Session 2: OML4R 1.5.1 Transparency Layer
With Oracle Machine Learning

Mark Hornick, Senior Director
Oracle Machine Learning Product Management

November 2020
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, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Agenda

1. Introduction
2. Transparency Layer examples
3. Options for connecting to Oracle Database
4. R object persistence in Oracle Database
5. Support for Time Series data preparation
6. Ordering Framework
7. Global options
8. In-database sampling and random partitioning
9. Data types
What does “transparency” mean?
Transparency

The Transparency Layer supports in-database data exploration, data preparation, and data analysis en route to application of machine learning algorithms, where we have a mix of in-database and open source R techniques.

No need to learn a different programming paradigm or environment
Operate on database tables as though they were R objects using R syntax
Minimize change to base R scripts for database data
Implicitly translate R to SQL for in-database execution, performance, and scalability

Copyright © 2020 Oracle and/or its affiliates.
## OML4R Packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORE</td>
<td>Top Level Package for Oracle R Enterprise</td>
</tr>
<tr>
<td>OREbase</td>
<td>Corresponds to R's base package</td>
</tr>
<tr>
<td>OREstats</td>
<td>Corresponds to R's stat package</td>
</tr>
<tr>
<td>OREgraphics</td>
<td>Corresponds to R's graphics package</td>
</tr>
<tr>
<td>OREcommon</td>
<td>Common low-level functionality</td>
</tr>
<tr>
<td>OREDplyr</td>
<td>…must explicitly load package</td>
</tr>
<tr>
<td>OREeda</td>
<td>Exploratory data analysis package containing Base SAS PROC-equivalent function</td>
</tr>
<tr>
<td>OREembed</td>
<td>Embedded R Execution functionality</td>
</tr>
<tr>
<td>OREdm</td>
<td>Exposes Oracle Data Mining algorithms</td>
</tr>
<tr>
<td>OREmodels</td>
<td>ORE-provided advanced analytics algorithms</td>
</tr>
<tr>
<td>OREpredict</td>
<td>Enables scoring data in Oracle DB using R models</td>
</tr>
<tr>
<td>OREserver</td>
<td>Supports server-side functionality of OML4R</td>
</tr>
<tr>
<td>ORExml</td>
<td>Supports XML translation between R and Oracle Database</td>
</tr>
</tbody>
</table>
Documentation and Demos

OREShowDoc()

demo(package = "ORE")

demo("aggregate", package = "ORE")
## Demos in package 'ORE'

<table>
<thead>
<tr>
<th>aggregate</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
<td>Basic analysis &amp; data processing operations</td>
</tr>
<tr>
<td>basic</td>
<td>Basic connectivity to database</td>
</tr>
<tr>
<td>binning</td>
<td>Binning logic</td>
</tr>
<tr>
<td>columnfns</td>
<td>Column functions</td>
</tr>
<tr>
<td>cor</td>
<td>Correlation matrix</td>
</tr>
<tr>
<td>crosstab</td>
<td>Frequency cross tabulations</td>
</tr>
<tr>
<td>datastore</td>
<td>DataStore operations</td>
</tr>
<tr>
<td>datetime</td>
<td>Date/Time operations</td>
</tr>
<tr>
<td>derived</td>
<td>Handling of derived columns</td>
</tr>
<tr>
<td>distributions</td>
<td>Distribution, density, and quantile functions</td>
</tr>
<tr>
<td>do_eval</td>
<td>Embedded R processing</td>
</tr>
<tr>
<td>esm</td>
<td>Exponential smoothing method</td>
</tr>
<tr>
<td>freqanalysis</td>
<td>Frequency cross tabulations</td>
</tr>
<tr>
<td>glm</td>
<td>Generalized Linear Models</td>
</tr>
<tr>
<td>graphics</td>
<td>Demonstrates visual analysis</td>
</tr>
<tr>
<td>group_apply</td>
<td>Embedded R processing by group</td>
</tr>
<tr>
<td>hypothesis</td>
<td>Hypothesis testing functions</td>
</tr>
<tr>
<td>matrix</td>
<td>Matrix related operations</td>
</tr>
<tr>
<td>nulls</td>
<td>Handling of NULL in SQL vs. NA in R</td>
</tr>
<tr>
<td>odm_ai</td>
<td>Oracle Data Mining: attribute importance</td>
</tr>
<tr>
<td>odm_ar</td>
<td>Oracle Data Mining: association rules</td>
</tr>
<tr>
<td>odm_dt</td>
<td>Oracle Data Mining: decision trees</td>
</tr>
<tr>
<td>odm_glm</td>
<td>Oracle Data Mining: generalized linear models</td>
</tr>
<tr>
<td>odm_kmeans</td>
<td>Oracle Data Mining: enhanced k-means clustering</td>
</tr>
<tr>
<td>odm_nb</td>
<td>Oracle Data Mining: naive Bayes classification</td>
</tr>
<tr>
<td>odm_nmf</td>
<td>Oracle Data Mining: non-negative matrix factorization</td>
</tr>
<tr>
<td>odm_oc</td>
<td>Oracle Data Mining: o-cluster</td>
</tr>
<tr>
<td>odm_svm</td>
<td>Oracle Data Mining: support vector machines</td>
</tr>
<tr>
<td>ore_dplyr</td>
<td>Data manipulation similar to dplyr</td>
</tr>
<tr>
<td>pca</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>push_pull</td>
<td>RDBMS &lt;-&gt; R data transfer</td>
</tr>
<tr>
<td>randomForest</td>
<td>Random Forest classification algorithm</td>
</tr>
<tr>
<td>reg</td>
<td>Attributed-based ranking of observations</td>
</tr>
<tr>
<td>row_apply</td>
<td>Ordinary least squares linear regression</td>
</tr>
<tr>
<td>sampling</td>
<td>Embedded R processing by row chunks</td>
</tr>
<tr>
<td>script</td>
<td>Random row sampling and partitioning</td>
</tr>
<tr>
<td>sql_like</td>
<td>Mapping of R to SQL commands</td>
</tr>
<tr>
<td>stepwise</td>
<td>Stepwise OLS linear regression</td>
</tr>
<tr>
<td>summary</td>
<td>Summary functionality</td>
</tr>
<tr>
<td>table_apply</td>
<td>Embedded R processing of entire table</td>
</tr>
</tbody>
</table>
Proxy objects for Big Data

data.frame

Inherits from

Proxy ore.frame

Copyright © 2020 Oracle and/or its affiliates.
Manipulating Data

Column selection

```r
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

```r
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)
```
Manipulating Data – SQL equivalent

R

**Column selection**

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

**Row selection**

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

SQL

**Column selection**

```sql
create view df as
select YEAR, DEST, ARRDELAY
from ONTIME_S;
-- cannot do column selection by number & exclusion in SQL
```

**Row selection**

```sql
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'
```

**Benefits of OML4R transparency:**

- In-database execution
- Deferred execution
- Leverage column indexes, partitioning, query optimization, parallelism

Copyright © 2020 Oracle and/or its affiliates.
merge

Joining two tables (data frames)

```r
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")
```
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)
}

x <- ONTIME_S
attach(x)
x$DIVERTED <- diverted_fmt(DIVERTED)
x$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 <- 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)))

head(ONTIME)
Recoding data

Using `ore.recode()`

```r
> d <- ore.recode(ONTIME_S$DIVERTED, old=c(0,1),
                  new=c('No','Yes'))

summary(as.ore.factor(d))
```

```
No   Yes
219394 538
```
Connecting to Oracle Database

Working with ore.frame proxy objects
Establish a connection using `ore.connect`

```r
if (!ore.is.connected())
    ore.connect(user="rquser", sid="orcl",
    host="localhost", password="rquser",
    all=TRUE)
ore.ls()
```

`ore.is.connected` returns TRUE if you’re already connected to an Oracle Database

`ore.connect` parameters
- Port defaults to 1521
- “all” set to TRUE loads all tables from the schema into OML4R metadata and makes them available at the R command line as `ore.frame` objects

`ore.ls` lists all available tables by name

`ore.connect` does not support connecting to the database as `sys`
ore.connect / ore.disconnect

Establish connection to ORE-enabled database

• Must precede all other calls to OML4R functionality
• Only one OML4R connection can be active at a time
• Calling ore.connect during an active OML4R connection results in disconnecting the current connection before starting the new connection

An OML4R connection implicitly terminates when its R session ends, but can disconnect explicitly

Argument all, if TRUE, call functions 'ore.sync' and 'ore.attach' using their default arguments

Argument type: either ""ORACLE"" (default) or ""HIVE"" for OML4Spark users. If ""HIVE"", all other connection parameters ignored and obtained from the system environment

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

ore.connect("rquser", host="localhost",
            password="rquser", service_name="ORCL",
            all=TRUE)

ore.connect(conn_string = "<wallet_connect_string>")

ore.connect(user="rquser",password="rquser",
            conn_string = "sales-server:1521:sales")

ore.connect(user="rquser",password="rquser",
            conn_string = "(DESCRIPTION=
                          (ADDRESS=(PROTOCOL=tcp)(HOST=sales-server)(PORT=1521))
                          (CONNECT_DATA= (SERVICE_NAME=sales.us.acme.com)))")

ore.connect(user="rquser",password="rquser",
            conn_string = ") # connect to local DB

ore.disconnect()
```
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

