

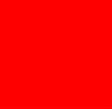


**ORACLE®**

## **Oracle R Enterprise Hands-on Lab**

Mark Hornick, Oracle Advanced Analytics

Tim Vlamis, Vlamis Software Solutions



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.

# Agenda

- Transparency Layer
- Embedded R Execution
  - R API
  - SQL API
- Predictive Analytics
- Advanced Examples
  - Bag of Little Bootstraps
  - Workflow for Model Building and Scoring

# ORE Transparency Layer

# Establish a connection using ore.connect

```
if (!ore.is.connected())  
  ore.connect("rquser", "orcl",  
             "localhost", "rquser", all=TRUE)  
ore.ls()
```

- Function **ore.connect** parameters
  - User id, SID, host, password
  - Port defaults to 1521
  - “all” set to TRUE loads all tables from the schema into ORE metadata and makes them available at the R command line as ore.frame objects
- Function **ore.ls** lists all the available tables by name

# Manipulating Data

- Column selection

```
df <- ONTIME_S[,c("YEAR", "DEST", "ARRDELAY")]
class(df)
```

```
head(df)
```

```
head(ONTIME_S[,c(1, 4, 23)])
```

```
head(ONTIME_S[, -(1:22)])
```

- Row selection

```
df1 <- df[df$DEST=="SFO",]
class(df1)
```

```
df2 <- df[df$DEST=="SFO",c(1, 3)]
```

```
df3 <- df[df$DEST=="SFO" | df$DEST=="BOS",1:3]
```

```
head(df1)
```

```
head(df2)
```

```
head(df3)
```

```
R> df <- ONTIME_S[,c("YEAR", "DEST", "ARRDELAY")]
```

```
R> class(df)
```

```
[1] "ore.frame"
```

```
attr(,"package")
```

```
[1] "OREbase"
```

```
R>
```

```
R> head(df)
```

```
YEAR DEST ARRDELAY
```

```
1 1987 MSP 4
```

```
2 1987 SJC 6
```

```
3 1987 OAK 7
```

```
4 1987 PHX 9
```

```
5 1987 CLT 0
```

```
6 1987 CVG 4
```

```
R> head(ONTIME_S[,c(1,4,23)])
```

```
YEAR DAYOFMONTH TAXIOUT
```

```
1 1987 1 NA
```

```
2 1987 1 NA
```

```
3 1987 1 NA
```

```
4 1987 1 NA
```

```
5 1987 1 NA
```

```
6 1987 1 NA
```

```
R> head(ONTIME_S[, -(1:22)])
```

```
TAXIOUT CANCELLED CANCELLATIONCODE
```

```
1 NA 0 <NA>
```

```
2 NA 0 <NA>
```

```
3 NA 0 <NA>
```

```
4 NA 0 <NA>
```

```
5 NA 0 <NA>
```

```
6 NA 0 <NA>
```

```
R> df1 <- df[df$DEST=="SFO",]
```

```
R> class(df1)
```

```
[1] "ore.frame"
```

```
attr(,"package")
```

```
[1] "OREbase"
```

```
R>
```

```
R> df2 <- df[df$DEST=="SFO",c(1,3)]
```

```
R> df3 <- df[df$DEST=="SFO" | df$DEST=="BOS",1:3]
```

```
R> head(df1)
```

```
YEAR DEST ARRDELAY
```

```
1 1987 SFO 24
```

```
2 1987 SFO 68
```

```
3 1987 SFO -3
```

```
4 1987 SFO 5
```

```
5 1987 SFO 37
```

```
6 1987 SFO 11
```

```
R> head(df2)
```

```
YEAR ARRDELAY
```

```
1 1987 24
```

```
2 1987 68
```

```
3 1987 -3
```

```
4 1987 5
```

```
5 1987 37
```

```
6 1987 11
```

```
R> head(df3)
```

```
YEAR DEST ARRDELAY
```

```
1 1987 SFO 24
```

```
2 1987 SFO 68
```

```
3 1987 SFO -3
```

```
4 1987 SFO 5
```

```
5 1987 SFO 37
```

```
6 1987 BOS NA
```

# Manipulating Data – SQL equivalent

## R

- Column selection

```
df <- ONTIME_S[,c("YEAR", "DEST", "ARRDELAY")]
head(df)
head(ONTIME_S[,c(1,4,23)])
head(ONTIME_S[,-(1:22)])
```

- Row selection

```
df1 <- df[df$DEST=="SFO",]
df2 <- df[df$DEST=="SFO",c(1,3)]
df3 <- df[df$DEST=="SFO" | df$DEST=="BOS",1:3]
```

### Benefits of ORE:

Deferred execution

Incremental query refinement

## SQL

- Column selection

```
create view df as
select YEAR, DEST, ARRDELAY
from ONTIME_S;
```

-- cannot do next two, e.g., by number & exclusion

- Row selection

```
create view df1 as
select * from df where DEST='SFO';
```

```
create view df2 as
select YEAR, ARRDELAY from df where DEST='SFO'
```

```
create view df3 as
select YEAR, DEST, ARRDELAY from df
where DEST='SFO' or DEST='BOS'
```

# merge

- Joining two tables (data frames)

```
df1 <- data.frame(x1=1:5, y1=letters[1:5])
df2 <- data.frame(x2=5:1, y2=letters[11:15])
merge (df1, df2, by.x="x1", by.y="x2")
```

```
ore.drop (table="TEST_DF1")
ore.drop (table="TEST_DF2")
```

```
ore.create (df1, table="TEST_DF1")
ore.create (df2, table="TEST_DF2")
merge (TEST_DF1, TEST_DF2,
      by.x="x1", by.y="x2")
```

```
R> df1 <- data.frame(x1=1:5, y1=letters[1:5])
R> df2 <- data.frame(x2=5:1, y2=letters[11:15])
R> merge (df1, df2, by.x="x1", by.y="x2")
  x1 y1 y2
1  1 a  o
2  2 b  n
3  3 c  m
4  4 d  l
5  5 e  k
```

```
R> ore.create(df1, table="TEST_DF1")
R> ore.create(df2, table="TEST_DF2")
R> merge (TEST_DF1, TEST_DF2,
+       by.x="X1", by.y="X2")
  X1 Y1 Y2
0  1 a  o
1  2 b  n
2  3 c  m
3  4 d  l
4  5 e  k
```

# Formatting data – Base SAS “format” equivalent

```
diverted_fmt <- function (x) {
  ifelse(x=='0', 'Not Diverted',
  ifelse(x=='1', 'Diverted', ''))
}

cancellationCode_fmt <- function(x) {
  ifelse(x=='A', 'A CODE',
  ifelse(x=='B', 'B CODE',
  ifelse(x=='C', 'C CODE',
  ifelse(x=='D', 'D CODE', 'NOT CANCELLED'))))
}

delayCategory_fmt <- function(x) {
  ifelse(x>200, 'LARGE',
  ifelse(x>=30, 'MEDIUM', 'SMALL'))
}

zscore <- function(x) {
  (x-mean(x, na.rm=TRUE))/sd(x, na.rm=TRUE)
}
```

```
ORE> x <- ONTIME_S
ORE> attach(x)
ORE>
ORE> head(DIVERTED)
[1] 0 0 0 0 0 0
Levels: 0 1
ORE> x$DIVERTED      <- diverted_fmt(DIVERTED)
ORE> head(x$DIVERTED)
[1] "Not Diverted" "Not Diverted" "Not Diverted" "Not Diverted"
[6] "Not Diverted"

x <- ONTIME_S
attach(x)

head(DIVERTED)
x$DIVERTED      <- diverted_fmt(DIVERTED)
head(x$DIVERTED)

x$CANCELLATIONCODE <-
  cancellationCode_fmt(CANCELLATIONCODE)
x$ARRDELAY        <- delayCategory_fmt(ARRDELAY)
x$DEPDELAY        <- delayCategory_fmt(DEPDELAY)
x$DISTANCE_ZSCORE <- zscore(DISTANCE)
detach(x)

head(x)
```

# Formatting data – Base SAS “format” equivalent

## *Using transform ( )*

```
ONTIME_S <- transform(ONTIME_S,  
  DIVERTED = ifelse(DIVERTED == 0, 'Not Diverted',  
    ifelse(DIVERTED == 1, 'Diverted', '')),  
  
  CANCELLATIONCODE =  
    ifelse(CANCELLATIONCODE == 'A', 'A CODE',  
    ifelse(CANCELLATIONCODE == 'B', 'B CODE',  
    ifelse(CANCELLATIONCODE == 'C', 'C CODE',  
    ifelse(CANCELLATIONCODE == 'D', 'D CODE', 'NOT CANCELLED'))),  
  
  ARRDELAY = ifelse(ARRDELAY > 200, 'LARGE',  
    ifelse(ARRDELAY >= 30, 'MEDIUM', 'SMALL')),  
  
  DEPDELAY = ifelse(DEPDELAY > 200, 'LARGE',  
    ifelse(DEPDELAY >= 30, 'MEDIUM', 'SMALL')),  
  
  DISTANCE_ZSCORE = (DISTANCE - mean(DISTANCE, na.rm=TRUE)) / sd(DISTANCE, na.rm=TRUE))
```

# Documentation and Demos

```
OREShowDoc ()
```

```
demo (package = "ORE")
```

```
demo ("aggregate", package = "ORE")
```

# Demos in package 'ORE'

<code>aggregate</code>	Aggregation	<code>nulls</code>	Handling of NULL in SQL vs. NA in R
<code>analysis</code>	Basic analysis & data processing	<code>odm_ai</code>	Oracle Data Mining: attribute importance
<code>basic</code>	Basic connectivity to database	<code>odm_dt</code>	ODM: decision trees
<code>binning</code>	Binning logic	<code>odm_glm</code>	ODM: generalized linear models
<code>columnfns</code>	Column functions	<code>odm_kmeans</code>	ODM: enhanced k-means clustering
<code>cor</code>	Correlation matrix	<code>odm_nb</code>	ODM: naive Bayes classification
<code>crosstab</code>	Frequency cross tabulations	<code>odm_svm</code>	ODM: support vector machines
<code>datastore</code>	DataStore operations	<code>push_pull</code>	RDBMS <-> R data transfer
<code>datetime</code>	Date/Time operations	<code>rank</code>	Attributed-based ranking of observations
<code>derived</code>	Handling of derived columns	<code>reg</code>	Ordinary least squares linear regression
<code>distributions</code>	Distr., density, and quantile functions	<code>row_apply</code>	Embedded R processing by row chunks
<code>do_eval</code>	Embedded R processing	<code>sampling</code>	Random row sampling and partitioning
<code>freqanalysis</code>	Frequency cross tabulations	<code>sql_like</code>	Mapping of R to SQL commands
<code>glm</code>	Generalized Linear Models	<code>stepwise</code>	Stepwise OLS linear regression
<code>graphics</code>	Demonstrates visual analysis	<code>summary</code>	Summary functionality
<code>group_apply</code>	Embedded R processing by group	<code>table_apply</code>	Embedded R processing of entire table
<code>hypothesis</code>	Hypothesis testing functions		
<code>matrix</code>	Matrix related operations		

# Example Dataset: "ONTIME" Airline Data

On-time arrival data for non-stop domestic flights by major air carriers.

Provides departure and arrival delays, origin and destination airports, flight numbers, scheduled and actual departure and arrival times, cancelled or diverted flights, taxi-out and taxi-in times, air time, and non-stop distance.

Most talked about data set when discussing general scalability in the R community

- Full Data

- 123M records
- 22 years
- 29 airlines

- Sample Data

- ~220K records
- ~10K / year
- ONTIME\_S

```
R> names(ONTIME_S)
 [1] "YEAR"           "MONTH"           "MONTH2"           "DAYOFMONTH"
 [5] "DAYOFMONTH2"   "DAYOFWEEK"       "DEPTIME"          "CRSDEPTIME"
 [9] "ARRTIME"       "CRSARRTIME"      "UNIQUECARRIER"  "FLIGHTNUM"
[13] "TAILNUM"       "ACTUALELAPSEDTIME" "CRSELAPSEDTIME"  "AIRTIME"
[17] "ARRDELAY"      "DEPDELAY"        "ORIGIN"           "DEST"
[21] "DISTANCE"      "TAXIIN"          "TAXIOUT"          "CANCELLED"
[25] "CANCELLATIONCODE" "DIVERTED"

R>
R> dim(ONTIME_S)
 [1] 219932 26
R>
R> str(ONTIME_S)
'data.frame':   219932 obs. of  26 variables:
 formal class 'ore.frame' [package "OREbase"] with 12 slots
 ..@ .Data      : list()
 ..@ dataDry   : Named chr "( select  \\"YEAR\\" VAL001,\\"MONTH\\" VAL002,\\"MONTH2\\" VAL003,\\"DAYOFMONTH\\" VAL
 ..@ .- attr(*, "names")= chr "17_7"
 ..@ dataObj   : chr "17_7"
 ..@ desc      : data.frame':   26 obs. of  2 variables:
 ..@ .$. name  : chr "YEAR" "MONTH" "MONTH2" "DAYOFMONTH" ...
 ..@ .$. Sclass: chr "numeric" "numeric" "factor" "numeric" ...
 ..@ sqlName   : chr
 ..@ sqlValue  : chr "\\"YEAR\\""" "\\"MONTH\\""" "\\"MONTH2\\""" "\\"DAYOFMONTH\\""" ...
 ..@ sqlTable  : chr "\\"ROUSER\\".\\"ONTIME_S\\"""
 ..@ sqlPred   : chr ""
 ..@ extRef    : list()
 ..@ names     : chr
 ..@ row.names : int
 ..@ .S3Class : chr "data.frame"
```

# ORE functions for interacting with database data

```
ore.sync()  
ore.sync("RUSER")  
ore.sync(table=c("ONTIME_S", "NARROW"))  
ore.sync("RUSER", table=c("ONTIME_S", "NARROW"))  
  
v <- ore.push(c(1,2,3,4,5))  
class(v)  
df <- ore.push(data.frame(a=1:5, b=2:6))  
class(df)  
  
ore.exists("ONTIME_S", "RUSER")
```

← Synchronize ORE proxy objects in R with tables/views available in database, on a per schema basis

← Store R object in database as temporary object, returns handle to object. Data frame, matrix, and vector to table, list/model/others to serialized object

← Returns TRUE if named table or view exists in schema

# ORE functions for interacting with database data

```
ore.attach("RUSER")
ore.attach("RUSER", pos=2)
search()

ore.ls()
ore.ls("RUSER")
ore.ls("RUSER",all.names=TRUE)
ore.ls("RUSER",all.names=TRUE, pattern= "NAR")

t <- ore.get("ONTIME_S","RUSER")
dim(t)

ore.detach("RUSER")

ore.rm("ONTIME_S")
ore.exists("ONTIME_S", "RUSER")
ore.sync()
ore.exists("ONTIME_S", "RUSER")
ore.rm(c("ONTIME_S","NARROW"), "RUSER")
ore.sync()
```

Make database objects visible in R for named schema. Can place corresponding environment in specific position in env path.

List the objects available in ORE environment mapped to database schema.

All.names=FALSE excludes names starting with a '.'

Obtain object to named table/view in schema.

Remove schema's environment from the object search path.

Remove table or view from schema's R environment.

# Creating and dropping tables

```
ore.exec("create table F2 as select * from ONTIME_S")
```

```
ore.create( ONTIME_S, table = "NEW_ONTIME_S")
```

```
ore.create( ONTIME_S, view = "NEW_ONTIME_S_VIEW")
```

```
ore.drop(table="NEW_ONTIME_S")
```

```
ore.drop(view="NEW_ONTIME_S_VIEW")
```

```
df <- data.frame(A=1:26, B=letters[1:26])
```

```
class(df)
```

```
ore.create(df, table="TEST_DF")
```

```
ore.ls(pattern="TEST_DF")
```

```
class(TEST_DF)
```

```
head(TEST_DF)
```

```
ore.drop(table="TEST_DF")
```

```
ontime <- ore.pull(ONTIME_S)
```

```
class(ONTIME_S)
```

```
class(ontime)
```

Execute SQL or PL/SQL without return value

Create a database table from a data.frame, ore.frame. Create a view from an ore.frame.

