Lecture on plyr on Oct 10th 2013

R package plyr

To install plyr, start R and type

```
install.packages("plyr")
```

Choose a CRAN mirror, preferably one of US mirrors

Load plyr

```
library(plyr)
```

First Example

Make up a data.frame `df` of 5 rows and 3 columns

```
> df
  gender grades score
1      M      A     1
2      M      A     2
3      M      B     3
4      F      A     4
5      F      B     5
```

Here's the code for it

```
df = data.frame(
        gender=rep(c("M","F"),times=c(3,2)),
        grades=c("A","A","B","A","B"),
        score=1:5
)
```

First Example Cont.

Let's find the average score for male and female

Make the output a nice data.frame df.ave with 2 rows and 2 columns

```
> df.ave
  gender average
1      F     4.5
2      M     2.0
```

First Example Cont.

Use base R functions

eh ...

Use tapply()

```
ave = tapply(df$score, INDEX=df$gender, FUN=mean)
df.ave = data.frame(
        gender=names(ave),
        average=as.numeric(ave)
)
```

Use plyr functions

```
df.ave = ddply(df, "gender", function(r) {
        c(average=mean(r$score))
})
```

First Example Cont.

Let's find the min and max score for combinations of gender and grade

Make the output a nice data.frame df.range with 4 rows and 4 columns

```
> df.range
  gender grades average
1      F      A     4.0
2      F      B     5.0
3      M      A     1.5
4      M      B     3.0
```

First Example Cont.

Use base R functions

eh ... ...

Use tapply()

```
ran = tapply(df$score,
    INDEX = list(df$gender, df$grades),
    FUN = function(r) { c(min(r), max(r)) }
)
```

eh ...

Use plyr functions

```
df.range = ddply(df,c("gender","grades"),function(r){
        c(min=min(r$score), max=max(r$score))
})
```

Split-Apply-Combine (SAC) Strategy for Data Analysis

Split-apply-combine is a common data analysis pattern/strategy

Split

Break up a big problem into manageable pieces

Apply

Operate on each piece independently

Combine

Put all pieces together

Examples of SAC

SAC is in all stages of analysis

Data preparation

Create new variables on a per-group basis

e.g., group-wise ranking, standardization

Exploratory analysis

Create group-wise summaries for display or analysis

e.g., calculate marginal means

Modelling

Fit separate models to each panel of panel data

Similar and Existing Tools

Many tools implemented SAC:

- Excel's pivot tables
- SQL's `group by` operator
- SAS's `by` argument to many procedures
- Hadoop's map-reduce strategy for processing large data

In R:

- Trellis display and lattice package for visualization
- "apply" family of functions for general data analysis
- plyr package for general data analysis

plyr Package

Developed by Hadley Wickham

A set of tools for SAC

For more details, refer to
http://plyr.had.co.nz/

plyr vs Loops

plyr provides a replacement for loops for SAC type of problems

Not because loops are slow

Loops do not clearly express intent

plyr illuminates the key components of the computation

plyr vs Loops Cont.

plyr assumes each piece of data will be processed only once and indepedently of all other pieces

Use loop when

- Iteration requires overlapping data
- Iteration depends on previous iteration

plyr vs apply functions

`apply()`, `lapply()`, `tapply()` can be used for SAC

plyr generalises and standardises the apply family of functions

- Consistent names, arguments and outputs
- Input from and output to data.frames, matrices and lists
- Labels are maintained across all operations

plyr is as fast as, or faster than, the built-in functions

plyr Function Names

Function names

aaply	adply	alply	a_ply
daply	ddply	dlply	d_ply
laply	ldply	llply	l_ply
raply	rdply	rlply	r_ply

Functions are named according to input type and output type

Fist character for input Second character for output

Input types:

a = array, d = data frame, l = list, r = number of interations

Output types:

a, d, l, _ means output discarded

Effects of input type and output type are orthogonal

Functions a*ply()

Input type: array

Arrays are sliced by dimension into lower-d arrays

```
a*ply(.data, .margins, .fun, ...)
```

My own print function for better display

```
myprint = function(x, ...) {
    cat("\n")
    print(x, ...)
    cat("------End of Print------\n")
}
```

Functions a*ply() Cont.

Slicing and printing a 2-d array

Make up an array

```
a2 = array(data=1:6, dim=c(2,3))
```

Split by one dimension

```
a_ply(.data=a2, .margins=1, .fun=myprint)
a_ply(.data=a2, .margins=2, .fun=myprint)
```

Split by two dimensions

```
a_ply(.data=a2, .margins=c(1,2), .fun=myprint)
```

Functions a*ply() Cont.

Slicing and printing a 3-d array

Make up an array

```
a3 = array(data=1:24, dim=c(2,3,4))
```

Split by one dimension

```
a_ply(.data=a3, .margins=3, .fun=myprint)
a_ply(.data=a3, .margins=2, .fun=myprint)
```

Split by two dimensions

```
a_ply(.data=a3, .margins=c(2,3), .fun=myprint)
```

Split by all three dimensions

```
a_ply(.data=a3, .margins=c(1,2,3), .fun=myprint)
```

Functions l*ply()

Input type: list

Lists are split by element

```
l*ply(.data, .fun, ...)
```

Make up a list

```
l3 = list(a=1, b=2:3, c=4:6)
```

Split by element

```
l_ply(.data=l3, .fun=myprint)
```

Functions d*ply()

Input type: data.frame

Data frame are subsetted by combinations of variables

```
d*ply(.data, .variables, .fun, ...)
```

Make up a data.frame

```
df = data.frame(
        gender=rep(c("M","F"),times=c(3,2)),
        grades=c("A","A","B","A","B"),
        score=1:5
)
```

Functions d*ply() Cont.

Split by gender

```
d_ply(.data=df, .variables="gender",
        .fun=myprint
)
```

Split by both gender and grades

```
d_ply(.data=df, .variables=c("gender","grades"),
        .fun=myprint
)
```

Functions d*ply() Cont.

Variable specification syntax

Character

```
d_ply(.data=df, .variables="gender",
        .fun=myprint
)
```

Numeric

```
d_ply(.data=df, .variables=1, .fun=myprint)
```

Formula

```
d_ply(.data=df, .variables=~gender+grades,
        .fun=myprint
)
```

Special

```
d_ply(.data=df, .variables=.(gender,grades),
        .fun=myprint
)
```

Case Study: Barley Data

Read in data

```
library(lattice)
data(barley)
?barley
```

Yield for 10 varieties at sites in each of two years

120 records

4 variables: yield, variety, year, site

We are going to calculate five-number summary of yield across varieties for every site in every year

Yield at One Site in One Year

This is a working unit of this analysis

Subset data at one site in one year

```
one = subset(barley,
    subset=(site=="University Farm" & year==1931)
)
```

Compute the five number summary

```
result = quantile(one$yield)
```

Make it a function

```
five.num = function(data) {
    quantile(data$yield)
}
result = five.num(one)
```

Yields at Every Site in Every year: User Base R Functions

Split to pieces

```
pieces = split(barley,
               list(barley$site, barley$year)
)
```

Initialize results

```
results = vector(mode="list", length=length(pieces))
```

Apply to pieces

```
for(i in seq_along(pieces)) {
    piece = pieces[[i]]
    results[[i]] = five.num(piece)
}
```

Combine pieces

```
results = do.call("rbind", results)
results = as.data.frame(results)
```

Yields at Every Site in Every year: User Base R Functions Cont.

Not done yet, need proper labels

```
groups = names(pieces)
groups = strsplit(groups, split="\\.")
groups = do.call("rbind", groups)
groups = as.data.frame(groups)
names(groups) = c("site", "year")
results.r = cbind(groups, results)
```

Yields at Every Site in Every year: User plyr Functions

Use `ddply()`

```
results.plyr = ddply(
        .data=barley,
        .variables=c("site","year"),
        .fun=five.num
)
```