ORACLE

Session 2a: Oracle Machine Learning for R

Transparency Layer - dplyr

Mark Hornick, Senior Director
Oracle Machine Learning Product Management

November 2020

Safe harbor statement

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. What is dplyr?
2. Functionality of OREdplyr
3. Examples using OREdplyr

What is dplyr?

What is dplyr?

A grammar for data manipulation

An R package that provides fast, consistent tool for working with data frame like objects, both in memory and out of memory

Operates on data.frame or numeric vector objects

Widely used package that also interfaces to database management systems

https://cran.r-project.org/web/packages/dplyr/index.html

dplyr + Oracle Database via OML4R…

OREdplyr

A subset of dplyr functionality extending ORE transparency layer

Use ore.frames instead of data.frames for in-database execution

Avoid costly movement of data

Scale to larger data volumes since not constrained by R Client memory

Functionality of OREdplyr

OREdplyr functions in ORE 1.5.1

OREdplyr functionality maps closely to CRAN dplyr package, e.g., function and args

OREdplyr operates on ore.frame or ore.numeric objects

Functions support non-standard evaluation (NSE) and standard evaluation (SE) interface

- Difference noted with a _ at the end of function name, e.g.,
  - NSE → select, filter, arrange, mutate, transmute
  - SE → select_, filter_, arrange_, mutate_, transmute_
- NSE interface is good for interactive use while SE ones are convenient for programming
- See https://cran.r-project.org/web/packages/dplyr/vignettes/programming.html for details

OREdplyr functions by category

Data manipulation

- select, filter, arrange, rename, mutate, transmute, distinct, slice, desc, select_, filter_, arrange_, rename_, mutate_, transmute_ , distinct_, slice_, inner_join, left_join, right_join, full_join

Grouping

- group_by, groups, ungroup, group_size, n_groups, group_by_

Aggregation

- summarise, summarise_, tally, count, count_

Sampling

- sample_n, sample_frac

Ranking

- row_number, min_rank, dense_rank, percent_rank, cume_dist, ntile, nth, first, last, n_distinct, top_n

Examples using OREdplyr

Content adapted from original dplyr vignettes (e.g., link)

Examples: basic operations

```
library(OREdplyr)

library(nycflights13)    # contains data sets

# Import data to Oracle Database

ore.drop("FLIGHTS") # remove database table, if exists
# create table from data.frame
ore.create(as.data.frame(flights), table="FLIGHTS")

dim(FLIGHTS)     # get # rows and # columns
names(FLIGHTS)   # view names of columns
head(FLIGHTS)    # verify data.frame appears as expected

# Basic operations

select(FLIGHTS, year, month, day, dep_delay, arr_delay)
       %>% head() # select columns
select(FLIGHTS, -year,-month, -day)
       %>% head() # exclude columns
```

```
select(FLIGHTS, tail_num = tailnum)
        %>% head() # rename columns, but drops others
rename(FLIGHTS, tail_num = tailnum)
        %>% head() # rename columns

filter(FLIGHTS, month == 1, day == 1)
        %>% head() # filter rows
filter(FLIGHTS, dep_delay > 240) %>% head()
filter(FLIGHTS, month == 1 | month == 2) %>% head()

arrange(FLIGHTS, year, month, day)
        %>% head() # sort rows by specified columns
arrange(FLIGHTS, desc(arr_delay))
        %>% head() # sort in descending order

distinct(FLIGHTS, tailnum)
         %>% head() # see distinct values
distinct(FLIGHTS, origin, dest)
         %>% head() # see distinct pairs
```

Examples: basic operations

```
mutate(FLIGHTS, speed = air_time / distance)
       %>% head()  # compute and add new columns
mutate(FLIGHTS,    # keeps existing columns
       gain = arr_delay - dep_delay,
       speed = distance / air_time * 60) %>% head()

transmute(FLIGHTS, # only keeps new computed columns
  gain = arr_delay - dep_delay,
  gain_per_hour = (arr_delay - dep_delay) / (air_time / 60))
      %>% head()

summarise(FLIGHTS, # aggregates the specified column values
          mean_delay = mean(dep_time,na.rm=TRUE),
          min_delay = min(dep_time,na.rm=TRUE),
          max_delay = max(dep_time,na.rm=TRUE),
          sd_delay = sd(dep_time,na.rm=TRUE))
```

```
# Row indexing requires setting row.names or have primary key
FLIGHTS[1,]  # Fails
row.names(FLIGHTS) <- FLIGHTS$tailnum   # set row.names
FLIGHTS[1,]                             # Succeeds

# requires ordered ore.frame, returns specified rows
slice(FLIGHTS, 10:20)

sample_n(FLIGHTS, 10) # take a random sample of N rows
dim(sample_frac(FLIGHTS, 0.01)) # take a random sample of p %

# take a random sample of N rows with replacement
sample_n(FLIGHTS, 10, replace = TRUE)
```


Examples

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

# select specified columns
names(select(IRIS, Petal.Length))
names(select(IRIS, petal_length = Petal.Length))

# drop specified column
names(select(IRIS, -Petal.Length))
names(select_(IRIS, ~Petal.Length))
names(select_(IRIS, petal_length = quote(Petal.Length)))
names(select_(IRIS, .dots = list("-Petal.Length")))

# rename() keeps all variables
names(rename(IRIS, petal_length = Petal.Length))

# Programming with select
head(select_(IRIS, ~Petal.Length))
head(select_(IRIS, "Petal.Length"))
head(select_(IRIS, quote(-Petal.Length),
             quote(-Petal.Width)))
head(select_(IRIS, .dots = list(quote(-Petal.Length),
             quote(-Petal.Width))))
```

```
# arrange ore.frame
MTCARS <- ore.push(mtcars)
arrange(MTCARS, cyl, disp)
arrange(MTCARS, desc(disp))

# filter ore.frame
head(filter(MTCARS, cyl == 8))
head(filter(MTCARS, cyl < 6))

# Multiple criteria
head(filter(MTCARS, cyl < 6 & vs == 1))
head(filter(MTCARS, cyl < 6 | vs == 1))

# Multiple arguments are equivalent to and
head(filter(MTCARS, cyl < 6, vs == 1))

head(mutate(MTCARS, displ_l = disp / 61.0237))
head(transmute(MTCARS, displ_l = disp / 61.0237))
head(mutate(MTCARS, cyl = NULL))
head(mutate(MTCARS, cyl = NULL, hp = NULL,
            displ_l = disp / 61.0237))
```

Examples

```
MTCARS <- ore.push(mtcars)
by_cyl <- group_by(MTCARS, cyl)
arrange(summarise(by_cyl, mean(disp), mean(hp)), cyl)

# summarise drops one layer of grouping
by_vs_am <- group_by(MTCARS, vs, am)
by_vs    <- summarise(by_vs_am, n = n())
arrange(by_vs, vs, am)
arrange(summarise(by_vs, n = sum(n_CNT)), vs)

# remove grouping
summarise(ungroup(by_vs), n = sum(n_CNT))

# group by expressions with mutate
arrange(group_size(group_by(mutate(MTCARS,
                                   vsam = vs + am),
                            vsam)), vsam)

# rename the grouping column
groups(rename(group_by(MTCARS, vs), vs2 = vs))
```

```
# add more grouping columns
groups(group_by(by_cyl, vs, am))
groups(group_by(by_cyl, vs, am, add = TRUE))

# Duplicate groups are dropped
groups(group_by(by_cyl, cyl, cyl))

library(magrittr)
by_cyl_gear_carb <- MTCARS %>% group_by(cyl, gear, carb)
n_groups(by_cyl_gear_carb)
arrange(group_size(by_cyl_gear_carb), cyl, gear, carb)

by_cyl <- MTCARS %>% group_by(cyl)

# number of groups
n_groups(by_cyl)

# size of each group
arrange(group_size(by_cyl), cyl)
```

Examples: stacking and grouping

```
# Stacking operations - lazy evaluation

c1 <- filter(FLIGHTS, year == 2013, month == 1, day == 1)
c2 <- select(c1, year, month, day,
             carrier, dep_delay, air_time, distance)
c3 <- mutate(c2,
             speed = distance / air_time * 60) # compute col
c4 <- arrange(c3, year, month, day, carrier) # sort result
head(c4)
dim(c4)
class(c4)

#-- Retrieve all data to a local data.frame

c4_local <- ore.pull(c4) # as opposed to 'collect' from dplyr
dim(c4_local)
class(c4_local)
```

```
# Grouping
by_tailnum <- group_by(FLIGHTS, tailnum)  # group by tailnum
head(by_tailnum)
# For each tailnum, compute count, avg distance, arrival delay
delay <- summarise(by_tailnum,
                   count = n(),
                   dist = mean(distance,na.rm=TRUE),
                   delay = mean(arr_delay,na.rm=TRUE)
)
head(delay)
# filter rows by count and distance
delay <- filter(delay, count > 20, dist < 2000)
head(delay)
```


Examples: grouping, etc.

```
library(ggplot2)
delay.local <- ore.pull (delay)  # pull data to client to
generate plot
ggplot(delay.local, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2, color='green') +
  geom_smooth() +
  scale_size_area()

# Group by year and month
monthly <- group_by(FLIGHTS, year, month)

# Find the most and least delayed flight each month
bestworst <- monthly %>%
  select(year, month, flight, arr_delay) %>%
  filter(min_rank(arr_delay) == 1 |
         min_rank(desc(arr_delay)) == 1)

bestworst %>% arrange(month, arr_delay)
```

```
# Rank each flight within the month
ranked <- monthly %>%
  select(arr_delay,year,month) %>%
  mutate(rank = rank(desc(arr_delay)))
head(ranked)
class(ranked)

ranked_sorted <- arrange(ranked, rank) # sort data by rank
head(ranked_sorted)

destinations <- group_by(FLIGHTS, dest) # group by destination
destinations %>% transmute(dest, planes = dense_rank(tailnum))
     %>% top_n(1) %>% unique

# determine # flights/day
daily <- group_by(FLIGHTS, year, month, day)
per_day   <- summarise(daily, flights = n())
head(per_day)

# number of flights per month
(per_month <- summarise(per_day, flights = sum(flights)))
# number of flights per year
(per_year  <- summarise(per_month, flights = sum(flights)))
```

Examples: chaining

```
a1 <- group_by(FLIGHTS, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
a3 <- summarise(a2,
                arr = mean(arr_delay, na.rm = TRUE),
                dep = mean(dep_delay, na.rm = TRUE))
a4 <- filter(a3, arr > 30 | dep > 30)
head(a4)
```

```
res <- filter(
  summarise(
    select(
      group_by(FLIGHTS, year, month, day),
      arr_delay, dep_delay),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)),
  arr > 30 | dep > 30)
head(res)
```

```
res <- FLIGHTS %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
head(res)
```


Examples: tally and count

```
# Tally and count
ore.drop("MTCARS")
ore.create(mtcars, table="MTCARS")

# count cars by # cylinders, sort by # cylinders
arrange(tally(group_by(MTCARS, cyl)), cyl)
# same, but sort by count
tally(group_by(MTCARS, cyl), sort = TRUE)

#-- Multiple tallys progressively roll up the groups
cyl_by_gear <- tally(group_by(MTCARS, cyl, gear), sort = TRUE)
tally(cyl_by_gear, sort = TRUE)
tally(tally(cyl_by_gear))

cyl_by_gear <- tally(group_by(MTCARS, cyl, gear),
                     wt = hp,  sort = TRUE)
tally(cyl_by_gear, sort = TRUE)
tally(tally(cyl_by_gear))
```