- **Full Data**
  - 123M records
  - 22 years
  - 29 airlines

- **Sample Data**
  - ~220K records
  - ~10K / year

```r
> names(ONTIME_S)
[1] "YEAR"    "MONTH"  "MONTH2"  "DAYOFMONTH"
[5] "DAYOFMONTH2" "DAYOFWEEK" "DEPDEP" "CANCELLATIONCODE"
[9] "ARRDEP" "ARRCANCEL" "UNIQUECARRIER" "DEP_DELAY"
[13] "DEP_TIME" "DEP_DELAY" "CANCELLATIONCODE" "DEP_DELAY" "DEP_TIME"
[17] "CANCELLATIONCODE" "DIVERTED" "CANCELLATIONCODE"

> dim(ONTIME_S)
[1] 219332 26

> str(ONTIME_S)
'data.frame': 219332 obs. of 26 variables:
  Formal class 'data.frame' [package "OREbase"] with 12 slots
  
  ..$ .Data : list()
  ..$ dataObj : Named chr "+YEAR" VAL001,\"MONTH\" VAL002,\"MONTH2\", VAL003,\"DAYOFMONTH\", VAL
  ..$ .attr(, "name") = chr "17_7"
  ..$ dataObj : chr "17_7"

  ..$ desc : 'data.frame': 26 obs. of 2 variables:
      ...$ .name : chr "YEAR" "MONTH" "MONTH2" "DAYOFMONTH" ...
      ...$ .class: chr "numeric" "numeric" "factor" "numeric" ...
      ...$ .siValue : chr "+YEAR\", '"MONTH\", '"MONTH2\", '"DAYOFMONTH\", ... 
      ...$ .siTable : chr "+YEAR\", '"MONTH\", '"MONTH2\", '"DAYOFMONTH\", ... 
      ...$ .siPred : chr "+YEAR\", '"MONTH\", '"MONTH2\", '"DAYOFMONTH\", ...
      ...$ .siExt : list()
      ...$ .names : chr "17_7"
      ...$ .row.names : int
      ...$ .SSClass : chr "data.frame"
```
Examine the structure of the ore.frame object

- `str(ONTIME_S)`

```r
> str(ONTIME_S)
'data.frame': 218210 obs. of 26 variables:
  Formal class 'ore.frame' [package "OREbase"] with 12 slots
  ..@ .Data : list()
  ..@ dataqry : Named chr "{ select \"YEAR\" VAL001,\"MONTH\" VAL002,\"MONTH2\"
  VAL003,\"DAYOFMONTH\" VAL004,\"DAYOFMONTH2\" VAL005,\"DAYOFWEEK\" VAL006,"|__truncated__
  ...
  ..- attr(" , "names")= chr "107_8"
  ..@ dataobj : chr "107_8"
  ..@ 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 "{\"STUDENT12\"}\", "{\"ONTIME_S\"}
  ..@ sqlPred : chr ""
  ..@ extRef : list()
  ..@ names : chr
  ..@ row.names: int
  ..@ .S3Class : chr "data.frame"
```
ore.frame – Proxy object for database table

Examine slot “dataQry” of the ore.frame object

• ONTIME_S$dataQry

```R
> ONTIME_S$dataQry

107_8
"( select "YEAR" VAL001,"MONTH" VAL002,"MONTH2" VAL003,"DAYOFMONTH" VAL004,"DAYOFMONTH2"
VAL005,"DAYOFWEEK" VAL006,"DEPTIME" VAL007,"CRSDEPTIME" VAL008,"ARRTIME"
VAL022,"TAXIOUT" VAL023,"CANCELLED" VAL024,"CANCELLATIONCODE" VAL025,"DIVERTED" VAL026 from
"STUDENT12"."ONTIME_S" )"
```
ore.frame – Proxy object for database table

Examine slot “desc” of the ore.frame object

- ONTIME_S@desc

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Sclass</td>
</tr>
<tr>
<td>1</td>
<td>YEAR numeric</td>
</tr>
<tr>
<td>2</td>
<td>MONTH numeric</td>
</tr>
<tr>
<td>3</td>
<td>MONTH2 factor</td>
</tr>
<tr>
<td>4</td>
<td>DAYOFMONTH numeric</td>
</tr>
<tr>
<td>5</td>
<td>DAYOFMONTH2 factor</td>
</tr>
<tr>
<td>6</td>
<td>DAYOFWEEK numeric</td>
</tr>
<tr>
<td>7</td>
<td>DEPTIME numeric</td>
</tr>
<tr>
<td>8</td>
<td>CRSDEPTIME numeric</td>
</tr>
<tr>
<td>9</td>
<td>ARRTIME numeric</td>
</tr>
<tr>
<td>10</td>
<td>CRSARRTIME numeric</td>
</tr>
<tr>
<td>11</td>
<td>UNIQUECARRIER factor</td>
</tr>
<tr>
<td>12</td>
<td>FLIGHTNUM numeric</td>
</tr>
<tr>
<td>13</td>
<td>TAILNUM factor</td>
</tr>
<tr>
<td>14</td>
<td>ACTUALELAPSEDTIME numeric</td>
</tr>
<tr>
<td>15</td>
<td>CRSELAPSEDTIME numeric</td>
</tr>
<tr>
<td>16</td>
<td>AIRTIME numeric</td>
</tr>
<tr>
<td>17</td>
<td>ARRDELAY numeric</td>
</tr>
<tr>
<td>18</td>
<td>DEPDELAY numeric</td>
</tr>
<tr>
<td>19</td>
<td>ORIGIN factor</td>
</tr>
<tr>
<td>20</td>
<td>DEST factor</td>
</tr>
<tr>
<td>21</td>
<td>DISTANCE numeric</td>
</tr>
<tr>
<td>22</td>
<td>TAXIIN numeric</td>
</tr>
<tr>
<td>23</td>
<td>TAXIOUT numeric</td>
</tr>
<tr>
<td>24</td>
<td>CANCELLED numeric</td>
</tr>
<tr>
<td>25</td>
<td>CANCELLATIONCODE factor</td>
</tr>
<tr>
<td>26</td>
<td>DIVERTED factor</td>
</tr>
</tbody>
</table>
OML4R functions for interacting with database data

Synchronize OML4R proxy objects in R with tables/views available in database, on a per schema basis

```r
ore.sync()
ore.sync("RQUSER")
ore.sync(table=c("ONTIME_S", "NARROW"))
ore.sync("RQUSER", table=c("ONTIME_S", "NARROW"))
ore.sync(query = c("QUERY1" = "select 0 X, 1 Y from dual",
                    "QUERY2" = "select 1 X, 0 Y from dual"))
ore.ls()
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", "RQUSER")
```

Caveat for ore.sync:
Data types long, long raw, UDTs, and reference types not supported
When encountered, warning issued and table not available, e.g., via ore.ls()
OML4R functions for interacting with database data

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

List the objects available in OML4R 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

---

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

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

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

ore.detach("RQUSER")

ore.rm("ONTIME_S")
ore.exists("ONTIME_S", "RQUSER")
ore.sync()
ore.exists("ONTIME_S", "RQUSER")
ore.rm(c("ONTIME_S","NARROW"), "RQUSER")
ore.sync()
ore.attach()
```
Creating and dropping tables

- **Execute SQL or PL/SQL without return value**
  ```r
  ore.exec("create table F2 as select * from ONTIME_S")
  ```

- **Create a database table from a data.frame or ore.frame. Create a view from an ore.frame**
  ```r
  ore.create(ONTIME_S, table = "NEW_ONTIME_S")
  ore.create(ONTIME_S, view = "NEW_ONTIME_S_VIEW")
  ```

- **Drop table or view in database**
  ```r
  ore.drop(table="F2")
  ore.drop(table="NEW_ONTIME_S")
  ore.drop(view="NEW_ONTIME_S_VIEW")
  ```

- **Create a data.frame and then create a database table from it, then clean up**
  ```r
  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")
  ```

- **Load data (pull) from database**
  ```r
  ontime <- ore.pull(ONTIME_S)
  class(ONTIME_S)
  class(ontime)
  ```
OREbase package

as.ore*
ore.vector
ore.character
ore.factor
ore.frame
ore.matrix
Convert R type to OML4R type

as.ore.character
as.ore.numeric
as.ore.vector
as.ore.matrix
as.ore.frame
as.ore

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

dim(df)
class(df)

# Convert data frame to OML4R
ore.f <- as.ore(df)
class(ore.f)
dim(ore.f)
head(ore.f)
```
ore.vector functions

