ORACLE®

Learning R Series

Session 3: Oracle R Enterprise 1.3 Embedded R Execution

Mark Hornick, Senior Manager, Development
Oracle Advanced Analytics

Learning R Series 2012

Session	Title
Session 1	Introduction to Oracle's R Technologies and Oracle R Enterprise 1.3
Session 2	Oracle R Enterprise 1.3 Transparency Layer
Session 3	Oracle R Enterprise 1.3 Embedded R Execution
Session 4	Oracle R Enterprise 1.3 Predictive Analytics
Session 5	Oracle R Enterprise 1.3 Integrating R Results and Images with OBIEE Dashboards
Session 6	Oracle R Connector for Hadoop 2.0 New features and Use Cases

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.

Topics

- Introduction to Embedded R Execution: What and Why?
- Embedded R Scripts
  - Execution through the R interface
  - Execution through the SQL interface
- ORE 1.3 New Features
  - Working with connections and auto-connect
  - Generating PNG image streams
  - ORE-defined graphics function examples
- Example of ORE Workflow for Model Building and Scoring
- Summary

Embedded R Execution

- Ability to execute R code on the database server
- Execution controlled and managed by Oracle Database
- Eliminates loading data to the user's R engine and result write-back to Oracle Database
- Enables data- and task-parallel execution of R functions
- Enables SQL access to R: invocation and results
- Supports use of open source CRAN packages at the database server
- R scripts can be stored and managed in the database
- Schedule R scripts for automatic execution

Motivation – why embedded R execution?

- Facilitate application use of R script results
  - Develop/test R scripts interactively with R interface
  - Invoke R scripts directly from SQL for production applications
  - R Scripts stored in Oracle Database
- Improved performance and throughput
  - Oracle Database data- and task-parallelism
  - Compute and memory resources of database server, e.g., Exadata
  - More efficient read/write of data between Oracle Database and R Engine
  - Parallel simulations
- Image generation at database server
  - Available to OBIEE and BI Publisher, or any such consumer
  - Rich XML, image streams

Embedded R Execution – R Interface

Embedded Script Execution – R Interface

Execute R scripts at the database server

R Interface function	Purpose
ore.doEval()	Invoke stand-alone R script
ore.tableApply()	Invoke R script with ore.frame as input
ore.rowApply()	Invoke R script on one row at a time, or multiple rows in chunks from ore.frame
ore.groupApply()	Invoke R script on data partitioned by grouping column of an ore.frame
ore.indexApply()	Invoke R script N times
ore.scriptCreate()	Create an R script in the database
ore.scriptDrop()	Drop an R script in the database

Embedded Script Execution – R Interface

ORE function	Signature
ore.doEval	ore.doEval(FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL)
ore.tableApply	ore.tableApply(X, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, parallel = FALSE)
ore.rowApply	ore.rowApply(X, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, rows = 1, parallel = FALSE)
ore.groupApply	ore.groupApply(X, INDEX, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, parallel = FALSE)
ore.indexApply	ore.indexApply(times, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, parallel = FALSE)
ore.scriptDrop	ore.scriptDrop(name)
ore.scriptCreate	ore.scriptCreate(name, FUN)

Embedded Script Execution – R Interface

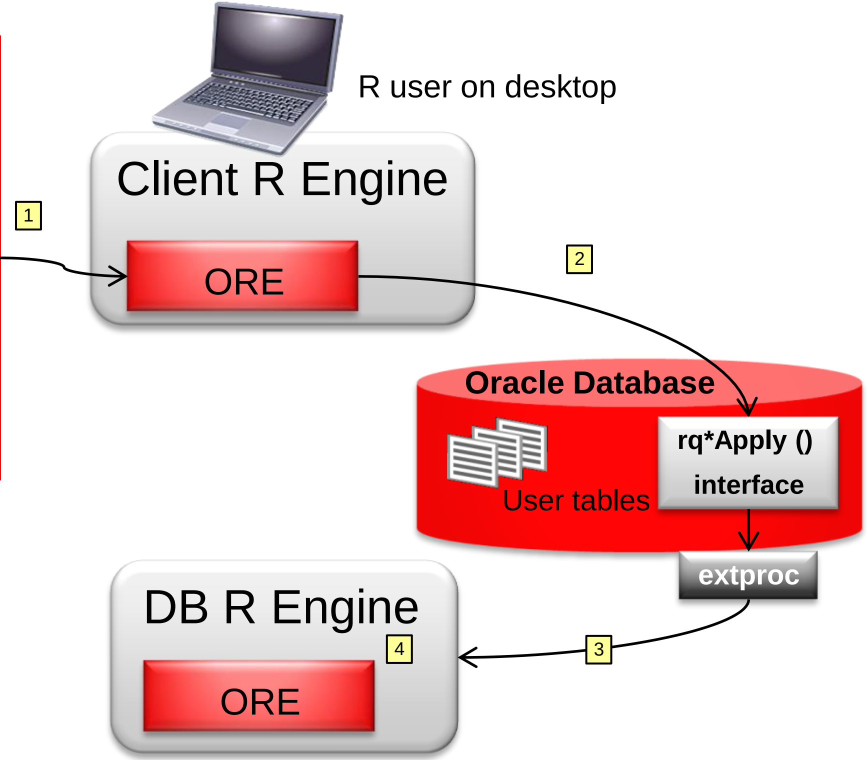
ORE function	Input data	FUN.VALUE	Arguments	Function	Special
ore.doEval()	None Generated within R function Load via ore.pull Transparency layer	NULL (returns ore.object) *or* data.frame or ore.frame used as a template for the return value (returns ore.frame)	Optional control arguments	FUN.NAME= name of function stored in R script repository *or* FUN = function *NOTE: For table/row/groupApply, first argument corresponds to input data as data.frame object. For indexApply, first argument corresponds to index number.*	Not applicable
ore.tableApply()	X = ore.frame		… arguments to function can be NULL or of the form <argument> = <value> Optional control arguments		parallel=T/F
ore.rowApply()					rows >= 1, the maximum number of rows in each chunk parallel=T/F
ore.groupApply()					INDEX = list or ore.frame object referencing ore.factor objects/columns with same length as X parallel=T/F
ore.indexApply()	None Generated within R function Load via ore.pull Transparency layer				times = number of times to execute the function parallel=T/F

ore.doEval – invoking a simple R script

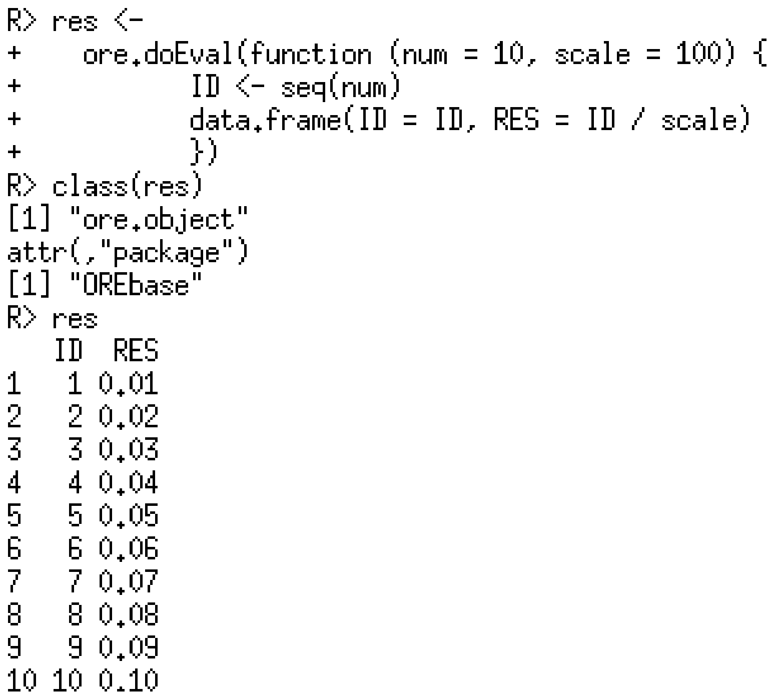
```
res <-
   ore.doEval(function (num = 10, scale = 100) {
          ID <- seq(num)
          data.frame(ID = ID, RES = ID / scale)
          })
class(res)
res
local_res <- ore.pull(res)
class(local_res)
local_res
```

Goal: scales the first n integers by value provided

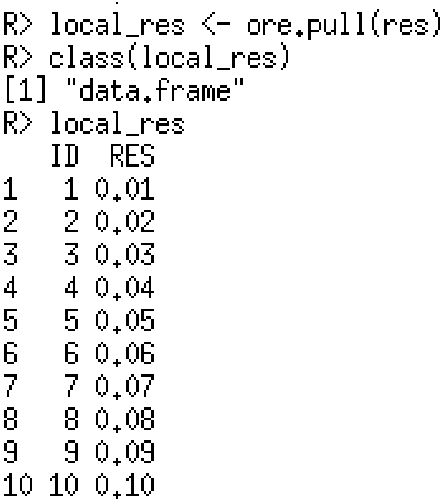
Result: a serialized R data.frame



Results



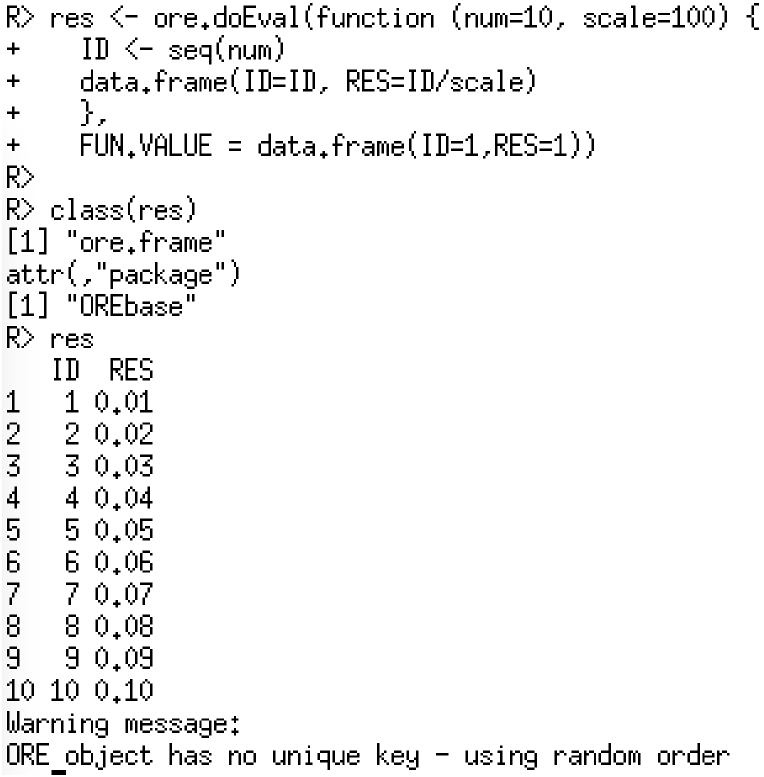
```
R> res <-
+    ore.doEval(function (num = 10, scale = 100) {
+             ID <- seq(num)
+             data.frame(ID = ID, RES = ID / scale)
+             })
R> class(res)
[1] "ore.object"
attr(,"package")
[1] "OREbase"
R> res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
```



```
R> local_res <- ore.pull(res)
R> class(local_res)
[1] "data.frame"
R> local_res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
```

ore.doEval – specifying return value

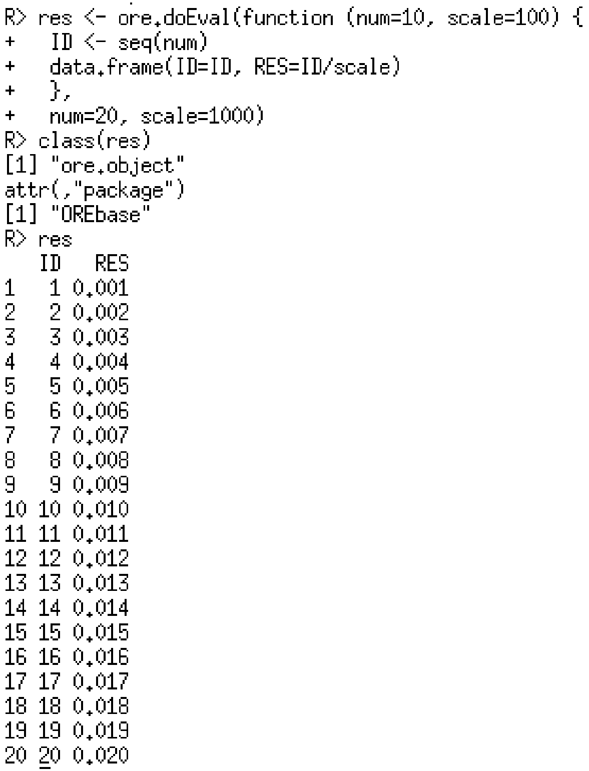
```
res <-
  ore.doEval(function (num = 10, scale = 100) {
                ID <- seq(num)
                data.frame(ID = ID, RES = ID / scale)
                },
         FUN.VALUE = data.frame(ID = 1, RES = 1))
class(res)
res
```



```
R> res <- ore.doEval(function (num=10, scale=100) {
+     ID <- seq(num)
+     data.frame(ID=ID, RES=ID/scale)
+     },
+     FUN.VALUE = data.frame(ID=1,RES=1))
R>
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
Warning message:
ORE_object has no unique key - using random order
```