```
cyl_by_gear <- count(MTCARS, cyl, gear, wt = hp + mpg,
                           sort = TRUE)
tally(cyl_by_gear, sort = TRUE)
tally(tally(cyl_by_gear))

MTCARS %>% group_by(cyl) %>% tally(sort = TRUE)

# count is more succinct and performs grouping
MTCARS %>% count(cyl) %>% arrange(cyl)

MTCARS %>% count(cyl, wt = hp) %>% arrange(cyl)

MTCARS[MTCARS$cyl==4, "hp"]
sum(MTCARS[MTCARS$cyl==4, "hp"])

MTCARS %>% count_("cyl", wt = hp, sort = TRUE)
```


Examples: tally and count

```
#-- Grouped tally

tally(group_by(FLIGHTS, month)) # count of flights per month
tally(group_by(FLIGHTS, month), sort = TRUE) # sorted by count

#-- Nested tally invocations progressively roll up the groups
origin_by_month <- tally(group_by(FLIGHTS, origin, month),
                         sort = TRUE)
tally(origin_by_month, sort = TRUE)
tally(tally(origin_by_month))

# Use the infix %>% operator
FLIGHTS %>% group_by(month) %>% tally(sort = TRUE)

# count is more succinct - also does grouping
FLIGHTS %>% count(month,sort=TRUE)
```

```
# Non-Standard Evaluation (NSE) vs Standard Evaluation (SE)

# NSE version:
summarise(MTCARS, mean(mpg))

# SE versions:
summarise_(MTCARS, ~mean(mpg))

summarise_(MTCARS, quote(mean(mpg)))

summarise_(MTCARS, "mean(mpg)")

n <- 10
dots <- list(~mean(mpg), ~n)
summarise_(MTCARS, .dots = dots)
```


Examples: two table functions – joins

```
# create the needed tables from the nycflights13 data sets
ore.drop("AIRLINES")
ore.create(as.data.frame(airlines), table="AIRLINES")
ore.drop("WEATHER")
ore.create(as.data.frame(weather), table="WEATHER")
ore.drop("PLANES")
ore.create(as.data.frame(planes), table="PLANES")
ore.drop("AIRPORTS")
ore.create(as.data.frame(airports), table="AIRPORTS")

#-- select subset of columns for the following examples

flights2 <- FLIGHTS %>% select(year,month,day, hour,
                               origin, dest, tailnum, carrier)
head(flights2)
dim(flights2)
```

```
# create a database table index, if desired
ore.exec('CREATE INDEX carrier_idx on FLIGHTS("carrier")')

# joins on carrier - "natural join"
res <- flights2 %>% left_join(AIRLINES)
dim(res)

# joins on year, month, day, origin - "natural join"
res <- flights2 %>% left_join(WEATHER)
dim(res)

# specify column to join by
res <- flights2 %>% left_join(PLANES, by = "tailnum")
dim(res)

# specify which columns to join
res <- flights2 %>% left_join(AIRPORTS, c("dest" = "faa"))
dim(res)

# join on origin instead of dest
res <- flights2 %>% left_join(AIRPORTS, c("origin" = "faa"))
dim(res)
```


Examples: other join-related functions

```
(df1 <- data_frame(x = c(1, 2), y = 2:1))  # create some data
(df2 <- data_frame(x = c(1, 3), a = 10, b = "a"))

# store in the database as tables
ore.drop("DF1")
ore.create(as.data.frame(df1), table="DF1")
ore.drop("DF2")
ore.create(as.data.frame(df2), table="DF2")

# returns rows when there is a match in both tables
DF1 %>% inner_join(DF2)
```

```
# returns all rows from the left table,
#   even if no matches in the right table
DF1 %>% left_join(DF2)

