data sources defined in Windows:



toaster package

Now, you can install **toaster** package in R. You can download its latest version here. Again, see here on how install package in R or install package **toaster** with RStudio as follows:

- *Tools -> Install Packages…*
- Change *Install* from: *Package Archive* to *File* and then browse for ***toaster_0.x.zip*** file downloaded
- *Install* (keep all defaults)
- Load package using *require* command (or *library* command):

```
require(toaster)
```

How to get help

Using standard R help facility get more information on any **toaster** function, for example to get help on `computeBarchart`, the command is

```
> ?computeBarchart
```

or

```
> help(computeBarchart)
```

Demo data sets

All examples presented are for special demo data sets constructed for Aster from 2 sources: Lahman's Baseball database. and StratoCasters Flig ht XII telemetries provided by Matt Angelo. You can loosely follow these instructions to get it installed and available for testing the package:

- Make sure you have Aster client tools including *act* command line utility installed on your computer.
- Have Aster Analytics Foundation 5.10 (AF 5.11 or later preferred) installed on the cluster.
- In case it is not yet released use the following version of new histogram functions (uninstall existing and install these): hist_map and hist_ reduce (they contain the latest enhancements not available yet with AF 5.1).
- Download and expand folder baseball to local drive.
- Run shell script *load_baseball_data.sh* with at least 3 parameters, for example:
  `sh load_baseball_data.sh -d mydbname -U beehive -w beehive`
- Baseball data set is available in Aster now. In particular, tables **pitching_enh**, **batting_enh**, **teams_enh** contain few more derived statistics plus *decadeID* column.
- Download and expand folder stratocasters to local drive.
- Run shell script *create_stratocasters_data.sh* as above.
- Stratocasters data set is available in Aster now. In particular, table **flights**.

Getting Started with toaster

First follow steps above to configure Aster, R, ODBC to Aster, baseball demo data, Stratocasters demo data (optional), and **toaster**.

Test toaster and database connectivity in R

Run following commands in R:

Test toaster and connectivity

```
> library(toaster)
> dsn = "PresalesCluster"   # name of ODBC data source defined on your desktop (see
ODBC above)
> uid = "beehive"
> pwd = "beehive"
> asterConn = odbcConnect(dsn, uid, pwd)   # obtain connection to Aster database
> odbcGetInfo(asterConn)
               DBMS_Name                  DBMS_Ver           Driver_ODBC_Ver
Data_Source_Name
              "nCluster"            "05.10.00.00"                  "03.52"
"PresalesCluster"
             Driver_Name               Driver_Ver                 ODBC_Ver
Server_Name
       "NCLUSTERODBC.DLL"              "04.6.0000"             "03.80.0000"
"xxx.xxx.xxx.xxx"
```

After running last command `asterConn` you should receive output as shown above. `asterConn` is connection object as returned by RODBC function `odbcConnect` - it is required in all `computeXXXX` functions in **toaster** (i.e. functions that access database).

As with ODBC connection in any language close connection to release resources:

Closing ODBC connection

```
> close(asterConn)
> odbcGetInfo(asterConn)
Error in odbcGetInfo(asterConn) : argument is not an open RODBC channel
```

Examples will no longer repeat boiler plate code to connect to the database - refer to the example above when code contains `asterConn` or `con n` objects.

Summary Statistics

Function *getTableSummary* computes summary statistics on all or selected columns in the database table. Summary statistics include

getTableSummary Examples

```
pitchingInfo = getTableSummary(conn, 'pitching_enh')
# compute statistics on table pitching_enh

battingInfo = getTableSummary(conn, 'batting_enh', modeValue=TRUE, percentiles=c(5,
95),    # compute statistics on a subset of table batting_enh,
                              where='decadeid >= 1980')
# compute default statistics and mode, 5th and 95th percentiles

# (25th, 50th (median), and 75th percentiles are always computed)
```

No more boilerplate
Boilerplate code (lines 1-4, 11) that opens and closes ODBC connection using **RODBC** package will be omitted in the rest of examples.

Both `pitchingInfo` and `battingInfo` are data frames (R tables) with resuts (column statistics) in format: each row describes single column from table. To see what info and statistics are available list data frame column names:

List data frame column names

```
names(pitchingInfo)
```

To see table statistics in spreadsheet-style viewer in R (or RStudio) use function `viewTableSummary`:

View Statistics Table

```
viewTableSummary(battingInfo)

viewTableSummary(pitchingInfo, basic=TRUE, percentiles=TRUE)
```

Exploratory Analysis with *showData*

Syntax and General Use

Function *showData* is designed for quick visual exploratory analysis with Aster. As mentioned before, it combines both computational and visualization steps in a single call:

Usage: showData

```
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, sampleFraction = NULL, sampleSize = NULL,
         facetName = NULL, regressionLine = FALSE,
         corrLabel = 'none', digits = 2,
         shape = 21, shapeSizeRange = c(1,10),
         facet = FALSE, ncol = 4, scales = ifelse(facet & format=='boxplot',"free",
"fixed"),
         coordFlip = FALSE, paletteName = "Set1",
         baseSize = 12, baseFamily = "sans",
         defaultTheme = theme_bw(base_size = baseSize), themeExtra = NULL,
         where = NULL)
```

Time Saver
It is a good idea to compute and save column statistics with *`getTableSummary`* into R variable. Then pass it as`tableInfo` parameter to *`showData`* like this:

Boxplots

Boxplot is a graphical way of depicting descriptive statistics of numerical data through their quartiles. Below are series of boxplot visuals made with toaster showData function that show basic descriptive stats for pitching and batting between years 2000 and 2012:

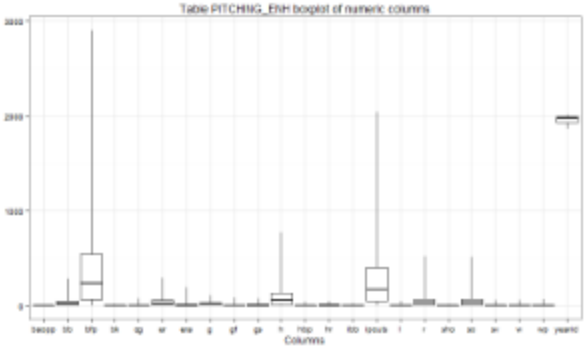
These table statistics will be used in all showData calls below

```
pitchingInfo = getTableSummary(asterConn, 'pitching_enh', where='yearid between 2000
and 2013')
battingInfo = getTableSummary(asterConn, 'batting_enh', where='yearid between 2000 and
2013')
teamsInfo = getTableSummary(asterConn, tableName='teams_enh', where='yearid between
1960 and 2013')
```

Straight forward example without any refinements:

All numeric columns in pitching

```
showData(asterConn, tableName='pitching_enh', tableInfo=pitchingInfo,
format='boxplot')
```



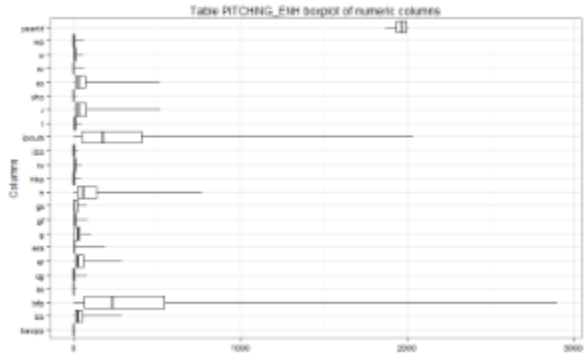
Prebuilding table stats for using with showData

```
pitcingInfo = getTableSummary(conn, 'pitching_enh', where = "lgid in ('AL','NL') and
decadedid >= 1970")

showData(conn, 'pitching_enh', tableInfo = pitchingInfo, format = "boxplot", where =
"lgid in ('AL','NL') and decadedid >= 1970")

showData(conn, 'pitching_enh', tableInfo = pitchingInfo, format = "corr", where =
"lgid in ('AL','NL') and decadedid >= 1970")

showData(conn, 'pitching_enh', tableInfo = pitchingInfo, format = "histogram", include
= c('era','h','bb','hr'), where = "lgid in ('AL','NL') and decadedid >= 1970")
```

Consistency of where clause
Remember to keep parameter `where` consistent across these calls (when using it) - otherwise computed summary statistics are not compatible with data shown.