ore.doEval – changing parameters

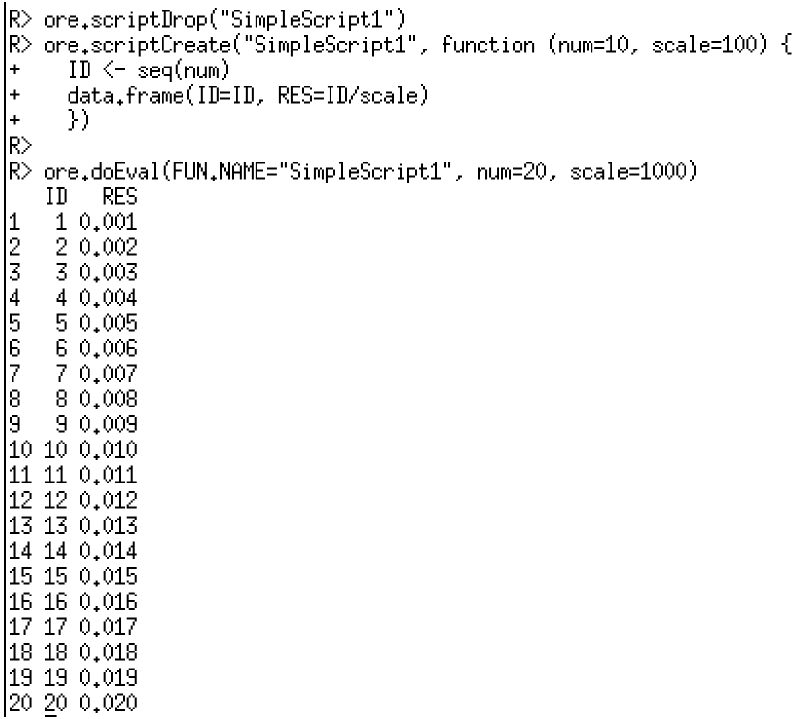
```
res <-
   ore.doEval(function (num = 10, scale = 100) {
                  ID <- seq(num)
                  data.frame(ID = ID, RES = ID / scale)
                  },
             num = 20, scale = 1000)
class(res)
res
```



```
R> res <- ore.doEval(function (num=10, scale=100) {
+   ID <- seq(num)
+   data.frame(ID=ID, RES=ID/scale)
+   },
+   num=20, scale=1000)
R> class(res)
[1] "ore.object"
attr(,"package")
[1] "OREbase"
R> res
   ID   RES
1   1 0.001
2   2 0.002
3   3 0.003
4   4 0.004
5   5 0.005
6   6 0.006
7   7 0.007
8   8 0.008
9   9 0.009
10 10 0.010
11 11 0.011
12 12 0.012
13 13 0.013
14 14 0.014
15 15 0.015
16 16 0.016
17 17 0.017
18 18 0.018
19 19 0.019
20 20 0.020
```

ore.doEval – using R script repository

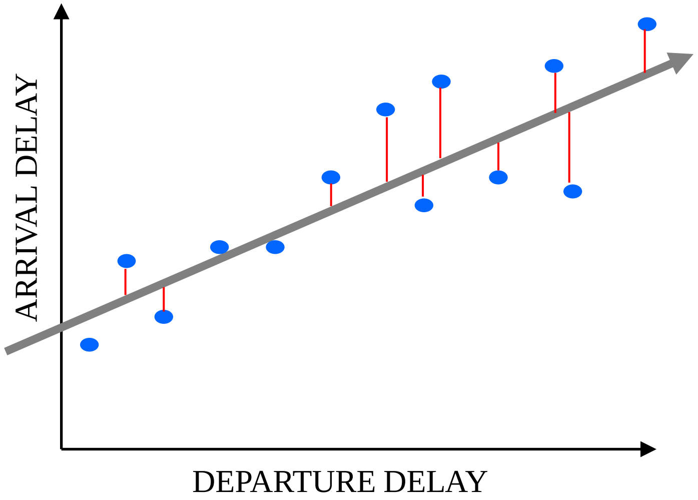
```
ore.scriptDrop("SimpleScript1”)
ore.scriptCreate("SimpleScript1”,
          function (num = 10, scale = 100) {
            ID <- seq(num)
            data.frame(ID = ID, RES = ID / scale)
            })
res <- ore.doEval(FUN.NAME="SimpleScript1",
                  num = 20, scale = 1000)
```



```
R> ore.scriptDrop("SimpleScript1")
R> ore.scriptCreate("SimpleScript1", function (num=10, scale=100) {
+    ID <- seq(num)
+    data.frame(ID=ID, RES=ID/scale)
+    })
R>
R> ore.doEval(FUN.NAME="SimpleScript1", num=20, scale=1000)
   ID   RES
1   1 0.001
2   2 0.002
3   3 0.003
4   4 0.004
5   5 0.005
6   6 0.006
7   7 0.007
8   8 0.008
9   9 0.009
10 10 0.010
11 11 0.011
12 12 0.012
13 13 0.013
14 14 0.014
15 15 0.015
16 16 0.016
17 17 0.017
18 18 0.018
19 19 0.019
20 20 0.020
```

Regression – e.g. using lm or ore.lm

Predict a continuous numerical value



For a simple dataset with two variables,
a line can be used to approximate the values

$$y = mx + b$$

Build a *model*, i.e., compute coefficients, that can be expressed in terms of values (m, b)

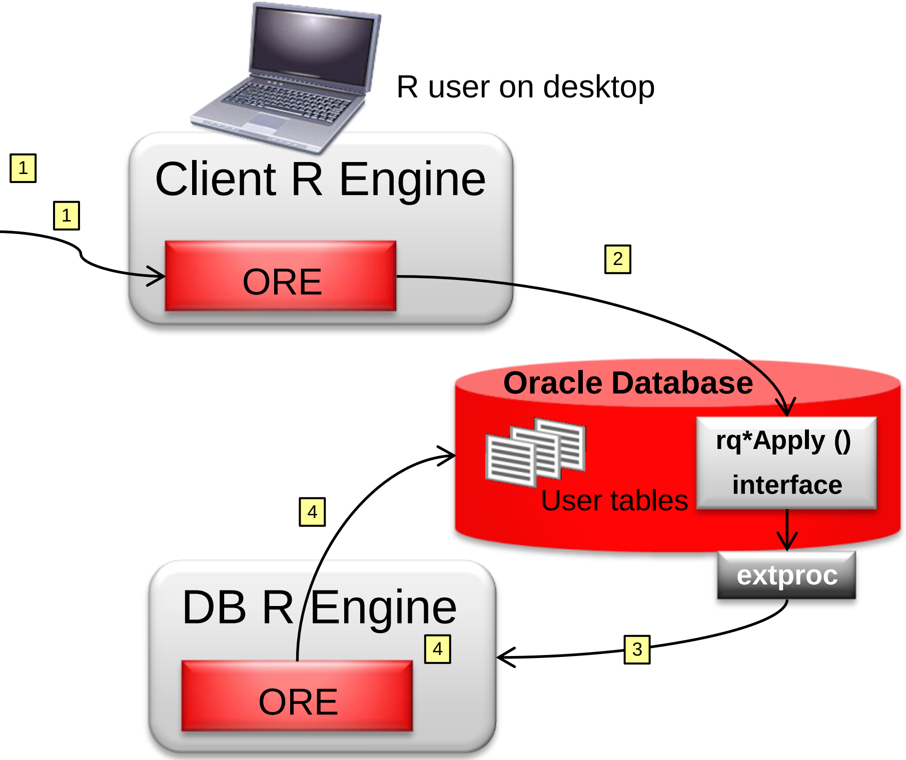
Models aren't perfect…when used for scoring, or making predictions, they may have an error component

Metrics like Root Mean Square Error (RMSE) are useful for assessing and comparing models

Scoring can be *batch* or *real-time*

ore.doEval – pulling data from Oracle Database

```
mod <- ore.doEval(
   function() {
     ore.sync(table="ONTIME_S")
     dat <- ore.pull(ore.get("ONTIME_S"))
     lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
   },
   ore.connect = TRUE);
mod_local <- ore.pull(mod)
class(mod_local)
summary(mod_local)
```

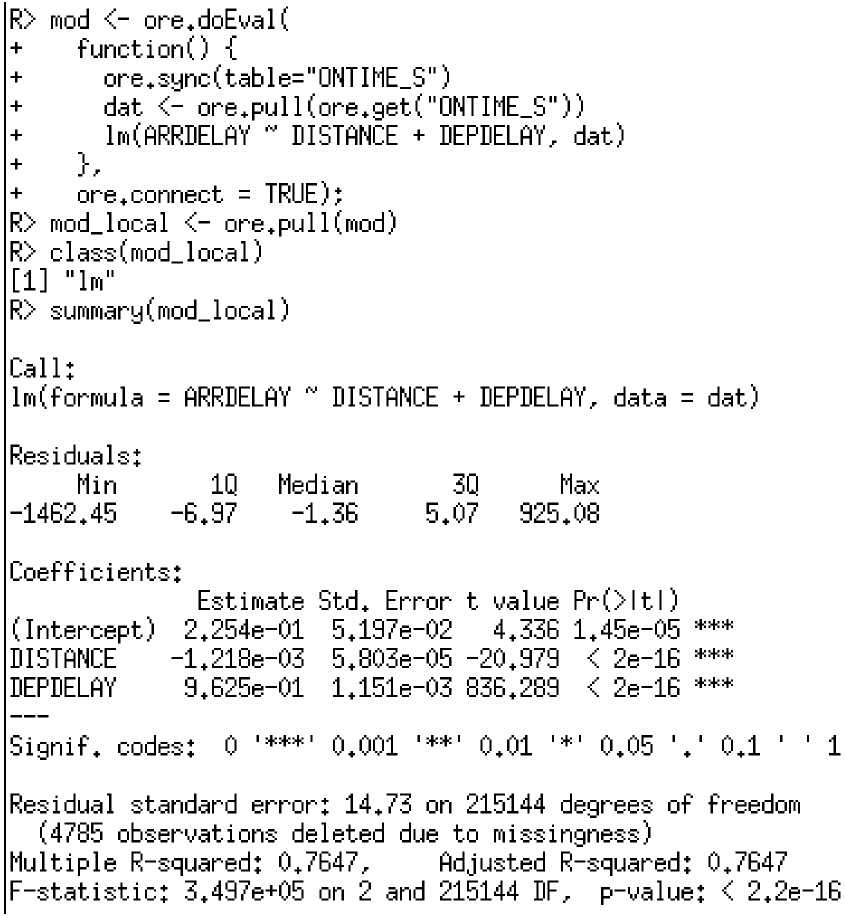


Goal: Build a single regression model retrieving data using Transparency Layer

Data explicitly loaded into R memory at DB R Engine using ore.pull()

Result "mod" returned as an R model object

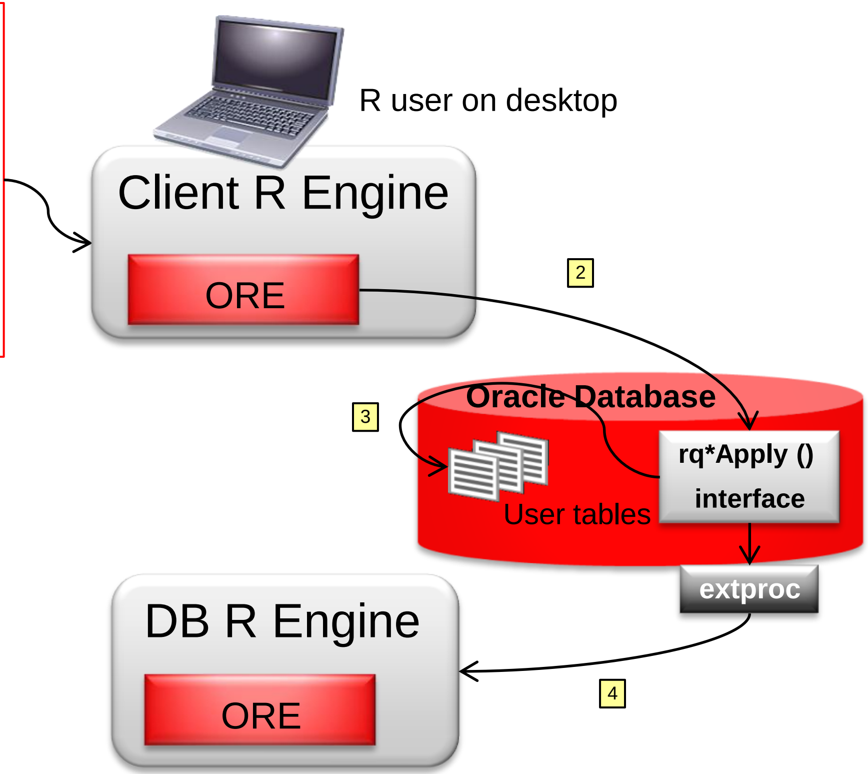
Results



```
R> mod <- ore.doEval(
+    function() {
+      ore.sync(table="ONTIME_S")
+      dat <- ore.pull(ore.get("ONTIME_S"))
+      lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
+    },
+    ore.connect = TRUE);
R> mod_local <- ore.pull(mod)
R> class(mod_local)
[1] "lm"
R> summary(mod_local)

Call:
lm(formula = ARRDELAY ~ DISTANCE + DEPDELAY, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-1462.45    -6.97    -1.36     5.07   925.08

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.254e-01  5.197e-02   4.336 1.45e-05 ***
DISTANCE    -1.218e-03  5.803e-05 -20.979  < 2e-16 ***
DEPDELAY     9.625e-01  1.151e-03 836.289  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.73 on 215144 degrees of freedom
  (4785 observations deleted due to missingness)
Multiple R-squared: 0.7647,     Adjusted R-squared: 0.7647
F-statistic: 3.497e+05 on 2 and 215144 DF,  p-value: < 2.2e-16
```

ore.tableApply – with parameter passing

```
modCoef <- ore.tableApply(
  ONTIME_S[,c("ARRDELAY","DISTANCE","DEPDELAY")],
  function(dat, family) {
    mod <- glm(ARRDELAY ~ DISTANCE + DEPDELAY,
                    data=dat, family=family)
    coef(mod)
    }, family=gaussian());
modCoef
```

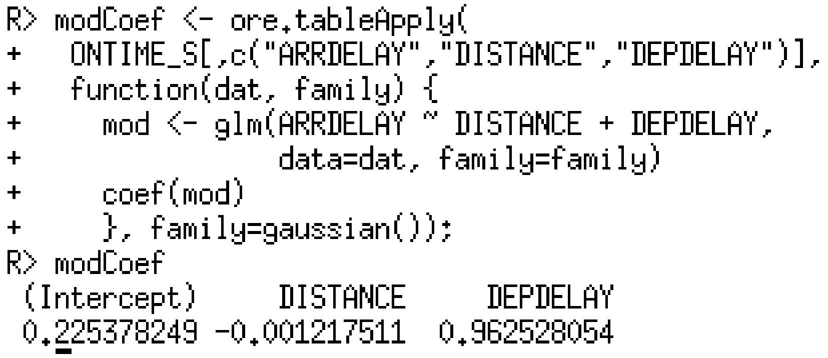


Goal: Build model on data from input cursor with parameter family = gaussian().

Data set loaded into R memory at DB R Engine and passed to function as first argument, *x*

Result coefficient(mod) returned as R object

Results



```
R> modCoef <- ore.tableApply(
+   ONTIME_S[,c("ARRDELAY","DISTANCE","DEPDELAY")],
+   function(dat, family) {
+     mod <- glm(ARRDELAY ~ DISTANCE + DEPDELAY,
+                data=dat, family=family)
+     coef(mod)
+     }, family=gaussian());
R> modCoef
 (Intercept)     DISTANCE     DEPDELAY
 0.225378249 -0.001217511  0.962528054
```

ore.tableApply – using CRAN package

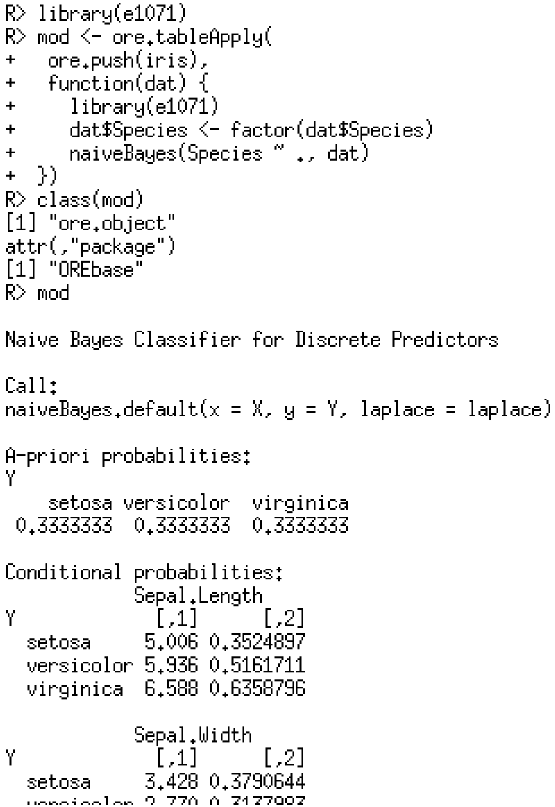
```
library(e1071)
mod <- ore.tableApply(
  ore.push(iris),
  function(dat) {
    library(e1071)
    dat$Species <- as.factor(dat$Species)
    naiveBayes(Species ~ ., dat)
 })
class(mod)
mod
```

Goal: Build model on data from input cursor

Package e1071loaded at DB R Engine

Data set pushed to database and then loaded into R memory at DB R Engine and passed to function

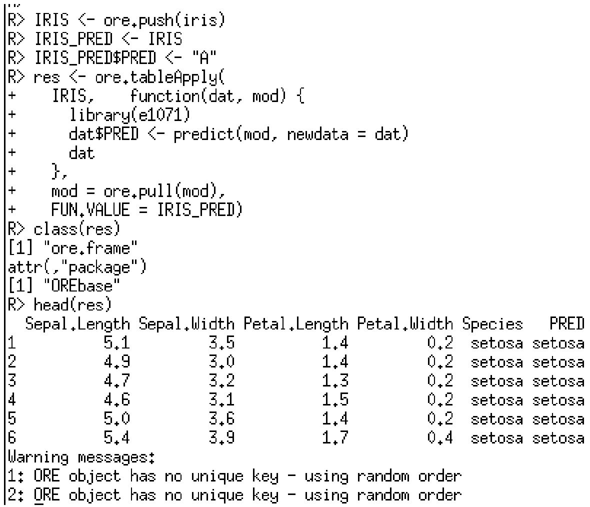
Result “mod” returned as serialized object



```
R> library(e1071)
R> mod <- ore.tableApply(
+   ore.push(iris),
+   function(dat) {
+     library(e1071)
+     dat$Species <- factor(dat$Species)
+     naiveBayes(Species ~ ., dat)
+  })
R> class(mod)
[1] "ore.object"
attr(,"package")
[1] "OREbase"
R> mod

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Conditional probabilities:
            Sepal.Length
Y             [,1]      [,2]
  setosa     5.006 0.3524897
  versicolor 5.936 0.5161711
  virginica  6.588 0.6358796

            Sepal.Width
Y             [,1]      [,2]
  setosa     3.428 0.3790644
  versicolor 2.770 0.3137983
```

ore.tableApply – batch scoring returning ore.frame

```
IRIS <- ore.push(iris)
IRIS_PRED <- IRIS
IRIS_PRED$PRED <- "A"
res <- ore.tableApply(
   IRIS,
   function(dat, mod) {
     library(e1071)
     dat$PRED <- predict(mod, newdata = dat)
     dat
   },
   mod = ore.pull(mod),
   FUN.VALUE = IRIS_PRED)
class(res)
head(res)
```



```
R> IRIS <- ore.push(iris)
R> IRIS_PRED <- IRIS
R> IRIS_PRED$PRED <- "A"
R> res <- ore.tableApply(
+    IRIS,    function(dat, mod) {
+      library(e1071)
+      dat$PRED <- predict(mod, newdata = dat)
+      dat
+    },
+    mod = ore.pull(mod),
+    FUN.VALUE = IRIS_PRED)
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> head(res)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species   PRED
1          5.1         3.5          1.4         0.2  setosa setosa
2          4.9         3.0          1.4         0.2  setosa setosa
3          4.7         3.2          1.3         0.2  setosa setosa
4          4.6         3.1          1.5         0.2  setosa setosa
5          5.0         3.6          1.4         0.2  setosa setosa
6          5.4         3.9          1.7         0.4  setosa setosa
Warning messages:
1: ORE object has no unique key - using random order
2: ORE object has no unique key - using random order
```

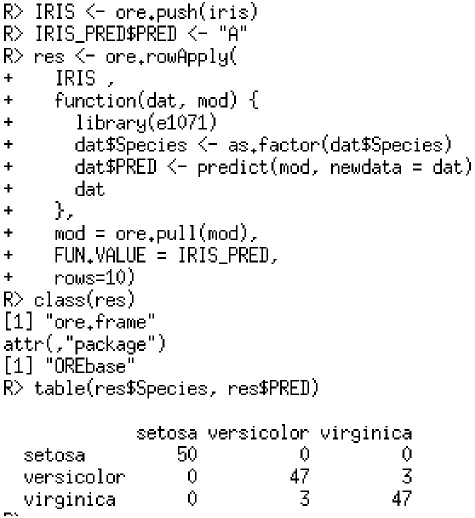
Goal: Score data using model with data from ore.frame

Return value specified using IRIS_PRED as *example* representation.

Result returned as ore.frame

ore.rowApply – data parallel scoring

```
IRIS <- ore.push(iris)
IRIS_PRED$PRED <- "A"
res <- ore.rowApply(
   IRIS ,
   function(dat, mod) {
     library(e1071)
     dat$Species <- as.factor(dat$Species)
     dat$PRED <- predict(mod, newdata = dat)
     dat
   },
   mod = ore.pull(mod),
   FUN.VALUE = IRIS_PRED,
   rows=10)
class(res)
table(res$Species, res$PRED)
```



```
R> IRIS <- ore.push(iris)
R> IRIS_PRED$PRED <- "A"
R> res <- ore.rowApply(
+    IRIS ,
+    function(dat, mod) {
+      library(e1071)
+      dat$Species <- as.factor(dat$Species)
+      dat$PRED <- predict(mod, newdata = dat)
+      dat
+    },
+    mod = ore.pull(mod),
+    FUN.VALUE = IRIS_PRED,
+    rows=10)
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> table(res$Species, res$PRED)

             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47
```

Goal: Score data in batch (rows=10) using data from input ore.frame

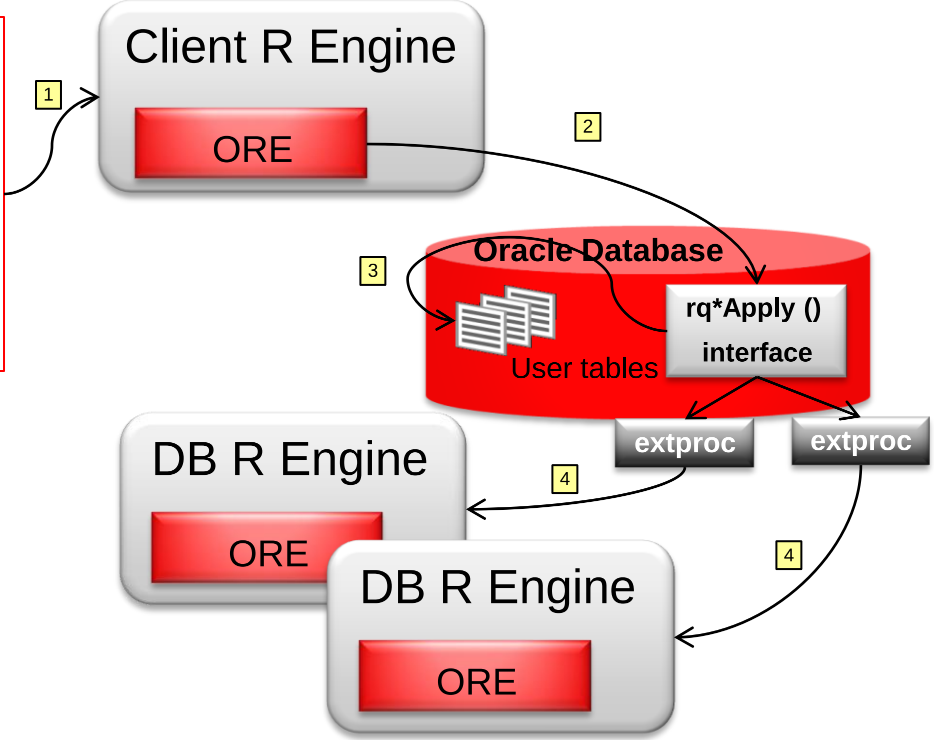
Data set loaded into R memory at database R Engine and passed to function

Return value specified using IRIS_PRED as *example* representation.