- show
- length
- c
- is.vector
- as.vector
- [ ]
- head
- tail
- l
- compare
  - ==, >, <, !=, <=, >=
- is.na
- cut
- %in%
- unique
- split
- sort
- rank
- order
- table
- paste
- interaction
- sapply
- tapply
- by
cut - binning

Divides the range of 'x' into intervals
Codes the values in 'x' according to which interval they fall
Leftmost interval corresponds to level one, the next leftmost to level two, etc.

```r
x <- ONTIME_S
x$ARRDELAY_BINNED = cut(x$ARRDELAY,
                     breaks=c(-1000,-100,-50,-10,0,10,50,100,1000))
class(x$ARRDELAY_BINNED)    # [1] "ore.factor"

# OR
x$ARRDELAY_BINNED = cut(x$ARRDELAY,
                     breaks=c(-1000,-100,-50,-10,0,10,50,100,1000),
                     labels=FALSE)
class(x$ARRDELAY_BINNED)    # [1] "ore.integer"

# EXPLICITLY convert this column to a factor if labels = FALSE

# Include x$DISTANCE_BINNED into an ore.glm, ore.lm,
# or ore.neural formula, it’s treated as categorical
x$DISTANCE_BINNED = cut(x$DISTANCE,
                     breaks=c(-1000,-100,-50,-10,0,10,50,100,1000))
fit = ore.glm(data=x,
             formula=CANCELLED ~ DISTANCE_BINNED,
             family=binomial(), trace=2)
```
split, sapply

split() divides the data in the vector \( x \) into the groups defined by the factor \( g \).

The result is a list with elements corresponding to the partitioned data.

```r
n <- 10; nn <- 100
x <- rnorm(n * nn) + sqrt(as.numeric(g))
X <- as.ore(x)
G <- as.ore(g)

XG <- split(X, G)

boxplot(XG, col = "lavender", notch = TRUE, varwidth = TRUE)

sapply(XG, length)
sapply(XG, mean)
```

```
R> sapply(XG, length)
  0  1  2  3  4  5  6  7  8  9 10
46 110  94 106  96 102  96 110  81 108  51
R> sapply(XG, mean)
  0   1   2   3   4   5   6   7   8   9  10
1.100634 1.438460 1.770479 2.011026 2.205655 2.461026 2.822435 2.756720 3.069607 3.244167 3.481981
```
split, sapply

split() divides the data in the vector x into the groups defined by the factor g. The result is a list with elements corresponding to the partitioned data.

sapply() invokes the function on the list and returns a vector or matrix of the same length.
sapply is a user-friendly version and wrapper of lapply by default returning a vector or matrix.

```r
dat <- ONTIME_S[ONTIME_S$ARRDELAY < 100 & ONTIME_S$ARRDELAY > -100,]
ad <- with(dat, split(ARRDELAY, UNIQUECARRIER))

boxplot(ad, col = "blue", notch = TRUE, cex=0.5, varwidth = TRUE)
sapply(ad, length)
sapply(ad, mean, na.rm=TRUE)
```
split, sapply results

```r
ORE> dat <- ONTIME_S[(ONTIME_S$ARR_DELAY < 100 &
    ONTIME_S$ARR_DELAY > -100,]
ORE> ad <- with(dat,sapply(ARR_DELAY, UNIQUECARRIER))
ORE> boxplot(ad, col = "blue", notch = TRUE, cex=0.5,
    varwidth = TRUE)
ORE> sapply(ad, length)
      9E  AA  AQ  AS  B6  CO  DH  DL  EA  EV  F9  FL  HA
         695 26494 226 4990 1129 14726 968 29327 2194 2192 477 1677 352
      HP  ML(1)  MQ  NW  OH  OO  PA(1)  PI  PS  TW  TZ  UA  US
         6608 131 5562 17917 1957 4112 710 2272 399 7217 284 23171 24818
      WN  XE  YV
         26734 3093 1075
ORE> sapply(ad, mean, na.rm=TRUE)
      9E  AA  AQ  AS  B6  CO  DH  DL  EA  EV  F9  FL  HA
-0.27625899 3.2267832 0.06637168 5.85931864 4.12754650 3.27930191
      HP  ML(1)  MQ  NW  OH  OO  PA(1)  PI  PS  TW  TZ  UA  US
0.21737521 5.46455935 4.11394713 4.58257299 1.92872117 5.18306500
      WN  XE  YV
-2.63636364 5.44718523 3.19083969 3.77202424 2.91309929 3.25958099
      UA  US  WN  XE  YV
2.06104086 3.35070423 8.14788732 11.06015038 4.62006374 0.77112676
      4.81908420 4.37001370 4.01911424 3.45328160 4.80093023
```
table

table() uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels
The result is an object of type “table”

```r
(t <- table(ONTIME_S$DAYOFWEEK))
class(t)
barplot(t)

table(ONTIME_S$DAYOFWEEK, ONTIME_S$Cancelled)

with (ONTIME_S,
    table(DAYOFWEEK,CANCELLED,DIVERTED))
```
table results

ORE> (t <- table(ONTIME_S$DAYOFWEEK))

           1  2  3  4  5  6  7
 32383 32254 32181 32165 32138 28335 30476
ORE> class(t)
[1] "table"
ORE> plot(t)
ORE>
ORE> table(ONTIME_S$DAYOFWEEK, ONTIME_S$CANCELLED)

          0  1
1 31783 600
2 31582 672
3 31579 602
4 31610 555
5 31603 535
6 27868 467
7 30047 429

ORE> with (ONTIME_S, table(DAYOFWEEK,CANCELLED,DIVERTED))

          0  1
1 31706 600
2 31514 672
3 31494 602
4 31519 555
5 31516 535
6 27807 467
7 29978 429

              0  1
CANCELLED
DAYOFWEEK
1    77     0
2    68     0
3    65     0
4    91     0
5    87     0
6    61     0
7    69     0
ore.character functions

nchar
tolower
toupper
casefold
chartr
sub
gsub
substr

```
x <- as.ore.character("MiXeD cAsE 123")
chartr("iXs", "why", x)
chartr("a-cX", "D-Fw", x)
```

```
ORE> x <- as.ore.character("MiXeD cAsE 123")
ORE>
ORE> chartr("iXs", "why", x)
[1] "MwheD cAyE 123"
ORE>
ORE> chartr("a-cX", "D-Fw", x)
[1] "MiweD FAsE 123"
```
ore.factor functions

levels
is.factor
as.factor
summary

```r
levels(ONTIME_S$CANCELLATIONCODE)
[1] "A" "B" "C" "D"

summary(ONTIME_S$CANCELLATIONCODE)
  A   B   C   D  NA's
  421 378 172   2 218959
```
ore.frame functions

- `show`
- `attach`
- `[`
- `$`
- `[[`
- `head`
- `tail`
- `length`
- `nrow`
- `ncol`
- `dim`
- `names`
- `colnames`
- `dimnames`
- `merge`
- `as.list`
- `unlist`
- `summary`
- `summary`
- `rbind`
- `cbind`
- `data.frame`
- `as.data.frame`
- `as.env`
- `eval`
- `subset`
- `with`
- `within`
- `transform`
- `arith`
  - Unary: `+`, `-`
  - Binary: `+`, `-`, `*`, `^`, `%%`, `%/%`, `/`
- `compare`
  - `==`, `>`, `<`, `!=`, `<=`, `>=`
- `!`
- `xor`
ore.frame functions

is.na
is.finite
is.nan
is.infinite
Math
   abs, sign, sqrt, ceiling, floor, trunc,
   cummax, cummin, cumprod, cumsum,
   exp, expm1, log, log10, log2, log1p, cos,
   cosh, sin, sinh, tan, tanh, acos, acosh,
   asin, asinh, atan, atanh, gamma,
   lgamma, digamma, trigamma

Summary
   max, min, range, prod, sum, any, all
   rowSums
   colSums
   rowMeans
   colMeans
   scale
   Interaction
   split
   unique
   by
   princomp

Copyright © 2020 Oracle and/or its affiliates.
Return subsets of vectors, matrices or data frames which meet conditions.

```r
ad <- ONTIME_S$ARRDELAY
ad <- subset(ad, ad < 200 & ad > -200)

hist(ad, breaks = 100)

add <- ONTIME_S[, c("ARRDELAY", "DEPDELAY")]
add <- subset(add, ARRDELAY < 100 & ARRDELAY > -100 & DEPDELAY < 100)

boxplot(add$ARRDELAY, add$DEPDELAY)
```
`scale` centers and/or scales the columns of a numeric `ore.frame`. Also referred to as “normalizing”.

```r
X <- ONTIME_S[,c("ARRDELAY", "DEPDELAY")]
centered.X <- scale(X, scale=FALSE)
head(centered.X)
scaled.X <- scale(X, scale=TRUE)
head(scaled.X)
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRDELAY</td>
<td>DEPDELAY</td>
</tr>
<tr>
<td>0</td>
<td>0.306837594</td>
</tr>
<tr>
<td>1</td>
<td>0.08837594</td>
</tr>
<tr>
<td>2</td>
<td>-0.08837594</td>
</tr>
<tr>
<td>3</td>
<td>0.34162406</td>
</tr>
<tr>
<td>4</td>
<td>-0.70837594</td>
</tr>
<tr>
<td>5</td>
<td>-0.08837594</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRDELAY</td>
<td>DEPDELAY</td>
</tr>
<tr>
<td>0</td>
<td>0.100734629</td>
</tr>
<tr>
<td>1</td>
<td>0.03486004</td>
</tr>
<tr>
<td>2</td>
<td>-0.001922746</td>
</tr>
<tr>
<td>3</td>
<td>0.063951844</td>
</tr>
</tbody>
</table>
princomp

Performs a principal component analysis on the given numeric ore.frame and returns the results as an object of class 'princomp'

```r
USA <- ore.push(USArrests)
princomp(USA, cor = TRUE)
```

```
R> USA <- ore.push(USArrests)
R> princomp(USA, cor = TRUE)
Call:
princomp(USA, cor = TRUE)

Standard deviations:
          Comp.1     Comp.2     Comp.3     Comp.4
[1,] 1.5748783 0.9948694 0.5971291 0.4164494

4 variables and 50 observations.
```
transform

Applies transformations to a data.frame / ore.frame

```r
X <- ONTIME_S[,c("ARRDELAY", "DEPDELAY")]
X <- transform(X,scale.ARRDELAY=scale(ARRDELAY),
               scale.DEPDELAY=scale(DEPDELAY))
head(X)
```
### ore.matrix functions

- **show**
- **is.matrix**
- **as.matrix**
- **nrow**
- **ncol**
- **dim**
- **rownames**
- **colnames**
- **dimnames**
- **t**
- **tabulate**

- **Arith**
  - Unary: `+`, `-`
  - Binary: `+`, `-`, `*`, `^`, `%%`, `%/%`, `/`
- **Math**
  - `abs`, `sign`, `sqrt`, `ceiling`, `floor`, `trunc`, `cummax`, `cummin`, `cumprod`, `cumsum`, `exp`, `expm1`, `log`, `log10`, `log2`, `log1p`, `cos`, `cosh`, `sin`, `sinh`, `tan`, `tanh`, `acos`, `acosh`, `asin`, `asinh`, `atan`, `atanh`, `gamma`, `lgamma`, `digamma`, `trigamma`
- **Summary**
  - `max`, `min`, `range`, `prod`, `sum`, `any`, `all`
  - `mean`

- **Bessel**
  - `bessel I`, `K`, `J`, `Y`
- `%*%`
- **crossprod**
- **tcrossprod**
- **solve**
- **backsolve**
- **forwardsolve**
Matrix multiplication `%*%`

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

```r
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,]  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> Y %*% Z
[,1] [,2] [,3]
[1,]  1  5  9
[2,]  4 12 20
[3,]  9 21 33
[4,] 16 32 48
```
solve

solves the equation $a \cdot x = b$ for $x$, where $b$ can be either a vector or a matrix.

```r
hilbert <- function(n) {
  i <- 1:n
  1 / outer(i - 1, i, "+")
}

h8 <- hilbert(8); h8
sh8 <- solve(h8)
round(sh8 %% h8, 3)

H8 <- ore.push(h8)
SH8 <- solve(H8)
round(SH8 %*% H8, 3)

# Same result...
```
# OREgraphics package functions

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrows</td>
</tr>
<tr>
<td>boxplot</td>
</tr>
<tr>
<td>boxplot.stats</td>
</tr>
<tr>
<td>cdplot</td>
</tr>
<tr>
<td>coplot</td>
</tr>
<tr>
<td>hist</td>
</tr>
<tr>
<td>identify</td>
</tr>
<tr>
<td>lines</td>
</tr>
<tr>
<td>matplot</td>
</tr>
<tr>
<td>pairs</td>
</tr>
<tr>
<td>plot</td>
</tr>
<tr>
<td>points</td>
</tr>
<tr>
<td>polygon</td>
</tr>
<tr>
<td>polypath</td>
</tr>
<tr>
<td>rug</td>
</tr>
<tr>
<td>segments</td>
</tr>
<tr>
<td>smoothScatter</td>
</tr>
<tr>
<td>sunflowerplot</td>
</tr>
<tr>
<td>symbols</td>
</tr>
<tr>
<td>text</td>
</tr>
<tr>
<td>xspline</td>
</tr>
<tr>
<td>xy.coords</td>
</tr>
</tbody>
</table>

Copyright © 2020 Oracle and/or its affiliates.
OREstat package function

IQR
aggregate
binom.test
chisq.test
cor
cov
fitdistr
ks.test
mad
median
model.frame
model.matrix

aa.omit
quantile
reorder
rnorm
sd
t.test
terms
var
var.test
wilcox.test

Copyright © 2020 Oracle and/or its affiliates.
Invoke in-database aggregation function

Source data is an ore.frame ONTIME_S, which resides in Oracle Database.
Overloaded aggregate() function accepts ORE frames.
aggregate() transparently switches between code that works with standard R data.frame and ore.frame objects.
Returns an ore.frame.

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

select DEST, count(*)
from ONTIME_S
group by DEST
ks.test – Kolmogorov-Smirnov test

Tests for the equality of continuous (numeric) vector probability distributions

Compares…

• a sample with a reference probability distribution (one-sample KS test)
• Two samples (two-sample KS test)

```r
x <- ore.push(rnorm(500))
y <- ore.push(runif(300))
# Do x and y come from the same distribution?
ks.test(x, x)
ks.test(x, y)
```

```r
x <- ONTIME_S$ARRDELAY
y <- ONTIME_S$DEPDELAY
# Do x and y come from the same distribution?
ks.test(x, y)
```
OREeda package functions

exploratory data analysis

ore.corr
ore.crosstab
ore.freq
ore.lm
ore.rank
ore.sort
ore.summary
ore.univariate
Solve problems 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

- Outliers
- 1.5 IQR
- 3rd Quartile
- Median
- 1st Quartile
- 1.5 IQR

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

Run one line at a time and view the result

```r
ontime <- ONTIME_S
aggrdata <- aggregate(ontime$DEST, by = list(ontime$DEST), FUN = length)
minx <- min(head(sort(aggrdata$x, decreasing = TRUE), 36))
busiest_airports <- aggrdata$Group.1[aggrdata$x >= minx, drop = TRUE]
delay <- ontime$ARRDELAY[ontime$DEST %in% busiest_airports & ontime$YEAR == 2007]
dest <- ontime$DEST[ontime$DEST %in% busiest_airports & ontime$YEAR == 2007, drop = TRUE]
dest <- reorder(dest, delay, FUN = median, na.rm = TRUE)
bdd <- split(delay, dest)
boxplot(bdd, notch = TRUE, col = "gold", cex = 0.5,
        outline = FALSE, horizontal = TRUE, yaxt = "n",
        main = "2007 Flight Delays by Airport -- top 36 busiest",
        ylab = "Delay (minutes)", xlab = "Airport")
labels <- levels(dest)
text(par("usr")[1] - 3, 1:length(labels), srt = 0, adj = 1,
      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?

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

```r
ontime <- ONTIME_S
delay <- ontime$ARRDELAY
dayofweek <- ontime$DAYOFWEEK
bd <- split(delay, dayofweek)
boxplot(bd, notch = TRUE, col = "red", cex = 0.5,
       outline = FALSE, axes = FALSE,
       main = "Airline Flight Delay by Day of Week",
       ylab = "Delay (minutes)", xlab = "Day of Week")
axis(1, at=1:7, labels=c("Monday", "Tuesday", "Wednesday", "Thursday",
                         "Friday", "Saturday", "Sunday"))
axis(2)
```
Are select airlines getting better or worse?

Mean annual delay by Year

Copyright © 2020 Oracle and/or its affiliates.
Are select airlines getting better or worse?