Drop table or view in database

Create a data.frame and then create a database table from it, then clean up

Load data (pull) from database

# Matrix multiplication %\*%

- %\*% - multiplies two matrices, if they are conformable

```
x <- 1:4
y <- diag(x)
z <- matrix(1:12, ncol = 3, nrow = 4)
X <- ore.push(x); Y <- ore.push(y); Z <- ore.push(z)
X %*% Z
Y %*% X
X %*% Z
Y %*% Z
```

```
R> Y
      [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    3    0
[4,]    0    0    0    4
R> Z
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
R>
R> Y %*% Z
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    4   12   20
[3,]    9   21   33
[4,]   16   32   48
```

# solve

- solve - solves the equation  $a \%*\% x = b$  for  $x$ , where  $b$  can be either a vector or a matrix.

```
hilbert <- function(n) {  
  i <- 1:n  
  1 / outer(i - 1, i, "+")  
}  
h8 <- hilbert(8); h8  
sh8 <- solve(h8)  
round(sh8 %** h8, 3)
```

```
R> hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }  
R> h8 <- hilbert(8); h8  
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]  
[1,] 1,0000000 0,5000000 0,3333333 0,2500000 0,2000000 0,16666667 0,14285714 0,12500000  
[2,] 0,5000000 0,3333333 0,2500000 0,2000000 0,16666667 0,14285714 0,12500000 0,11111111  
[3,] 0,3333333 0,2500000 0,2000000 0,16666667 0,14285714 0,12500000 0,11111111 0,10000000  
[4,] 0,2500000 0,2000000 0,16666667 0,14285714 0,12500000 0,11111111 0,10000000 0,09090909  
[5,] 0,2000000 0,16666667 0,1428571 0,12500000 0,11111111 0,10000000 0,09090909 0,08333333  
[6,] 0,16666667 0,1428571 0,1250000 0,11111111 0,10000000 0,09090909 0,08333333 0,07692308  
[7,] 0,1428571 0,1250000 0,1111111 0,10000000 0,09090909 0,08333333 0,07692308 0,07142857  
[8,] 0,1250000 0,1111111 0,1000000 0,09090909 0,08333333 0,07692308 0,07142857 0,06666667  
R> sh8 <- solve(h8)  
R> round(sh8 %** h8, 3)  
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]  
[1,] 1 0 0 0 0 0 0 0  
[2,] 0 1 0 0 0 0 0 0  
[3,] 0 0 1 0 0 0 0 0  
[4,] 0 0 0 1 0 0 0 0  
[5,] 0 0 0 0 1 0 0 0  
[6,] 0 0 0 0 0 1 0 0  
[7,] 0 0 0 0 0 0 1 0  
[8,] 0 0 0 0 0 0 0 1
```

# Invoke in-database aggregation function

```
aggdata <- aggregate(ONTIME_S$DEST,  
                     by = list(ONTIME_S$DEST),  
                     FUN = length)  
  
class(aggdata)  
head(aggdata)
```

```
R> aggdata <- aggregate(ONTIME_S$DEST,  
+                       by = list(ONTIME_S$DEST),  
+                       FUN = length)  
R> class(aggdata)  
[1] "ore.frame"  
attr(,"package")  
[1] "OREbase"  
R> head(aggdata)  
  Group,1  x  
0     ABE 237  
1     ABI 34  
2     ABQ 1357  
3     ABY 10  
4     ACK 3  
5     _  ACT 33
```



R user on desktop

Client R Engine

Other R packages

Transparency Layer

Oracle R package

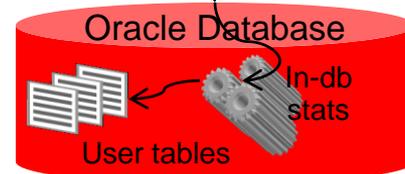
```
select DEST, count(*)  
from ONTIME_S  
group by DEST
```

Source data is an ore.frame ONTIME\_S, which resides in Oracle Database

The aggregate() function has been overloaded to accept ORE frames

aggregate() transparently switches between code that works with standard R data.frames and ore.frames

Returns an ore.frame

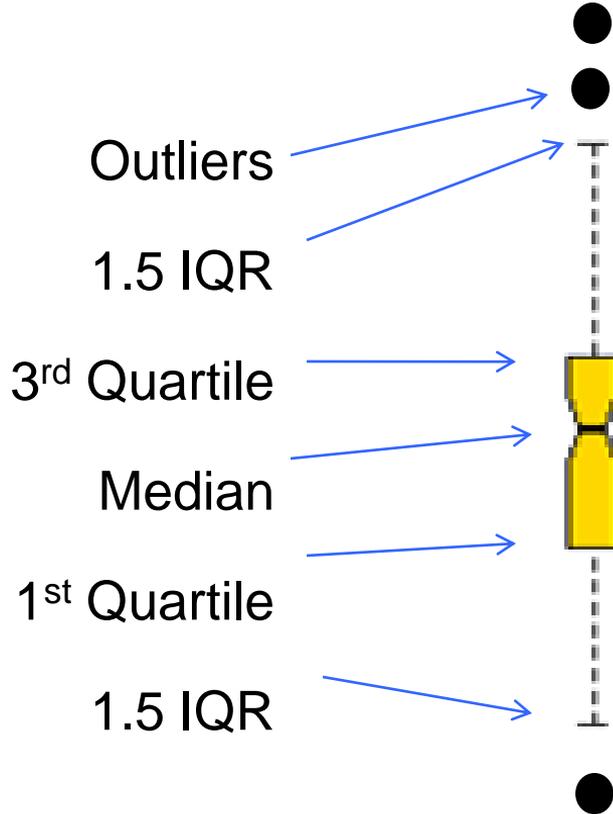


Analytics involving ONTIME data set

# Investigation Questions on ONTIME\_S

- Are some airports more prone to delays than others?
- Are some days of the week likely to see fewer delays than others?
  - Are these differences significant?
- How do arrival delay distributions differ for the best and worst 3 airlines compared to the industry?
  - Are there significant differences among airlines?
- For American Airlines, how has the distribution of delays for departures and arrivals evolved over time?
- How do average annual arrival delays compare across select airlines?
  - What is the underlying trend for each airline?

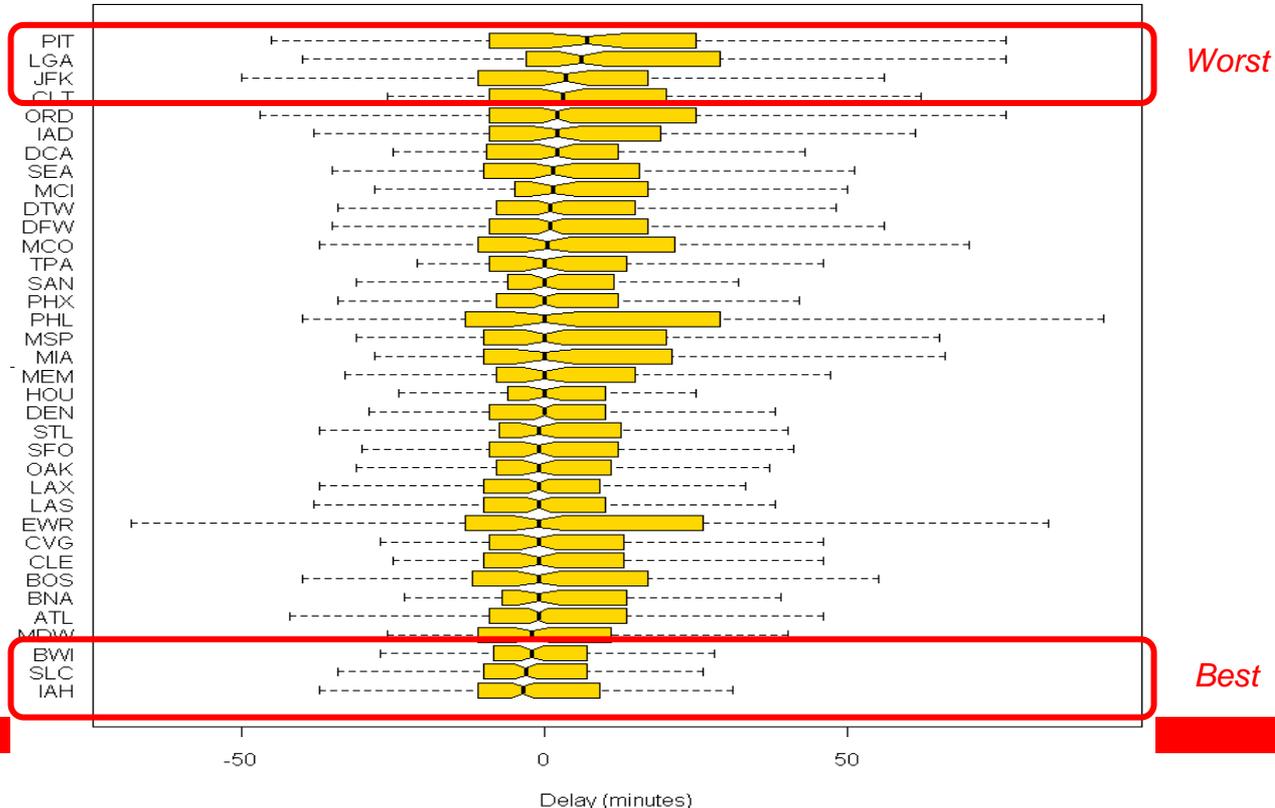
# Interpreting a Box Plot



- Facilitates comparison among multiple variables
- Limited number of quantities summarize each distribution
- *Interquartile range* measures spread of distribution (middle 50% of data)
- Median position indicates *skew*
- *Notch* gives roughly 95% confidence interval for the median

# Of the 36 busiest airports, which are the best/worst for Arrival Delay?

2007 Flight Delays by Airport -- top 36 busiest

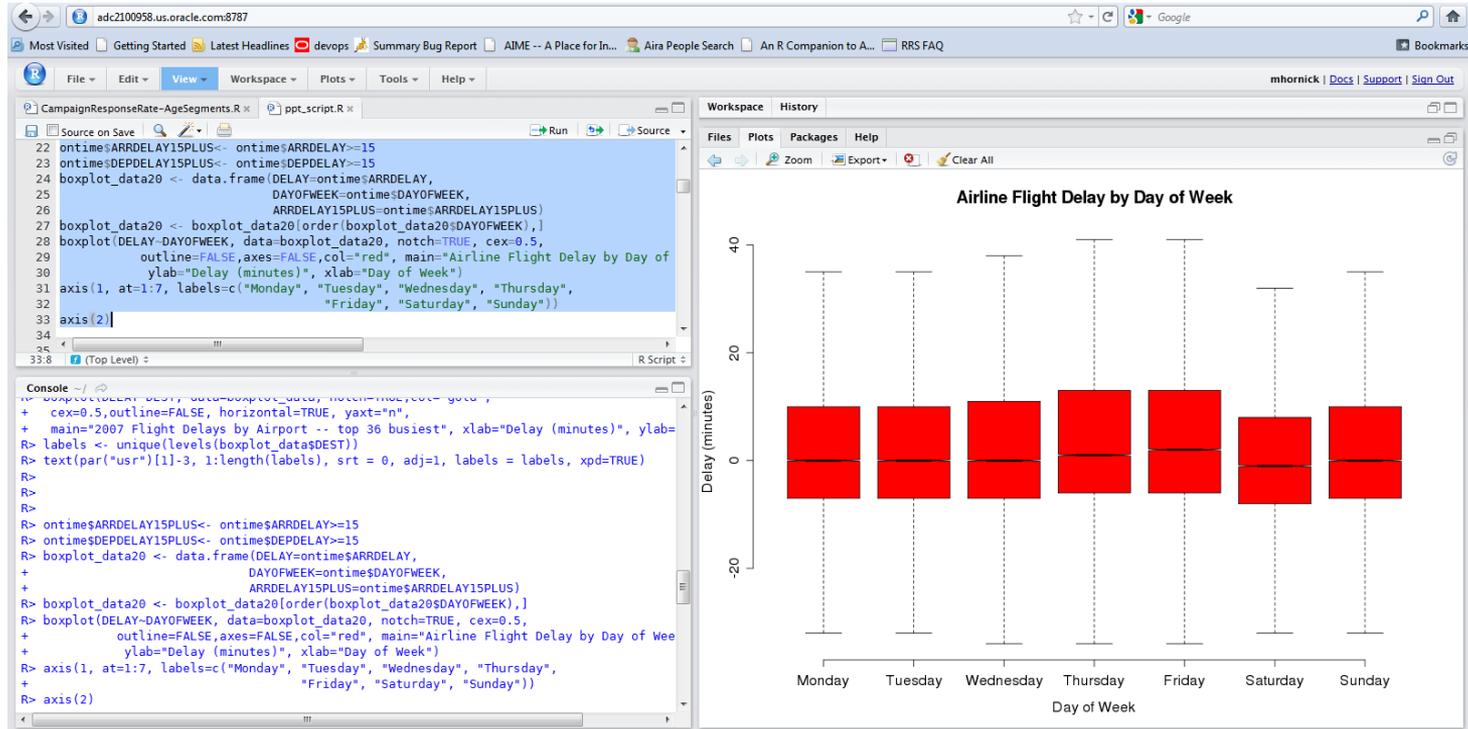


# Of the 36 busiest airports, which are the best/worst for Arrival Delay?

*Execute one line at a time and view the result*

```
1 ontime <- ONTIME_S
2 aggdata <- aggregate(ontime$DEST, by = list(ontime$DEST), FUN = length)
3 minx <- min(head(sort(aggdata$x, decreasing = TRUE), 36))
4 busiest_airports <- aggdata$Group.1[aggdata$x >= minx, drop = TRUE]
5 delay <- ontime$ARRDELAY[ontime$DEST %in% busiest_airports & ontime$YEAR == 2007]
6 dest <- ontime$DEST[ontime$DEST %in% busiest_airports & ontime$YEAR == 2007, drop = TRUE]
7 dest <- reorder(dest, delay, FUN = median, na.rm = TRUE)
8 bd <- split(delay, dest)
9 boxplot(bd, notch = TRUE, col = "gold", cex = 0.5,
10         outline = FALSE, horizontal = TRUE, yaxt = "n",
11         main = "2007 Flight Delays by Airport -- top 36 busiest",
12         ylab = "Delay (minutes)", xlab = "Airport")
13 labels <- levels(dest)
14 text(par("usr")[1] - 3, 1:length(labels), srt = 0, adj = 1,
15      labels = labels, xpd = TRUE, cex = 0.75)
```

# Which days were the worst to fly for delays over the past 22 years?



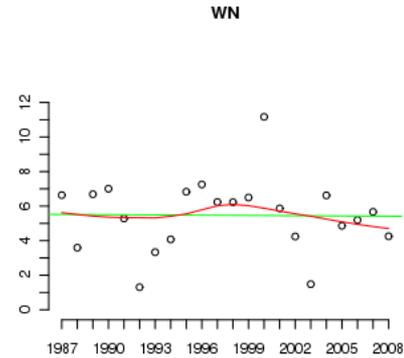
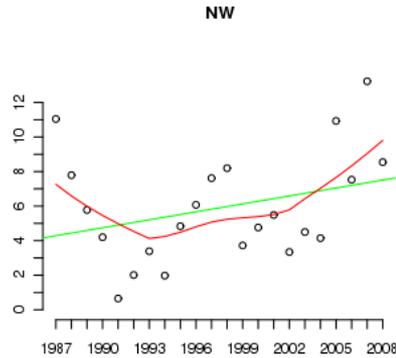
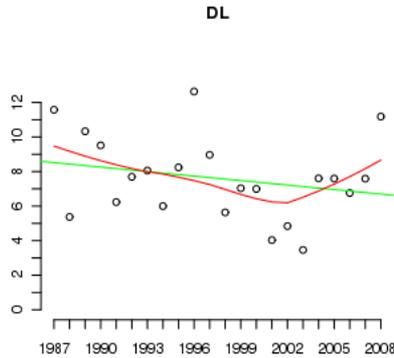
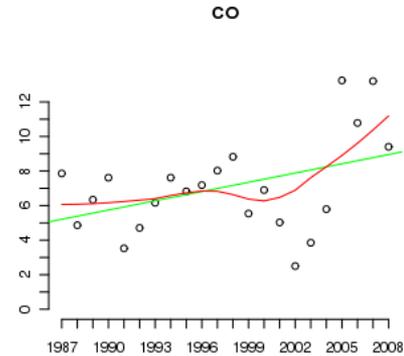
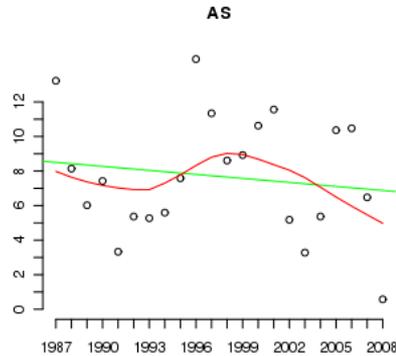
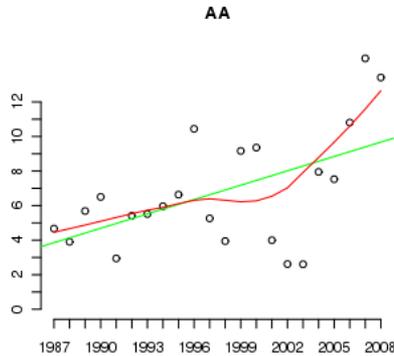
# Which days were the worst to fly for delays over the past 22 years?