Result returned as ore.frame

ore.groupApply – partitioned data flow

```
modList <- ore.groupApply(
   X=ONTIME_S,
   INDEX=ONTIME_S$DEST,
   function(dat) {
      lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
   });
modList_local <- ore.pull(modList)
summary(modList_local$BOS) ## return model for BOS
```

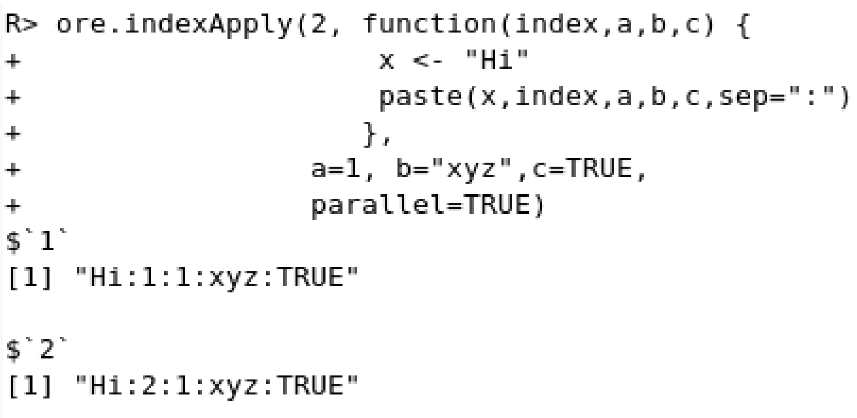


ore.indexApply – task-parallel execution

```
ore.indexApply(2,
               function(index,a,b,c) {
                 x <- "Hi"
                 paste(x,index,a,b,c,sep=":")
               },
               a=1, b="xyz",c=TRUE,
               parallel=TRUE)
```

Goal: illustrate using index as input to vary behavior of function.

Return ore.list, one element per index



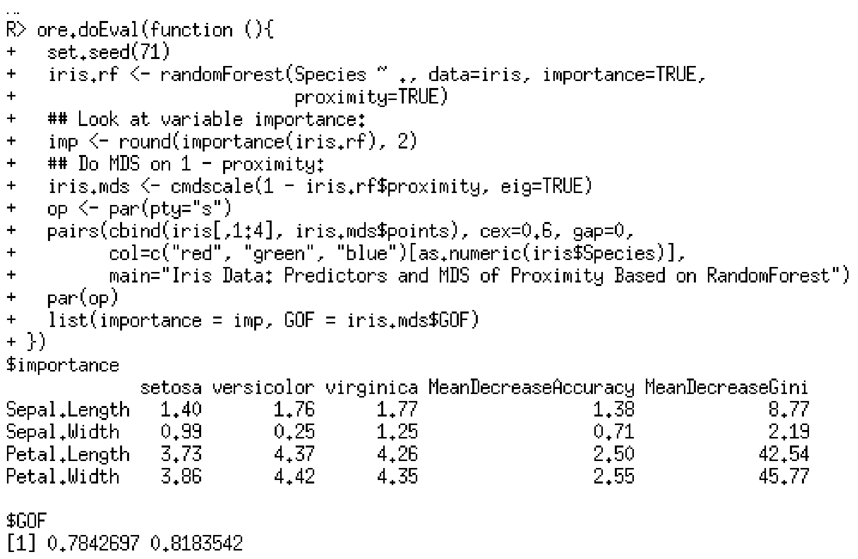
```
R> ore.indexApply(2, function(index,a,b,c) {
+                      x <- "Hi"
+                      paste(x,index,a,b,c,sep=":")
+                    },
+                  a=1, b="xyz",c=TRUE,
+                  parallel=TRUE)
$`1`
[1] "Hi:1:1:xyz:TRUE"

$`2`
[1] "Hi:2:1:xyz:TRUE"
```

Viewing database server-generated graphics in client

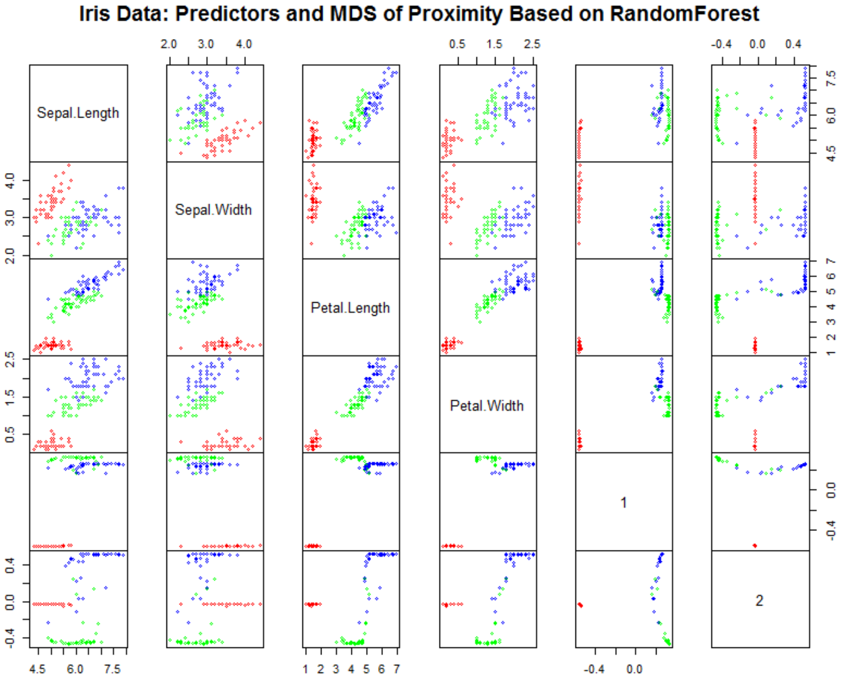
```
ore.doEval(function (){
  set.seed(71)
  iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE, proximity=TRUE)
  ## Look at variable importance:
  imp <- round(importance(iris.rf), 2)
  ## Do MDS on 1 - proximity:
  iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE)
  op <- par(pty="s")
  pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
        col=c("red", "green", "blue")[as.numeric(iris$Species)],
        main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
  par(op)
  list(importance = imp, GOF = iris.mds$GOF)
})
```

Goal: generate graph at database server, view on client and return importance from randomForest model

Results



```
...
R> ore.doEval(function (){
+   set.seed(71)
+   iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE,
+                           proximity=TRUE)
+   ## Look at variable importance:
+   imp <- round(importance(iris.rf), 2)
+   ## Do MDS on 1 - proximity:
+   iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE)
+   op <- par(pty="s")
+   pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
+         col=c("red", "green", "blue")[as.numeric(iris$Species)],
+         main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
+   par(op)
+   list(importance = imp, GOF = iris.mds$GOF)
+ })
$importance
             setosa versicolor virginica MeanDecreaseAccuracy MeanDecreaseGini
Sepal.Length   1.40       1.76      1.77                 1.38             8.77
Sepal.Width    0.99       0.25      1.25                 0.71             2.19
Petal.Length   3.73       4.37      4.26                 2.50            42.54
Petal.Width    3.86       4.42      4.35                 2.55            45.77

$GOF
[1] 0.7842697 0.8183542
```



```
ore.doEval(function (){
   …
}, ore.graphics=TRUE, ore.png.height=700, ore.png.width=500)
```

Parameterizing server-generated graphics in client

```
ore.doEval(function (rounding = 2, colorVec= c("red", "green", "blue")){
  set.seed(71)
  iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE,
                          proximity=TRUE)
  ## Look at variable importance:
  imp <- round(importance(iris.rf), rounding)
  ## Do MDS on 1 - proximity:
  iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE)
  op <- par(pty="s")
  pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
        col=colorVec[as.numeric(iris$Species)],
        main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
  par(op)
  list(importance = imp, GOF = iris.mds$GOF)
  },
  rounding = 3, colorVec = c("purple","black","pink"))
```

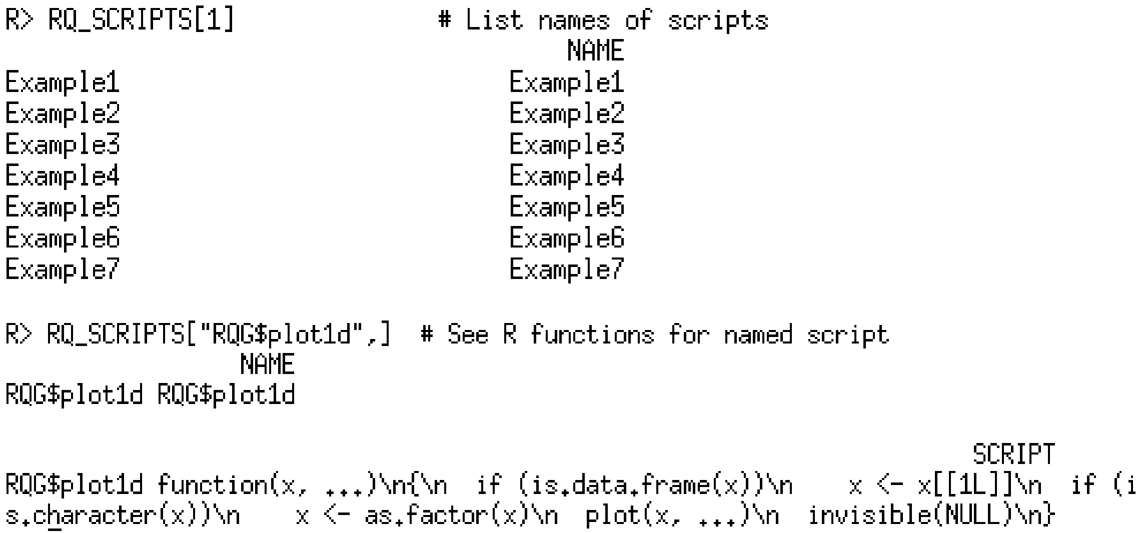
at database server, view on client
e from randomForest model

Control Arguments Summary

- Arguments starting with 'ore.' are special control arguments
  - Not passed to the function specified by 'FUN' or 'FUN.NAME' arguments
  - Controls what happens before or after the execution of the funtion (closure)
- Supported control arguments include:
  - **ore.drop** - contols the input data. If TRUE, a one column input data.frame will be converted to a vector (default: TRUE)
  - **ore.connect** - controls whether to automatically connect to ORE inside the closure. This is equivalent to doing an ore.connect call with the same credentials as the client session. (default: FALSE)
  - **ore.graphics** - controls whether to start a graphical driver and look for images (default: TRUE)
  - **ore.png.*** - if ore.graphics=TRUE, provides additional parameters for png graphics device driver. Use “ore.png.” prefix to arguments of png function. E.g., if ore.png.height is supplied, argument “height” will be passed to the png function. If not set, the standard default values for the png function are used.

