REVOLUTION ANALYTICS

Introduction to the Apply Function Family

Revolution Analytics

Overview

In this session we use `apply` functions on elements of lists.

Objectives:

- Understand what are the `apply()` functions
- Understand how `apply()` and family simplifies your code
- Understand different uses of the `apply()` functions

Outline

Example Dataset

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Summary statistics for multiple variable

- Imagine you want to compute a mean for each column in a data frame.
- Traditional programming languages: Write a loop. For example:

```
cars.means <- numeric(ncol(mtcars))
for (i in 1:ncol(mtcars)) {
  cars.means[i] <- mean(mtcars[, i])
}
cars.means
```

```
##  [1]  20.0906   6.1875 230.7219 146.6875   3.5966   3.2172  17.8487
##  [8]   0.4375   0.4062   3.6875   2.8125
```

A simpler way by using `apply()`

The function `apply()` allows you to compute on the dimensions (e.g. rows and columns) an array or dataframe

```
apply(X, MARGIN, FUN, ...)
```

`X`	an array, including a matrix.
`MARGIN`	a vector giving the subscripts which the function will be applied over. For a matrix, 1 indicates rows, 2 indicates columns.
`FUN`	the function to be applied: see 'Details'. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted.
	optional arguments to `FUN`.

Looping Across Columns

Remember that the dimensions of a table (and data frame) are:

1. Rows
2. Columns

So we apply the function `mean()` (or others) to the second dimension of `mtcars`:

```
apply(mtcars, 2, mean)
```

```
##      mpg      cyl     disp       hp     drat       wt     qsec       vs
##  20.0906   6.1875 230.7219 146.6875   3.5966   3.2172  17.8487   0.4375
##       am     gear     carb
##   0.4062   3.6875   2.8125
```

Exercise

Your turn:

- Use `apply()` to calculate the median of each column in the `mtcars` dataset
- Use `apply()` to calculate the 75th %ile of each column (Hint: `?quantile`)

apply() vs. loops

See help(apply) for function documentation.

Note the case of c(1,2) for MARGIN

```
set.seed(42)
dims <- c(360, 720, 120)
mat.stack <- array(rnorm(prod(dims)), dim = dims)
avgs.mat1 <- matrix(NA, nrow(mat.stack), ncol(mat.stack))
avgs.mat2 <- avgs.mat1
```

apply() vs. loops (cont'd)

```
system.time({
  for (i in 1:nrow(mat.stack)) {
    for (j in 1:ncol(mat.stack)) {
      avgs.mat1[i, j] <- mean(mat.stack[i, j, ])
    }
  }
})
```

```
##    user  system elapsed
##   6.636   0.008   6.642
```

```
system.time({
  avgs.mat2 <- apply(mat.stack, c(1, 2), mean)
})
```

```
##    user  system elapsed
##   4.804   0.104   4.910
```

Using `apply()`

Note that the actual data sets are identical.

All differences are 0.

```
table(avgs.mat1 - avgs.mat2)
```

```
##
##      0
## 259200
```

Exercise: Calculate `min()` and `max()`

Your turn:

- Use `apply()` to calculate the minimum and maximum values of each column in `mtcars`.
- Use `apply()` to calculate the 90th percentile for each variable in `mtcars` (see `quantile`)

Outline

Using `lapply()`

The function `lapply()` applies a function to each element of a list, returning the results in list format.

It can be used very much like `apply()` for columns:

```
mins.list.lapply <- lapply(mtcars, min)
mins.list.apply <- apply(mtcars, 2, min)
```

lapply() Results

```
mins.list.lapply
```

```
## $mpg
## [1] 10.4
##
## $cyl
## [1] 4
##
## $disp
...
```

```
mins.list.apply
```

```
##    mpg    cyl   disp     hp   drat     wt   qsec     vs     am   gear
## 10.400  4.000 71.100 52.000  2.760  1.513 14.500  0.000  0.000  3.000
##   carb
##  1.000
```

Why do both work?

```
is.list(mtcars)
```

```
## [1] TRUE
```

```
is.matrix(mtcars)
```

```
## [1] FALSE
```

Both give the same output, but in a different structure

Different data structures

- `apply()` only works on matrices, so it coerces the dataset to a matrix first.
- `lapply()` works on lists

OK sometimes, but you must watch out!

Outline

Add a `character` Variable

```
head(mtcars2)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
##                     efficient
## Mazda RX4           efficient
## Mazda RX4 Wag       efficient
## Datsun 710          efficient
## Hornet 4 Drive      efficient
## Hornet Sportabout gas-guzzler
## Valiant           gas-guzzler
```

Unexpected results with `apply()`

```
apply(mtcars2, 2, mean)
```

```
##       mpg       cyl      disp        hp      drat        wt      qsec
##        NA        NA        NA        NA        NA        NA        NA
##        vs        am      gear      carb efficient
##        NA        NA        NA        NA        NA
```

`lapply()` still works!

```
lapply(mtcars2, mean)
```

```
## $mpg
## [1] 20.09
##
## $cyl
## [1] 6.188
##
## $disp
...
```

Why does apply fail?

```
apply(mtcars2, 2, is.numeric)
```

```
##       mpg       cyl      disp        hp      drat        wt      qsec
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
##        vs        am      gear      carb efficient
##     FALSE     FALSE     FALSE     FALSE     FALSE
```

Why does apply fail?

```
apply(mtcars2, 2, is.character)
```

```
##       mpg       cyl      disp        hp      drat        wt      qsec
##      TRUE      TRUE      TRUE      TRUE      TRUE      TRUE      TRUE
##        vs        am      gear      carb efficient
##      TRUE      TRUE      TRUE      TRUE      TRUE
```

Remember: matrices can only store 1 `mode` of information, so it coerces all values to `character`.