Mean annual delay by Year

```r
ontimeSubset <- subset(ONTIME_S, UNIQUECARRIER %in% c("AA", "AS", "CO", "DL", "WN", "NW"))
res22 <- with(ontimeSubset, tapply(ARRDELAY, list(UNIQUECARRIER, YEAR), mean, na.rm = TRUE))
g_range <- range(0, res22, na.rm = TRUE)
riindex <- seq_len(nrow(res22))
cindex <- seq_len(ncol(res22))
par(mfrow = c(2,3))
for(i in riindex) {
  temp <- data.frame(index = cindex, avg_delay = res22[i,])
  plot(avg_delay ~ index, data = temp, col = "black",
       axes = FALSE, ylim = g_range, xlab = "", ylab = "",
       main = attr(res22, "dimnames")[[1]][i])
  axis(1, at = cindex, labels = attr(res22, "dimnames")[[2]])
  axis(2, at = 0:ceiling(g_range[2]))
  abline(lm(avg_delay ~ index, data = temp), col = "green")
  lines(lowess(temp$index, temp$avg_delay), col="red")
}
```
R Object Persistence in Oracle Database
R Object Persistence

What does R provide?

save()
load()

Serialize and unserialize R objects to files
Standard R functions to persist R objects do not interoperate with Oracle Database

Use cases include
- Persist models for subsequent data scoring
- Save entire R state to reload next R session

```r
x1 <- lm(...)  
x2 <- data.frame(...)  
save(x1,x2,file="myRObjects.RData")
```

```
load("myRObjects.RData")  
ls()  
"x1" "x2"
```
R Object Persistence with OML4R

ore.save()
ore.load()

Provide database storage to save/restore R and OML4R 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 OML4R objects across R sessions

x1 <- ore.lm(...)  # Function call
x2 <- ore.frame(...)  # Function call
ore.save(x1,x2,name="ds1")  # Save
ore.load(name="ds1")  # Load
ls()  # List variables
"x1"  "x2"  # Names of variables

R Datastore
ds1 {x1,x2}
Datastore Details

Each schema has its own datastore table where R objects are saved as named datastores

Maintain referential integrity of saved objects
- Account for objects auto-deleted at end of session
- Database objects, such as tables, ODM models, etc., not used by any saved R object is deleted when R session ends

Functions
- ore.save, ore.load
- ore.datastore, ore.datastoreSummary
- ore.delete
- ore.grant, ore.revoke
ore.save

```r
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`.

ore.save arguments

- `...` — the R variables of the objects to be saved
- `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
- `grantable` — a logical value, if TRUE, read access to datastore can be granted to others
- `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
- `envAsEmptyenv` — a logical value indicating whether referenced environments in R objects to be saved should be replaced with an empty environment during serialization
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
  • 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
  • type – An optional scalar character string specifying the type of datastore to list: 'user'
    (default), 'private', 'all', 'grantable', 'grant', or 'granted'
ore.datastore example

```r
R> ore.datastore()

<table>
<thead>
<tr>
<th>datastore name</th>
<th>object.count</th>
<th>size</th>
<th>creation.date</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myDatastore</td>
<td>1</td>
<td>64461835</td>
<td>2012-11-14 17:12:36</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>myIrisData</td>
<td>1</td>
<td>5789</td>
<td>2012-11-14 19:07:18</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>myModels</td>
<td>3</td>
<td>45782</td>
<td>2012-11-14 18:56:32</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>

R> ore.datastore(pattern="*Mod*")

<table>
<thead>
<tr>
<th>datastore name</th>
<th>object.count</th>
<th>size</th>
<th>creation.date</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myModels</td>
<td>3</td>
<td>45782</td>
<td>2012-11-14 18:56:32</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>
```
ore.datastoreSummary

```r
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
- owner — optional character string specifying the owner of datastore to summarize
```r
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       5
```
ore.delete

ore.delete(name = "myModels", list = character(0))

Deletes named datastore in connected schema and its corresponding objects
If objects saved in other datastores referenced the same objects, referenced objects are only cleaned up when there are no more references

Argument
• name — name of datastore under current user schema from which to return data
• list – optional character vector containing names of objects to delete from datastore. If not specified, entire datastore is deleted

Copyright © 2020 Oracle and/or its affiliates.
ore.save(iris, name="ds_1", grantable=TRUE)         # create grantable datastores
ore.save(mtcars, name="ds_2", grantable=TRUE)

ore.grant(name="ds_1", type="datastore", user=NULL) # grant read to all users

ore.datastore(type="all")[,-5]              # show all datastores
ore.datastore(type="grantable")[,-4]       # show grantable datastores
ore.datastore(type="grant")                 # show datastores where read granted

ore.revoke(name="ds_1", type="datastore", user=NULL)  # revoke grant

ore.delete(name="ds_1")  # clean up
ore.delete(name="ds_2")

ore.grant and ore.revoke

Arguments

• name – string name of R datastore
• type – string “datastore” or “rqscript” namespace within which to grant/revoke the read privilege
• user – optional string indicating user being granted/revoked read privilege. Default of NULL grants to all users, i.e., PUBLIC

```r
# create grantable datastores
ore.save(iris, name="ds_1", grantable=TRUE)
ore.save(mtcars, name="ds_2", grantable=TRUE)

# grant read to all users
ore.grant(name="ds_1", type="datastore", user=NULL) 

# show all datastores
ore.datastore(type="all")

# show grantable datastores
ore.datastore(type="grantable")

# show datastores where read granted
ore.datastore(type="grant")

# revoke grant
ore.revoke(name="ds_1", type="datastore", user=NULL)

# clean up
ore.delete(name="ds_1")
ore.delete(name="ds_2")
```
Corresponding SQL API

List available datastores
   select * from rquser_DataStoreList

View contents of a given datastore
   select * from rquser_DataStoreContents
       where dsname = 'ds_name';

Delete a datastore
   rqDropDataStore(''<ds_name>''
Using datastore objects in embedded R SQL API

begin
-- sys.rqScriptDrop('buildmodel_1');
sys.rqScriptCreate('buildmodel_1',
    'function(dat, out.dsname, out.objname) {
        assign(out.objname, lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat, model = FALSE))
        ore.save(list=out.objname, name = out.dsname, overwrite=TRUE)
        cbind(dsname=out.dsname, ore.datastoreSummary(name= out.dsname))
    }');
end;
/

-- build model
select * from table(rqTableEval(
    cursor(select ARRDELAY, DISTANCE, DEPDELAY from ONTIME_S),
    cursor(select 'ontime_model' as "out.dsname",'lm.mod' as "out.objname",
        1 as "ore.connect" from dual),
    'select * from rquser_datastoreContents',
    'buildmodel_1'));

Copyright © 2020 Oracle and/or its affiliates.
Data Preparation support for Time Series Analytics
Time Series Analysis

Motivation

Time series data is widely prevalent
- Stock / trading data
- Sales data
- Employment data

Need to understand trends, seasonable effects, residuals
Time Series Analysis

Aggregation and moving window analysis of large time series data

Equivalent functionality from popular R packages for data preparation available in-database
Support for Time Series Data

Oracle data types
• DATE, TIMESTAMP
• TIMESTAMP WITH TIME ZONE
• TIMESTAMP WITH LOCAL TIME ZONE

Analytic capabilities
• Date arithmetic, Aggregations & Percentiles
• Moving window calculations:
  ore.rollmax ore.rollmean ore.rollmin ore.rolls
  ore.rollsum ore.rollvar
Date and Time

Motivation

Support for key data types in Oracle Database and R

Date and time handling essential for time series data

Date and time representation unified in Oracle Database, but R lacks a standard structure and functions

- E.g., Date, POSIXct, POSIXlt, difftime in R base package
- Mapping data types important for transparent database access
# Mapping Oracle Date and Time Data Types to R

<table>
<thead>
<tr>
<th>Oracle SQL Data Type</th>
<th>OML4R Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>ore.datetime</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>ore.datetime</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>ore.datetime</td>
</tr>
<tr>
<td>TIMESTAMP WITH LOCAL TIME ZONE</td>
<td>ore.datetime</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>ore.character</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>ore.difftime</td>
</tr>
</tbody>
</table>

Copyright © 2020 Oracle and/or its affiliates.
Date and Time Transparency Layer functions

Binary operations
- Arithmetic (+, -, *, /)
- Comparison (==, <, >, !, <=, >=)

Row functions
- Component extraction (year, month, day, etc.)
- General operations (is.na, %in%, etc.)
- Number-like operations (round, trunc, etc.)

Vector operations
- Subsetting ("[", head, tail)
- Distinct values (unique)

Aggregates
- Date-time quantiles (min, max, median, quantile)
- Tabulations (table)

Set operations
- Row filtering by date-time comparisons)
- Row splitting/grouping by date-time (split)
- Joining by date-time (merge)

Group by analysis
- Univariate fixed group aggregations by date-time characteristics (aggregate, tapply, by, etc.)

Moving window aggregation
- Univariate moving window aggregations of ordered data (ordering may or may not be date-time related)
Date and Time aggregates

```r
N <- 500
mydata <- data.frame(
  datetime = seq(as.POSIXct("2001/01/01"),
                 as.POSIXct("2001/12/31"),
                 length.out = N),
  difftime = as.difftime(runif(N),
                         units = "mins"),
  x = rnorm(N))
MYDATA <- ore.push(mydata)
class(MYDATA)
class(MYDATA$datetime)
head(MYDATA,3)
```
Date and Time aggregates

```r
## statistic aggregates
min(MYDATA$datetime)
max(MYDATA$datetime)
range(MYDATA$datetime)
median(MYDATA$datetime)
quantile(MYDATA$datetime, probs = c(0, 0.05, 0.10))
```

```r
## Order statistic aggregates
R> min(MYDATA$datetime)
[1] "2001-01-01 EST"
R> max(MYDATA$datetime)
[1] "2001-12-31 EST"
R> range(MYDATA$datetime)
[1] "2001-01-01 EST" "2001-12-31 EST"
R> median(MYDATA$datetime)
[1] "2001-07-02 01:00:00 EDT"
R> quantile(MYDATA$datetime, 
  +     probs = c(0, 0.05, 0.10))
  0%      5%     10%
"2001-01-01 00:00:00 EST" "2001-01-19 04:48:00 EST" "2001-02-06 09:36:00 EST"
```
Date and Time arithmetic

```r
## Arithmetic

day1Shift <- MYDATA$datetime +
    as.difftime(1, units = "days")
class(day1Shift)
head(MYDATA$datetime,3)
head(day1Shift,3)

lag1Diff <- diff(MYDATA$datetime)
class(lag1Diff)
head(lag1Diff,3)
```

```
R> ## Arithmetic
R> day1Shift <- MYDATA$datetime + as.difftime(1, units = "days")
R> class(day1Shift)
[1] "ord.datetime"
attr("package")
[1] "OREbase"
R> head(day1Shift,3)
[1] "2001-01-02 00:00:00 EST" "2001-01-02 17:30:25 EST" 
[3] "2001-01-03 11:00:50 EST"
R>
R> lag1Diff <- diff(MYDATA$datetime)
R> class(lag1Diff)
[1] "ord.difftime"
attr("package")
[1] "OREbase"
R> head(lag1Diff,3)
Time differences in secs
[1] 63025.25 63025.25 63025.25
```
**Date and Time comparisons**

```r
isQ1 <- MYDATA$datetime < as.Date("2001/04/01")
class(isQ1)
head(isQ1,3)

isMarch <- isQ1 & MYDATA$datetime > as.Date("2001/03/01")
class(isMarch)
head(isMarch,3)
sum(isMarch)

eoySubset <- MYDATA[MYDATA$datetime > as.Date("2001/12/27")]
class(eoySubset)
head(eoySubset,3)
```

---

R> isQ1 <- MYDATA$datetime < as.Date("2001/04/01")
R> class(isQ1)
[1] "ore.logical"
attr(,"package")
[1] "OREbase"
R> head(isQ1,3)
[1] TRUE TRUE TRUE
R>
R> isMarch <- isQ1 & MYDATA$datetime > as.Date("2001/03/01")
R> class(isMarch)
[1] "ore.logical"
attr(,"package")
[1] "OREbase"
R> head(isMarch,3)
[1] FALSE FALSE FALSE
R> sum(isMarch)
[1] 43
R>
R> eoySubset <- MYDATA[MYDATA$datetime > as.Date("2001/12/27")]
R> class(eoySubset)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> head(eoySubset,3)

<table>
<thead>
<tr>
<th>datetime</th>
<th>difftime</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-12-27</td>
<td>08:27:53</td>
<td>5.909428 secs 1.148956</td>
</tr>
<tr>
<td>2001-12-28</td>
<td>01:58:18</td>
<td>36.967094 secs -1.631579</td>
</tr>
<tr>
<td>2001-12-28</td>
<td>19:28:44</td>
<td>29.381100 secs 1.317894</td>
</tr>
</tbody>
</table>
```
## Date/time accessors

```r
## Date/time accessors
year <- ore.year(MYDATA$datetime)
unique(year)

month <- ore.month(MYDATA$datetime)
range(month)

dayOfMonth <- ore.mday(MYDATA$datetime)
range(dayOfMonth)

hour <- ore.hour(MYDATA$datetime)
range(hour)

minute <- ore.minute(MYDATA$datetime)
range(minute)

second <- ore.second(MYDATA$datetime)
range(second)
```

```r
grep("2001", unique(year))
## [1] 2001

grep("1 12", range(month))
## [1] 1 12

grep("1 31", range(dayOfMonth))
## [1] 1 31

grep("0 23", range(hour))
## [1] 0 23

grep("0 59", range(minute))
## [1] 0 59

grep("0.00000 59.87976", range(second))
## [1] 0.00000 59.87976
```
Date and Time coercion

dateOnly <- as.ore.date(MYDATA$datetime)
class(dateOnly)
head(sort(unique(dateOnly)),3)

nameOfDay <- as.ore.character(MYDATA$datetime, 
format = "DAY")
class(nameOfDay)
sort(unique(nameOfDay))

dayOfYear <-
as.integer(as.character(MYDATA$datetime, 
format = "DDD"))
class(dayOfYear)
range(dayOfYear)

quarter <-
as.integer(as.character(MYDATA$datetime, 
format = "Q"))
class(quarter)
sort(unique(quarter))

R> dateOnly <- as.ore.date(MYDATA$datetime)
R> class(dateOnly)
[1] "ore.date"
attr("package")
[1] "OREbase"
R> head(sort(unique(dateOnly)),3)
[1] "2001-01-01" "2001-01-02" "2001-01-03"
R>
R> nameOfDay <- as.ore.character(MYDATA$datetime, format = "DAY")
R> class(nameOfDay)
[1] "ore.character"
attr("package")
[1] "OREbase"
R> sort(unique(nameOfDay))
[1] "FRIDAY " "MONDAY " "SATURDAY " "SUNDAY " "THURSDAY " "TUESDAY " 
[7] "WEDNESDAY"
R>
R> dayOfYear <- as.integer(as.character(MYDATA$datetime, format = "DDD"))
R> class(dayOfYear)
[1] "ore.integer"
attr("package")
[1] "OREbase"
R> range(dayOfYear)
[1] 1 365
R>
R> quarter <- as.integer(as.character(MYDATA$datetime, format = "Q"))
R> class(quarter)
[1] "ore.integer"
attr("package")
[1] "OREbase"
R> sort(unique(quarter))
[1] 1 2 3 4

Copyright © 2020 Oracle and/or its affiliates.
Arima example with rolling mean window function

```r
row.names(MYDATA) <- MYDATA$datetime
MYDATA$rollmean5 <- ore.rollmean(MYDATA$x, k = 5)
MYDATA$rollsd5  <- ore.rolls (MYDATA$x, k = 5)
head(MYDATA)

marchData <- ore.pull(MYDATA[isMarch,])
tseries.x <- ts(marchData$x)
arima110.x <- arima(tseries.x, c(1,1,0))
predict(arima110.x, 3)

tseries.rm5 <- ts(marchData$rollmean5)
arima110.rm5 <- arima(tseries.rm5, c(1,1,0))
predict(arima110.rm5, 3)
```

```r
R> row.names(MYDATA) <- MYDATA$datetime
R> MYDATA$rollmean5 <- ore.rollmean(MYDATA$x, k = 5)
R> MYDATA$rollsd5  <- ore.rolls (MYDATA$x, k = 5)
R> head(MYDATA, 3)
datetime  difftime       x rollmean5  rollsd5
2001-01-01  00:00:00  00:00:00     18,10307 -0.7592010  0.1176402  0.8656136
2001-01-01  17:30:25  2001-01-01  17:30:25     16,68061  0.9715714  0.1310861  0.7072820
2001-01-02  11:00:50  2001-01-02  11:00:50     37,77584 -0.1405502 -0.1508768  0.8790343
R>
R> marchData <- ore.pull(MYDATA[isMarch,])
R> tseries.x <- ts(marchData$x)
R> arima110.x <- arima(tseries.x, c(1,1,0))
R> predict(arima110.x, 3)

$pred
Time Series:
Start = 44
End = 46
Frequency = 1
[1] -0.7801517 -0.8719835 -0.8383381

sse
Time Series:
Start = 44
End = 46
Frequency = 1
[1] 0.4557869 0.4557631 0.4554896

R> tseries.rm5 <- ts(marchData$rollmean5)
R> arima110.rm5 <- arima(tseries.rm5, c(1,1,0))
R> predict(arima110.rm5, 3)

$pred
Time Series:
Start = 44
End = 46
Frequency = 1
[1] 1.2239001 1.4589741 1.738810

sse
Time Series:
Start = 44
End = 46
Frequency = 1
[1] 0.2923064 0.4718523 0.6129770
```
Ordering Framework
Data Ordering
Contrasting R and Database behavior

R
R’s in-memory nature has a well-defined, implicit ordering of elements in vectors or objects based on vectors, e.g., data frames
R supports integer indexing by default, e.g., df[1:4,]
Notion of “unordered” data doesn’t really exist in R

DB
RDBMS data, e.g., tables and views, do not define an implicit ordering
Enabling ordering involves having a primary key for tables and views
Explicit ordering of data possible via ORDER BY clause, provided a unique ordering is possible, e.g., via single or multi-column key, but can impose a performance penalty
Features of Ordering Framework

Ordering enables integer and character indexing on ore.frames

Distinguish between functions that require ordering and those that do not

- Throw error if unordered data types provided to functions requiring ordering
- Provide alternative semantics where possible if functions would normally require ordering and generate a warning only
  - e.g., head() and tail() can return sample of n rows instead of the top or bottom rows
- Ability to turn off ordering warnings

Use `row.names` and `row.names<-` functions for ordered frames

- Enables specifying unique identifier on an unordered ore.frame to make it ordered
- Convert between ordered and unordered types by setting/clearing row.names
- Can be comprised of multiple values, supporting multi-column keys

For tables with unique constraints, option to create ordered frames during ore.sync()
Ordered and unordered ore.frame objects

An ore.frame is *ordered* if...

- A primary key is defined on the underlying table
- It is produced by certain functions, e.g., “aggregate” and “cbind”
- The row names of the ore.frame are set to unique values
- All input ore.frames to relevant OML4R functions are ordered

An ore.frame is *unordered* if...

- No primary key is defined on the underlying table
- Even with a primary key is specified, ore.sync parameter use.keys is set to FALSE
- No row names are specified for the ore.frame
- Row names have been set to NULL
- One or more input ore.frames to relevant OML4R functions are unordered
# R
library(kernlab)
data(spam)
s <- spam
s$TS <- as.integer(1:nrow(s)+1000)
s$USERID <- rep(1:50+350, each=2, len=nrow(s))
ore.drop(table='SPAM_PK')
ore.drop(table='SPAM_NOPK')
ore.create(s[,c(59:60,1:28)], table='SPAM_PK')
ore.create(s[,c(59:60,1:28)], table='SPAM_NOPK')

ore.exec('alter table SPAM_PK
add constraint SPAM_PK primary key ("USERID","TS")')

# R
head(SPAM_PK[,1:8])
head(SPAM_NOPK[,1:8])
Using Keys

ore.sync(use.keys = FALSE)

head(SPAM_PK[,1:4],3)
head(SPAM_NOPK[,1:4],3)

ore.sync() # use.keys TRUE default

head(SPAM_PK[,1:4],3)
head(SPAM_NOPK[,1:4],3)

is.null(row.names(SPAM_PK))

Load proxy objects as unordered ore.frames

Notice that row names are sequential numbers by default

Warning issued

Same for SPAM_NOPK table

Load proxy objects as ordered ore.frames if primary key specified for table

Notice that row names consist of TS and USERID values separated by ‘|’ character forming the row name

Notice also that since SPAM_NOPK does not have a primary key specified, it is still an unordered frame
Using row.names

```r
a <-
ore.push(data.frame(a=c(1:10, 10:1), b=letters[c(1:10, 10:1)]))
a$b
row.names(head(a))
row.names(head(SPAM_NOPK))
row.names(head(SPAM_PK))
row.names(SPAM_PK) <- SPAM_PK$TS
row.names(head(SPAM_PK[,1:4]))
head(SPAM_PK[,1:4])
```

ore.frame created as ordered by default
note: no warnings

Default row names

SPAM_NOPK has no unique key, so row.names raises error

Row names consist of TS ‘|’ USERID

Reassign row names with TS only

Row names now correspond to TS value only
### Indexing `data.frame`

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SPAM_PK[&quot;2060&quot;, 1:4]</code></td>
<td>Index to a specifically named row</td>
</tr>
<tr>
<td><code>SPAM_PK[as.character(2060:2064), 1:4]</code></td>
<td>Index to a range of rows by row names</td>
</tr>
<tr>
<td><code>SPAM_PK[2060:2062, 1:4]</code></td>
<td>Index to a range of rows by integer index</td>
</tr>
</tbody>
</table>
### Merge Example

```r
x <- SPAM_NOPK[,1:4]  
y <- SPAM_NOPK[,c(1,2,4,5)]  

m1 <- merge(x, y, by="USERID")  
head(m1,3)
```

Set up data for illustrating merge

```r
x <- SPAM_PK[,1:4]  
y <- SPAM_PK[,c(1,2,4,5)]  

m1 <- merge(x, y, by="USERID")  
head(m1,3)
```

Merged result completes with warning since not an ordered frame

```
1: ORE object has no unique key - using random order  
2: ORE object has no unique key - using random order  
```

Warning messages:

Set up data for illustrating merge

```
Warning messages:  
1: ORE object has no unique key - using random order  
2: ORE object has no unique key - using random order  
```

Merged result completes with warning since not an ordered frame

```
1: ORE object has no unique key - using random order  
2: ORE object has no unique key - using random order  
```

Merged result with no warning since ordered frame

Notice that row names are concatenation of row names from x and y

Copyright © 2020 Oracle and/or its affiliates.
Ordering Framework Options

```r
options("ore.warn.order")
options("ore.warn.order" = TRUE)
options("ore.warn.order" = FALSE)

options("ore.sep")
options("ore.sep" = "/")
options("ore.sep" = "|")

row.names(NARROW) <- NARROW[,c("ID", "AGE")]
ore.pull(head(NARROW), sep = "+")
```

```r
c> row.names(NARROW) <- NARROW[,c("ID", "AGE")]
c> ore.pull(head(NARROW), sep = "+")

<table>
<thead>
<tr>
<th>ID</th>
<th>GENDER</th>
<th>AGE</th>
<th>MARITAL_STATUS</th>
<th>COUNTRY</th>
<th>EDUCATION</th>
<th>OCCUPATION</th>
<th>YRS_RESIDENCE</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>101501+41</td>
<td>101501</td>
<td>41</td>
<td>Never Married</td>
<td>United States of America</td>
<td>Masters</td>
<td>Prof.</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>101502+27</td>
<td>101502</td>
<td>27</td>
<td>Never Married</td>
<td>United States of America</td>
<td>Bach.</td>
<td>Sales</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>101503+20</td>
<td>101503</td>
<td>20</td>
<td>Never Married</td>
<td>United States of America</td>
<td>HS-grad</td>
<td>Cleric.</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>101504+45</td>
<td>101504</td>
<td>45</td>
<td>Married</td>
<td>United States of America</td>
<td>Bach.</td>
<td>Exec.</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>101505+34</td>
<td>101505</td>
<td>34</td>
<td>Never Married</td>
<td>United States of America</td>
<td>Masters</td>
<td>Sales</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>101506+38</td>
<td>101506</td>
<td>38</td>
<td>Married</td>
<td>United States of America</td>
<td>HS-grad</td>
<td>Other</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Ordering Recommended Practice

Ordering is expensive in the database
Most operations in R do not need ordering

In `ore.sync()`, set `use.keys = FALSE` almost always UNLESS you know that you need more
If you are sampling data or you need integer indexing for any other purpose, then set `use.keys = TRUE` as you need ordered ore frames
Global Options in OML4R
Global Options in OML4R

help(ore.options)

Options for Reporting

- **ore.trace**: A logical value indicating whether iterative OML4R functions should print output at each iteration. Default: FALSE

Options for Row Ordering

- **ore.sep**: A character string specifying the separator to use between multiple column row names of an 'ore.frame'. Default: "|
- **ore.warn.order**: A logical value indicating whether a warning should be issued when pulling an 'ore.frame' that lacks row names or an 'ore.vector' that lacks element names into memory. Default: 'TRUE'

Copyright © 2020 Oracle and/or its affiliates.
Global Options in OML4R

Options for Server Execution:

• **ore.parallel**: A preferred degree of parallelism to use in the embedded R job; either a positive integer greater than or equal to '2' for a specific degree of parallelism, a value of 'FALSE' or '1' for no parallelism, a value of 'TRUE' for the database's default for parallelism, or 'NULL' for the database default for the operation. OML4R will use the same DOP for all operations based on the ore.parallel setting. Default: NULL