Viewing R Script Repository Contents

```
ore.sync(table = "RQ_SCRIPTS", schema = "SYS")
ore.attach(schema = "SYS")
row.names(RQ_SCRIPTS) <- RQ_SCRIPTS$NAME

RQ_SCRIPTS[1]                # List names of scripts
RQ_SCRIPTS["RQG$plot1d",]   # See R functions for named script
```



```
R> RQ_SCRIPTS[1]              # List names of scripts
                                        NAME
Example1                            Example1
Example2                            Example2
Example3                            Example3
Example4                            Example4
Example5                            Example5
Example6                            Example6
Example7                            Example7

R> RQ_SCRIPTS["RQG$plot1d",]  # See R functions for named script
                 NAME
RQG$plot1d RQG$plot1d

                                                                    SCRIPT
RQG$plot1d function(x, ...)\n{\n  if (is.data.frame(x))\n    x <- x[[1L]]\n  if (i
s.character(x))\n    x <- as.factor(x)\n  plot(x, ...)\n  invisible(NULL)\n}
```

Embedded R Scripts – SQL Interface

Embedded Script Execution – SQL Interface

SQL Interface function	Purpose
rqEval()	Invoke stand-alone R script
rqTableEval()	Invoke R script with full table as input
rqRowEval()	Invoke R script on one row at a time, or multiple rows in chunks
“rqGroupEval()”	Invoke R script on data partitioned by grouping column
sys.rqScriptCreate	Create named R script
sys.rqScriptDrop	Drop named R script

rq*Eval() Table Functions

rqEval, rqTableEval, “rqGroupEval”, rqRowEval

```
rq*Eval(
  cursor(select * from <table-1>),
  cursor(select * from <table-2>),
  'select <column list> from <table-3> t',
  <grouping col-name from table-1
   or num rows>,
  '<R-script-name>')
```

- Input cursor – Depending on the function, input passed as a whole table, group, or one row at a time to the R closure (not for rqEval)
- Parameters cursor – Parameters are specified through a select statement, scalars only – single row
- Output table definition – a query specifying the format of the result
  If NULL, output is a serialized BLOB
- Group name (optional) – Name of the grouping column
- Number of rows (optional) – number of rows to provide to function at one time
- Name of R function (closure) – The name of the function to execute.

Embedded Script Execution – SQL Interface

ORE function	Input data	FUN.VALUE	Arguments	R Script *	Special
rqEval	None Generated within R function Load via ore.pull Transparency layer	NULL (returns chunked blob) table signature (returns table) XML PNG	NULL or Cursor with single row select statement with scalar values	R script name	Not applicable
rqTableEval	table cursor *				Not applicable
rqRowEval					Integer >= 1
“rqGroupEval”					Single column name *
sys.rqScriptCreate	Not applicable	Not applicable	Not applicable	R Closure (function)	Script Name*
sys.rqScriptDrop	Not applicable	Not applicable	Not applicable	Not applicable	Script Name*

*** required**

Passing parameters

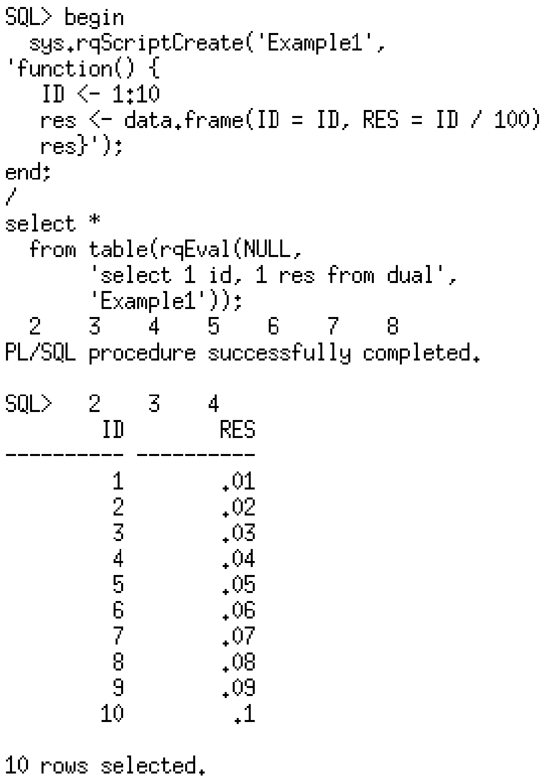
- Directly pass **scalar numeric and string** as R parameters via parameter cursor

```
select count(*)
from table(rqTableEval(
  cursor ( select x as "x", y as "y", parameter_value as "z"
           from geological_model_grid),
  cursor( select 30 as "levels", '/oracle/image.png' as "filename",
          1 "ore.connect" from dual),
  NULL,
  'Example5'));
```

- To pass non-scalar R parameter (e.g., a model), use a *datastore* object

rqEval – invoking a simple R script

```
begin
  sys.rqScriptCreate('Example1',
'function() {
   ID <- 1:10
   res <- data.frame(ID = ID, RES = ID / 100)
   res}');
end;
/
select *
  from table(rqEval(NULL,
       'select 1 id, 1 res from dual',
       'Example1'));
```



```
SQL> begin
  sys.rqScriptCreate('Example1',
'function() {
   ID <- 1:10
   res <- data.frame(ID = ID, RES = ID / 100)
   res}');
end;
/
select *
  from table(rqEval(NULL,
       'select 1 id, 1 res from dual',
       'Example1'));
  2    3    4    5    6    7    8
PL/SQL procedure successfully completed.

SQL>   2    3    4
        ID        RES
---------- ----------
         1        .01
         2        .02
         3        .03
         4        .04
         5        .05
         6        .06
         7        .07
         8        .08
         9        .09
        10         .1

10 rows selected.
```

Embedded R Execution – SQL Interface

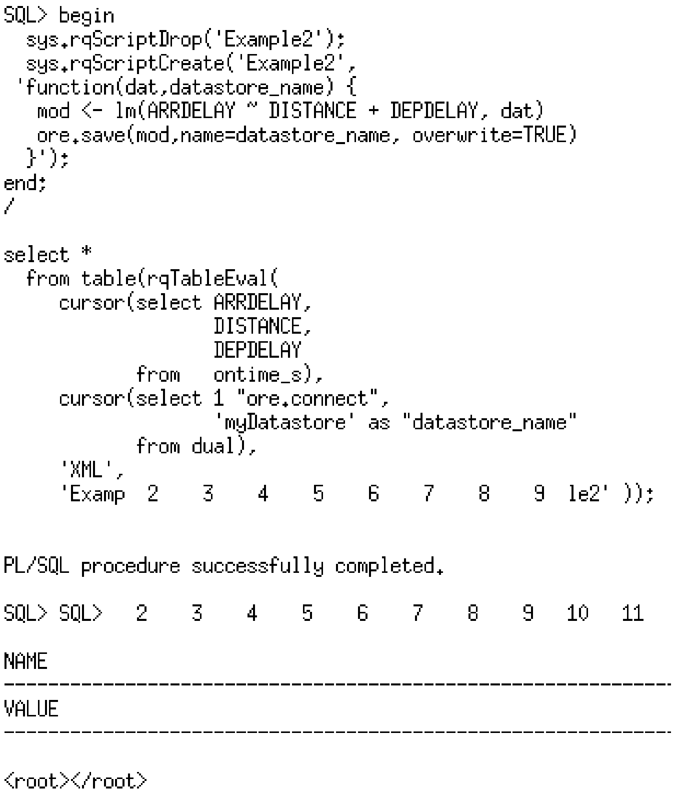
For model build and batch scoring

```
begin
  sys.rqScriptDrop('Example2');
  sys.rqScriptCreate('Example2',
 'function(dat,datastore_name) {
   mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
   ore.save(mod,name=datastore_name, overwrite=TRUE)
  }');
end;
/

select *
  from table(rqTableEval(
     cursor(select ARRDELAY,
                   DISTANCE,
                   DEPDELAY
            from   ontime_s),
     cursor(select 1 as "ore.connect",
                   'myDatastore' as "datastore_name"
            from dual),
     'XML',
     'Example2' ));
```

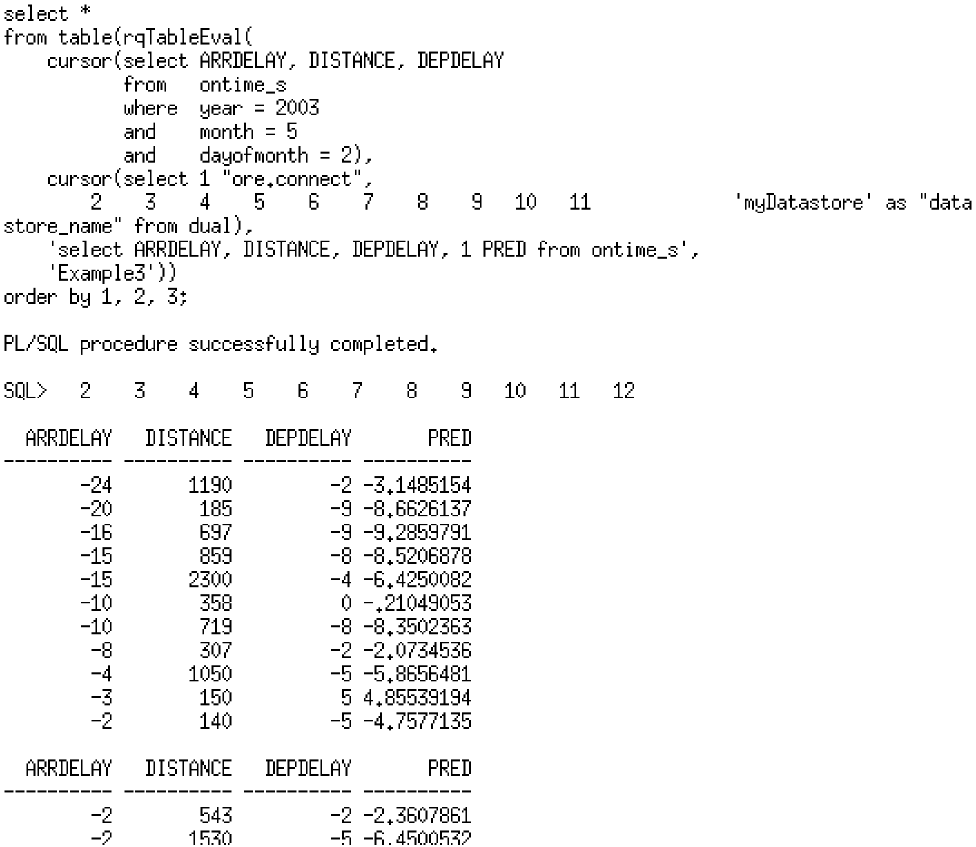
```
begin
 sys.rqScriptDrop('Example3');
 sys.rqScriptCreate('Example3',
 'function(dat, datastore_name) {
     ore.load(datastore_name)
     prd <- predict(mod, newdata=dat)
     prd[as.integer(rownames(prd))] <- prd
     res <- cbind(dat, PRED = prd)
     res}');
end;
/
select *
from table(rqTableEval(
    cursor(select ARRDELAY, DISTANCE, DEPDELAY
           from   ontime_s
           where  year = 2003
           and    month = 5
           and    dayofmonth = 2),
    cursor(select 1 as "ore.connect",
                 'myDatastore' as "datastore_name" from dual),
    'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
    'Example3'))
order by 1, 2, 3;
```

©2012 Oracle – All Rights Reserved

Results



```
SQL> begin
  sys.rqScriptDrop('Example2');
  sys.rqScriptCreate('Example2',
 'function(dat,datastore_name) {
   mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
   ore.save(mod,name=datastore_name, overwrite=TRUE)
  }');
end;
/

select *
  from table(rqTableEval(
     cursor(select ARRDELAY,
                   DISTANCE,
                   DEPDELAY
            from   ontime_s),
     cursor(select 1 "ore.connect",
                   'myDatastore' as "datastore_name"
            from dual),
     'XML',
     'Examp  2    3    4    5    6    7    8    9  le2' ));


PL/SQL procedure successfully completed.

SQL> SQL>  2    3    4    5    6    7    8    9   10   11

NAME
------------------------------------------------------------
VALUE
------------------------------------------------------------

<root></root>
```



```
select *
from table(rqTableEval(
    cursor(select ARRDELAY, DISTANCE, DEPDELAY
           from   ontime_s
           where  year = 2003
           and    month = 5
           and    dayofmonth = 2),
    cursor(select 1 "ore.connect",
        2    3    4    5    6    7    8    9   10   11             'myDatastore' as "data
store_name" from dual),
    'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
    'Example3'))
order by 1, 2, 3;

PL/SQL procedure successfully completed.

SQL>   2    3    4    5    6    7    8    9   10   11   12

  ARRDELAY   DISTANCE   DEPDELAY       PRED
---------- ---------- ---------- ----------
       -24       1190         -2 -3.1485154
       -20        185         -9 -8.6626137
       -16        697         -9 -9.2859791
       -15        859         -8 -8.5206878
       -15       2300         -4 -6.4250082
       -10        358          0 -.21049053
       -10        719         -8 -8.3502363
        -8        307         -2 -2.0734536
        -4       1050         -5 -5.8656481
        -3        150          5 4.85539194
        -2        140         -5 -4.7577135

  ARRDELAY   DISTANCE   DEPDELAY       PRED
---------- ---------- ---------- ----------
        -2        543         -2 -2.3607861
        -2       1530         -5 -6.4500532
```

rqTableEval – singleton / real-time scoring

```
select *
from   table(rqTableEval(
         cursor(select 23 ARRDELAY, 3210 DISTANCE, 45 DEPDELAY
                from   dual),
         cursor(select 'myDatastore' "datastore_name",
                        1 "ore.connect" from dual),
        'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
         'Example3'))
order by 1, 2, 3;
```

rq*Eval functions: XML and PNG Image generation

Motivation

XML Generation

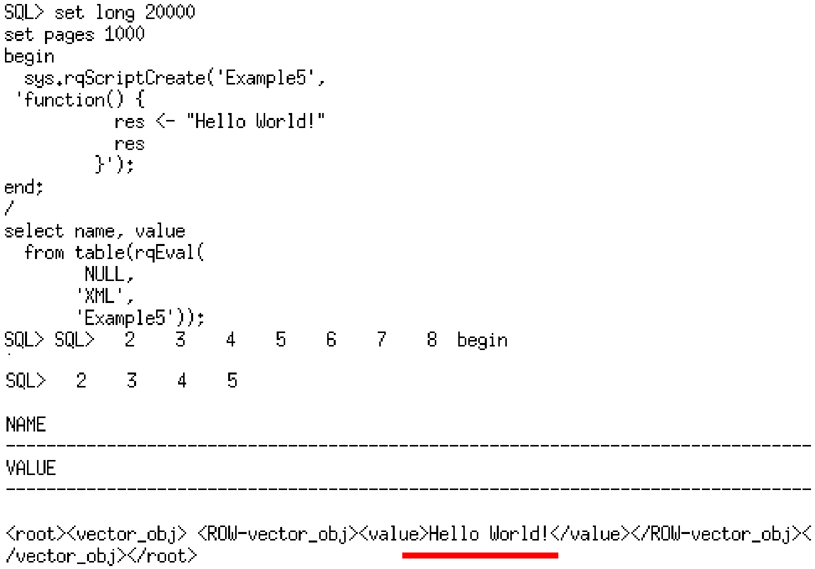
- R script output is often dynamic – not conforming to pre-defined structure
  - XML is very flexible and expressive
- R applications generate heterogeneous data
  - Statistics, new data, graphics, complex objects
  - Applications R results may often need these results
- Web-based applications typically can consume XML output
- Database-based applications need ready integration of R executed via SQL

PNG Image Generation

- Database-based applications can consume images directly from tables
- R scripts can generate multiple images
  - Enable returning image stream from R script
  - Images directly returned as a table consisting of identifier and BLOB columns
- Such results can be directly integrated with OBIEE 11.1.1.7 RPD for direct image access in dashboards

rqEval – “Hello World!” XML Example

```
set long 20000
set pages 1000
begin
  sys.rqScriptCreate('Example5',
 'function() {"Hello World!"}');
end;
/
select name, value
  from table(rqEval(
        NULL,
       'XML',
       'Example5'));
```

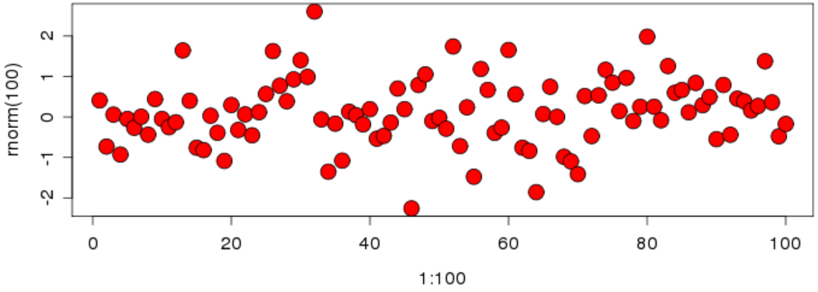


```
SQL> set long 20000
set pages 1000
begin
  sys.rqScriptCreate('Example5',
 'function() {
          res <- "Hello World!"
          res
        }');
end;
/
select name, value
  from table(rqEval(
        NULL,
       'XML',
       'Example5'));
SQL> SQL>   2    3    4    5    6    7    8  begin

SQL>   2    3    4    5

NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------

<root><vector_obj> <ROW-vector_obj><value>Hello World!</value></ROW-vector_obj><
/vector_obj></root>
```

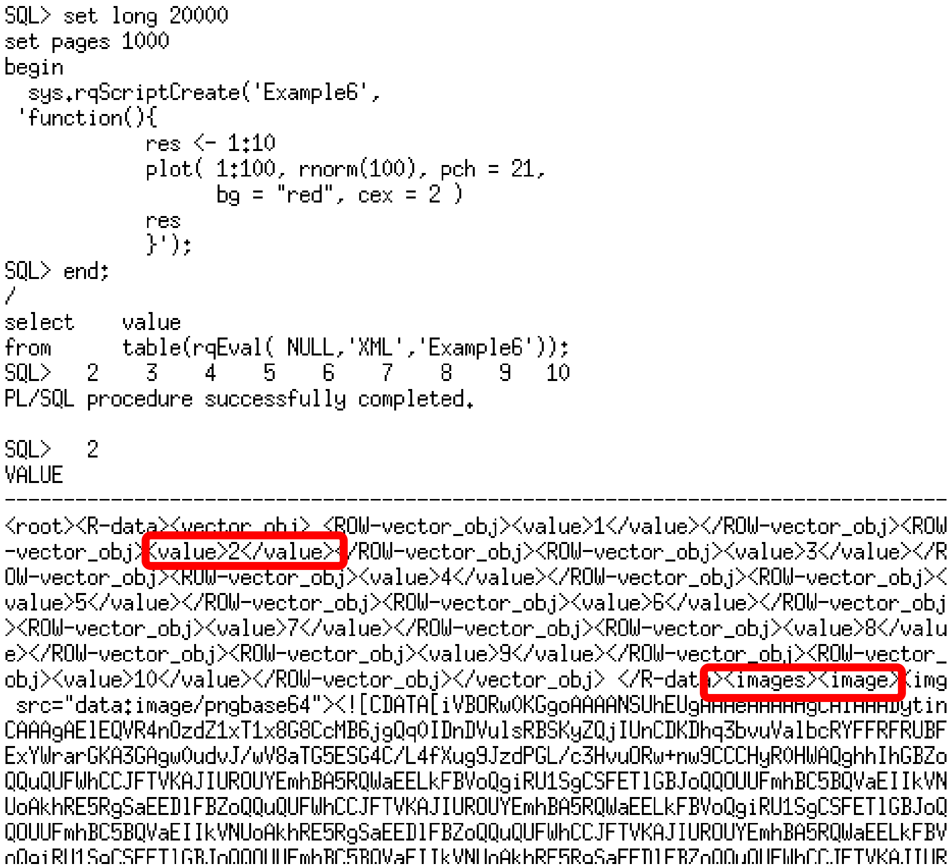
rqEval – generate XML string for graphic output

```
set long 20000
set pages 1000
begin
   sys.rqScriptCreate('Example6',
  'function(){
                 res <- 1:10
                 plot( 1:100, rnorm(100), pch = 21,
                          bg = "red", cex = 2 )
                 res
                 }');
end;
/
select    value
from      table(rqEval( NULL,'XML','Example6'));
```

- Execute the function that plots 100 random numbers
- Returns a vector with values 1 to 10
- No parameters are specified
- Return the results as XML
- View the XML VALUE returned, which can be consumed by BI Publisher



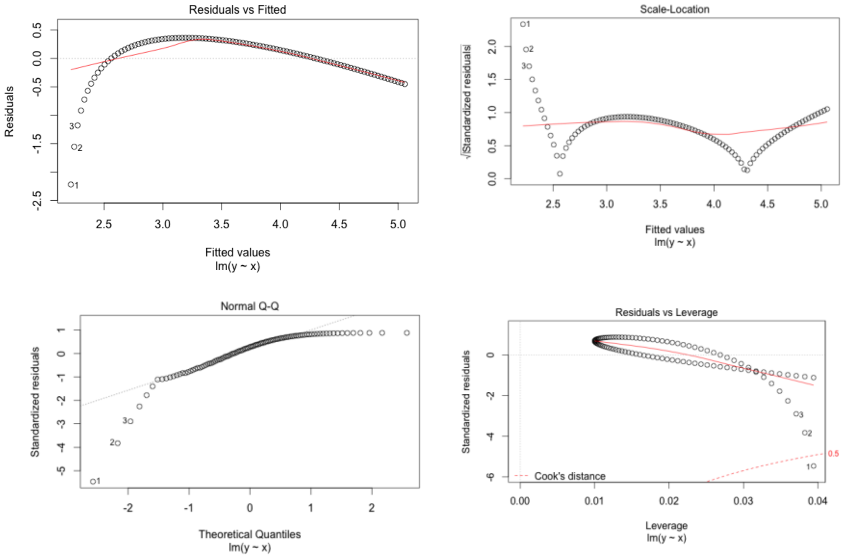
Results



```
SQL> set long 20000
set pages 1000
begin
  sys.rqScriptCreate('Example6',
 'function(){
            res <- 1:10
            plot( 1:100, rnorm(100), pch = 21,
                  bg = "red", cex = 2 )
            res
            }');
SQL> end;
/
select    value
from      table(rqEval( NULL,'XML','Example6'));
SQL>   2    3    4    5    6    7    8    9   10
PL/SQL procedure successfully completed.

SQL>   2
VALUE
--------------------------------------------------------------------------------
<root><R-data><vector_obj> <ROW-vector_obj><value>1</value></ROW-vector_obj><ROW
-vector_obj><value>2</value></ROW-vector_obj><ROW-vector_obj><value>3</value></R
OW-vector_obj><ROW-vector_obj><value>4</value></ROW-vector_obj><ROW-vector_obj><
value>5</value></ROW-vector_obj><ROW-vector_obj><value>6</value></ROW-vector_obj
><ROW-vector_obj><value>7</value></ROW-vector_obj><ROW-vector_obj><value>8</valu
e></ROW-vector_obj><ROW-vector_obj><value>9</value></ROW-vector_obj><ROW-vector_
obj><value>10</value></ROW-vector_obj></vector_obj> </R-data><images><image><img
 src="data:image/pngbase64"><![CDATA[iVBORw0KGgoAAAANSUhEUgAAAeAAAAAgCAIAAADytin
CAAAgAElEQVR4nOzdZ1xT1x8G8CcMB6jgQq0IDnDVulsRBSKyZQjIUnCDKDhq3bvuValbcRYFFRUBF
ExYWrarGKA3GAgw0udvJ/wV8aTG5ESG4C/L4fXug9JzdPGL/c3HvuORw+nw9CCCHyROHWAQghhIhGBZo
QQuQUFWhCCJFTVKAJIUROUYEmhBA5RQWaEELkFBVoQgiRU1SgCSFETlGBJoQQQOUUFmhBC5BQVaEIIkVN
UoAkhRE5RgSaEEDlFBZoQQuQUFWhCCJFTVKAJIUROUYEmhBA5RQWaEELkFBVoQgiRU1SgCSFETlGBJoQ
QOUUFmhBC5BQVaEIIkVNUoAkhRE5RgSaEEDlFBZoQQuQUFWhCCJFTVKAJIUROUYEmhBA5RQWaEELkFBV
oQgiRU1SgCSFETlGBJoQQOUUFmhBC5BQVaEIIkVNUoAkhRE5RgSaEEDlFBZoQQuQUFWhCCJFTVKAJIUR
```

rqEval – generate PNG image stream

```
begin
  sys.rqScriptDrop('Example7');
  sys.rqScriptCreate('Example7',
 'function(){
            dat <- data.frame(y=log(1:100), x = 1:100)
            plot(lm(y ~ x, dat))
            }');
end;
/
select    name, id, image
from      table(rqEval( NULL,'PNG','Example7'));
```



rq*Eval Output Specification Summary

Output Type Parameter Value	Data Returned
SQL table specification string e.g., “select 1 ID, ‘aaa’ VAL from dual”	Table – streamed structured data Image stream is discarded
NULL	Serialized R object(s) May contain both data and image objects
‘XML’	XML string May contain both data and image data Images represented as base 64 encoding of PNG
‘PNG’	Table with 1 image per row NAME varchar2(4000) ID number IMAGE blob

Embedded R Execution – Privileges

Database Roles	R Interface	SQL Interface
RQADMIN	Execute ore.*Apply functions Use FUN argument to dynamically create R scripts Execute ore.scriptCreate and ore.scriptDrop functions Access SYS.RQ_SCRIPTS table	Execute rq*Eval functions Execute sys.rqScriptCreate and sys.rqScriptDrop functions Access SYS.RQ_SCRIPTS table

```
grant RQADMIN to <USER>;
```

Working with Connections

Connecting to databases from an embedded R function

- Enable embedded R function executing in database to access database tables without requiring explicit login (when possible)
- Scenario 1: Connect to the same database in which embedded R execution originated
  - Login credentials are already available from the current active database session
  - Steps: Obtain connection object. Use connection to execute queries. Disconnect
  - Example

```
con = dbConnect(Extproc())
...
dbGetQuery(con, 'query’)
dbDisconnect(con)
```

- Scenario 2: Connect to other databases or more than 1 database
  - Login credentials not available since desired connection is to a different schema or different database instance
  - Steps: Obtain connection object via explicit login, Use connection to execute queries, Disconnect when done
  - Example

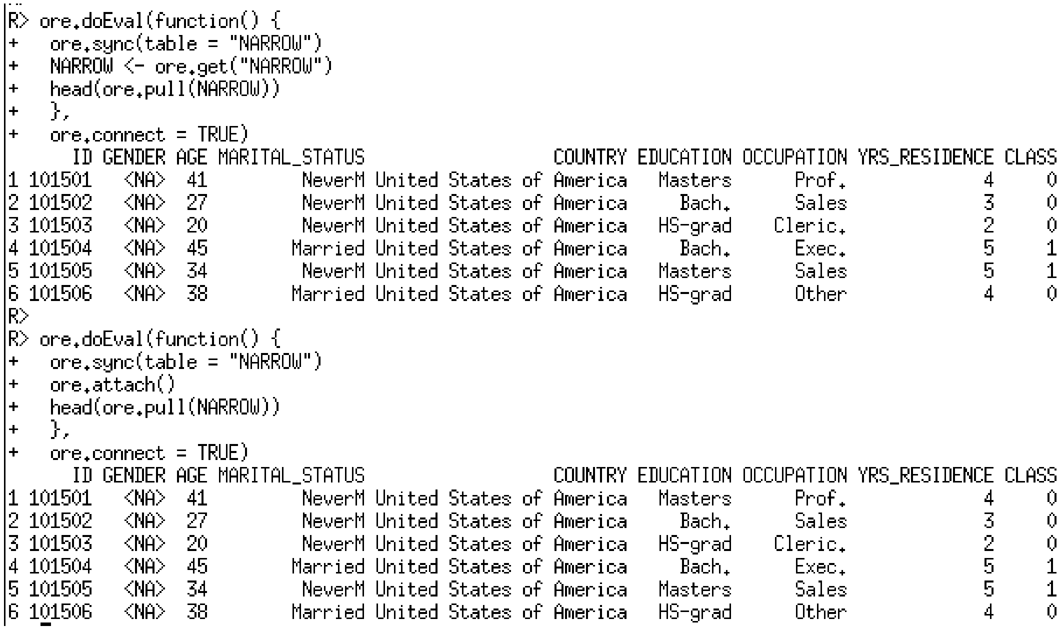
```
con = dbConnect(Oracle(), "login credentials/connect string”)
      # OR con = dbConnect(Oracle(), "WALLET”)
dbGetQuery(con, 'query');
dbDisconnect(con)
```

A few examples…

```
ore.doEval(function(){
  ore.is.connected()}  # returns FALSE
)

ore.doEval(function(){
  ore.is.connected()},  # returns TRUE
  ore.connect = TRUE
)

ore.doEval(function(){
  library(ORE)
  ore.connect("rquser", password = "rquser", conn_string = "inst1")
  ore.is.connected()   # returns TRUE
})
```

A few examples…

```
ore.doEval(function() {
  ore.sync(table = "NARROW")
  NARROW <- ore.get("NARROW")
  head(ore.pull(NARROW))
  },
  ore.connect = TRUE)

ore.doEval(function() {
  ore.sync(table = "NARROW")
  ore.attach()
  head(ore.pull(NARROW))
  },
  ore.connect = TRUE)
```



```
R> ore.doEval(function() {
+   ore.sync(table = "NARROW")
+   NARROW <- ore.get("NARROW")
+   head(ore.pull(NARROW))
+   },
+   ore.connect = TRUE)
      ID GENDER AGE MARITAL_STATUS                  COUNTRY EDUCATION OCCUPATION YRS_RESIDENCE CLASS
1 101501   <NA>  41        NeverM United States of America   Masters      Prof.             4     0
2 101502   <NA>  27        NeverM United States of America     Bach.      Sales             3     0
3 101503   <NA>  20        NeverM United States of America   HS-grad    Cleric.             2     0
4 101504   <NA>  45       Married United States of America     Bach.      Exec.             5     1
5 101505   <NA>  34        NeverM United States of America   Masters      Sales             5     1
6 101506   <NA>  38       Married United States of America   HS-grad      Other             4     0
R>
R> ore.doEval(function() {
+   ore.sync(table = "NARROW")
+   ore.attach()
+   head(ore.pull(NARROW))
+   },
+   ore.connect = TRUE)
      ID GENDER AGE MARITAL_STATUS                  COUNTRY EDUCATION OCCUPATION YRS_RESIDENCE CLASS
1 101501   <NA>  41        NeverM United States of America   Masters      Prof.             4     0
2 101502   <NA>  27        NeverM United States of America     Bach.      Sales             3     0
3 101503   <NA>  20        NeverM United States of America   HS-grad    Cleric.             2     0
4 101504   <NA>  45       Married United States of America     Bach.      Exec.             5     1
5 101505   <NA>  34        NeverM United States of America   Masters      Sales             5     1
6 101506   <NA>  38       Married United States of America   HS-grad      Other             4     0
```

©2012 Oracle – All Rights Reserved

Embedded Graphic Function Examples

Why use embedded R graphic functions?

- Same reasons for embedded R in general
  - More powerful database server
  - More efficient transfer of data between database and R engine
  - Execute scripts from SQL

ORE-defined graphic function examples

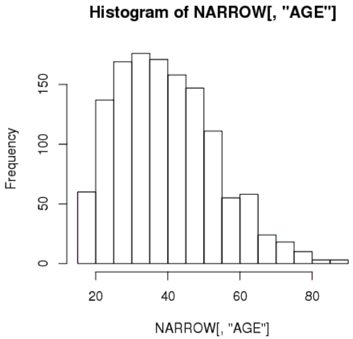
- ORE-defined scripts with a reserved name prefix: ‘RQG$'
- Prefix followed by a function name from 'graphics’ package that the script wraps
- Depending on function, takes either the first, the first and second, or all columns of the input 'data.frame'
- For use with
  - ore.tableApply
  - ore.groupApply
  - ore.rowApply
- Each function allows '...' parameters to enable passing graphics function parameters to the wrapped function

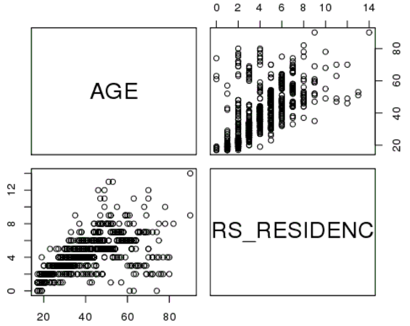
ORE-defined graphics function examples

ORE Embedded R Function	Wraps R Function	Performs function on … of input ore.frame object
RQG$plot1d	plot	first column
RQG$plot2d	plot	first two columns
RQG$hist	hist	first column
RQG$boxplot	boxplot	first column
RQG$smoothScatter	smoothScatter	first two columns
RQG$cdplot	cdplot	first two columns
RQG$pairs	pairs	all columns
RQG$matplot	matplot	all columns

rqEval – invoking a simple R script

```
hist(NARROW[,"AGE"])
pairs(NARROW[,c("AGE”,"YRS_RESIDENCE"))
```



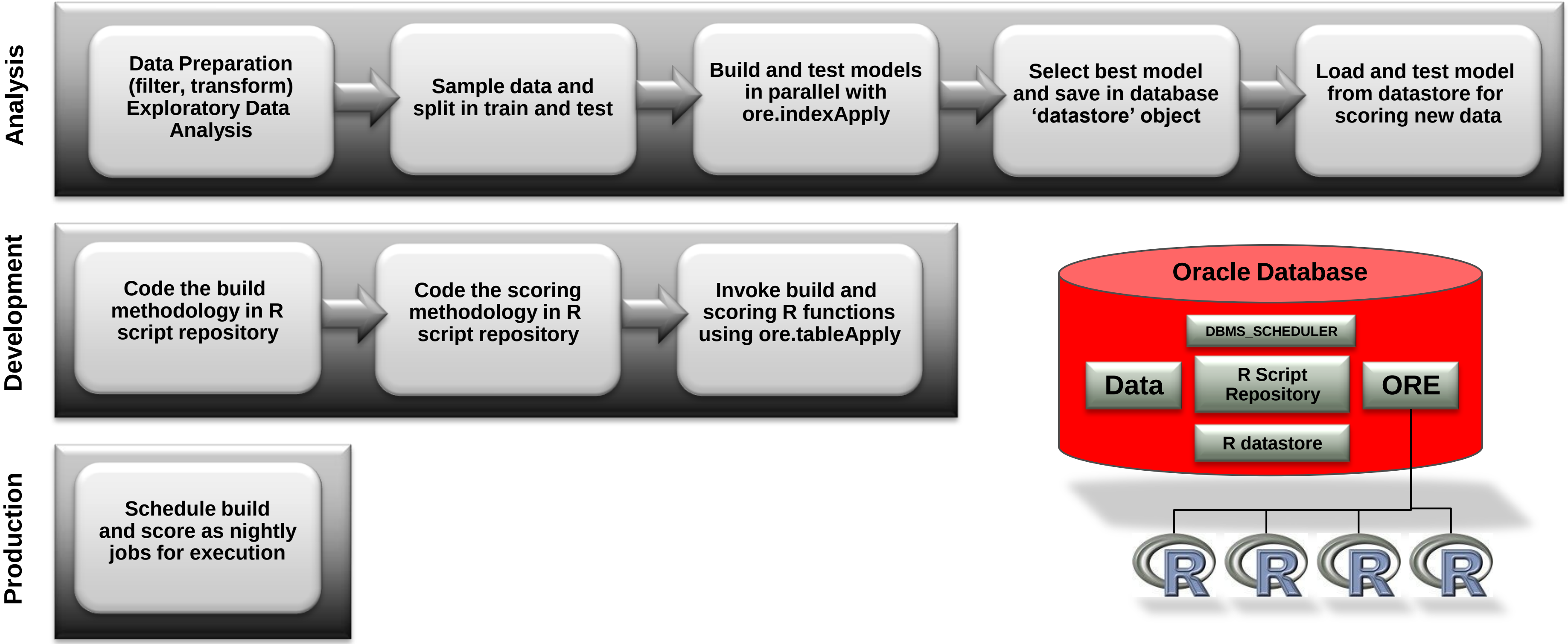


```
select *
  from table(rqTableEval(cursor(select AGE from NARROW), NULL,
                              'PNG',
                              'RQG$hist'));

select *
  from table(rqTableEval(cursor(select AGE, YRS_RESIDENCE from NARROW), NULL,
                              'PNG',
                              'RQG$pairs'));
```

Example of ORE Workflow for Model Building and Scoring

ORE as framework for Model Building and Scoring

Workflow example



Analysis

Data Preparation (filter, transform) Exploratory Data Analysis

Sample data and split in train and test

Build and test models in parallel with ore.indexApply

Select best model and save in database 'datastore' object

Load and test model from datastore for scoring new data

Development

Code the build methodology in R script repository

Code the scoring methodology in R script repository

Invoke build and scoring R functions using ore.tableApply

Production

Schedule build and score as nightly jobs for execution

Oracle Database

DBMS_SCHEDULER

Data

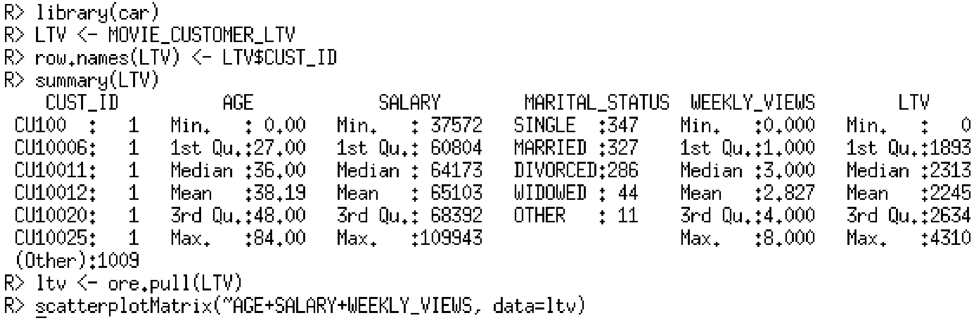
R Script Repository

ORE

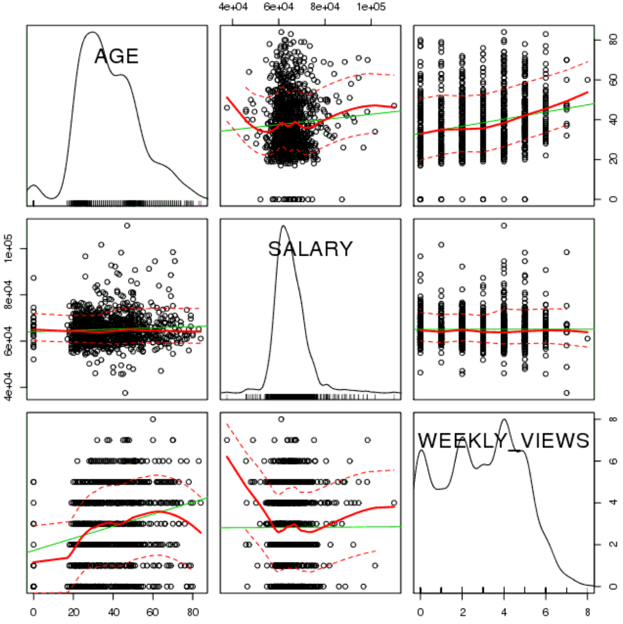
R datastore

Data exploration

```
library(car)
LTV <- MOVIE_CUSTOMER_LTV
row.names(LTV) <- LTV$CUST_ID
summary(LTV)
ltv <- ore.pull(LTV)
scatterplotMatrix(~AGE+SALARY+WEEKLY_VIEWS,
  data=ltv)
```

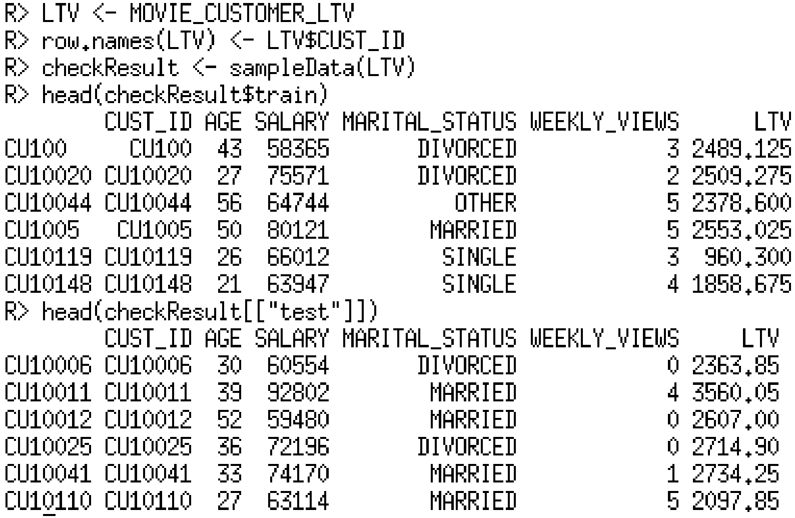


```
R> library(car)
R> LTV <- MOVIE_CUSTOMER_LTV
R> row.names(LTV) <- LTV$CUST_ID
R> summary(LTV)
    CUST_ID          AGE             SALARY          MARITAL_STATUS  WEEKLY_VIEWS        LTV
 CU100   :   1   Min.   : 0.00   Min.   : 37572   SINGLE  :347    Min.   :0.000   Min.   :   0
 CU10006:   1   1st Qu.:27.00   1st Qu.: 60804   MARRIED :327    1st Qu.:1.000   1st Qu.:1893
 CU10011:   1   Median :36.00   Median : 64173   DIVORCED:286    Median :3.000   Median :2313
 CU10012:   1   Mean   :38.19   Mean   : 65103   WIDOWED : 44    Mean   :2.827   Mean   :2245
 CU10020:   1   3rd Qu.:48.00   3rd Qu.: 68392   OTHER   : 11    3rd Qu.:4.000   3rd Qu.:2634
 CU10025:   1   Max.   :84.00   Max.   :109943                   Max.   :8.000   Max.   :4310
 (Other):1009
R> ltv <- ore.pull(LTV)
R> scatterplotMatrix(~AGE+SALARY+WEEKLY_VIEWS, data=ltv)
```



Sample data into train and test sets

```
sampleData <- function(data) {
  nrows <- nrow(data)
  train.size <- as.integer(nrows * 0.6)
  ind <- sample(1:nrows,train.size)
  group <- as.integer(1:nrows %in% ind)
  trainData <- data[group==TRUE,]
  testData <- data[group==FALSE,]
  list(train=trainData, test=testData)
}

LTV <- MOVIE_CUSTOMER_LTV
row.names(LTV) <- LTV$CUST_ID
checkResult <- sampleData(LTV)
head(checkResult$train)
head(checkResult[["test"]])
```



```
R> LTV <- MOVIE_CUSTOMER_LTV
R> row.names(LTV) <- LTV$CUST_ID
R> checkResult <- sampleData(LTV)
R> head(checkResult$train)
        CUST_ID AGE SALARY MARITAL_STATUS WEEKLY_VIEWS      LTV
CU100     CU100  43  58365       DIVORCED            3 2489.125
CU10020 CU10020  27  75571       DIVORCED            2 2509.275
CU10044 CU10044  56  64744          OTHER            5 2378.600
CU1005   CU1005  50  80121        MARRIED            5 2553.025
CU10119 CU10119  26  66012         SINGLE            3  960.300
CU10148 CU10148  21  63947         SINGLE            4 1858.675
R> head(checkResult[["test"]])
        CUST_ID AGE SALARY MARITAL_STATUS WEEKLY_VIEWS     LTV
CU10006 CU10006  30  60554       DIVORCED            0 2363.85
CU10011 CU10011  39  92802        MARRIED            4 3560.05
CU10012 CU10012  52  59480        MARRIED            0 2607.00
CU10025 CU10025  36  72196       DIVORCED            0 2714.90
CU10041 CU10041  33  74170        MARRIED            1 2734.25
CU10110 CU10110  27  63114        MARRIED            5 2097.85
```

Build and test models in parallel with ore.indexApply

```
produceModels <- function(models.list, trainData, model.datastore, overwrite=FALSE, parallel = FALSE) {
  # local function that builds model with trainData
  local.build.model <- function (idx, test.models, dat, model.datastore) {
    model.name <- names(test.models)[idx]
    assign(model.name, do.call(test.models[[idx]], list(dat)) )
    ore.save(list = model.name, name = model.datastore, append=TRUE)
    model.name
  }
  # check overwrite
  if (overwrite && nrow(ore.datastore(name=model.datastore)) > 0L)
    ore.delete(name=model.datastore)

  # build models
  trainData <- ore.pull(trainData)
  models.success <- ore.pull(ore.indexApply(length(models.list), local.build.model,
                                            test.models=models.list, dat=trainData,
                                            model.datastore=model.datastore, parallel=parallel,
                                            ore.connect=TRUE))
  as.character(models.success)
}
```

Select best model and save in database ‘datastore’ object

Part 1

```
selectBestModel <- function(testData, evaluate.func,
                            model.datastore, modelnames.list=character(0),
                            production.datastore=character(0), parallel=FALSE) {
  # get names of models to select from
  modelNames <- ore.datastoreSummary(name = model.datastore)$object.name
  modelNames <- intersect(modelNames, modelnames.list)

  # local function that scores model with test data
  local.model.score <- function(idx, model.names, datastore.name, dat, evaluate) {
    modName <- model.names[idx]
    ore.load(list=modName, name=datastore.name)
    mod <- get(modName)
    predicted <- predict(mod, dat)
    do.call(evaluate, list(modName, dat, predicted))
  }
```

Select best model and save in database ‘datastore’ object

Part 2

```
  # score these models
  testData <- ore.pull(testData)
  scores <- ore.pull(ore.indexApply(length(modelNames), local.model.score,
                                    model.names=modelNames,
                                    datastore.name=model.datastore, dat=testData,
                                    evaluate=evaluate.func, parallel=parallel,
                                    ore.connect=TRUE))
  # get best model based upon scores
  bestmodel.idx <- order(as.numeric(scores))[1]
  bestmodel.score <- scores[[bestmodel.idx]]
  bestmodel.name <- modelNames[bestmodel.idx]
  ore.load(list=bestmodel.name, name=model.datastore)
  if (length(production.datastore) > 0L)
    ore.save(list=bestmodel.name, name=production.datastore, append=TRUE)
  names(bestmodel.score) <- bestmodel.name
  bestmodel.score
}
```

Generate the Best Model

```
generateBestModel <- function(data, datastore.name, models.list, evaluate.func, parallel=FALSE) {
  data <- sampleData(data)
  trainData <- data$train
  testData  <- data$test
  produceModels(models.list, trainData, model.datastore="ds.tempModelset",
                overwrite=TRUE, parallel=parallel)
  bestModelName <- names(selectBestModel(testData, evaluate.func,
                         model.datastore="ds.tempModelset",
                         production.datastore=datastore.name, parallel=parallel))
  bestModelName
}
```

Test production script

Part 1

```
LTV <- MOVIE_CUSTOMER_LTV
row.names(LTV) <- LTV$CUST_ID

f1 <- function(trainData) glm(LTV ~ AGE + SALARY, data = trainData)
f2 <- function(trainData) glm(LTV ~ AGE + WEEKLY_VIEWS, data = trainData)
f3 <- function(trainData) lm(LTV ~ AGE + SALARY + WEEKLY_VIEWS, data = trainData)
models <- list(mod.glm.AS=f1, mod.glm.AW=f2, mod.lm.ASW=f3)

evaluate <- function(modelName, testData, predictedValue) {
  sqrt(sum((predictedValue - testData$LTV)^2)/length(testData$LTV))
}
```

Test production script

Part 2

```
bestModel <- generateBestModel(data=LTV, datastore.name="ds.production",
                  models.list=models, evaluate.func=evaluate, parallel=TRUE)

# production score
ore.load(list=bestModel, name="ds.production")

data <- LTV
data$PRED <- ore.predict(get(bestModel), data)
ore.create(data[,c("CUST_ID","PRED")],table='BATCH_SCORES')
```

ORE 1.3 New Features Summary

For Embedded R Execution

- Auto-connect
  - No need to load ORE library or login to database server to use transparency layer
    - Automatically uses same credentials as user already connected to Oracle Database
    - Use ore.sync() followed by ore.attach() or ore.get()
  - Use ROracle connection without providing credentials
- Connect to 1 or more databases via ROracle
  - Connect to the same database in which embedded R execution originated
  - Connect to other databases or more than 1 database
- “Control” arguments to ore.*Apply and rq*Eval
- ORE-defined graphics functions in R script repository
  - RQG$ plot1d, plot2d, hist, boxplot, smoothscatter, cdplot, pairs, matplot
  - Generate graphs at database server
- PNG image streams through SQL interface
  - Generates three column output table: NAME, ID, IMAGE
  - IMAGE is of type BLOB that can be included in OBIEE 11.1.1.7 RPD for display in dashboards
  - Supports multiple images returned as separate rows

Summary

- Embed R scripts in applications and operational systems
  - Control and secure R code that runs in Oracle Database
- ORE provides data- and task-flow parallelism for R
  - Interface function enable parallelism using multiple database R engines
  - Supports parallel simulations capability
- Rq*Eval enables
  - Rich XML and PNG image stream output for integration with BI Publisher and OBIEE, or any tool or system that can consume such data
  - SQL access to R

Resources

- **Blog:** https://blogs.oracle.com/R/
- **Forum:** https://forums.oracle.com/forums/forum.jspa?forumID=1397
- **Oracle R Distribution:** http://www.oracle.com/technetwork/indexes/downloads/r-distribution-1532464.html
- **ROracle:** http://cran.r-project.org/web/packages/ROracle
- **Oracle R Enterprise:** http://www.oracle.com/technetwork/database/options/advanced-analytics/r-enterprise
- **Oracle R Connector for Hadoop:** http://www.oracle.com/us/products/database/big-data-connectors/overview

ORACLE®