*Execute one line at a time and view the result*

```
1 ontime <- ONTIME_S
2 delay <- ontime$ARRDELAY
3 dayofweek <- ontime$DAYOFWEEK
4 bd <- split(delay, dayofweek)
5 boxplot(bd, notch = TRUE, col = "red", cex = 0.5,
6         outline = FALSE, axes = FALSE,
7         main = "Airline Flight Delay by Day of Week",
8         ylab = "Delay (minutes)", xlab = "Day of Week")
9 axis(1, at=1:7, labels=c("Monday", "Tuesday", "Wednesday", "Thursday",
10                          "Friday", "Saturday", "Sunday"))
11 axis(2)
```

# Are select airlines getting better or worse?

*Mean annual delay by Year*



# Are select airlines getting better or worse?

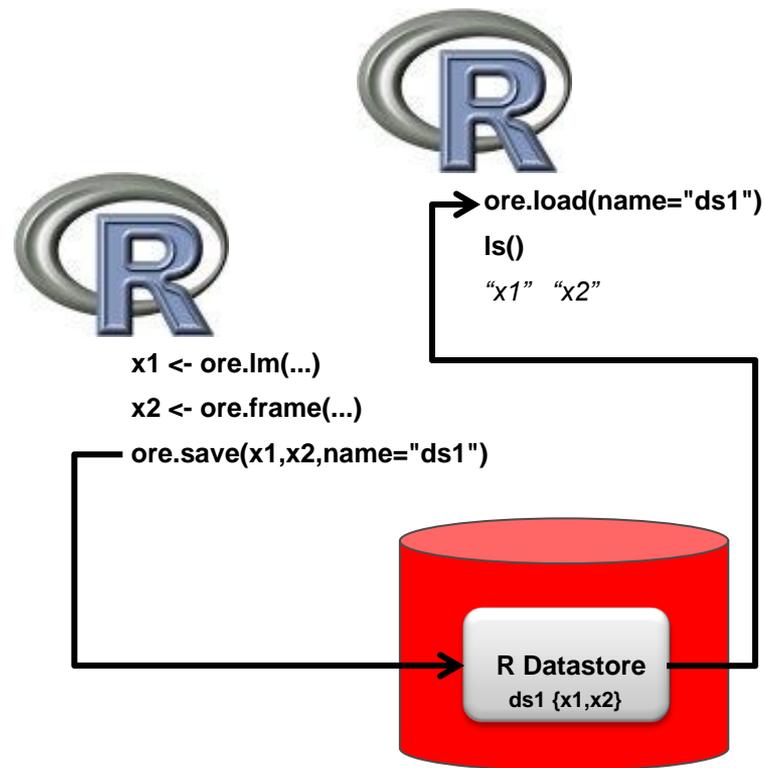
## *Mean annual delay by Year*

```
1 ontimeSubset <- subset(ONTIME_S, UNIQUECARRIER %in% c("AA", "AS", "CO", "DL", "WN", "NW"))
2
3 res22 <- with(ontimeSubset, tapply(ARRDELAY, list(UNIQUECARRIER, YEAR), mean, na.rm = TRUE))
4
5 g_range <- range(0, res22, na.rm = TRUE)
6 rindex <- seq_len(nrow(res22))
7 cindex <- seq_len(ncol(res22))
8 par(mfrow = c(2,3))
9 for(i in rindex) {
10   temp <- data.frame(index = cindex, avg_delay = res22[i,])
11   plot(avg_delay ~ index, data = temp, col = "black",
12        axes = FALSE, ylim = g_range, xlab = "", ylab = "",
13        main = attr(res22, "dimnames")[[1]][i])
14   axis(1, at = cindex, labels = attr(res22, "dimnames")[[2]])
15   axis(2, at = 0:ceiling(g_range[2]))
16   abline(lm(avg_delay ~ index, data = temp), col = "green")
17   lines(lowess(temp$index, temp$avg_delay), col="red")
18 }
```

# R Object Persistence in Oracle Database

# R Object Persistence with ORE

- `ore.save()`
- `ore.load()`
- Provide database storage to save/restore R and ORE objects across R sessions
- Use cases include
  - Enable passing of predictive model for embedded R execution, instead of recreating them inside the R functions
  - Passing arguments to R functions with embedded R execution
  - Preserve ORE objects across R sessions



# ore.save

```
DAT1      <- ore.push(ONTIME_S[,c("ARRDELAY", "DEPDELAY", "DISTANCE")])
ore.lm.mod <- ore.lm(ARRDELAY ~ DISTANCE + DEPDELAY, DAT1 )
lm.mod    <- lm(mpg ~ cyl + disp + hp + wt + gear, mtcars)
nb.mod    <- ore.odmNB(YEAR ~ ARRDELAY + DEPDELAY + log(DISTANCE), ONTIME_S)
ore.save(ore.lm.mod, lm.mod, nb.mod, name = "myModels")
```

- R objects and their referenced data tables are saved into the datastore of the connected schema
- Saved R objects are identified with datastore name *myModels*
- Arguments
  - ... the names of the objects to be saved (as symbols or character strings)
  - list — a character vector containing the names of objects to be saved
  - name — datastore name to identify the set of saved R objects in current user's schema
  - envir — environment to search for objects to be saved
  - overwrite — boolean indicating whether to overwrite the existing named datastore
  - append — boolean indicating whether to append to the named datastore
  - description -- comments about the datastore

# ore.load

```
ore.load(name = "myModels")
```

- Accesses the R objects stored in the connected schema with datastore name "*myModels*"
- These are restored to the R `.GlobalEnv` environment
- Objects `ore.lm.mod`, `lm.mod`, `nb.mod` can now be referenced and used
- Arguments
  - name — datastore name under current user schema in the connected schema
  - list — a character vector containing the names of objects to be loaded from the datastore, default is all objects
  - envir — the environment where R objects should be loaded in R

# ore.datastore

```
dsinfo <- ore.datastore(pattern = "my*")
```

- List basic information about R datastore in connected schema
- Result *dsinfo* is a data.frame
  - Columns: datastore.name, object.count (# objects in datastore), size (in bytes), creation.date, description
  - Rows: one per datastore object in schema
- Arguments – one of the following
  - name — name of datastore under current user schema from which to return data
  - pattern — optional regular expression. Only the datastores whose names match the pattern are returned. By default, all the R datastores under the schema are returned

## ore.datastore example

```
R> ore.datastore()
  datastore.name object.count   size   creation.date description
1   myDatastore           1 64461835 2012-11-14 17:12:36      <NA>
2   myIrisData            1    5789 2012-11-14 19:07:18      <NA>
3    myModels             3   45782 2012-11-14 18:56:32      <NA>
R>
R> ore.datastore(pattern="*Mod*")
  datastore.name object.count   size   creation.date description
1    myModels             3   45782 2012-11-14 18:56:32      <NA>
```

# ore.datastoreSummary

```
objinfo <- ore.datastoreSummary(name = "myModels")
```

- List names of R objects that are saved within named datastore in connected schema
- Result *objinfo* is a data.frame
  - Columns:  
object.name, class, size (in bytes), length (if vector),  
row.count (if data.frame), col.count (if data.frame)
  - Rows: one per datastore object in schema
- Argument
  - name — name of datastore under current user schema from which to list object contents

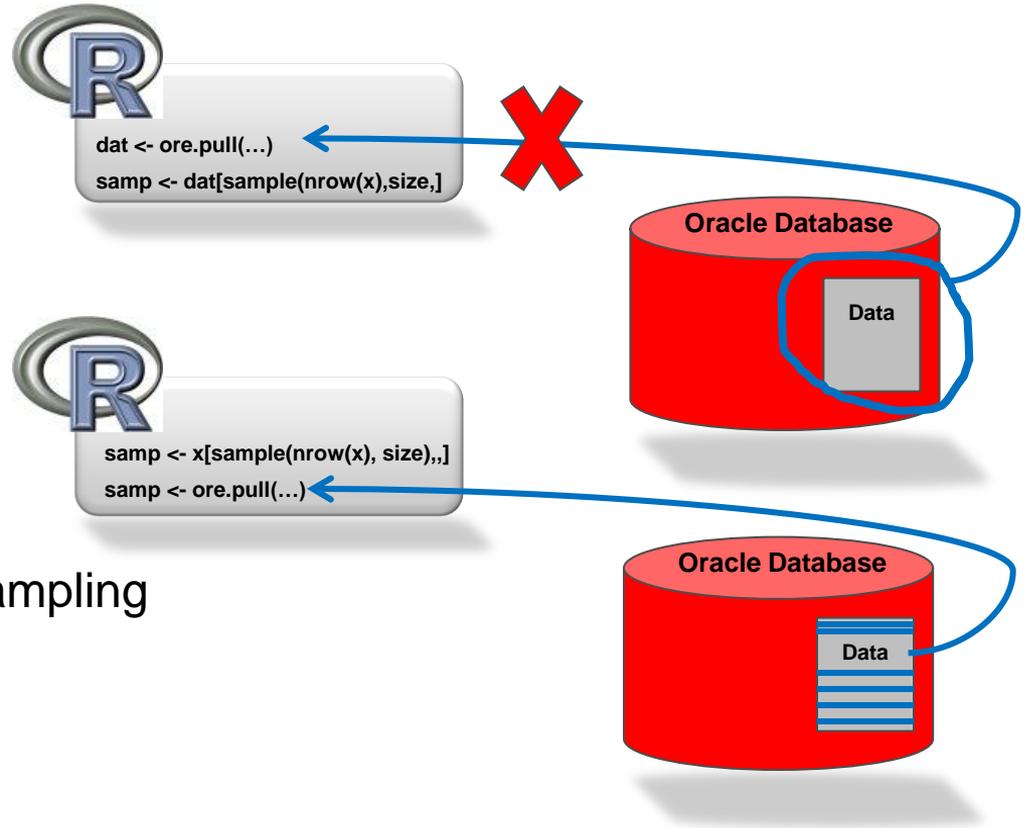
## ore.datastoreSummary example

```
R> ore.datastoreSummary("myModels")
  object.name      class  size length row.count col.count
1      lm.mod      lm 10352   12      NA      NA
2      nb.mod ore.odmNB 27015    9      NA      NA
3 ore.lm.mod      ore.lm  8415   11      NA      NA
R>
R> ore.datastoreSummary("myIrisData")
  object.name      class size length row.count col.count
1      iris data.frame 5789    5      150      5
```

# In-database Sampling

# In-database sampling techniques

- Simple random sampling
- Split data sampling
- Systematic sampling
- Stratified sampling
- Cluster sampling
- Quota sampling
- Accidental / Convenience sampling
  - via row order access
  - via hashing



# Simple random sampling

## Select rows at random

```
set.seed(1)
N <- 20
myData <- data.frame(a=1:N,b=letters[1:N])
MYDATA <- ore.push(myData)
head(MYDATA)
sampleSize <- 5
simpleRandomSample <- MYDATA[sample(nrow(MYDATA),
                                   sampleSize), ,
                             drop=FALSE]

class(simpleRandomSample)
simpleRandomSample
```

```
R> set.seed(1)
R> N <- 20
R> myData <- data.frame(a=1:N,b=letters[1:N])
R> MYDATA <- ore.push(myData)
R> head(MYDATA)
  a b
1 1 a
2 2 b
3 3 c
4 4 d
5 5 e
6 6 f
R> sampleSize <- 5
R> simpleRandomSample <- MYDATA[sample(nrow(MYDATA),
+                                     sampleSize), ,
+                                   drop=FALSE]
R> class(simpleRandomSample)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> simpleRandomSample
  a b
4  4 d
6  6 f
8  8 h
11 11 k
16 16 p
```

# Split data sampling

*Randomly partition data in train and test sets*

```
set.seed(1)

sampleSize <- 5
ind <- sample(1:nrow(MYDATA), sampleSize)
group <- as.integer(1:nrow(MYDATA) %in% ind)

MYDATA.train <- MYDATA[group==FALSE,]
dim(MYDATA.train)

class(MYDATA.train)

MYDATA.test <- MYDATA[group==TRUE,]
dim(MYDATA.test)
```

```
R> set.seed(1)
R>
R> sampleSize <- 5
R> ind <- sample(1:nrow(MYDATA), sampleSize)
R> group <- as.integer(1:nrow(MYDATA) %in% ind)
R>
R> MYDATA.train <- MYDATA[group==FALSE,]
R> dim(MYDATA.train)
[1] 15 2
R>
R> class(MYDATA.train)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R>
R> MYDATA.test <- MYDATA[group==TRUE,]
R> dim(MYDATA.test)
[1] 5 2
```

# Stratified sampling

## *ore.stratified.sample*

```
ore.drop("NARROW_SAMPLE_G")
ss <- ore.stratified.sample(x=NARROW, by="GENDER",
                           pct=0.1,
                           res.nm="NARROW_SAMPLE_G")

dim(NARROW_SAMPLE_G)
summary(NARROW_SAMPLE_G$GENDER)

ore.drop("R1_SAMPLE_G_MS")
res <- ore.stratified.sample(x=NARROW,
                             by=c("GENDER", "MARITAL_STATUS"),
                             pct=0.1,
                             res.nm="R1_SAMPLE_G_MS")

dim(R1_SAMPLE_G_MS)
summary(R1_SAMPLE_G_MS$GENDER)
summary(R1_SAMPLE_G_MS$MARITAL_STATUS)
with(R1_SAMPLE_G_MS, table(GENDER, MARITAL_STATUS))
```

```
R> ore.drop("NARROW_SAMPLE_G")
R> ss <- ore.stratified.sample(x=NARROW, by="GENDER",
+                             pct=0.1,
+                             res.nm="NARROW_SAMPLE_G")
[1] "# of stratum(s) to sample = 3, approx. # of sample = 150"
R> dim(NARROW_SAMPLE_G)
[1] 108 9
R> summary(NARROW_SAMPLE_G$GENDER)
 M F
77 31
R>
R> ore.drop("R1_SAMPLE_G_MS")
R> res <- ore.stratified.sample(x=NARROW,
+                               by=c("GENDER", "MARITAL_STATUS"),
+                               pct=0.1,
+                               res.nm="R1_SAMPLE_G_MS")
[1] "# of stratum(s) to sample = 21, approx. # of sample = 150"
[1] "10% is done "
[1] "20% is done "
[1] "29% is done "
[1] "39% is done "
[1] "48% is done "
[1] "58% is done "
[1] "67% is done "
[1] "77% is done "
[1] "86% is done "
[1] "96% is done "
R> dim(R1_SAMPLE_G_MS)
[1] 127 9
R> summary(R1_SAMPLE_G_MS$GENDER)
 M F
81 46
R> summary(R1_SAMPLE_G_MS$MARITAL_STATUS)
Married NeverM Divorc. Separ. Widowed Mabsent
 53      33      26      7      6      2
R> with(R1_SAMPLE_G_MS, table(GENDER, MARITAL_STATUS))
      MARITAL_STATUS
GENDER Divorc. Mabsent Married NeverM Separ. Widowed
 F      17      2      5      11      5      6
 M      9      0      48     22      2      0
```

# ORE Embedded R Execution

# 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?

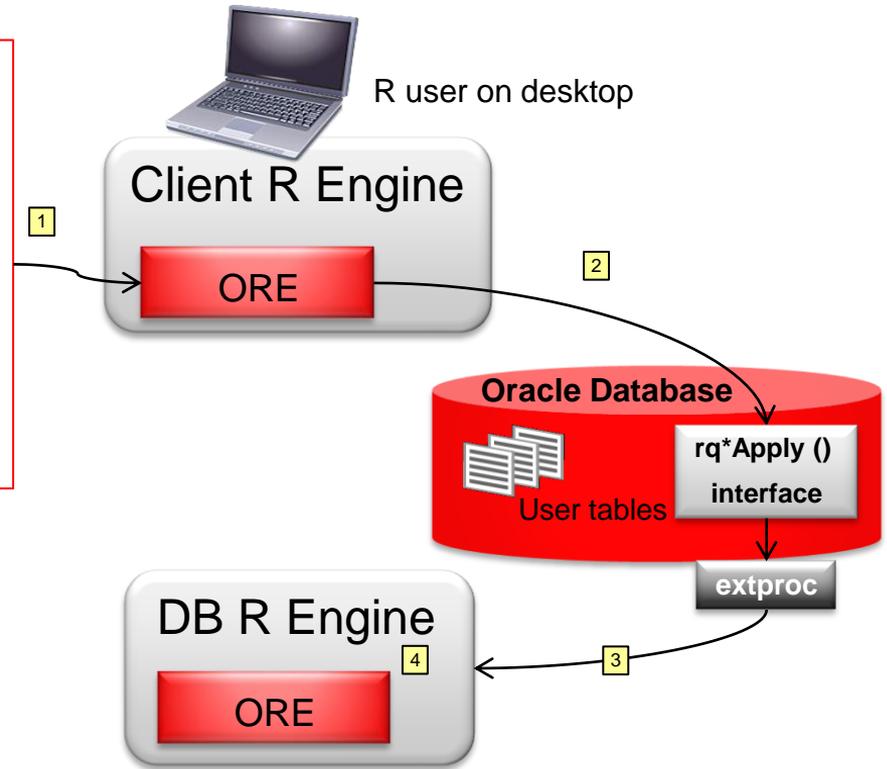
- Facilitate application use of R script results
  - Develop/test R scripts interactively with R interface
  - Invoke R scripts directly from SQL for production applications
  - R Scripts stored in Oracle Database
- Improved performance and throughput
  - Oracle Database data- and task-parallelism
  - 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

# 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 – passing 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

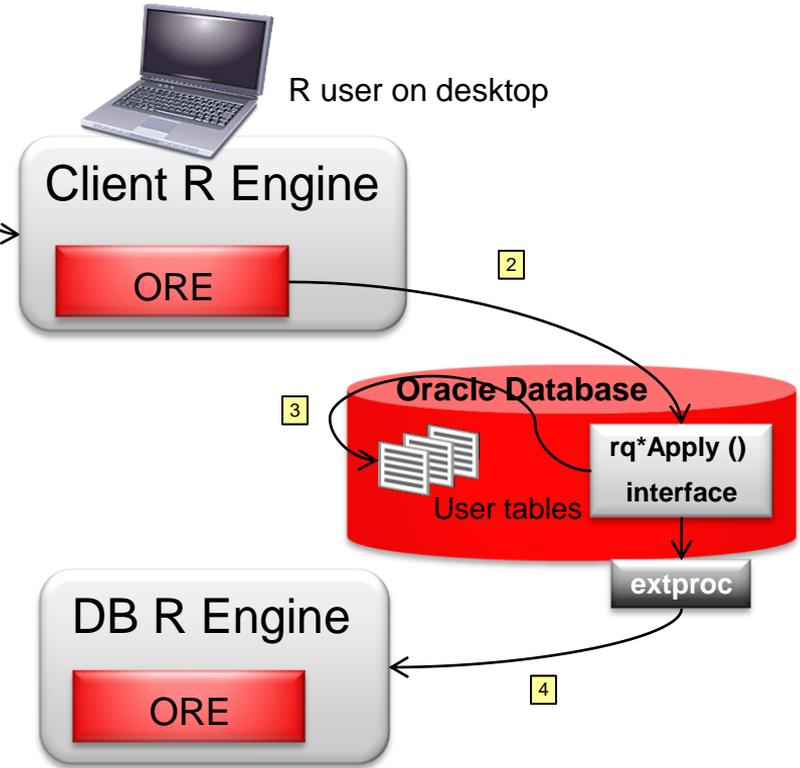
```
ore.scriptDrop("SimpleScript1")
ore.scriptCreate("SimpleScript1",
                function (num = 10, scale = 100) {
                    ID <- seq(num)
                    data.frame(ID = ID, RES = ID / scale)
                })
res <- ore.doEval(FUN.NAME="SimpleScript1",
                  num =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

# 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 e1071 loaded 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 0.2127002
```

# 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)
+   },
+   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)
```

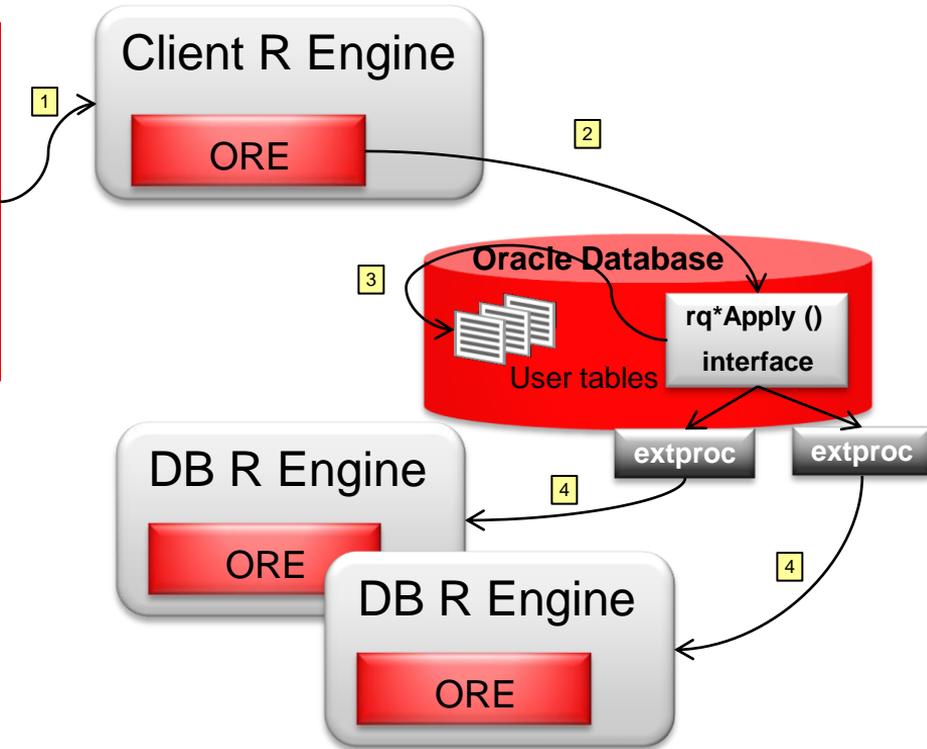
```
R> IRIS <- ore.push(iris)
R> IRIS_PRED$PRED <- "A"
R> res <- ore.rowApply(
+   IRIS ,
+   function(dat, mod) {
+     library(e1071)
+     dat$Species <- as.factor(dat$Species)
+     dat$PRED <- predict(mod, newdata = dat)
+     dat
+   },
+   mod = ore.pull(mod),
+   FUN.VALUE = IRIS_PRED,
+   rows=10)
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.groupApply – returning a single data.frame

```
IRIS <- ore.push(iris)
test <- ore.groupApply(IRIS, IRIS$Species,
  function(dat) {
    species <- as.character(dat$Species)
    mod <- lm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, dat)
    prd <- predict(mod, newdata=dat)
    prd[as.integer(rownames(prd))] <- prd
    data.frame(Species = species, PRED= prd, stringsAsFactors = FALSE)
  },
  FUN.VALUE = data.frame(Species = character(),
    PRED = numeric(),
    stringsAsFactors = FALSE),
  parallel = TRUE)

# save results in database table TEST
ore.create(test, "TEST")
```

# Viewing database server-generated graphics in client

```
ore.doEval(function () {  
  set.seed(71)  
  library(randomForest)  
  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)  
})
```

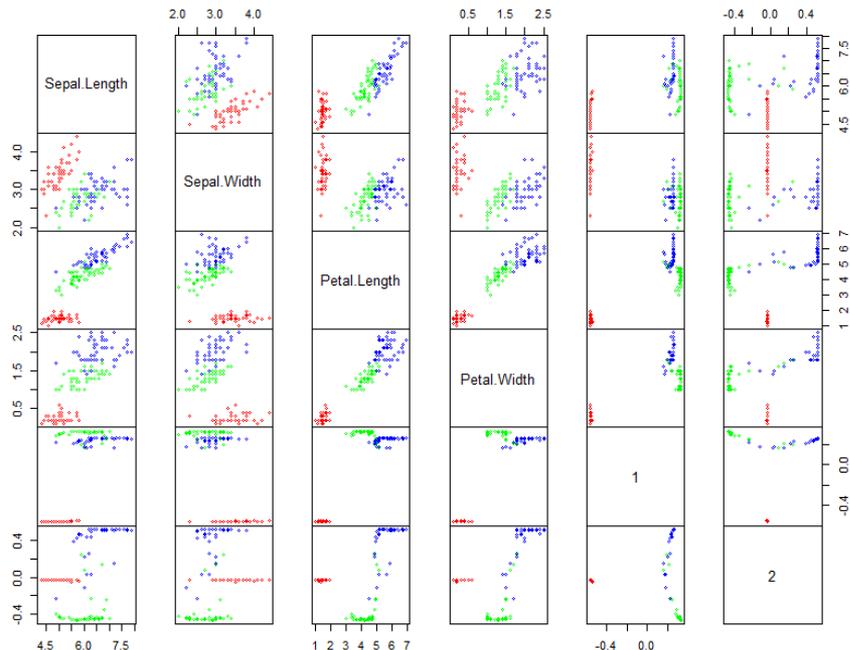
Goal: generate graph at database server, view on client and return importance from randomForest model

# 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
```

Iris Data: Predictors and MDS of Proximity Based on RandomForest



```
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)
  library(randomForest)
  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"))
```

# Embedded R Script Execution – R Interface

*Execute R scripts at the database server*

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

## Embedded R Scripts – SQL Interface

# 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;
```

# 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 1e2' ));
```

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
            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

# “Hello World!” XML Example

```
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> 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>
```

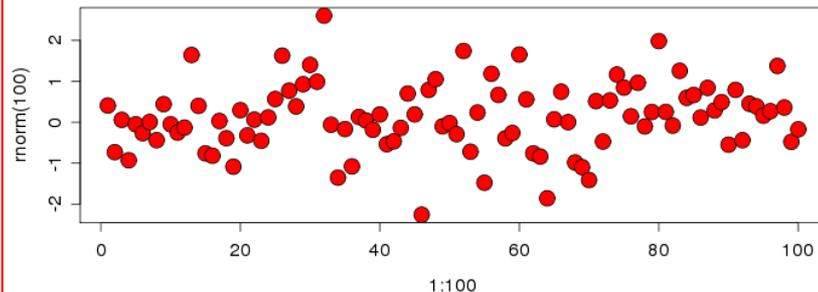
# Production Deployment – same R function, multiple uses

```
begin
  sys.rqScriptDrop('RandomRedDots');
  sys.rqScriptCreate('RandomRedDots',
'function() {
  id <- 1:10
  plot(1:100, rnorm(100), pch=21, bg="red", cex =2)
  data.frame(id=id, val=id / 100)
}');
end;
/
```

```
select value
from table(rqEval( NULL, 'XML', 'RandomRedDots' ));

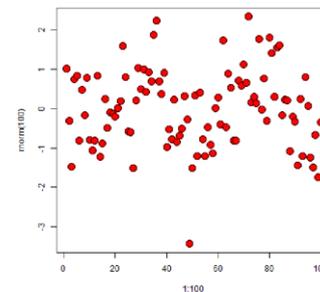
select ID, IMAGE
from table(rqEval( NULL, 'PNG', 'RandomRedDots' ));

select *
from table(rqEval( NULL,
  'select 1 id, 1 val from dual', 'RandomRedDots' ));
```



```
> ore.doEval(FUN.NAME="RandomRedDots")
```

id	val
1	0.01
2	0.02
3	0.03
4	0.04
5	0.05
6	0.06
7	0.07
8	0.08
9	0.09
10	0.10



# Results

## 'PNG' result

ID	IMAGE
1	1 (BLOB)

## 'select 1 id, 1 val from dual' result

ID	VAL
1	0.01
2	0.02
3	0.03
4	0.04
5	0.05
6	0.06
7	0.07
8	0.08
9	0.09
10	0.1

## 'XML' result