Options for Subsetting

• **ore.na.extract**: A logical value used during logical subscripting of an ore.frame or ore.vector object. When TRUE, rows or elements with an NA logical subscript produces rows or elements with NA values. When FALSE an NA logical subscript is interpreted as a FALSE value, resulting in the removal of the corresponding row or element. Default is FALSE, whereas TRUE would mimic how R treats missing value logical subscripting of data.frame and vector objects.
In-database Sampling and Random Partitioning
High performance 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
In-database Sampling

Motivation

R provides basic sampling capabilities, but requires data to be pre-loaded into memory

Catch 22

- Data too large to fit in memory, so need to sample
- Can’t sample because data won’t fit in memory

Minimize data movement by sampling in Oracle Database with OML4R ordering framework’s integer row indexing
Simple random sampling

*Select rows at random*

```r
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
```
Split data sampling
Randomly partition data in train and test sets

```r
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)
```
Systematic sampling
Select rows at regular intervals

```
set.seed(1)
N <- 20
myData <- data.frame(a=1:20,b=letters[1:N])
MYDATA <- ore.push(myData)
head(MYDATA)
start <- 2
by <- 3
systematicSample <- MYDATA[seq(start, nrow(MYDATA), by = by), , drop=FALSE]
class(systematicSample)
systematicSample
```
Stratified sampling
Select rows within each group

```r
set.seed(1)
N <- 200
myData <- data.frame(a=1:N, b=round(rnorm(N), 2),
                    group=round(rnorm(N, 4), 0))
MYDATA <- ore.push(myData)
head(MYDATA)
sampleSize <- 10
stratifiedSample <-
do.call(rbind,
    lapply(split(MYDATA, MYDATA$group),
          function(y) {
            ny <- nrow(y)
            y[sample(ny, sampleSize*ny/N),, drop = FALSE]
          })))
class(stratifiedSample)
stratifiedSample
```
Stratified sampling

### ore.stratified.sample

```r
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 code

```r
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(groups) to sample = 3, approx. # of sample = 150"
R> dim(NARROW_SAMPLE_G)
[1] 108 9
R> summary(NARROW_SAMPLE_G$GENDER)
   M  F
  77 31
R>
R> ore.drop("R1_SAMPLE_G_MS")
R> res <- ore.stratified.sample(x=NARROW,
  +   by=c("GENDER","MARITAL_STATUS"),
  +   pct=0.1,
  +   res.nm="R1_SAMPLE_G_MS")
[1] "# of stratum(groups) to sample = 21, approx. # of sample = 150"
[1] "10% is done"
[1] "20% is done"
[1] "23% is done"
[1] "33% is done"
[1] "48% is done"
[1] "58% is done"
[1] "67% is done"
[1] "77% is done"
[1] "86% is done"
[1] "96% is done"
R> dim(R1_SAMPLE_G_MS)
[1] 127 9
R> summary(R1_SAMPLE_G_MS$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
```
Cluster sampling

Select whole groups at random

```r
set.seed(1)
N <- 200
myData <- data.frame(a=1:N,b=round(runif(N),2),
                     group=round(rnorm(N,4),0))
MYDATA <- ore.push(myData)
head(MYDATA)
sampleSize <- 5
clusterSample <- do.call(rbind,
                          sample(split(MYDATA, MYDATA$group), 2))
class(clusterSample)
unique(clusterSample$group)
```
Quota sampling

*Select first $N$ rows*

```r
set.seed(1)
N <- 200
myData <- data.frame(a=1:N, b=round(runif(N), 2))
MYDATA <- ore.push(myData)

sampleSize <- 10
quotaSample1 <- head(MYDATA, sampleSize)
quotaSample1
```

```
  a b
1 1 0.27
2 2 0.37
3 3 0.57
4 4 0.91
5 5 0.20
6 6 0.90
7 7 0.94
8 8 0.66
9 9 0.63
10 10 0.06
```
Data Types
## Data Types

**Mapping between R and Oracle Database**

<table>
<thead>
<tr>
<th>SQL – ROracle Read</th>
<th>R</th>
<th>SQL – ROracle Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>varchar2, char, clob, rowid</td>
<td>character</td>
<td>varchar2(4000)</td>
</tr>
<tr>
<td>number, float, binary_float,</td>
<td>numeric</td>
<td>if(ora.number==T) number else binary_double</td>
</tr>
<tr>
<td>binary_double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>date, timestamp</td>
<td>POSIXct</td>
<td>timestamp</td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>timestamp</td>
</tr>
<tr>
<td>interval day to second</td>
<td>difftime</td>
<td>interval day to second</td>
</tr>
<tr>
<td></td>
<td>factor (and other types)</td>
<td>character</td>
</tr>
</tbody>
</table>
Create a `data.frame` with various types, then `ore.frame`

```r
df <- data.frame(a="abc",
b=1.456,
c=TRUE,
d=as.integer(1),
e=Sys.Date(),
f=as.difftime(c("0:3:20", "11:23:15")))
str(df)
DF <- ore.push(df)
str(DF)
DF@desc$Sclass

DF$a <- as.ore.character(DF$a)
```
```r
R> df <- data.frame(a = "abc", b = 1.456, c = TRUE, d = as.integer(1), e = Sys.Date(), 
+f = as.difftime(c("01:30", "11:23:15")))
R>
R> str(df)
'data.frame': 2 obs. of 6 variables:
$ a : Factor w/ 1 level "abc": 1 1
$ b: num 1.45 1.45
$ c: logi TRUE TRUE
$ d: int 1 1
$ e: Date, format: "2014-01-26 "2014-01-26"
$ f:Class 'difftime' atomic [1:2] 335 635.25
 ... attr(*, "tzone")= chr "EST5EDT"
 ... attr(*, "units")= chr "minute"
R> DF <- ore.push(df)
R> str(DF)
'data.frame': 2 obs. of 6 variables:
Formal class 'ore.frame' [package "OREbase"] with 12 slots
 .. @ data : list()
 .. @ data0ry : Named chr '{ select VAL007 NAME001,VAL008 NAME002, VAL001 ,VAL002 ,VAL003 ,VAL004 ,VAL005 ,VAL006 from ORE1_459 }'
 .. @ atraits: chr "1_460"
 .. @ data0bj : chr "1_460"
 .. @ desc : 'data.frame': 6 obs. of 2 variables:
 .. ..@ name : chr "a" "b" "c" "d" ...
 .. ..@ Sclass : chr "factor" "numeric" "logical" "integer" ...
 .. @ sqlName : Named chr "VAL007" "VAL008"
 .. @ atraits: chr "1_460"
 .. @ sqlValue : chr "VAL001" "VAL002" "VAL003" "VAL004" ...
 .. @ sqlTable : chr "ORE1_459"
 .. @ sqlPred : chr ""
 .. @ extRef :List of 1
 .. ..@ (environment: 0x10fd5158)
 .. @ names : chr
 .. @ row.names: int
 .. @ SClass : chr "data.frame"
R> DF@desc$Sclass
[1] "factor" "numeric" "logical" "integer" "Date" "difftime"
R>
R> DF$a <- as.ore.character(DF$a)
R> DF@desc$Sclass
[1] "character" "numeric" "logical" "integer" "Date" "difftime"
```
CLOB and BLOB support in `ore.push` and `ore.pull`

```r
vbraw <- raw(3000L)
attr(vbraw, "ora.type") <- "blob"
oreBRaw <- ore.push(vbraw)
class(oreBRaw)
[1] "ore.raw"
attr(,"package")
[1] "OREbase"
new.vbraw <- ore.pull(oreBRaw)
class(new.vbraw)
[1] "raw"
length(new.vbraw)
[1] 3000
oreBRaw@sqlTable
[1] "/RQUSER"."ORE$3_18"
```

```
SQL> desc ORE$3_18
Name       Null? Type
----------- -------- ------------------------
VAL001      BLOB
VAL002      NUMBER(38)
VAL003      NUMBER(38)
```
Summary

The purpose of the Transparency Layer is to support in-database data exploration, data preparation, and data analysis en route to application of machine learning algorithms, where we have a mix of in-database and CRAN techniques.

OML4R provides transparency for in-database execution from R

It’s transparent...

• R users need use only R syntax
• No need to learn a different programming paradigm or environment
• Users see database objects as proxy R objects to simplify interaction and manipulation
For more information...

oracle.com/machine-learning

Oracle Machine Learning

The Oracle Machine Learning product family enables scalable data science projects. Data scientists, analysts, developers, and IT can achieve data science project goals faster while taking full advantage of the Oracle platform.

Oracle Machine Learning consists of complementary components supporting scalable machine learning algorithms for in-database and big data environments, notebook technology, SQL and R APIs, and Hadoop/Spark environments.

See also AskTOM OML Office Hours
Thank You

Mark Hornick
Oracle Machine Learning Product Management