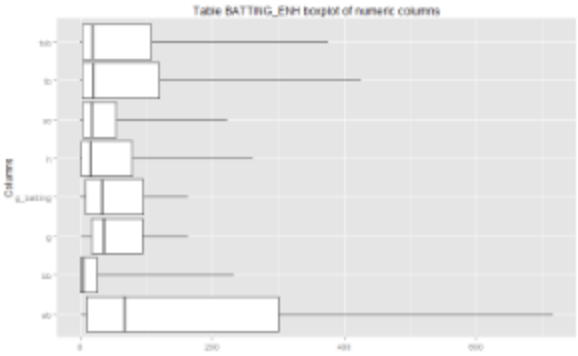
The same with coordinates flipped:

```
showData(asterConn, tableName='pitching_enh', tableInfo=pitchingInfo,
format='boxplot',
         coordFlip=TRUE)
```



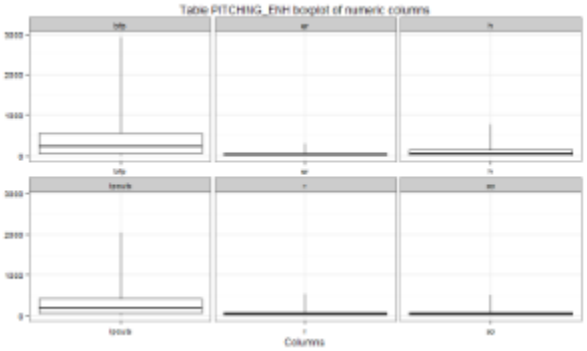
Include only certain variables (columns) and change background (theme) to grey:

```
showData(asterConn, tableName='batting_enh', tableInfo=battingInfo, format='boxplot',
         include = c('ab','tb','tob','so','h','bb','g_batting','g'),
         coordFlip=TRUE,
         defaultTheme=theme_grey(),
         where='yearid between 2000 and 2013')
```



and lastly place each boxplot in its own space (using facets):

```
showData(asterConn, tableName='pitching_enh', tableInfo=pitchingInfo,
format='boxplot',
         include=c('bfp','er','h','ipouts','r','so'), ncol=3,
         facet=TRUE, scale="free_x")
```

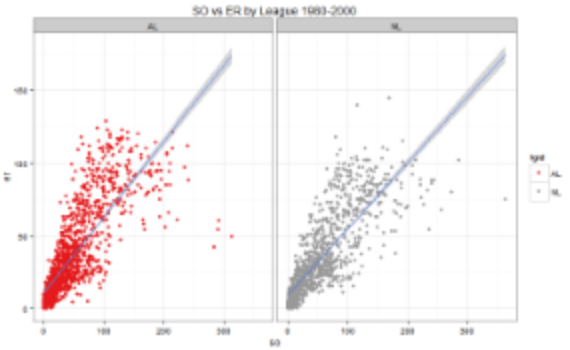


Scatterplots

Scatterplot is a type of mathematical diagram to display values for two variables. When combined with fitting function (linear, polynomial, splines, smoothing, etc.) scatterplot could be a powerful visualization tool to understand and visualize relationships in 2 dimensions. **toaster** adds faceting to scatterplots to extend visualizations across 3d and 4th (commonly categorical) dimensions.

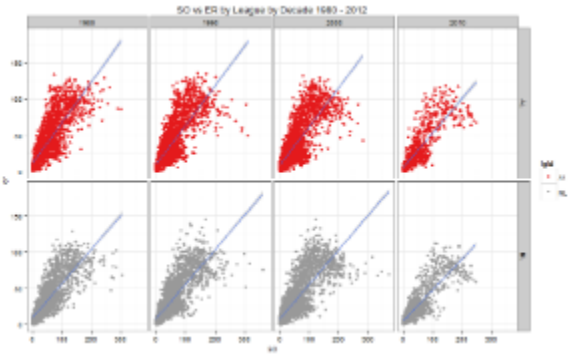
Let's compare pitching between two leagues between for the time period 1980 - 2000 (using facets of AL and NL leagues):

```
showData(asterConn, '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')
```



Compare how pitching trended by decade across both leagues (using facets of leagues as above plus facets for decades) :

```
showData(asterConn, 'pitching_enh', format='scatterplot',
         include=c('so','er'), facetName=c('lgid','decadeid'), pointColour="lgid",
         sampleSize=50000, regressionLine=TRUE,
         title="SO vs ER by League by Decade 1980 - 2012",
         where='yearid between 1980 and 2012')
```



Computational and Visualization Functions

Moving beyond *showData* we enter specialized functions that compute in Aster and bring results back into R. Thus, they utilize Aster map-reduce and massively parallel processing (MPP) SQL database. **toaster** conventionally uses pattern *computeXXXX* for such functions. Additionally, each compute function has corresponding plotting function that presents computed results with proper visualization. **toaster** uses pattern createYYYY in their names.

Percentiles and Boxplots

We begin with summary statistics function to compute percentiles on Aster columns *computePercentiles*:

Usage: computePercentiles function

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

Computing single set of percentiles on a table column and use *createBoxplot* to visualize results with box plot:

Compute BA percentiles on all players

```
allBAPerc = computePercentiles(conn, "batting_enh", "ba")

createBoxplot(allBAPerc, fill=NULL, title="BA Boxplot")
```

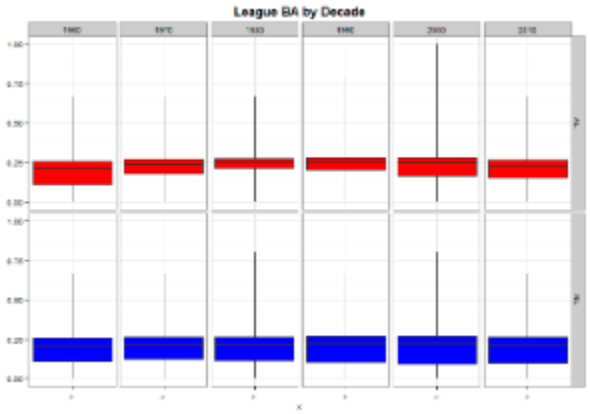
Last function displays:

The real power of createPercentiles is in its ability to compute multiple percentiles on a column by varying across one or more other categories. For this we use *by* parameter:

BA Percenttiles by League and Decade

```
teamsBAbyDecadePerc = computePercentiles(conn, "batting_enh", "ba",
                                         by=c("lgid","decadeid"),
                                         where="yearid>=1960 and lgid in ('AL','NL')")

createBoxplot(teamsBAbyDecadePerc, facet=c("lgid","decadeid"), fill="lgid",
              paletteValues = c("red","blue"), legendPosition="none",
              title="League BA by Decade")
```

produces this plot:



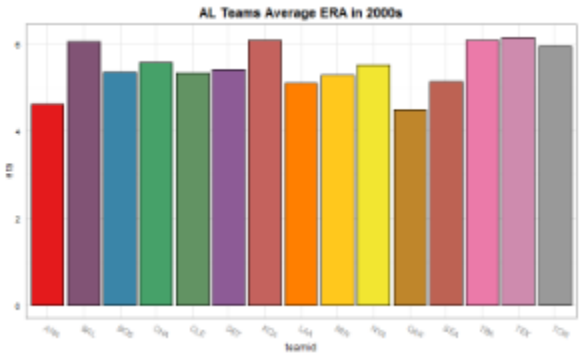
Bar Charts

Given column with categorical values **toaster** computes aggregates with *computeBarchart* (step 1) and visualizes them as bar chart with *createHistogram* (step 2).

computeBarchart

```
bc = computeBarchart(channel=conn, tableName="pitching_enh", category="teamid",
                     aggregates="AVG(era) era",
                     where="yearid >= 2000 and lgid='AL'")

createHistogram(bc, "teamid", "era", fill="teamid",
                title = "AL Teams Average ERA in 2000s", legendPosition="none")
```

creates this bar chart:

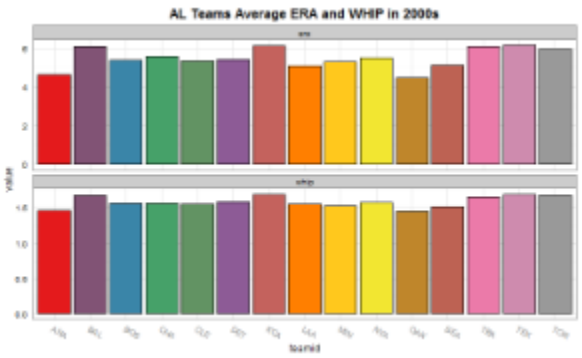


To compute and visualize multiple aggregates of the same category use vector for *`aggregates`* and *`withMelt`*=TRUE to un-pivot results. Note using different names of columns inside of *`createHistogram`* (due to *`withMelt`*=TRUE) and use of parameter `facet` in *`createHistogram`*:

Bar Chart with Multiple Aggregates

```
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'")

createHistogram(bc, "teamid", "value", fill="teamid", facet="variable",
                title = "AL Teams Average ERA and WHIP in 2000s",
legendPosition="none")
```

creates this bar chart:



Parameters *`orderBy`* and *`top`* make sense when collecting top results for certain aggregate and they are not compatible with parameter *`by`*.When using both multiple aggregates and parameter *`by`* then faceting must be 2-dimensional.

Histograms

Aster 5.10 or later offers new enhanced SQL/MR functions to compute histograms and **toaster** utilizes new features introduced in them.

> **Using Aster AF 5.10**
> Version of **hist_map** in AF 5.10 uses group option `BY` while in versions starting with 5.11 it is `GROUP_COLUMNS`. AF 5.11 or later also supports multiple group columns in it. **toaster** fully utilizes this feature in **computeHistogram** function with its **by** parameter. To use older syntax with **computeHistogram** call it with parameter **`oldStyle=TRUE`**, otherwise it will default to latest (5.11 or later) version.

Linear Regression

Aster linear regression functions are now available using **toaster** function *computeLm*:

Linear Regression Coefficients

```
modelNL = computeLm(channel=conn, tableName="pitching_enh", expr= era ~ er + hr + bb +
so,
                    where = "yearid >= 2000 and lgid = 'NL'")
modelAL = computeLm(channel=conn, tableName="pitching_enh", expr= era ~ er + hr + bb +
so,
                    where = "yearid >= 2000 and lgid = 'AL'")
```

this produces:

Regression coefficients for AL and NL models

```
> modelNL
  coefficient_name coefficient_index       value
1                0                 0  6.65264864
2               er                 1  0.08736568
3               hr                 2 -0.07910826
4               bb                 3 -0.05912771
5               so                 4 -0.03660032
> modelAL
  coefficient_name coefficient_index       value
1                0                 0  6.67008710
2               er                 1  0.08225370
3               hr                 2 -0.07092817
4               bb                 3 -0.05395920
5               so                 4 -0.04061422
```

Maps

toaster plans to offer several mapping function. First implemented function `createUS48Map` creates visualizations over all or part of 48 continental US states:

Usage: createUS48Map (as of ver. 0.2)

```
createUS48Map(states = 'United States', zoom = "auto",
              maptype = "terrain",
              mapColor = c("color", "bw"),
              source = c("google", "osm", "stamen", "cloudmade"),
              data, locationName = NULL, locationNameBak = NULL,
              lonName = "LONGITUDE", latName = "LATITUDE",
              facet = NULL, ncol = 1, facetScales = "fixed",
              metricName, labelName = NULL,
              scaleRange = c(1,6),
              shapeColour = "gold2", textColour = shapeColour,
              geocodeFun,
              baseSize = 12, baseFamily = "sans",
              title = paste("US Map:", states, "of", metricName),
              legend.position = "right",
              defaultTheme = theme_bw(base_size = baseSize),
              themeExtra = NULL)
```

Internet Access
Using map functions requires access to internet to download map images and optionally to geocode locations. It may use various sources including Google Maps, OpenStreetMap and others, and may display messages like this:

Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=ITALY&sensor=false Google Maps API Terms of Service : http://developers.google.com/maps/terms

Without internet access map functions will fail with errors.

Example:

Team Attendance by Leage and Decade (2000-2012)

```
# step 1: compute in Aster
data = compute(asterConn, "pitching",
                columns = c("name || ', ' || park teamname", "lgid", "teamid",
"decadeid"),
                aggregates = c("min(name) name", "min(park) park", "avg(rank) rank",
"avg(attendance) attendance")
                )

# see Tip below about geocode and memoise functions
geocodeMem = memoise(geocode)

# step 2: visualize
createUS48Map(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='Yearly Game Attendance by Decade and League (yearly,
2000-2012)',
               geocodeFun=geocodeMem)
```

Cache geocode function with memoise
When using geocoding in maps it is advised to cache its results to minimize access to Google Maps API. For one reason it's slow, but also The Geocoding API has a number of request limitations in place to prevent abuse: an unspecied short-term rate limit is in place as well as a 24-hour limit of 2,500 requests.

For example use package memoise to avoid repeating calls to Geocoding API:

```
require(toaster)
require(memoise)

geocodeMem = memoise(geocode)
mymap = createUS48Map(..., geocodeFun=geocodeMem)
```

Charts Gallery

This section will contain all the image charts available through ***toaster***.

	Boxplots
	Scatterplots
	Histograms
	Barcharts
	Heatmaps
	Population Pyramids
	Maps

Known Problems

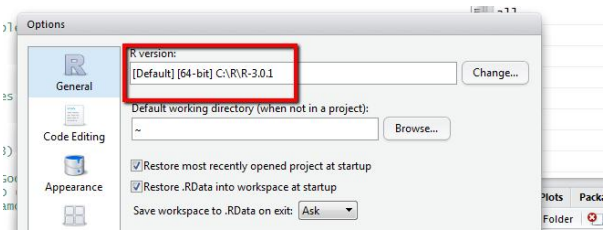
Environment and Configuration

32-bit vs. 64-bit R and ODBC versions

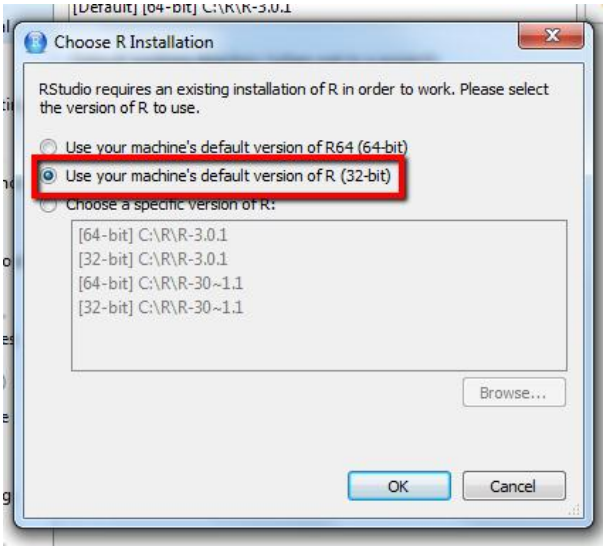
ODBC driver when attempting to connect to Aster in R throws error: "The specified DSN contains an architecture mismatch between the Driver and Application":

```
> require(toaster)
>
> dsn = 'Aster510'
> odbcConnect(dsn, 'beehive', 'beehive')
[1] -1
Warning messages:
1: In odbcDriverConnect("DSN=Aster510;UID=beehive;PWD=beehive") :
  [RODBC] ERROR: state IM014, code 0, message [Microsoft][ODBC Driver
Manager] The specified DSN contains an architecture mismatch between the
Driver and Application
2: In odbcDriverConnect("DSN=Aster510;UID=beehive;PWD=beehive") :
  ODBC connection failed
```

The problem is in mismatch between version of R and version of ODBC driver. If you use 64-bit R then use 64-bit ODBC driver, if you still use 32-bit R then ODBC driver must be 32-bit version as well.
You can find out about version of R in RStudio by selecting **Tools --> Options** in the menu:



and changing it there as necessary:



Updating toaster from package archive file

When updating a package that is already installed in R library follow instructions for installing it - just make sure package is unloaded from R environment first. For example, to update **toaster** from the archive unload it from memory first:

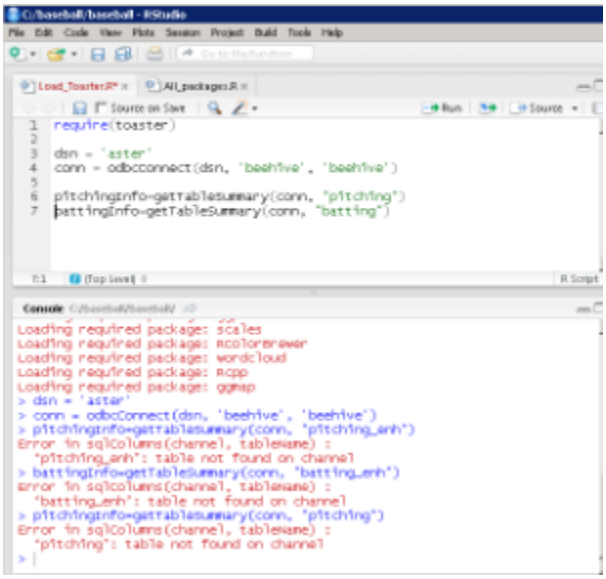
> ### Unloading package
>
> ```
> detach("package:toaster", unload=TRUE)
> ```

then install package (see **How to Install** above) and load it with *require* or *library* command.

Connectivity

Error 'table not found on channel'

When working with ODBC connections obtained with *odbcConnect* they time out after some period of inactivity. R will also reset connections every time it reloads **RODBC** or **toaster** package. When attempting using reset connection error ***'some-table-name-here': table not found on channel*** appears:



Make sure you close and open connection again to resolve it.

Permissions and Authorization

Error 'argument is of length zero'

While there are multiple scenarios for this error one typical case is lack of permission to execute SQL/MR function in Aster. The permission belongs to user id specified when connecting to Aster (obtaining connection object with RODBC odbcConnect function):

Aster connection credentials

```
dsn = "data-source-name"
uid = "beehive"
pwd = "beehive-password"
conn = odbcConnect(dsn, uid, pwd)
```

In example above Aster user *beehive* must have **EXECUTE** privilege to run SQL/MR function. The syntax to give the privilege is:

Grant EXECUTE privilege to run function

```
GRANT EXECUTE
   ON FUNCTION <schema-name>.<function-name>
   TO <user-name or group-name or PUBLIC>;
```

For details, see p.54 of Aster Analytics Foundation User Guide 5.11 (in the Chapter 2: Installing Analytical Functions).

References and Links

General

- The R Project for Statistical Computing
- RStudio IDE
- RODBC package
- ggplot2 package
- Lahman's Baseball Database
- Aster Client Tools (for ODBC driver)
- <placeholder>
- Baseball Statistics

Handy R Links

- Colors in R
- Documenting R Functions