# returns all rows from the right table,
#   even if no matches in the right table
DF1 %>% right_join(DF2)

# swap the tables and see different,
#   but similar results on a per row basis
DF2 %>% left_join(DF1)

# returns all rows from the left and right tables.
#    Combines the result of both LEFT and RIGHT joins
DF1 %>% full_join(DF2)
```

OREdplyr caveats

‘:’ not supported for range of column specification, e.g., V1:V10

Variables cannot be referenced within a mutate() and transmute()

- Restate computation where needed

Functions supported for summarise when using grouped ore.frame

- 'min', 'mean', 'max', 'median', 'length', 'IQR', 'prod', 'sum', 'range', 'quantile', 'fivenum', 'summary', 'sd','var', 'all', 'any‘

n_distinct()

- Works with non-grouped ore.frame
- Not supported for summarise with grouped ore.frame
  - Work around: use dense_rank, top_n, and unique

```
# compute number of distinct planes over destination
destinations %>% transmute(dest, planes = dense_rank(tailnum)) %>% top_n(1) %>% unique
```

filter() does not apply non-ranking function per group

Use ore.pull instead of dplyr collect

Summary

OREdplyr provides a subset of dplyr functionality working with ore.frames

Use popular API conveniently with Oracle Database tables

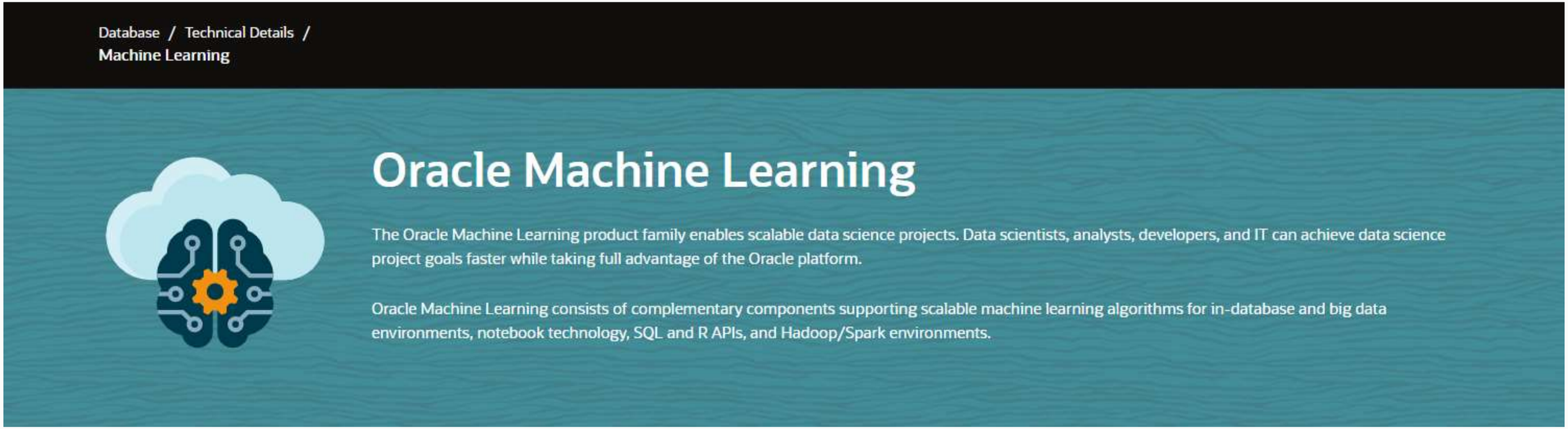
Avoid costly movement of data

Scale to larger data volumes since not constrained by R Client memory

Use Oracle Database as high performance compute engine

For more information…

oracle.com/machine-learning



See also AskTOM OML Office Hours

Thank You

Mark Hornick
Oracle Machine Learning Product Management