Outline

`lapply()` example

The function `lapply()` returns a list of the same length as `X`, each element of which is the result of applying `FUN` to the corresponding element of `X`.

```
test.frame <- data.frame(X = rnorm(20), Y = rnorm(20, 2, 10), Z = rnorm(20,
  2, 3))
test.results <- apply(test.frame, 2, t.test)  # apply t-test to columns of the dataframe

class(test.results)
```

```
## [1] "list"
```

```
length(test.results)
```

```
## [1] 3
```

How to extract the names of each result?

Traditional approach:

```
test.names1 <- list()
for (i in 1:length(test.results)) {
  test.names1[[i]] <- names(test.results[[i]])
}
names(test.names1) <- names(test.results)
test.names1
```

```
## $X
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
##
## $Y
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
...
```

lapply() approach

```
test.names <- lapply(test.results, names)
test.names
```

```
## $X
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
##
## $Y
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
...
```

```
length(test.names)
```

```
## [1] 3
```

Question:

Can we duplicate `test.names` using `apply()` instead of `lapply()`? If so, what would the MARGIN argument be?

```
apply(test.results, 2, names)
```

We cannot easily duplicate `test.names` using `apply()` instead of `lapply()` because `apply()` requires a matrix (i.e. rows and columns).

Extract a Different Element

```
test.pvals <- lapply(test.results, getElement, "p.value")
test.pvals
```

```
## $X
## [1] 0.05243
##
## $Y
## [1] 0.1106
##
## $Z
## [1] 0.0007369
```

```
length(test.pvals)
```

```
## [1] 3
```

Addtiional helper functions:

sapply() Use `lapply()`, then `simplify2array()`

vapply() Similar to `sapply()`, but you must specify the type of result you expect from each call to `FUN`. Benefit: tremendously faster than `sapply()`

Exercise:

You turn

- Create a new list of the `estimate` of each of the `t.test` results contained in `test.results`

Exercise: More practice with lapply

```
data.list <- lapply(sample(0:1, 100, replace = TRUE), function(x) if (x ==
  0) rnorm(10) else letters[sample(1:26, 1)])
data.list
```

```
## [[1]]
## [1] "z"
##
## [[2]]
## [1] "d"
##
## [[3]]
...
```

Exercise: More practice with lapply

- Use `lapply()` on `data.list` to return a list indicating which elements are numeric and which are not.
- For the numeric elements, write a function that returns the average of that element. For the non-numeric elements return the fill value `NA`.
- Use `sapply()` to do the last step again, except with the output simplified as a vector.

Summary and Questions?

- `apply()` for matrices
- `lapply()` and sapply() for lists

Notes: Be careful with apply() and data.frames!

Outline

Using by()

The function by() is an "object-oriented wrapper" that is applied over subsets of dataframes and matrices defined by the indices.

For example, mtcars can be split by row into data frames subset by the levels of cyl, and function FUN is applied to each subset in turn:

```
# Average of all Variables at each level of Cyl

by(mtcars, mtcars$cyl, colMeans)
```

```
## mtcars$cyl: 4
##      mpg      cyl     disp       hp     drat       wt     qsec       vs
##  26.6636   4.0000 105.1364  82.6364   4.0709   2.2857  19.1373   0.9091
##       am     gear     carb
##   0.7273   4.0909   1.5455
## ------------------------------------------------------------
## mtcars$cyl: 6
```

Exercise: Practice with `by()`

Your turn:

- Use `by()` to return summary statistics for all columns of `mtcars` data using the summary function for all combinations of `am` and `vs`.

Outline

Using `replicate()`

The function `replicate()` conveniently evaluates an expression multiple times.

For example, say we want to bootstrap a sample of $1:100$ ten times:

```
replicate(10, sample(1:100, 100, replace = TRUE))
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##   [1,]   52   56   84  100   99   33   89   41   19    99
##   [2,]   23   97   19   44   61   89   31   77   83    22
##   [3,]   51   31   36   64   27   46   79   23    4    11
##   [4,]   48    8   41   56   39   35   46   47    9    60
##   [5,]   46    3   18   94    4   11   35   64   80    36
##   [6,]   92   17   12   12   95   17   75   87   73    67
...
```

Background: The Central Limit Theorem

Central limit theorem (CLT):

- Regardless of the underlying distribution, the distribution of the sum and average of a large number of independent, identically distributed (iid) variables will be approximately normal.

CLT: Background (cont'd)

For large sample sizes,

- The expected value of the sampling distribution of the mean is the population mean: $m_{\bar{X}=\mu}$
- The s.d. of the sampling distribution of the mean is the population: $s_{\bar{X}=\frac{\sigma}{\sqrt{N}}}$

Useful web page about central limit theorem

Exercise: The CLT and `replicate()`

Your turn:

- Using the sample on the previous slide, generate 100 bootstrap samples of 100 observations from an exponential distribution with a rate parameter of 3.
- Calculate the sample mean of each bootstrap sample.
- Calculate the means and standard deviation of the bootstrapped means. Is this what you would expect?
- Display the results graphically with `hist()`

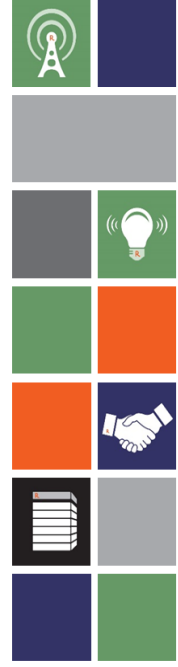
Hint: Exponential distribution: Use `rexp()`

Module review questions

- What, in a nutshell, is an `apply` function?
- What are the advantages of using `apply` functions?
- What are the `apply` functions we covered in this module, and what does each one do?

Questions?

REVOLUTION ANALYTICS



Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR