MANGOSOLUTIONS
data analysis that delivers

Apply Functions

Agenda

- Why Apply?
- The “apply” family of functions
  - Primary Functions
  - Other Functions
- The “plyr” package
- Summary

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Why Apply?

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Why Apply?

- The functions we use are simple things …

```
mean(x, trim = 0, na.rm = TRUE)
```

No “by” argument

- We need a framework to allow us to allow simple functions in a more structured way

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The “Apply” family of Functions

MANGOSOLUTIONS

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Which Functions?

- Getting a full list is tricky

```
 apropos("apply")
[1] ".mapply"    "apply"      "dendrapply" "eapply"
[5] "kernapply"  "lapply"     "mapply"     "rapply"
[9] "sapply"     "tapply"     "vapply"
```

- Most challenging aspect: remembering which function to use

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Primary Apply Functions

	Description	Input	Output
apply	Applys a function over dimensions of a data structure	Structure with a "dimension"	A single mode structure
lapply	Applys a function to elements of a list or vector	A list or vector	A list
sapply	Applys a function to elements of a list or vector	A list or vector	A "simplified list"
tapply	Applys a function to a vector by factor(s)	A Vector + Factor(s)	Depends on # factors
by	Applys a function to a data frame by factor(s)	A Data Frame + Factor(s)	A "by" object (which is a list)
aggregate	Applys a function to columns of a data frame by factor(s)	A Data frame + Factor(s)	A Data frame

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The “Apply” family of Functions

The `apply` Function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `apply` Function

- Applies a simple function over dimensions of a data structure
- So input is anything that has rows, columns (matrix, data frame, array .. NOT lists or vectors)

```
> args(apply)
function (X, MARGIN, FUN, ...)
NULL
```

Structure with a dimension

Dimension number (1 = rows, 2 = columns)

Function to apply

Additional arguments to the function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

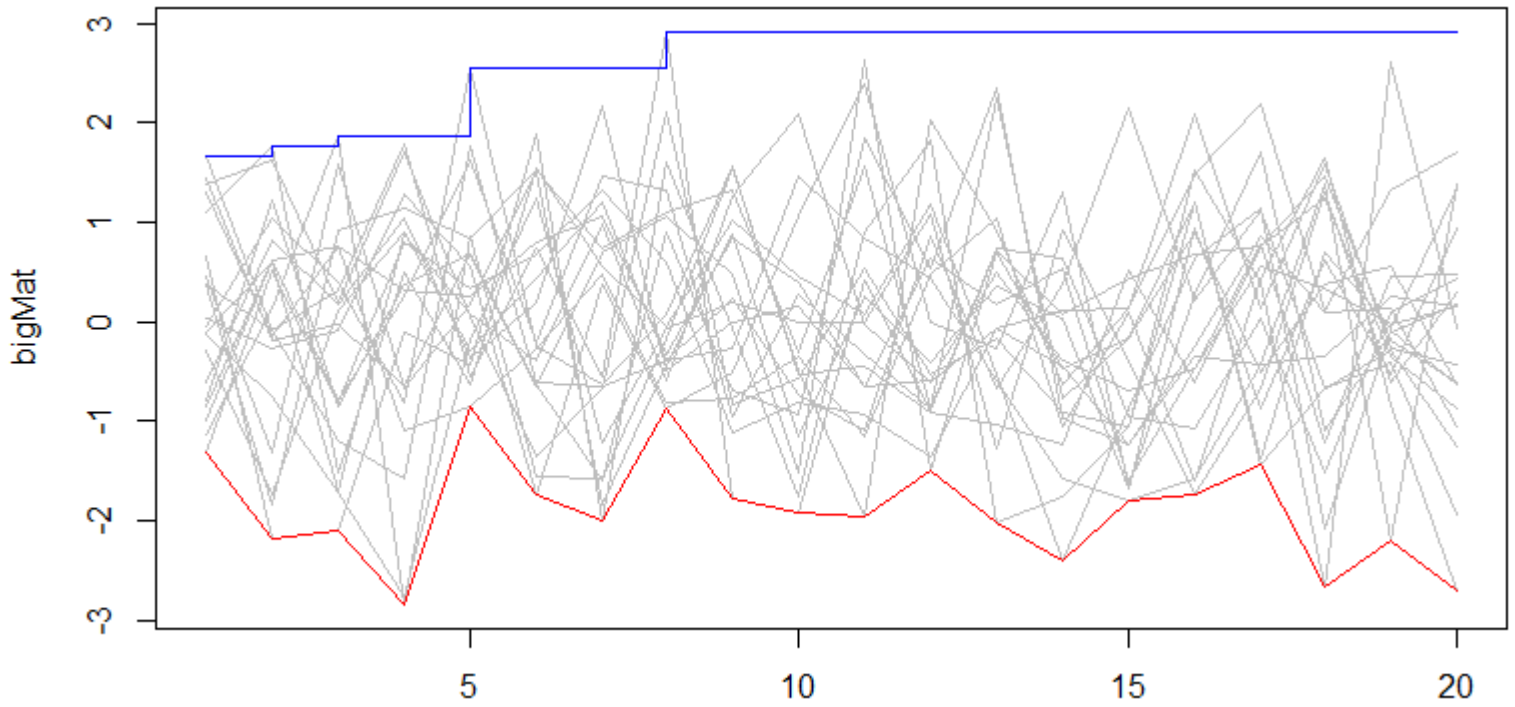
Basic `apply` Example

```
> myMatrix <- matrix( rpois(30, 3), 5)
> myMatrix
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    5    4    6    3    1
[2,]    2    4    3    5    4    2
[3,]    0    3    3    4    2    2
[4,]    2    8    2    4    2    2
[5,]    6    4    1    0    2    1
> apply(myMatrix, 1, min)  # Row minima
[1] 1 2 0 2 0
> apply(myMatrix, 2, max)  # Column maxima
[1] 6 8 4 6 4 2
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Mix it with graphics …

```
> bigMat <- matrix( rnorm(400), 20 )
> matplot(bigMat, type = "l", lty = 1, col = "grey")
> lines(1:20, apply(bigMat, 1, min), col = "red")
> lines(1:20, cummax(apply(bigMat, 1, max)), col = "blue", type = "s")
```



Apply with multiple dimensions

```
> myMatrix
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    5    4    6    3    1
[2,]    2    4    3    5    4    2
[3,]    0    3    3    4    2    2
[4,]    2    8    2    4    2    2
[5,]    6    4    1    0    2    1
> apply(myMatrix, 1:2, mean)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    5    4    6    3    1
[2,]    2    4    3    5    4    2
[3,]    0    3    3    4    2    2
[4,]    2    8    2    4    2    2
[5,]    6    4    1    0    2    1
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Apply over arrays

```
> myArray <- array( rpois(18, 3), dim = c(3, 3, 2))
> myArray
, , 1

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    5    4
[3,]    3    0    1

, , 2

     [,1] [,2] [,3]
[1,]    3    4    4
[2,]    2    5    1
[3,]    1    2    2
```

```
> apply(myArray, 3, diag)
     [,1] [,2]
[1,]    1    3
[2,]    5    5
[3,]    1    2
> apply(myArray, 1:2, max)
     [,1] [,2] [,3]
[1,]    3    4    4
[2,]    2    5    4
[3,]    3    2    2
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

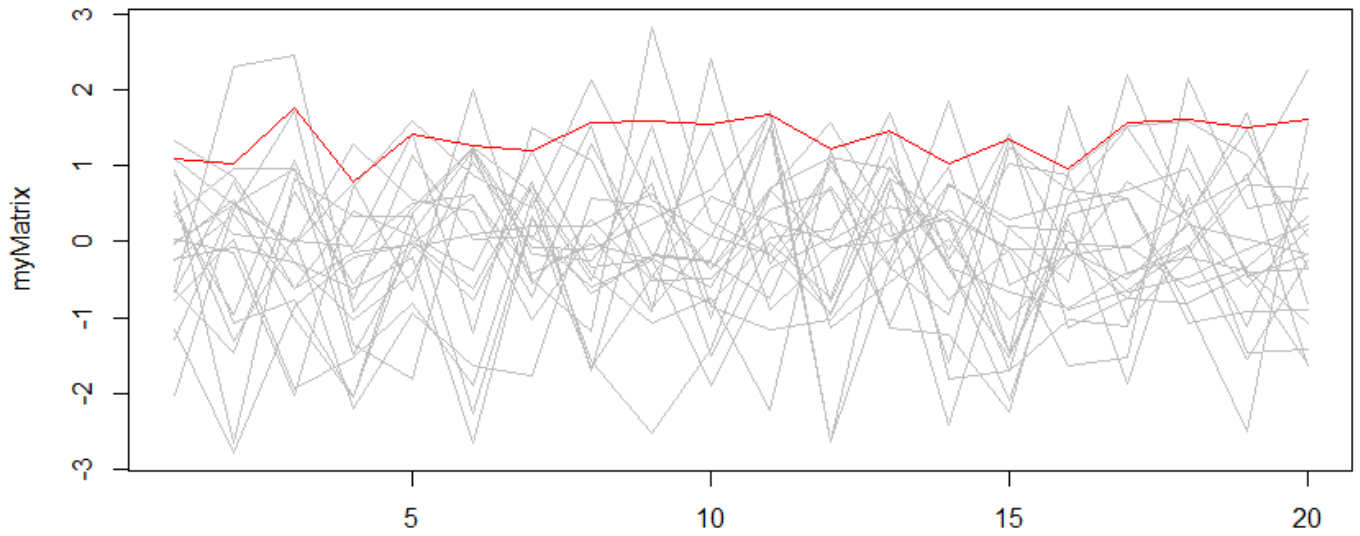
Passing Additional Arguments

```
> myMatrix[2, 2] <- myMatrix[4, 2] <- NA
> myMatrix
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    5    4    6    3    1
[2,]    2   NA    3    5    4    2
[3,]    0    3    3    4    2    2
[4,]    2   NA    2    4    2    2
[5,]    6    4    1    0    2    1
> apply(myMatrix, 2, mean)
[1] 3.0  NA 2.6 3.8 2.6 1.6
> apply(myMatrix, 2, mean, na.rm = TRUE)
[1] 3.0 4.0 2.6 3.8 2.6 1.6
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

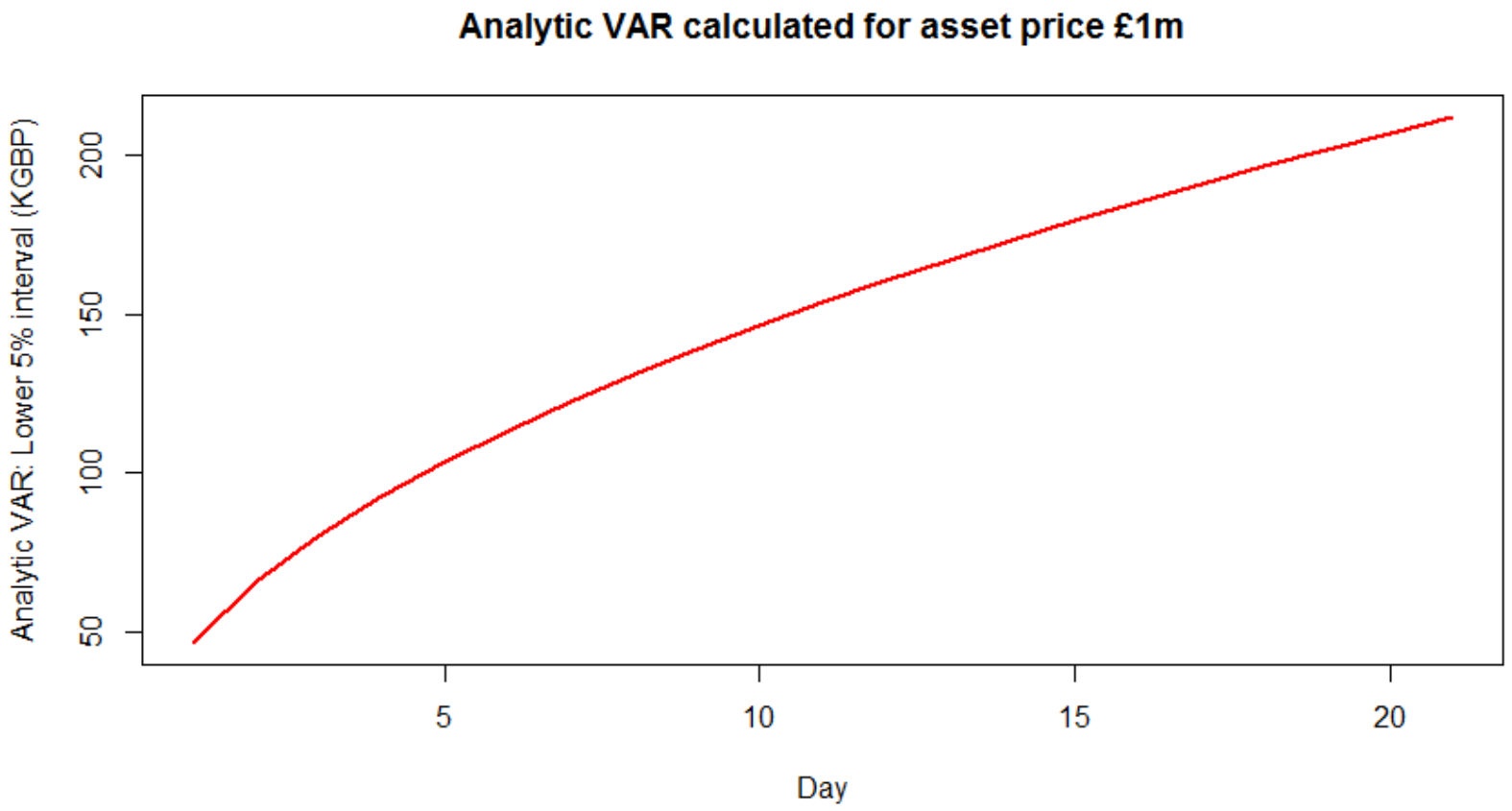
Using Custom Functions

```
> myFun <- function(x) quantile(x, .95)
> myMatrix <- matrix( rnorm(20^2), 20 )
> q95 <- apply(myMatrix, 1, myFun)
> q95
 [1] 1.0893245 1.0207158 1.7472680 0.7875372 1.4150316 1.2692083 1.1924562
 [8] 1.5534152 1.5946268 1.5326181 1.6687241 1.2074702 1.4628125 1.0186941
[15] 1.3470435 0.9614715 1.5547159 1.6110892 1.4897992 1.5976683
>
> matplot(myMatrix, type = "l", lty = 1, col = "grey")
> lines(1:20, q95, col = "red")
```



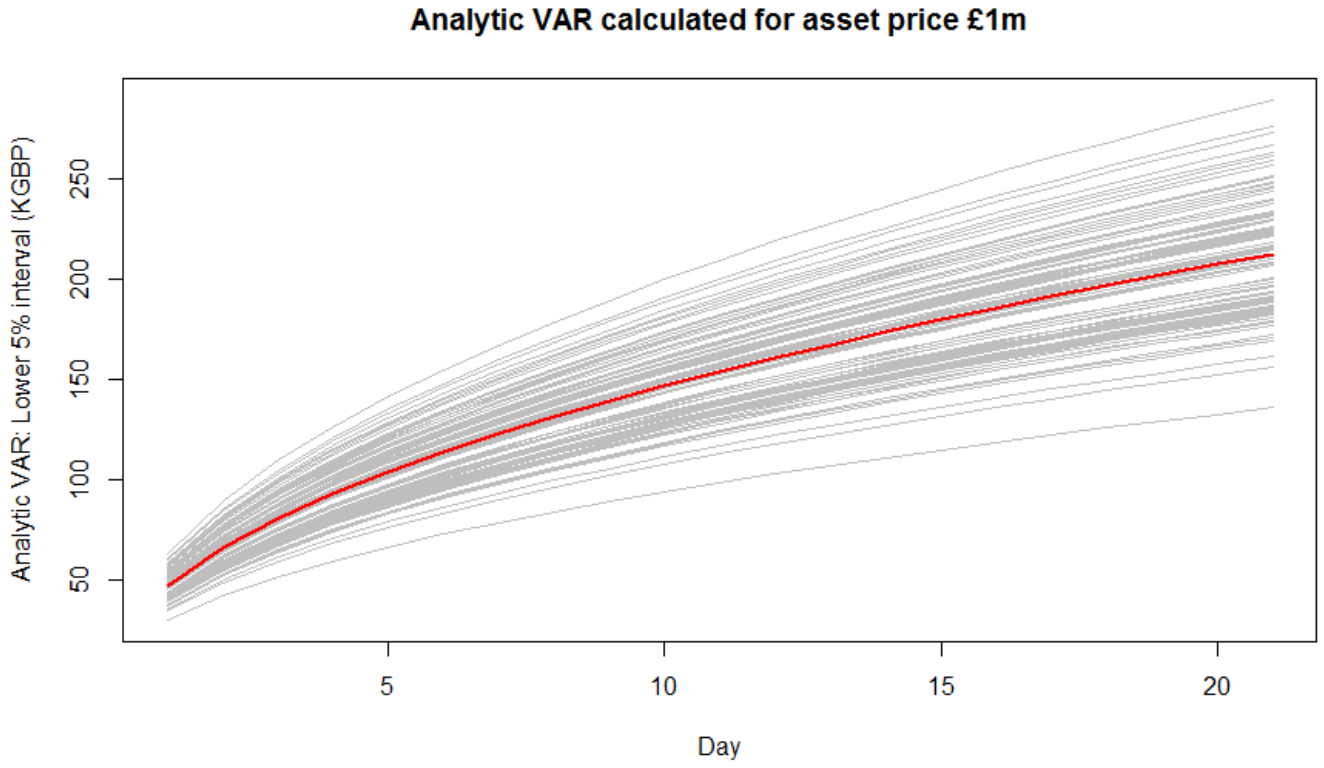
A Quick Example

```
> aVar95 <- function(days, value, mu, sigma) - (mu - qnorm(.95) * sigma) * value * sqrt(days)
> plot(1:21, aVar95(1:21, 1000000, .003, .03)/1000, type = "l", xlab = "Day",
+      ylab = "Analytic VAR: Lower 5% interval (KGBP)", col = "red",
+      main = "Analytic VAR calculated for asset price £1m", lwd = 2)
```



A Quick Example

```
> require(MASS)  # Load the MASS library
> theCov <- cbind(c(.00001, 1.13e-07), c( 1.13e-07, .00001) )
> varDist <- mvrnorm(100, c(.003, .03), theCov)  # Calculate parameters
> av95s <- apply(varDist, 1, function(pars) aVar95(1:21, 1000000, pars[1], pars[2])/1000)
> matplot(1:21, av95s, type = "l", col = "grey", lty = 1, xlab = "Day",
+         ylab = "Analytic VAR: Lower 5% interval (KGBP)",
+         main = "Analytic VAR calculated for asset price £1m")
> lines(1:21, aVar95(1:21, 1000000, .003, .03)/1000, col = "red", lwd = 2)
```



The "Apply" family of Functions

The `lapply` Function

MANGOSOLUTIONS

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `lapply` Function

- Applies a function over elements of a list
- Remember: a data frame is a list of named vectors!
- It always returns a list

```
> args(lapply)
function (X, FUN, ...)
NULL
```

List structure

Function to apply

Additional arguments to the function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple `lapply` Example

```
> myList <- list(Pois = rpois(10, 3), Norm = rnorm(5), Unif = runif(5, 1, 10))
> myList
$Pois
 [1] 3 0 7 7 4 2 2 2 1 7

$Norm
[1] -0.03116511  1.01138977  0.46652072  0.59080973 -1.13100916

$Unif
[1] 2.840396 7.659040 4.736725 9.415129 9.836543

> lapply(myList, mean)
$Pois
[1] 3.5

$Norm
[1] 0.1813092

$Unif
[1] 6.897567
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Using `split` to generate a list

- Will split an object (vector, data frame) based on 1 or more factors
- Great input to **`lapply`**!

```
> tubeData <- read.csv("tubeSubset.csv")
> head(tubeData)
  Month Excess     Line Type   WhenOpen Length
1     1   6.04 Bakerloo   DT After 1900  Short
2     2   6.54 Bakerloo   DT After 1900  Short
3     3   4.77 Bakerloo   DT After 1900  Short
4     4   5.40 Bakerloo   DT After 1900  Short
5     5   5.23 Bakerloo   DT After 1900  Short
6     6   5.03 Bakerloo   DT After 1900  Short
> with(tubeData, split(Excess, Line))
$Bakerloo
 [1] 6.04 6.54 4.77 5.40 5.23 5.03 5.14 5.73 4.80
[10] 5.95 4.76 6.00 6.67 5.24 4.83 5.50 6.19 5.60
[19] 4.64 4.74 6.96 5.72 5.40 5.11 5.65 4.37 5.30
[28] 4.36 4.48 5.45 4.80 4.54 3.99 5.41 4.78 5.04

$Central
 [1]  7.21  5.23  5.67  6.10  5.54  5.85  6.08
 [8]  7.95  7.27  6.64  6.33  6.09  7.01  6.33
[15]  6.78  7.04 10.10  6.91  5.74  6.17  6.79
[22]  5.45  4.95  5.36  6.18  5.68  4.69  4.78
[29]  6.47  8.20  5.28  6.05  5.34  5.71  6.16
[36]  4.40
```

Using `split` and `lapply`

```
> lapply(with(tubeData, split(Excess, Line)), mean)
$Bakerloo
[1] 5.282222

$Central
[1] 6.209167

$`Circle & Ham`
[1] 7.570556

$District
[1] 5.531111

$Jubilee
[1] 5.646944

$Metropolitan
[1] 8.232778

$Northern
[1] 6.611667

$Piccadilly
[1] 6.042778
```

Using `split` and `lapply`

```
> lapply(split(tubeData, tubeData$Line),
+        function(df) lm(log(Excess) ~ Month, data = df))
$Bakerloo

Call:
lm(formula = log(Excess) ~ Month, data = df)

Coefficients:
(Intercept)        Month
   1.745715    -0.004827


$Central

Call:
lm(formula = log(Excess) ~ Month, data = df)

Coefficients:
(Intercept)        Month
   1.900325    -0.004763


$`Circle & Ham`
```

Using `lapply` with vectors

```
> lapply(1:5, rnorm)
[[1]]
[1] 0.5253137

[[2]]
[1] -1.077698 -1.121485

[[3]]
[1]  1.00577204 -0.26125924  0.07158016

[[4]]
[1]  0.02008518 -0.04030489  0.28904921  1.04996314

[[5]]
[1] -1.56370326 -1.09166864 -0.09623626 -0.39157708  1.88692760
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Split processing using `lapply`

```
> largeXs <- lapply(split(tubeData, tubeData$Line), function(df) {
+   xSd <- sd(df$Excess)
+   xMean <- mean(df$Excess)
+   subset(df, Excess > xMean + 2 * xSd)
+ })
> largeXs
$Bakerloo
   Month Excess     Line Type   WhenOpen Length
13    13   6.67 Bakerloo   DT After 1900  Short
21    21   6.96 Bakerloo   DT After 1900  Short

$Central
   Month Excess    Line Type   WhenOpen Length
53    17   10.1 Central   DT After 1900   Long

$`Circle & Ham`
    Month Excess          Line Type    WhenOpen Length
102    30  16.08 Circle & Ham   SS Before 1900 Medium
103    31  11.38 Circle & Ham   SS Before 1900 Medium

$District
    Month Excess     Line Type    WhenOpen Length
138    30   7.61 District   SS Before 1900   Long
143    35   7.09 District   SS Before 1900   Long
```

Split processing using `lapply`

```
> do.call("rbind", largeXs)
                  Month Excess             Line Type    WhenOpen Length
Bakerloo.13          13   6.67         Bakerloo   DT  After 1900  Short
Bakerloo.21          21   6.96         Bakerloo   DT  After 1900  Short
Central              17  10.10          Central   DT  After 1900   Long
Circle & Ham.102     30  16.08    Circle & Ham   SS Before 1900 Medium
Circle & Ham.103     31  11.38    Circle & Ham   SS Before 1900 Medium
District.138         30   7.61        District   SS Before 1900   Long
District.143         35   7.09        District   SS Before 1900   Long
Jubilee              25   7.56         Jubilee   DT  After 1900 Medium
Metropolitan         30  17.60    Metropolitan   SS Before 1900   Long
Northern              8  22.25        Northern   DT Before 1900 Medium
Piccadilly.282       30  19.71       Piccadilly   DT  After 1900   Long
Piccadilly.283       31  12.07       Piccadilly   DT  After 1900   Long
Victoria.298         10   8.45         Victoria   DT  After 1900  Short
Victoria.301         13   7.54         Victoria   DT  After 1900  Short
Waterloo & City      18   3.26 Waterloo & City   DT Before 1900  Short
```

The “Apply” family of Functions

The `sapply` Function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `sapply` Function

- Calls **`lapply`** and simplifies the output

```
> sapply
function (X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
{
    FUN <- match.fun(FUN)
    answer <- lapply(X = X, FUN = FUN, ...)
    if (USE.NAMES && is.character(X) && is.null(names(answer)))
        names(answer) <- X
    if (!identical(simplify, FALSE) && length(answer))
        simplify2array(answer, higher = (simplify == "array"))
    else answer
}
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

sapply vs lapply

```
> myList <- list(Pois = rpois(10, 3), Norm = rnorm(5), Unif = runif(5, 1, 10))
> myList
$Pois
 [1] 4 4 5 3 3 1 3 2 1 2

$Norm
[1] -0.32033487  0.43315902 -0.09740632 -0.38475806 -0.07488000

$Unif
[1] 1.013038 5.436154 6.745381 5.676055 6.879456

> lapply(myList, mean)
$Pois
[1] 2.8

$Norm
[1] -0.08884405

$Unif
[1] 5.150017

> sapply(myList, mean)
       Pois        Norm        Unif
 2.80000000 -0.08884405  5.15001693
```

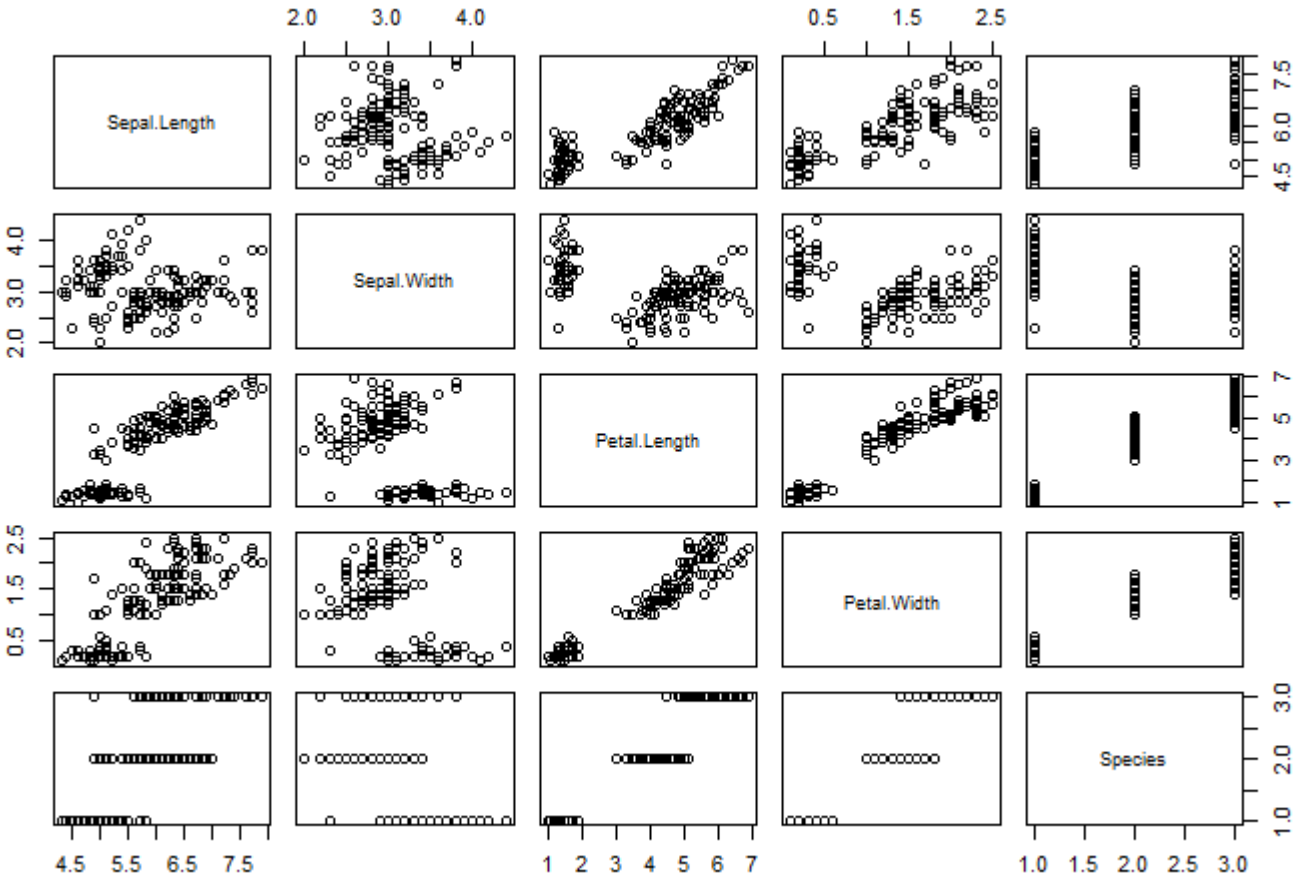
Using `sapply` and `split`

```
> sapply(with(tubeData, split(Excess, Line)), mean)
      Bakerloo         Central    Circle & Ham        District         Jubilee
      5.282222        6.209167        7.570556        5.531111        5.646944
   Metropolitan        Northern      Piccadilly        Victoria Waterloo & City
      8.232778        6.611667        6.042778        5.698889        2.040833
>
> t(sapply(split(tubeData, tubeData$Line),
+        function(df) coef(lm(log(Excess) ~ Month, data = df))))
                (Intercept)        Month
Bakerloo          1.7457150 -0.004827385
Central           1.9003246 -0.004762810
Circle & Ham      2.0340699 -0.001500276
District          1.6356038  0.003570620
Jubilee           1.6430340  0.004162640
Metropolitan      2.0246010  0.003384029
Northern          1.7716312  0.003548646
Piccadilly        1.5899706  0.008537010
Victoria          1.8541875 -0.006782791
Waterloo & City   0.6223147  0.002860249
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Bootstrap style `sapply` example

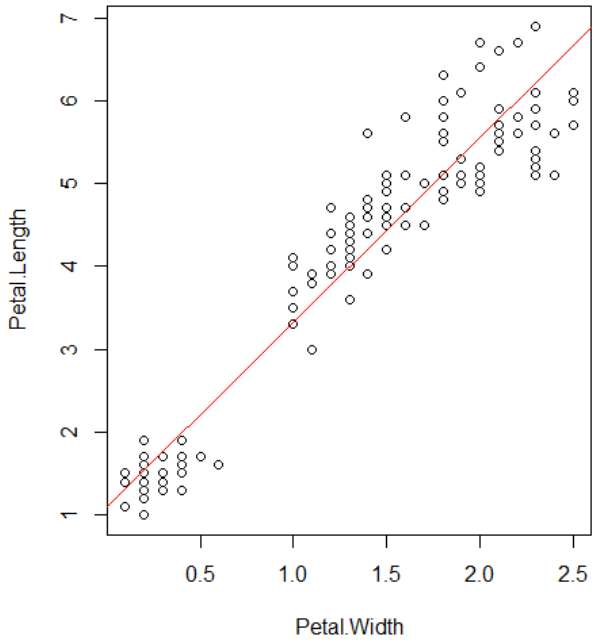
```
> nrow(iris)
[1] 150
> pairs(iris)
```



Bootstrap style `sapply` example

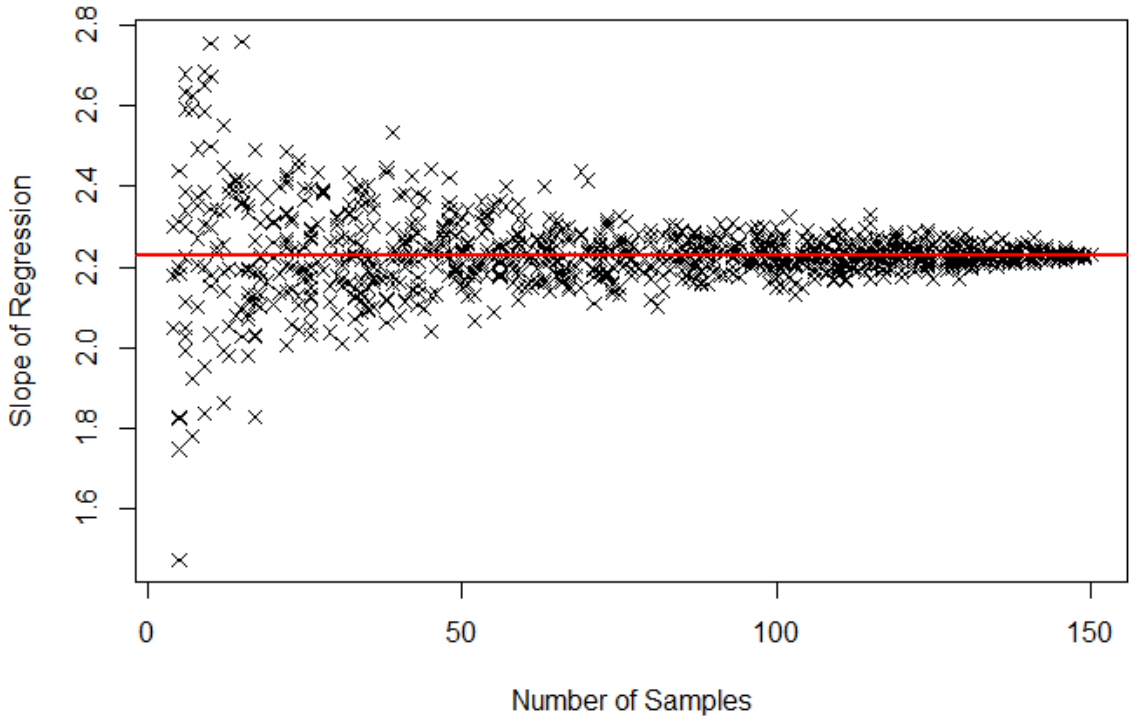
```
> myLm <- lm(Petal.Length ~ Petal.Width, data = iris)
> myLm

Call:
lm(formula = Petal.Length ~ Petal.Width, data = iris)

Coefficients:
(Intercept)  Petal.Width
      1.084        2.230
```

```
> with(iris, plot(Petal.Width, Petal.Length))
> abline(myLm, col = "red")
```