```
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-dat
<![CDATA[iVBORwOKGgoAAAANSUhcUgAAAAAAAAAAGCgAAAAAAAAA
CAAAgAE1EQVR4n0zdZ1xT1x8G8CcMB6jgQq0IDnDvulsRBSKyZQjIUncDKDhq3bvuvAlbcRYFFFRUBF
ExYlWnarGKA3GAgw0udvJ/wV8aTG5ESG4C/L4FXug9JzdPGL/c3Hvu0Rw+nw9CCCHyROHWAQghhIhGBZo
QQuQUFWhCCJFTVKAJIUROUYEmhBA5RQMaEELkFBVoQgiRU1SgCSFET1GBJoQQOUUFmhBC5BQVaEIIkVN
UoAkhRE5RgSaEED1FBZoQQuQUFWhCCJFTVKAJIUROUYEmhBA5RQMaEELkFBVoQgiRU1SgCSFET1GBJoQ
QUUUFmhBC5BQVaEIIkVNuoAkhRE5RgSaEED1FBZoQQuQUFWhCCJFTVKAJIUROUYEmhBA5RQMaEELkFBV
n0aiRlI1SoCSFFT1GR.In0000IIFmhBC5RQVaFTIkVNIInAkhRFR5R0SaFFT1FB7n0000IIFmhCCJFTVKAJTIIR
```

# 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’	Structured output data ignored Table with 1 image per row NAME    varchar2(4000) ID       number IMAGE   blob

# Predictive Analytics

# High performance in-database predictive techniques available through ORE packages

## OREdm

- Support Vector Machine
- GLM
- k-Means clustering
- OC clustering
- Naïve Bayes
- Decision Trees
- Association Rules
- Attribute Importance

## OREmodels

- Neural Networks
- Linear Regression
- Stepwise Regression
- Generalized Linear Model

# OREdm Features

- Function signatures conform to R norms
  - Use formula for specifying target and predictor variables
  - Use ore.frame objects as input data set for build and predict
  - Create R objects for model results
  - Use parameter names similar to corresponding R functions
  - Function parameters provide explicit default values to corresponding ODM settings, where applicable
- As in R, models are treated as transient objects
  - Automatically delete ODM model when corresponding R object no longer exists
  - Can be explicitly saved using datastore, via ore.save

# OREdm Algorithms

Algorithm	Main R Function	Mining Type / Function
Association Rules	ore.odmAssocRules	Association Rules
Minimum Description Length	ore.odmAI	Attribute Importance for Classification or Regression
Decision Tree	ore.odmDT	Classification
Generalized Linear Models	ore.odmGLM	Classification Regression
KMeans	ore.odmKMeans	Clustering
Naïve Bayes	ore.odmNB	Classification
Non-negative Matrix Factorization	ore.odmNFM	Feature Extraction
Orthogonal Partitioning	ore.odmOC	Clustering
Support Vector Machine	ore.odmSVM	Classification Regression Anomaly Detection

# Attribute Importance

- Compute the relative importance of predictor variables for predicting a response (target) variable
- Gain insight into the relevance of variables to guide manual variable selection or reduction, with the goal to reduce predictive model build time and/or improve model accuracy
- Attribute Importance uses a Minimum Description Length (MDL) based algorithm that ranks the relative importance of predictor variables in predicting a specified response (target) variable
- Pairwise only – each predictor with the target
- Supports categorical target (classification) and numeric target (regression)

# ore.odmAI - Example

## Attribute Importance

```
R> LONGLEY <- ore.push(longley)
```

```
R> head(LONGLEY)
```

	GNP.deflator	GNP	Unemployed	Armed.Forces	Population	Year	Employed	
?ore.odmAI	1947	83.0	234.289	235.6	159.0	107.608	1947	60.323
	1948	88.5	259.426	232.5	145.6	108.632	1948	61.122
	1949	88.2	258.054	368.2	161.6	109.773	1949	60.171
	1950	89.5	284.599	335.1	165.0	110.929	1950	61.187
	1951	96.2	328.975	209.9	309.9	112.075	1951	63.221
	1952	98.1	346.999	193.2	359.4	113.270	1952	63.639

```
R> ore.odmAI(Employed ~ ., LONGLEY)
```

Call:

```
ore.odmAI(formula = Employed ~ ., data = LONGLEY)
```

Importance:

	importance	rank
Year	0.4901166	1
Population	0.4901166	1
GNP	0.4901166	1
GNP.deflator	0.4901166	1
Armed.Forces	0.3648186	2
Unemployed	0.1318046	3

# ore.odmNB – Example

## Naïve Bayes

```
?ore.odmNB
library(ORE)
ore.connect("rquser","orcl","localhost","rquser",all=TRUE)
```

Login to database for transparent access via ORE

```
data(titanic3,package="PASWR")
```

Push data to db for transparent access

```
t3 <- ore.push(titanic3)
t3$survived <- ifelse(t3$survived == 1, "Yes", "No")
```

Recode column from 0/1 to No/Yes keeping data in database

```
n.rows <- nrow(t3)
set.seed(seed=6218945)
random.sample <- sample(1:n.rows, ceiling(n.rows/2))
t3.train <- t3[random.sample,]
t3.test <- t3[setdiff(1:n.rows,random.sample),]
```

Sample keeping data in database

```
priors <- data.frame(
  TARGET_VALUE = c("Yes", "No"),
  PRIOR_PROBABILITY = c(0.1, 0.9))
```

Create priors for model building

```
nb <- ore.odmNB(survived ~ pclass+sex+age+fare+embarked,
  t3.train, class.priors=priors)
```

Build model using R formula with transparency layer data

```
nb.res <- predict(nb, t3.test,"survived")
```

Score data using ore.frame with OREdm model object.

```
head(nb.res,10)
```

Display first 10 rows of data frame using transparency layer

```
with(nb.res, table(survived,PREDICTION, dnn = c("Actual","Predicted")))
```

Compute confusion matrix using transparency layer

```
library(verification)
res <- ore.pull(nb.res)
perf.auc <- roc.area(ifelse(res$survived == "Yes", 1, 0), res$"Yes")
auc.roc <- signif(perf.auc$a, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(res$survived == "Yes", 1, 0), res$"Yes", binormal=T,
  plot="both",
  xlab="False Positive Rate",
  ylab="True Postive Rate", main="Titanic survival ODM NB model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$a, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))
```

Retrieve result from database for using verification package

```
summary(nb)
```

View model object summary

```
ore.disconnect()
```

Disconnect from database

Model, train and test objects are automatically removed when session ends or R objects are removed

# ROC Curve

```
R> summary(nb)
```

```
Call:
ore.odmNB(formula = survived ~ pclass + sex + age + fare + embarked,
  data = t3,train, class.priors = priors)
```

```
Settings:
  value
prep.auto on
```

```
Apriori:
  No Yes
0.9 0.1
```

```
Tables:
$embarked
  'Cherbourg' 'Queenstown', 'Southampton'
No  0.1569620                0.8430380
Yes 0.3178295                0.6821705
```

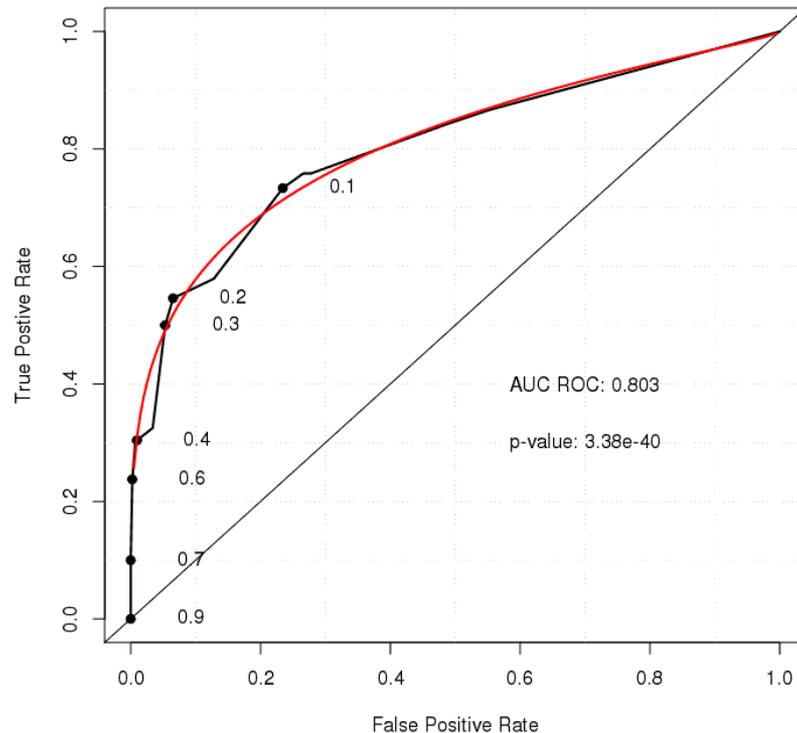
```
$fare
( ; 51.931249600000001), [51.931249600000001; 51.931249600000001] (51.931249600000001; )
No  0.91370558                0.08629442
Yes 0.67307692                0.32692308
```

```
$pclass
  '1st', '2nd'  '3rd'
No  0.3417722 0.6582278
Yes 0.6346154 0.3653846
```

```
$sex
  female  male
No 0.1670886 0.8329114
Yes 0.6769231 0.3230769
```

```
Levels:
[1] "No" "Yes"
```

Titanic survival ODM NB model ROC Curve



# ore.odmSVM – Example

## Support Vector Machine

```
?ore.odmSVM

x <- seq(0.1, 5, by = 0.02)
y <- log(x) + rnorm(x, sd = 0.2)
dat <- ore.push(data.frame(x=x, y=y))

# Regression
svm.mod <- ore.odmSVM(y~x, dat, "regression", kernel.function="linear")
summary(svm.mod)
coef(svm.mod)
svm.res <- predict(svm.mod, dat, supplemental.cols="x")
head(svm.res, 6)
```

# ore.odmSVM – Example

## Support Vector Machine

```
# Set up data set
m <- mtcars
m$gear <- as.factor(m$gear)
m$cyl <- as.factor(m$cyl)
m$vs <- as.factor(m$vs)
m$ID <- 1:nrow(m)
MTCARS <- ore.push(m)
```

```
# Classification
svm.mod <- ore.odmSVM(gear ~ .-ID, MTCARS,"classification")
summary(svm.mod)
coef(svm.mod)
svm.res <- predict(svm.mod, MTCARS,"gear")
head(svm.res)
svm.res <- predict(svm.mod, MTCARS,"gear",type="raw")
head(svm.res)
svm.res <- predict(svm.mod, MTCARS,"gear",type="class")
head(svm.res)
with(svm.res, table(gear,PREDICTION)) # confusion matrix

# Anomaly Detection
svm.mod <- ore.odmSVM(~ .-ID, MTCARS,"anomaly.detection")
summary(svm.mod)
svm.res <- predict(svm.mod, MTCARS, "ID")
head(svm.res)
table(svm.res$PREDICTION)
```

# ore.odmKMeans

## K-Means Clustering – model building

```
?ore.odmKMeans
```

```
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),  
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))  
colnames(x) <- c("x", "y")  
X <- ore.push (data.frame(x))  
km.mod1 <- ore.odmKMeans(~., X, num.centers=2, num.bins=5)  
summary(km.mod1)  
rules(km.mod1)  
clusterhists(km.mod1)  
histogram(km.mod1)
```

```
R> summary(km.mod1)
```

```
Call:  
ore.odmKMeans(formula = "~.", data = X, num.centers = 2, num.bins = 5)
```

```
Settings:  
                value  
clus.num.clusters 2  
block.growth      2  
conv.tolerance    0.01  
distance          euclidean  
iterations        3  
min.pct.attr.support 0.1  
num.bins          5  
split.criterion   variance  
prep.auto         on
```

```
Centers:  
      x      y  
2  1.05630476 1.0455933541  
3 -0.01131291 0.0001622473
```

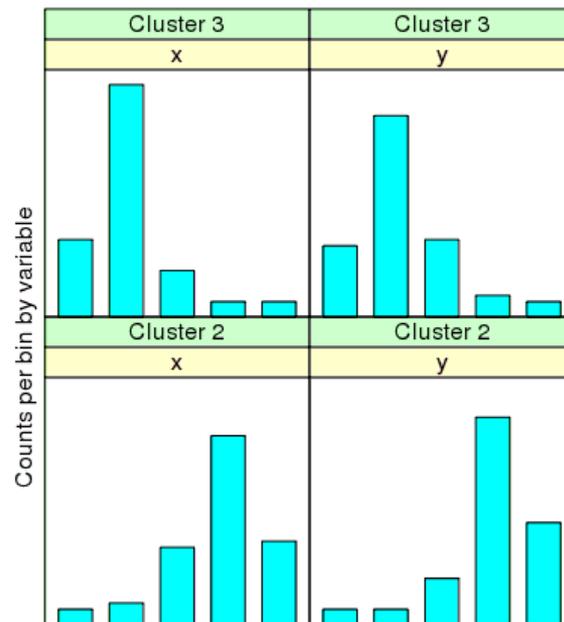
# ore.odmKMeans – results

```
R> rules(km_mod1)
rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.num.val lhs.var.chr.val lhs.var.support lhs.var.conf
1 1 100 1.0 90 0.9 x -0.2084763 <NA> 90 0.20
2 1 100 1.0 90 0.9 x 1.9020637 <NA> 90 0.20
3 1 100 1.0 90 0.9 y -0.2588969 <NA> 91 0.20
4 1 100 1.0 90 0.9 y 1.6761630 <NA> 91 0.20
5 2 50 0.5 45 0.9 x 0.3191587 <NA> 49 0.25
6 2 50 0.5 45 0.9 x 1.9020637 <NA> 49 0.25
7 2 50 0.5 45 0.9 y 0.7086331 <NA> 45 0.50
8 2 50 0.5 45 0.9 y 1.6761630 <NA> 45 0.50
9 3 50 0.5 45 0.9 x -0.7361113 <NA> 45 0.80
10 3 50 0.5 45 0.9 x 0.3191587 <NA> 45 0.80
11 3 50 0.5 45 0.9 y -0.7426619 <NA> 49 0.60
12 3 50 0.5 45 0.9 y 0.7086331 <NA> 49 0.60
$`1`
rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf predicate
2 1 100 1 90 0.9 x 90 0.2 <= 1.9021
1 1 100 1 90 0.9 x 90 0.2 >= -0.2085
4 1 100 1 90 0.9 y 91 0.2 <= 1.6762
3 1 100 1 90 0.9 y 91 0.2 >= -0.2589
$`2`
rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf predicate
6 2 50 0.5 45 0.9 x 49 0.25 <= 1.9021
5 2 50 0.5 45 0.9 x 49 0.25 >= 0.3192
8 2 50 0.5 45 0.9 y 45 0.50 <= 1.6762
7 2 50 0.5 45 0.9 y 45 0.50 >= 0.7086
$`3`
rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf predicate
10 3 50 0.5 45 0.9 x 45 0.8 <= 0.3192
9 3 50 0.5 45 0.9 x 45 0.8 >= -0.7361
12 3 50 0.5 45 0.9 y 49 0.6 <= 0.7086
11 3 50 0.5 45 0.9 y 49 0.6 >= -0.7427
```

# ore.odmKMeans – results

```
R> clusterhists(km.mod1)
cluster.id variable bin.id lower_bound upper_bound label count
1 1 x 1 -0.7361113 -0.2084763 -7.361E-01 ; -2.085E-01 10
2 1 x 2 -0.2084763 0.3191587 -2.085E-01 ; 3.192E-01 36
3 1 x 3 0.3191587 0.8467937 3.192E-01 ; 8.468E-01 15
4 1 x 4 0.8467937 1.3744287 8.468E-01 ; 1.374E+00 28
5 1 x 5 1.3744287 1.9020637 1.374E+00 ; 1.902E+00 11
6 1 y 1 -0.7426619 -0.2588969 -7.427E-01 ; -2.589E-01 9
7 1 y 2 -0.2588969 0.2248681 -2.589E-01 ; 2.249E-01 30
8 1 y 3 0.2248681 0.7086331 2.249E-01 ; 7.086E-01 15
9 1 y 4 0.7086331 1.1923980 7.086E-01 ; 1.192E+00 32
10 1 y 5 1.1923980 1.6761630 1.192E+00 ; 1.676E+00 14
11 2 x 1 -0.7361113 -0.2084763 -7.361E-01 ; -2.085E-01 0
12 2 x 2 -0.2084763 0.3191587 -2.085E-01 ; 3.192E-01 1
13 2 x 3 0.3191587 0.8467937 3.192E-01 ; 8.468E-01 10
14 2 x 4 0.8467937 1.3744287 8.468E-01 ; 1.374E+00 28
15 2 x 5 1.3744287 1.9020637 1.374E+00 ; 1.902E+00 11
16 2 y 1 -0.7426619 -0.2588969 -7.427E-01 ; -2.589E-01 0
17 2 y 2 -0.2588969 0.2248681 -2.589E-01 ; 2.249E-01 0
18 2 y 3 0.2248681 0.7086331 2.249E-01 ; 7.086E-01 5
19 2 y 4 0.7086331 1.1923980 7.086E-01 ; 1.192E+00 31
20 2 y 5 1.1923980 1.6761630 1.192E+00 ; 1.676E+00 14
21 3 x 1 -0.7361113 -0.2084763 -7.361E-01 ; -2.085E-01 10
22 3 x 2 -0.2084763 0.3191587 -2.085E-01 ; 3.192E-01 35
23 3 x 3 0.3191587 0.8467937 3.192E-01 ; 8.468E-01 5
24 3 x 4 0.8467937 1.3744287 8.468E-01 ; 1.374E+00 0
25 3 x 5 1.3744287 1.9020637 1.374E+00 ; 1.902E+00 0
26 3 y 1 -0.7426619 -0.2588969 -7.427E-01 ; -2.589E-01 9
27 3 y 2 -0.2588969 0.2248681 -2.589E-01 ; 2.249E-01 30
28 3 y 3 0.2248681 0.7086331 2.249E-01 ; 7.086E-01 10
29 3 y 4 0.7086331 1.1923980 7.086E-01 ; 1.192E+00 1
30 3 y 5 1.1923980 1.6761630 1.192E+00 ; 1.676E+00 0
```

Cluster Histograms



# ore.odmKMeans

## *K-Means Clustering – data scoring*

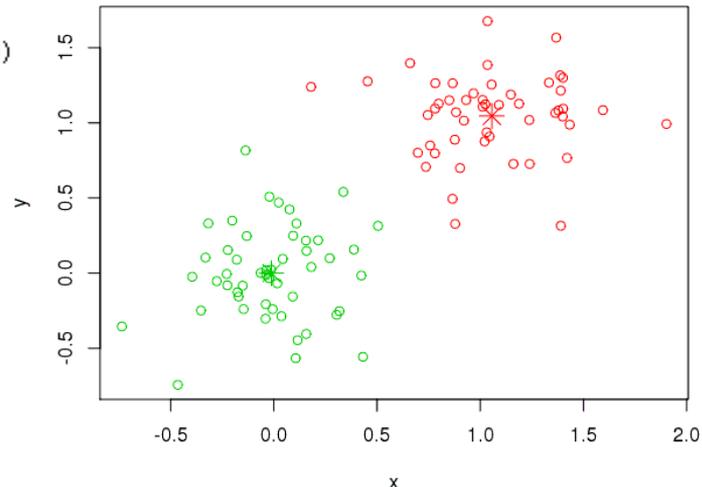
```
km.res1 <- predict(km.mod1,X,type="class",supplemental.cols=c("x","y"))
head(km.res1,3)
km.res1.local <- ore.pull(km.res1)

plot(data.frame(x=km.res1.local$x, y=km.res1.local$y), col=km.res1.local$CLUSTER_ID)
points(km.mod1$centers2, col = rownames(km.mod1$centers2), pch = 8, cex=2)

head(predict(km.mod1,X))
head(predict(km.mod1,X,type=c("class","raw"),supplemental.cols=c("x","y")),3)
head(predict(km.mod1,X,type="raw",supplemental.cols=c("x","y")),3)
```

# ore.odmKMeans – results

```
R> km.res1 <- predict(km.mod1,X,type="class",supplemental.cols=c("x","y"))
R> head(km.res1,3)
  x      y CLUSTER_ID
1 -0.03999935 -0.3029228      3
2  0.50486611  0.3145332      3
3 -0.20133745  0.3497027      3
R> km.res1.local <- ore.pull(km.res1)
R> plot(data.frame(x=km.res1.local$x, y=km.res1.local$y), col=km.res1.local$CLUSTER_ID)
R> points(km.mod1$centers2, col = rownames(km.mod1$centers2), pch = 8, cex=2)
R>
R> head(predict(km.mod1,X))
  '3'      '2' CLUSTER_ID
1 0.9999998 1.844763e-07      3
2 0.9338791 6.612089e-02      3
3 0.9999185 8.154833e-05      3
4 0.9999520 4.798267e-05      3
5 0.9999885 1.153331e-05      3
6 0.9995041 4.959004e-04      3
R> head(predict(km.mod1,X,type=c("class","raw"),supplemental.cols=c("x","y"),3))
  '3'      '2'      x      y CLUSTER_ID
1 0.9999998 1.844763e-07 -0.03999935 -0.3029228      3
2 0.9338791 6.612089e-02  0.50486611  0.3145332      3
3 0.9999185 8.154833e-05 -0.20133745  0.3497027      3
R> head(predict(km.mod1,X,type="raw",supplemental.cols=c("x","y"),3))
  x      y      '3'      '2'
1 -0.03999935 -0.3029228 0.9999998 1.844763e-07
2  0.50486611  0.3145332 0.9338791 6.612089e-02
3 -0.20133745  0.3497027 0.9999185 8.154833e-05
..      ..
```



# ore.odmDT

## Decision Tree Classification

```
?ore.odmDT

m <- mtcars
m$gear <- as.factor(m$gear)
m$cyl <- as.factor(m$cyl)
m$vs <- as.factor(m$vs)
m$ID <- 1:nrow(m)
MTCARS <- ore.push(m)
row.names(MTCARS) <- MTCARS

dt.mod <- ore.odmDT(gear ~ ., MTCARS)
summary(dt.mod)

dt.res <- predict(dt.mod, MTCARS, "gear")
# confusion matrix
with(dt.res, table(gear, PREDICTION))
```

```
> dt.mod <- ore.odmDT(gear ~ ., MTCARS)
> summary(dt.mod)
```

```
Call:
ore.odmDT(formula = gear ~ ., data = MTCARS)
n = 32
```

```
Nodes:
  parent node.id row.count prediction          split
1      NA         0         32          3      <NA>
2         0         1         16          4 (disp <= 196.2999)
3         0         2         16          3 (disp > 196.2999)

          surrogate          full.splits
1              <NA>              <NA>
2 (cyl in ("4" "6" )) (disp <= 196.299999999999995)
3 (cyl in ("8" )) (disp > 196.299999999999995)
```

```
Settings:
              value
prep.auto           on
impurity.metric  impurity.gini
term.max.depth     7
term.minpct.node   0.05
term.minpct.split  0.1
term.minrec.node   10
term.minrec.split  20
```

```
> dt.res <- predict(dt.mod, MTCARS, "gear")
> with(dt.res, table(gear, PREDICTION))
  PREDICTION
gear 3 4
   3 14 1
   4 0 12
   5 2 3
```

# Association (Market Basket Analysis)

## *Transactional Data and Rule Example*

Input Data:

User ID	Movies Viewed
1	{Movie1, Movie2, Movie3}
2	{Movie1, Movie4}
3	{Movie1, Movie3}
4	{Movie2, Movie5, Movie6}
...	...
N	{Movie3, Movie4, Movie6}

**Movie1 and Movie2 → Movie3**  
with support of .12 and confidence .78

# Association Rules

## Support and Confidence

User ID	Movies Viewed
1	{1, 2, 3}
2	{1, 4}
3	{1, 3}
4	{2, 5, 6}

$$\begin{aligned}\text{Support } (A \rightarrow B) &= P(AB) \\ &= \text{count}(A \& B) / \text{totalCount}\end{aligned}$$

$$\begin{aligned}\text{Confidence } (A \rightarrow B) &= P(AB)/P(A) \\ &= \text{count}(A \& B) / \text{count}(A)\end{aligned}$$

**1 → 3 :**

$$\text{Support} = 2/4 = 50\%$$

$$\text{Confidence} = 2/3 = 66\%$$

**3 → 1 :**

$$\text{Support} = 2/4 = 50\%$$

$$\text{Confidence} = 2/2 = 100\%$$

# ore.odmAssocRules

## Association Rules

```
# Relational data in a single-record case table.
ar.mod3 <- ore.odmAssocRules(~., NARROW,
  case.id.column = "ID",
  min.support=0.25, min.confidence=0.15,
  max.rule.length = 2)

rules = rules(ar.mod3)
itemsets = itemsets(ar.mod3)

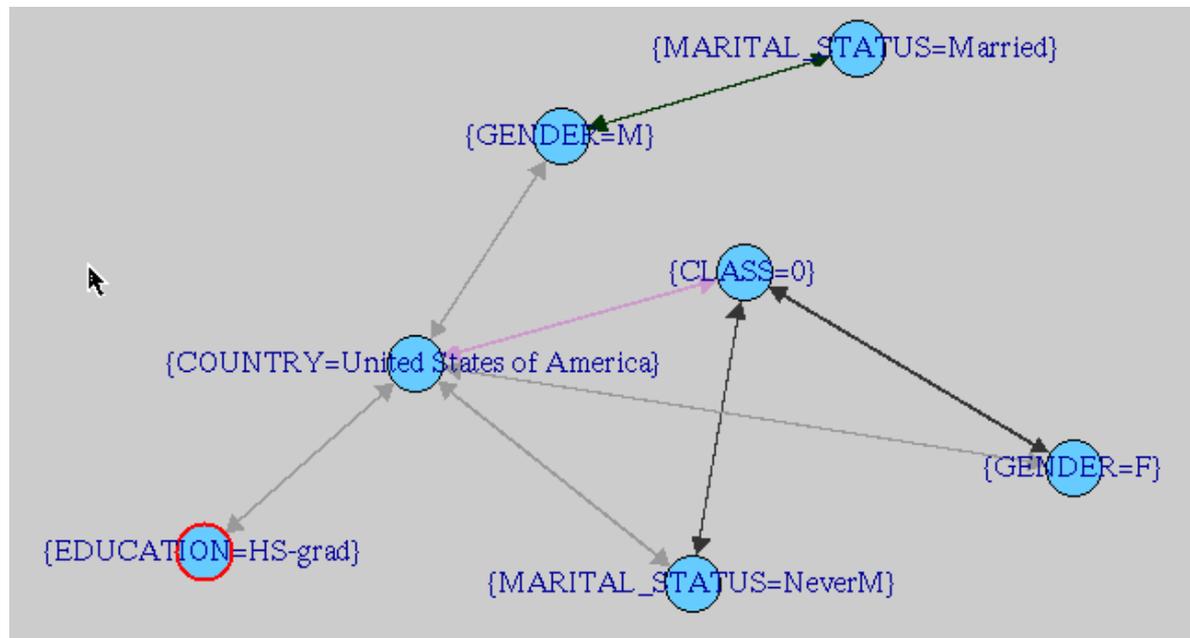
itemsets.arules <- ore.pull(itemsets)
inspect(itemsets.arules)

rules.arules <- ore.pull(rules)

plot(rules.arules, method = "graph",
  interactive = TRUE)
```

```
R> inspect(itemsets.arules)
  items                                     support
1 {COUNTRY=United States of America} 0.8960000
2 {CLASS=0}                             0.7466667
3 {CLASS=0,
  COUNTRY=United States of America} 0.6646667
4 {GENDER=M}                             0.5866667
5 {COUNTRY=United States of America,
  GENDER=M}                             0.5273333
6 {MARITAL_STATUS=Married}              0.4133333
7 {CLASS=0,
  GENDER=M}                             0.3986667
8 {COUNTRY=United States of America,
  MARITAL_STATUS=Married}              0.3646667
9 {GENDER=M,
  MARITAL_STATUS=Married}              0.3140000
10 {GENDER=F}                            0.2806667
11 {EDUCATION=HS-grad}                   0.2806667
12 {MARITAL_STATUS=NeverM}               0.2793333
13 {CLASS=0,
  MARITAL_STATUS=NeverM}               0.2633333
14 {COUNTRY=United States of America,
  EDUCATION=HS-grad}                   0.2586667
15 {CLASS=1}                             0.2533333
16 {COUNTRY=United States of America,
  MARITAL_STATUS=NeverM}               0.2533333
17 {CLASS=0,
  GENDER=F}                             0.2520000
18 {COUNTRY=United States of America,
  GENDER=F}                             0.2520000
```

```
plot(rules.arules, method = "graph",  
      interactive = TRUE)
```



# OREmodels Package

# ore.lm

```
LONGLEY <- ore.push(longley)

# Fit full model
oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
summary(oreFit1)
```

```
R> LONGLEY <- ore.push(longley)
R>
R> # Fit full model
R> oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
R> summary(oreFit1)

Call:
ore.lm(formula = Employed ~ ., data = LONGLEY)

Residuals:
    Min       1Q   Median       3Q      Max
-0.41011 -0.15980 -0.02816  0.15681  0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.482e+03  8.904e+02  -3.911  0.003560 **
GNP.deflator  1.506e-02  8.492e-02   0.177  0.863141
GNP          -3.582e-02  3.349e-02  -1.070  0.312681
Unemployed   -2.020e-02  4.884e-03  -4.136  0.002535 **
Armed.Forces -1.033e-02  2.143e-03  -4.822  0.000944 ***
Population   -5.110e-02  2.261e-01  -0.226  0.826212
Year         1.829e+00  4.555e-01   4.016  0.003037 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared:  0.9955,    Adjusted R-squared:  0.9925
F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

# Stepwise Regression: ore.stepwise

## Motivation

- Automatically selects predictive variables
- Produces models with fewer terms
- Enable handling data with complex patterns
  - Even for relatively small data sets (e.g., < 1M rows) R may not yield satisfactory results
- Increases performance
  - Side benefit of handling complex patterns is to dramatically boost performance
  - No need to pull data into memory from database
  - Leverage more powerful database machine
- Provide a stepwise regression that maps to SAS PROC REG

# ore.stepwise – example

```
LONGLEY <- ore.push(longley)

# Using ore.stepwise
oreStep1 <-
  ore.stepwise(Employed ~ .^2, data = LONGLEY,
              add.p = 0.1, drop.p = 0.1)
oreStep1
```

Build model with interaction terms

```
# Using R step with ore.lm
oreStep2 <-
  step(ore.lm(Employed ~ 1, data = LONGLEY),
       scope = terms(Employed ~ .^2, data = LONGLEY))
oreStep2
```

# ore.stepwise – results

```
R> oreStep1 <-
+   ore.stepwise(Employed ~ .^2, data = LONGLEY,
+               add.p = 0.1, drop.p = 0.1)
R> oreStep1
```

Aliased:

```
[1] "Unemployed:Armed.Forces" "Unemployed:Population" "Unemployed:Year" "Armed.Forces:Population"
"
[5] "Armed.Forces:Year" "Population:Year"
```

Steps:

	Add	Drop	RSS	Rank
1	GNP.deflator;Unemployed	<NA>	384,426	2
2		GNP:Year	218,957	3
3		GNP.deflator;GNP	130,525	4
4		GNP.deflator;Population	81,211	5
5		GNP;Armed.Forces	18,244	6
6		Year	14,492	7

Call:

```
ore.stepwise(formula = Employed ~ .^2, data = LONGLEY, add.p = 0.1,
             drop.p = 0.1)
```

Coefficients:

(Intercept)	Year	GNP.deflator;GNP	GNP.deflator;Unemployed	GNP,de
flator;Population -3,539e-01 2,303e-05	3,589e-05	-2,978e-03	2,326e-04	
GNP;Armed.Forces 6,875e-06	GNP:Year 2,007e-04			

# step with ore.lm – results

## Akaike information criterion (AIC)

- Measure of quality of a model

- Used for model selection

```
R> oreStep2 <-
+ step(ore.lm(Employed ~ 1, data = LONGLEY),
+ scope = terms(Employed ~ .^2, data = LONGLEY))
Start: AIC=41.17
Employed ~ 1
```

	Df	Sum of Sq	RSS	AIC
+ GNP	1	178.973	6.036	-11.597
+ Year	1	174.552	10.457	-2.806
+ GNP.deflator	1	174.397	10.611	-2.571
+ Population	1	170.643	14.366	2.276
+ Unemployed	1	46.716	138.293	38.509
+ Armed.Forces	1	38.691	146.318	39.411
<none>			185.009	41.165

```
Step: AIC=-11.6
Employed ~ GNP
```

	Df	Sum of Sq	RSS	AIC
+ Unemployed	1	2.457	3.579	-17.960
+ Population	1	2.162	3.874	-16.691
+ Year	1	1.125	4.911	-12.898
<none>			6.036	-11.597
+ GNP.deflator	1	0.212	5.824	-10.169
+ Armed.Forces	1	0.077	5.959	-9.802
- GNP	1	178.973	185.009	41.165

```
Step: AIC=-17.96
Employed ~ GNP + Unemployed
```

	Df	Sum of Sq	RSS	AIC
+ Armed.Forces	1	0.822	2.757	-20.137
<none>			3.579	-17.960
+ Year	1	0.340	3.239	-17.556
+ GNP:Unemployed	1	0.182	3.397	-16.795
+ Population	1	0.097	3.482	-16.399
+ GNP.deflator	1	0.019	3.560	-16.044
- Unemployed	1	2.457	6.036	-11.597
- GNP	1	134.714	138.293	38.509

```
Step: AIC=-20.14
Employed ~ GNP + Unemployed + Armed.Forces
```

	Df	Sum of Sq	RSS	AIC
+ Year	1	1.898	0.859	-36.799
+ GNP:Unemployed	1	0.614	2.143	-22.168
+ Population	1	0.390	2.367	-20.578
<none>			2.757	-20.137
+ Unemployed:Armed.Forces	1	0.083	2.673	-18.629
+ GNP.deflator	1	0.073	2.684	-18.566
+ GNP:Armed.Forces	1	0.060	2.697	-18.489
- Armed.Forces	1	0.822	3.579	-17.960
- Unemployed	1	3.203	5.959	-9.802
- GNP	1	78.494	81.250	31.999

```
Step: AIC=-36.8
Employed ~ GNP + Unemployed + Armed.Forces + Year
```

	Df	Sum of Sq	RSS	AIC
<none>			0.8587	-36.799
+ Unemployed:Year	1	0.0749	0.7838	-36.259
+ GNP:Unemployed	1	0.0678	0.7909	-36.115
+ Unemployed:Armed.Forces	1	0.0515	0.8072	-35.788
+ GNP:Armed.Forces	1	0.0367	0.8220	-35.498
+ Population	1	0.0193	0.8393	-35.163
+ GNP.deflator	1	0.0175	0.8412	-35.129
+ Armed.Forces:Year	1	0.0136	0.8451	-35.054
+ GNP:Year	1	0.0084	0.8502	-34.957
- GNP	1	0.4647	1.3234	-31.879
- Year	1	1.8980	2.7567	-20.137
- Armed.Forces	1	2.3806	3.2393	-17.556
- Unemployed	1	4.0491	4.9077	-10.908

```
R> oreStep2
```

```
Call:
ore.lm(formula = Employed ~ GNP + Unemployed + Armed.Forces +
Year, data = LONGLEY)
```

```
Coefficients:
(Intercept)      GNP      Unemployed  Armed.Forces      Year
-3.599e+03  -4.019e-02  -2.088e-02  -1.015e-02  1.887e+00
```

# Artificial Neural Networks

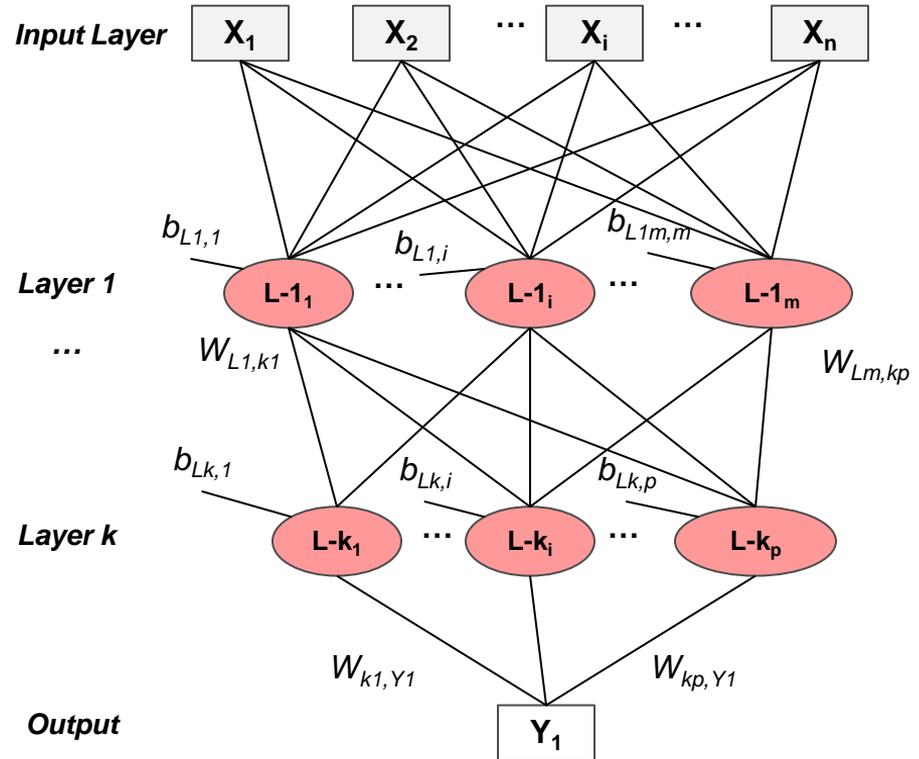
- Neural network (NN) is a mathematical model inspired by biological neural networks and in some sense mimics the functioning of a brain
  - Consists of an interconnected group of artificial neurons (nodes)
  - Non-linear statistical data modeling tools
  - Model complex nonlinear relationships between input and output variables
- Find patterns in data:
  - Function approximation: regression analysis, including time series prediction, fitness approximation, and modeling
  - Classification: including pattern and sequence recognition, novelty detection and sequential decision making
  - Data processing: including filtering, clustering, blind source separation and compression
  - Robotics: including directing manipulators, computer numerical control

# Artificial Neural Networks

- Well-suited to data with noisy and complex sensor data
- Problem characteristics
  - Potentially many (numeric) predictors, e.g., pixel values
  - Target may be discrete-valued, real-valued, or a vector of such
  - Training data may contain errors – robust to noise
  - Fast scoring
  - Model transparency not required – models difficult to interpret
- Universal approximator
  - Adding more neurons can lower error to be as small as desired
  - Not always the desired behavior

# Architecture Specification

- Input Layer
  - Numerical or categorical
  - No automatic normalization of data
  - Supports up to 1000 actual columns (due to database table limit)
  - No fixed limit on interactions
  - No fixed limit on cardinality of categorical variables
- Hidden Layers
  - Any number of hidden layers -  $k$
  - All nodes from previous layer are connected to nodes of next
  - Activation function applies to one layer
    - Bipolar Sigmoid default for hidden layers
- Output Layer
  - Currently single numeric target or binary categorical
  - Linear activation function default, all others also supported
- Calculate number of weights
  - $(\# \text{ input units}) \times (\# \text{ L1 nodes}) + (\# \text{ L1 nodes bias}) +$   
 $(\# \text{ L1 nodes}) \times (\# \text{ L2 nodes}) + (\# \text{ L2 nodes bias}) +$   
...
  - $(\# \text{ Lk nodes}) \times (\# \text{ output nodes})$
- Initialize weights
  - Change initialization with random seed
  - Set lower and upper bound, typically -0.25, 0.25



# Local Minima – comparison with nnet

```
R> fit.nn <- nnet(formula = T ~ A + B,
  data = d, size=2)
# weights:  9
initial  value 1.046487
iter   10 value 0.997966
iter   20 value 0.569304
iter   30 value 0.502784
iter   40 value 0.500426
iter   50 value 0.500050
final   value 0.500041
converged
R>
R> predict(fit.nn,d)
      [,1]
1 0.499970251
2 0.999986345
3 0.002586049
4 0.500004855
```

```
R> fit.ore <- ore.neural(formula = T ~ A + B, data
  = ore.push(d),
+       hiddenSizes = c(5000, 10, 10),
+       lowerBound=-1, upperBound=1)
R> predict(fit.ore,ore.push(d))
      pred_T
1  0.913355525
2  1.035549253
3 -0.020444140
4 -0.001179834
```

# Example

```
IRIS <- ore.push(iris)
```

```
fit <- ore.neural(Petal.Length ~ Petal.Width + Sepal.Length,  
  data = IRIS, hiddenSizes = c(20, 5),  
  activations = c('bSigmoid', 'tanh', 'linear'))
```

```
fit
```

```
R> fit  
Number of input units      2  
Number of output units    1  
Number of hidden layers   2  
Objective value            6.431877E+00  
Solution status           Optimal  
Hidden layer [1]          number of neurons 20, activation 'bSigmoid'  
Hidden layer [2]          number of neurons 5, activation 'tanh'  
Output layer              number of neurons 1, activation 'linear'  
Optimization solver       L-BFGS  
Scale Hessian inverse      1  
Number of L-BFGS updates  20
```

```
ans <- predict(fit, newdata = IRIS,  
  supplemental.cols = 'Petal.Length')  
localPredictions <- ore.pull(ans)
```

```
# Inspect some predictions  
head(localPredictions)
```

```
# Compute RMSE  
ore.rmse <- function (pred, obs) {  
  sqrt(mean(pred-obs)^2)  
}
```

```
R> ore.rmse(localPredictions$pred_Petal.Length,  
  localPredictions$Petal.Length)  
[1] 0.00148768
```

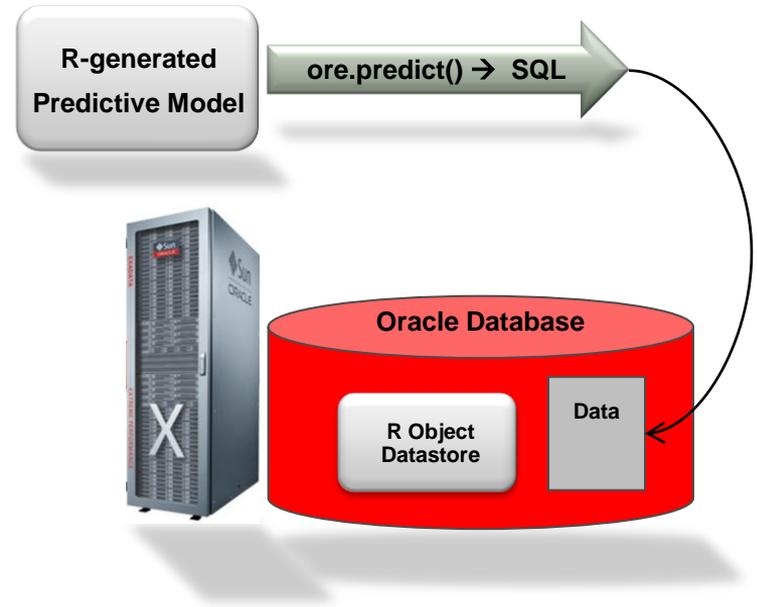
# ore.neural vs. nnet

- ORE is
  - Scalable
  - Choose activation functions
  - Generic topology
    - unrestricted number of hidden layers, including none
  - Parallel implementation

# OREpredict Package

# Exadata storage tier scoring for R models

- Fastest way to operationalize R-based models for scoring in Oracle Database
- Go from model to SQL scoring in one step
  - No dependencies on PMML or any other plugins
- R packages supported out-of-the-box include
  - glm, glm.nb, hclust, kmeans, lm, multinom, nnet, rpart
- Models can be managed in-database using ORE datastore



# OREpredict Package

- Provide a commercial grade scoring engine
  - High performance
  - Scalable
  - Simplify application workflow
- Use R-generated models to score in-database on ore.frame
- Maximizes use of Oracle Database as compute engine
- Function `ore.predict`
  - S4 generic function
  - A specific method for each model ORE supports

# ore.predict supported algorithms

Class	Package	Description
glm	stats	Generalized Linear Model
negbin	MASS	Negative binomial Generalized Linear Model
hclust	stats	Hierarchical Clustering
kmeans	stats	K-Means Clustering
lm	stats	Linear Model
multinom	nnet	Multinomial Log-Linear Model
nnet	nnet	Neural Network
rpart	rpart	Recursive Partitioning and Regression Tree

# Example using lm

```
irisModel <- lm(Sepal.Length ~ ., data = iris)
IRIS      <- ore.push(iris)
IRISpred  <- ore.predict(irisModel, IRIS, se.fit = TRUE,
                        interval = "prediction")
IRIS <- cbind(IRIS, IRISpred)
head(IRIS)
```

- Build a typical R lm model
- Use ore.predict to score data in Oracle Database using ore.frame, e.g., IRIS

```
R> irisModel <- lm(Sepal.Length ~ ., data = iris)
R> IRIS <- ore.push(iris)
R> IRISpred <- ore.predict(irisModel, IRIS, se.fit = TRUE,
+                          interval = "prediction")
R> IRIS <- cbind(IRIS, IRISpred)
R> head(IRIS)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  PRED  SE_PRED LOWER_PRED UPPER_PRED
1          5.1         3.5         1.4         0.2  setosa  5.004788 0.04479188  4.391895  5.617681
2          4.9         3.0         1.4         0.2  setosa  4.756844 0.05514933  4.140660  5.373027
3          4.7         3.2         1.3         0.2  setosa  4.773097 0.04690495  4.159587  5.386607
4          4.6         3.1         1.5         0.2  setosa  4.889357 0.05135928  4.274454  5.504259
5          5.0         3.6         1.4         0.2  setosa  5.054377 0.04736842  4.440727  5.668026
6          5.4         3.9         1.7         0.4  setosa  5.388886 0.05592364  4.772430  6.005342
```

# Example using glm

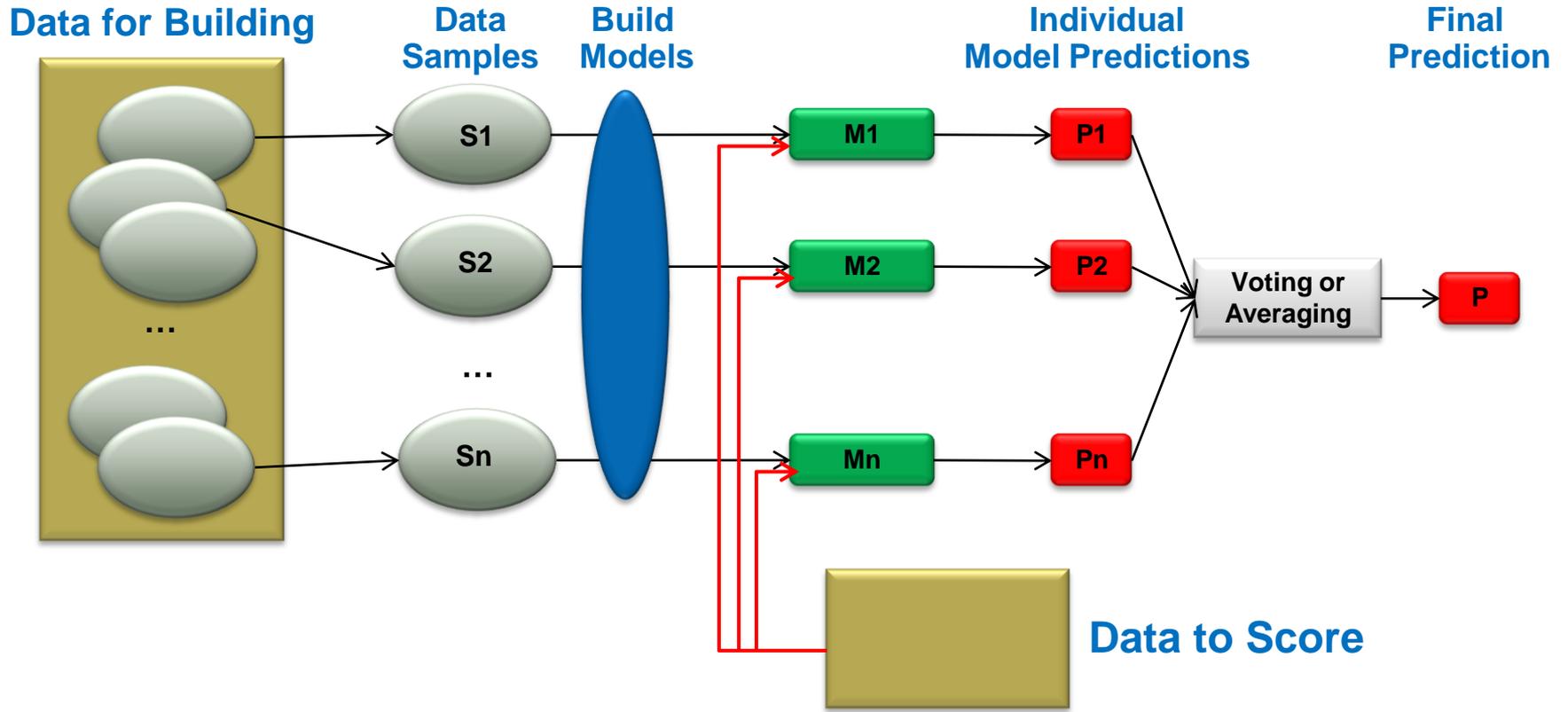
- Build an R glm model
- Use ore.predict to score data in Oracle Database using ore.frame, e.g., INFERT

```
R> infertModel <-  
+   glm(case ~ age + parity + education + spontaneous + induced,  
+       data = infert, family = binomial())  
R> INFERT <- ore.push(infert)  
R> INFERTpred <- ore.predict(infertModel, INFERT, type = "response",  
+                            se.fit = TRUE)  
R> INFERT <- cbind(INFERT, INFERTpred)  
R> head(INFERT)
```

	education	age	parity	induced	case	spontaneous	stratum	pooled.stratum	PRED	SE.PRED
1	0-5yrs	26	6	1	1	2	1	3	0.5721916	0.20630954
2	0-5yrs	42	1	1	1	0	2	1	0.7258539	0.17196245
3	0-5yrs	39	6	2	1	0	3	4	0.1194459	0.08617462
4	0-5yrs	34	4	2	1	0	4	2	0.3684102	0.17295285
5	6-11yrs	35	3	1	1	1	5	32	0.5104285	0.06944005
6	6-11yrs	36	4	2	1	1	6	36	0.6322269	0.10117919

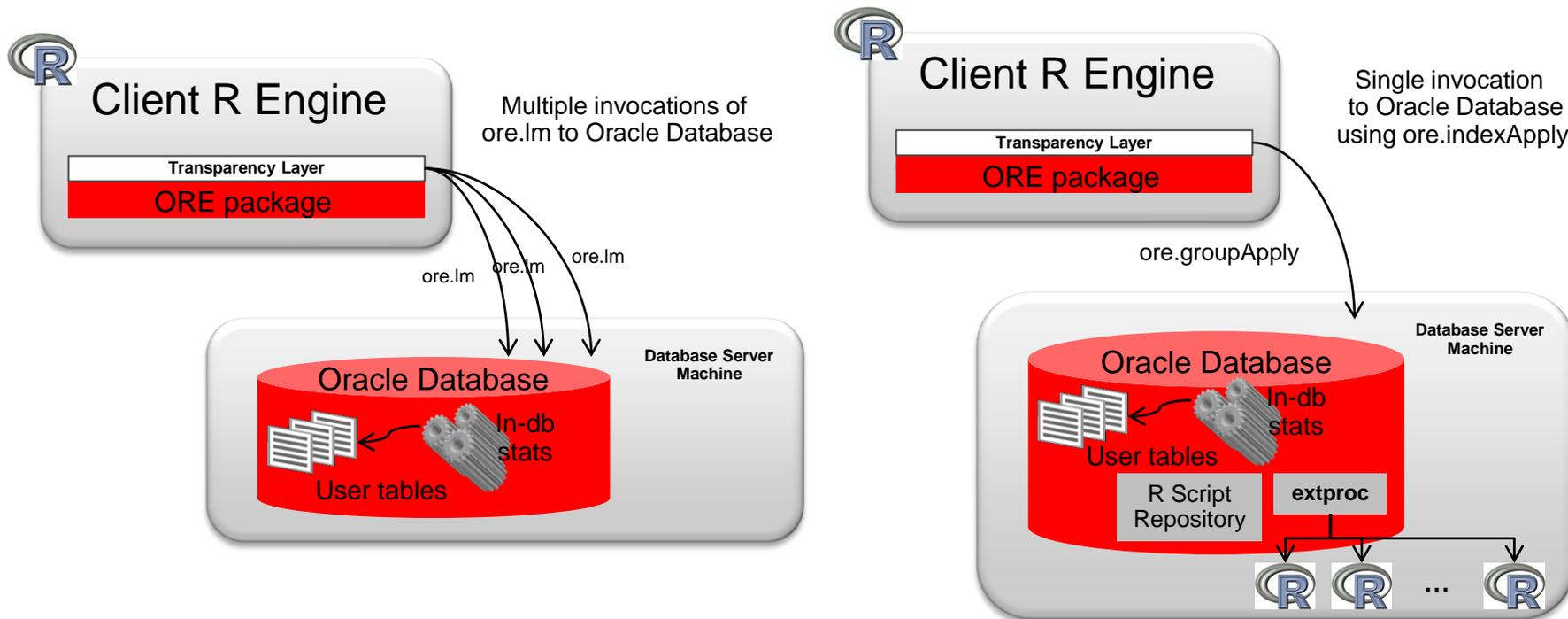
## Advanced Example with Bag of Little Bootstraps

# The “Bagging” Concept



# “Bagging” Execution Model

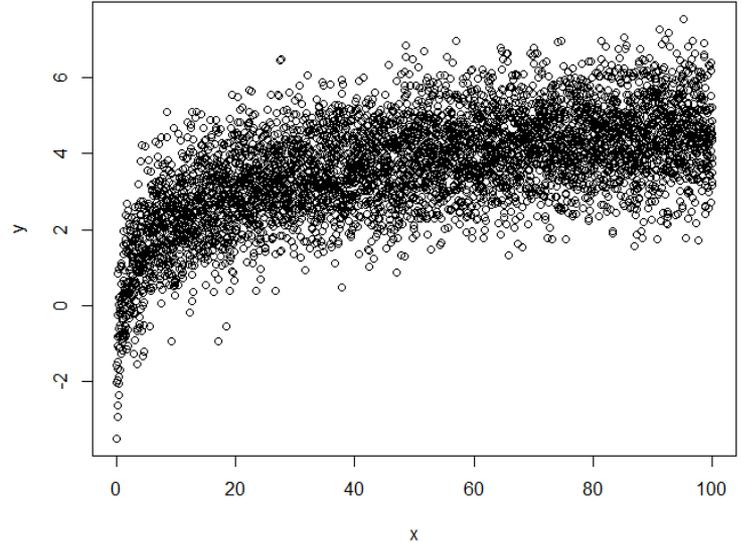
*Two options: client-controlled and database-controlled*



# Setting up the data

```
set.seed(2345)
x <- seq(0.1, 100, by = 0.02)
y <- log(x) + rnorm(x, sd = 1.0)
plot(x,y)

ID <- 1:length(x)
DAT <- ore.push(data.frame(ID=ID, X=x, Y=y))
myDAT <- DAT
row.names(myDAT) <- myDAT$ID
dim(myDAT)
```



# Function: bootstrapSample

```
bootstrapSample <- function (data, num.samples, size) {  
  num.rows <- nrow(data)  
  sample.indexes <- lapply(1:num.samples,  
                           function(x) sample(num.rows,  
                                               size, replace=TRUE))  
  
  create.sample <- function (sample.indexes, data) {  
    ore.push (data[sample.indexes,])  
  }  
  
  samples <- lapply(sample.indexes, create.sample, data)  
  samples  
}
```

# Sampling

```
sample(20, 5, replace=FALSE)
sample(20, 15, replace=TRUE)

inputData <- myDAT
num.samples <- 5
sample.size <- 500
samples.myDAT <- bootstrapSample (inputData,
                                   num.samples,
                                   sample.size)

lapply(samples.myDAT, head)
```

# Function: bagReg

```
bagReg <- function (data, formula, num.models, sample.size) {  
  samples <- bootstrapSample (data, num.samples = num.models,  
                              sample.size)  
  
  build.ore.lm <- function (data, formula, ...){  
    ore.lm(formula, data, ...)  
  }  
  
  models <- lapply(samples, build.ore.lm, formula)  
  models  
}
```

# Building a regression model

```
myMod <- ore.lm (Y~X, inputData)
myMod

myPred <- predict(myMod, inputData[1:10,])
myPred

formula_1 <- Y ~ X
sample.size <- 500
num.models <- 25
models <- bagReg (inputData, formula_1, num.models, sample.size)
models[[1]]
models[[25]]
```

# Function: predict.bagReg

```
predict.bagReg <- function (models, data, supp.cols) {  
  
  score.ore.lm <- function (model, data, supp.cols) {  
    res <- data.frame(data[,supp.cols])  
    res$PRED <- predict (model, data)  
    res  
  }  
  
  predictions <- lapply (models, score.ore.lm, data, supp.cols)  
  scores <- predictions[[1L]][,c(supp.cols)]  
  predValues <- lapply(predictions, function(y) y[, "PRED"])  
  scores$PRED_MIN <- do.call(pmin, predValues)  
  scores$PRED <- rowMeans(do.call(ore.frame, predValues))  
  scores$PRED_MAX <- do.call(pmax, predValues)  
  scores  
}
```

# Predict and evaluate with the bagged model

```
supp.cols <- c("ID", "Y")
scores <- predict.bagReg (models, inputData, supp.cols)
tail(scores, 30)

ore.rmse <- function (pred, obs) sqrt(mean(pred-obs)^2)

with(scores, ore.rmse (PRED, Y))

plot(x, y)
res <- lapply (models, abline, col="red")
```

# Advanced Example of ORE Workflow for Model Building and Scoring

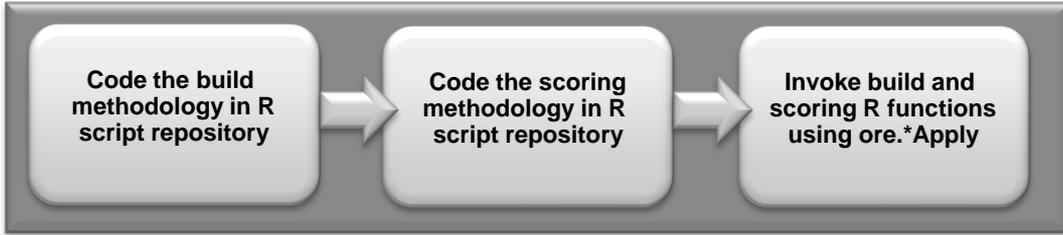
# Oracle R Enterprise as framework for Advanced Analytics

## Workflow example

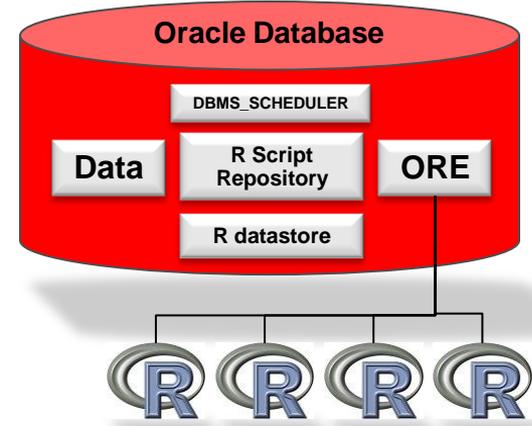
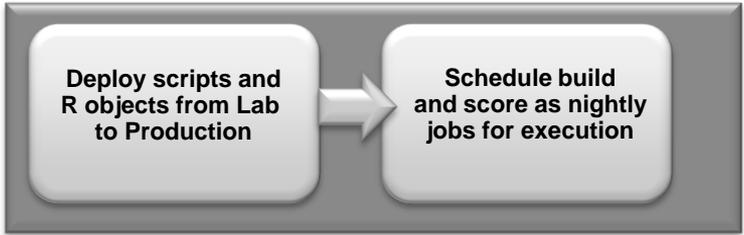
Analysis



Development



Production

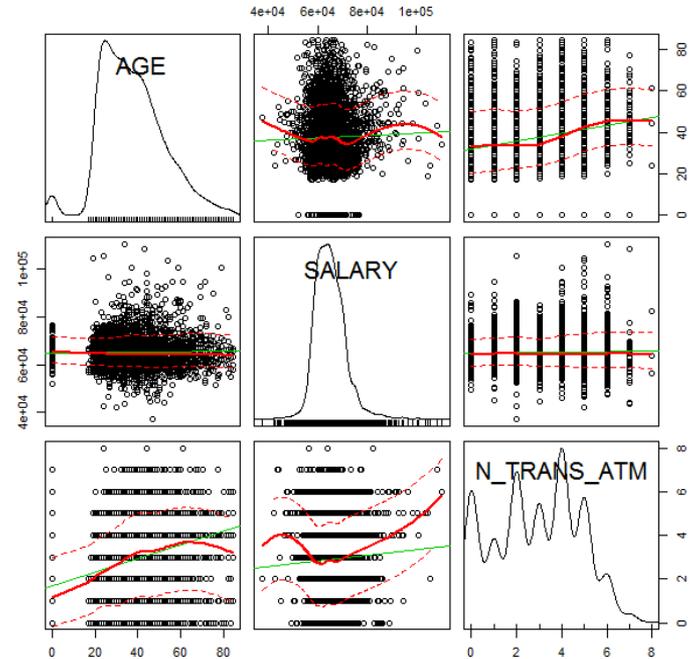


# Data exploration

```
library(car)
LTV <- CUSTOMER_LTV
row.names(LTV) <- LTV$CUST_ID
summary(LTV[,c("CUST_ID", "AGE", "SALARY",
              "MARITAL_STATUS", "N_TRANS_ATM", "LTV")])
ltv <- ore.pull(LTV)
ltv.sample <- ltv[sample(1:nrow(ltv), 4000), ]
scatterplotMatrix(~AGE+SALARY+N_TRANS_ATM,
                  data=ltv.sample)
```

```
> summary(LTV[,c("CUST_ID", "AGE", "SALARY", "MARITAL_STATUS",
+               "N_TRANS_ATM", "LTV")])
```

	CUST_ID	AGE	SALARY	MARITAL_STATUS	N_TRANS_ATM	LTV
CU1	: 1	Min. : 0.0	Min. : 37107	SINGLE :5359	Min. :0.00	Min. : 0
CU10	: 1	1st Qu.:27.0	1st Qu.: 60730	MARRIED :5284	1st Qu.:1.00	1st Qu.:18657
CU100	: 1	Median :36.0	Median : 64452	DIVORCED:3901	Median :3.00	Median :23054
CU1000	: 1	Mean :37.4	Mean : 65132	WIDOWED : 669	Mean :2.88	Mean :22267
CU10000	: 1	3rd Qu.:46.0	3rd Qu.: 68496	OTHER : 129	3rd Qu.:4.00	3rd Qu.:26104
CU10001	: 1	Max. :84.0	Max. :111605		Max. :8.00	Max. :47502
(Other)	:15336					



# 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 <- 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 <- CUSTOMER_LTV
row.names(LTV) <- LTV$CUST_ID

f1 <- function(trainData) glm(LTV ~ AGE + SALARY, data = trainData)
f2 <- function(trainData) glm(LTV ~ AGE + N_TRANS_ATM, data = trainData)
f3 <- function(trainData) lm(LTV ~ AGE + SALARY + N_TRANS_ATM, 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')
```

*This will fail, debug and determine why*

# Resources

<http://oracle.com/goto/R>

- **Book:** [Using R to Unlock the Value of Big Data](#), by Mark Hornick and Tom Plunkett
- **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®**