Bootstrap style `sapply` example

```
> nSamples <- sample(4:150, 1000, TRUE)
> theSlopes <- sapply(nSamples, function(i) {
+   coef(lm(Petal.Length ~ Petal.Width,
+           data = iris, subset = sample(1:nrow(iris), i)))[2]
+ })
> plot(nSamples, theSlopes, pch = 4, xlab = "Number of Samples",
+      ylab = "Slope of Regression")
> abline(h = coef(myLm)[2], col = "red", lwd = 2)
```



Using `sapply` with data frames

```
> sapply(tubeData, class)
    Month    Excess      Line      Type  WhenOpen    Length
"integer" "numeric"  "factor"  "factor"  "factor"  "factor"
>
> numIris <- sapply(iris, class) == "numeric"
> sapply(iris [ numIris ], mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The "Apply" family of Functions

The `tapply` Function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `tapply` Function

- Applies a function to a vector BY levels of other factor(s)
- A wrapper to **`lapply`** & **`split`**

```
> args(tapply)
function (X, INDEX, FUN = NULL, ..., simplify = TRUE)
NULL
```

Vector to summarise

Factor(s) by which to summarise

Function to apply

Other arguments to the function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple examples of `tapply`

```
> head(tubeData)
  Month Excess     Line Type   WhenOpen Length
1     1   6.04 Bakerloo   DT After 1900  Short
2     2   6.54 Bakerloo   DT After 1900  Short
3     3   4.77 Bakerloo   DT After 1900  Short
4     4   5.40 Bakerloo   DT After 1900  Short
5     5   5.23 Bakerloo   DT After 1900  Short
6     6   5.03 Bakerloo   DT After 1900  Short
> with(tubeData, tapply(Excess, Line, mean))
      Bakerloo         Central    Circle & Ham        District         Jubilee
      5.282222        6.209167        7.570556        5.531111        5.646944
   Metropolitan        Northern      Piccadilly        Victoria Waterloo & City
      8.232778        6.611667        6.042778        5.698889        2.040833
> with(tubeData, tapply(Excess, list(WhenOpen, Length), mean))
                 Long   Medium    Short
After 1900  6.125972 5.646944 5.490556
Before 1900 6.881944 7.091111 2.040833
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

tapply vs sapply + split

```
> with(tubeData, tapply(Excess, Line, mean))
      Bakerloo        Central   Circle & Ham       District        Jubilee
      5.282222       6.209167       7.570556       5.531111       5.646944
  Metropolitan       Northern     Piccadilly       Victoria Waterloo & City
      8.232778       6.611667       6.042778       5.698889       2.040833
> with(tubeData, sapply(split(Excess, Line), mean))
      Bakerloo        Central   Circle & Ham       District        Jubilee
      5.282222       6.209167       7.570556       5.531111       5.646944
  Metropolitan       Northern     Piccadilly       Victoria Waterloo & City
      8.232778       6.611667       6.042778       5.698889       2.040833
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

When `tapply` goes wrong

```
> with(tubeData, tapply(Excess, Line, range))
$Bakerloo
[1] 3.99 6.96

$Central
[1]  4.4 10.1

$`Circle & Ham`
[1]  5.99 16.08

$District
[1] 3.89 7.61

$Jubilee
[1] 4.02 7.56

$Metropolitan
[1]  5.43 17.60
```

```
> with(tubeData, tapply(Excess, list(WhenOpen, Length), range))
                Long      Medium    Short
After 1900  Numeric,2 Numeric,2 Numeric,2
Before 1900 Numeric,2 Numeric,2 Numeric,2
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The “Apply” family of Functions

The `by` Function

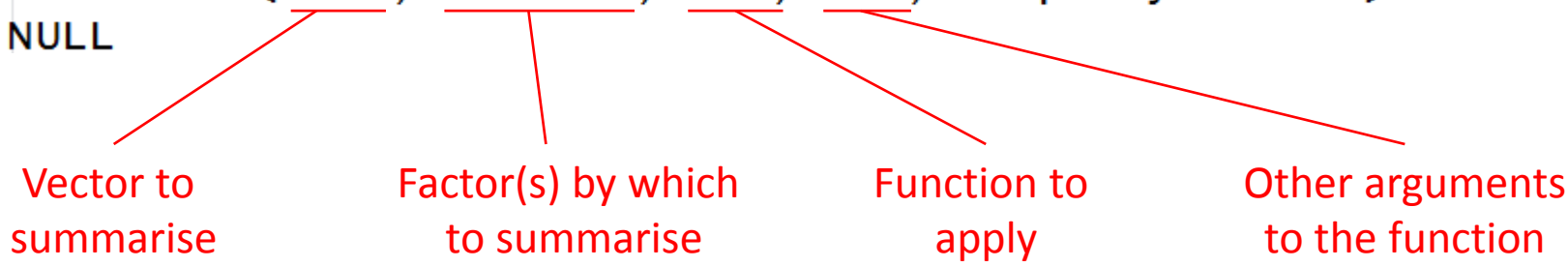
MANGOSOLUTIONS

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `by` Function

- The **`tapply`** function is restricted to vector inputs
- The **`by`** function applies functions to level of a data frame by one or more factors

```
> args(by)
function (data, INDICES, FUN, ..., simplify = TRUE)
NULL
```



Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple `by` example

```
> by( tubeData, tubeData$Line, head )
tubeData$Line: Bakerloo
  Month Excess     Line Type   WhenOpen Length
1     1   6.04 Bakerloo   DT After 1900  Short
2     2   6.54 Bakerloo   DT After 1900  Short
3     3   4.77 Bakerloo   DT After 1900  Short
4     4   5.40 Bakerloo   DT After 1900  Short
5     5   5.23 Bakerloo   DT After 1900  Short
6     6   5.03 Bakerloo   DT After 1900  Short
------------------------------------------------------------
tubeData$Line: Central
   Month Excess    Line Type   WhenOpen Length
37     1   7.21 Central   DT After 1900   Long
38     2   5.23 Central   DT After 1900   Long
39     3   5.67 Central   DT After 1900   Long
40     4   6.10 Central   DT After 1900   Long
41     5   5.54 Central   DT After 1900   Long
42     6   5.85 Central   DT After 1900   Long
------------------------------------------------------------
tubeData$Line: Circle & Ham
   Month Excess         Line Type    WhenOpen Length
73     1   7.50 Circle & Ham   SS Before 1900 Medium
74     2   7.92 Circle & Ham   SS Before 1900 Medium
75     3   8.46 Circle & Ham   SS Before 1900 Medium
76     4   6.94 Circle & Ham   SS Before 1900 Medium
77     5   7.76 Circle & Ham   SS Before 1900 Medium
78     6   8.19 Circle & Ham   SS Before 1900 Medium
------------------------------------------------------------
```

An object of class `by` (example 1)

```
> byOutput1 <- by( tubeData, tubeData$Line, head )
> class(byOutput1)      # What is the class?
[1] "by"
> names(byOutput1)      # What are the names of the object?
 [1] "Bakerloo"         "Central"          "Circle & Ham"     "District"
 [5] "Jubilee"          "Metropolitan"     "Northern"         "Piccadilly"
 [9] "Victoria"         "Waterloo & City"
> byOutput1$Victoria    # Treat it like a list
    Month Excess     Line Type   WhenOpen Length
289     1   6.13 Victoria   DT After 1900  Short
290     2   5.77 Victoria   DT After 1900  Short
291     3   5.90 Victoria   DT After 1900  Short
292     4   5.49 Victoria   DT After 1900  Short
293     5   6.89 Victoria   DT After 1900  Short
294     6   6.15 Victoria   DT After 1900  Short
```

An object of class `by` (example 2)

```
> byOutput2 <- by( tubeData, list(tubeData$WhenOpen, tubeData$Length),
+                  function(df) lm(log(Excess) ~ Month, data = df))
> class(byOutput2)
[1] "by"
> names(byOutput2)     # What are the names of the object?
NULL
> print.default(byOutput2)
            Long    Medium  Short
After 1900  List,12 List,12 List,12
Before 1900 List,12 List,12 List,12
attr(,"call")
by.data.frame(data = tubeData, INDICES = list(tubeData$WhenOpen,
    tubeData$Length), FUN = function(df) lm(log(Excess) ~ Month,
    data = df))
attr(,"class")
[1] "by"
```

The “Apply” family of Functions

The `aggregate` Function

MANGOSOLUTIONS

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `aggregate` Function

- The **`aggregate`** function allows us to apply functions to one or more variables by one or more factors
- It always returns a data frame

```
> args(aggregate.data.frame)
function (x, by, FUN, ..., simplify = TRUE)
NULL
```

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple `aggregate` Example

```
> aggregate( list(MeanExcess = tubeData$Excess),
+            tubeData[c("WhenOpen", "Type")], mean)
     WhenOpen Type MeanExcess
1  After 1900   DT   5.776000
2 Before 1900   DT   4.326250
3 Before 1900   SS   7.111481
```

```
> q5 <- aggregate( list(q5 = tubeData$Excess),
+             tubeData["Line"], quantile, probs = .05)
> q50 <- aggregate( list(q50 = tubeData$Excess),
+                   tubeData["Line"], quantile, probs = .5)
> q95 <- aggregate( list(q95 = tubeData$Excess),
+                    tubeData["Line"], quantile, probs = .95)
> merge(merge(q5, q50), q95)
              Line     q5   q50    q95
1         Bakerloo 4.3675 5.235 6.5725
2          Central 4.7575 6.095 8.0125
3     Circle & Ham 6.2675 7.145 9.2125
4         District 4.6825 5.390 6.9925
5          Jubilee 4.0975 5.670 6.9150
6     Metropolitan 6.3350 7.920 9.9000
7         Northern 4.8850 5.960 9.0950
8       Piccadilly 4.4425 5.430 8.4475
```

Alternative `aggregate` Usage

- The **`aggregate`** function allows us to apply functions to one or more variables by one or more factors
- It always returns a data frame

```
> args(stats:::aggregate.formula)
function (formula, data, FUN, ..., subset, na.action = na.omit)
NULL
```

Formula defining summary structure

Data frame to use

Function to apply

Other arguments to the function

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple `aggregate` Example

```
> aggregate(Excess ~ WhenOpen + Type, tubeData, mean)
     WhenOpen Type   Excess
1  After 1900   DT 5.776000
2 Before 1900   DT 4.326250
3 Before 1900   SS 7.111481
> aggregate(Excess ~ Line, tubeData, quantile, .95)
                Line Excess
1           Bakerloo 6.5725
2            Central 8.0125
3     Circle & Ham 9.2125
4           District 6.9925
5            Jubilee 6.9150
6     Metropolitan 9.9000
7           Northern 9.0950
8         Piccadilly 8.4475
9           Victoria 7.4125
10 Waterloo & City 3.1850
```

The “Apply” family of Functions

Other Functions

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Other “apply” Functions

Function	Usage
`eapply`	Apply a Function Over Values in an Environment
`mapply`	Apply a Function to Multiple List or Vector Arguments
`rapply`	Recursively Apply a Function to a List
`vapply`	Similar to sapply with a pre-specified “template” return value
`replicate`	Wrapper for sapply, for repeated evaluation of an expression

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

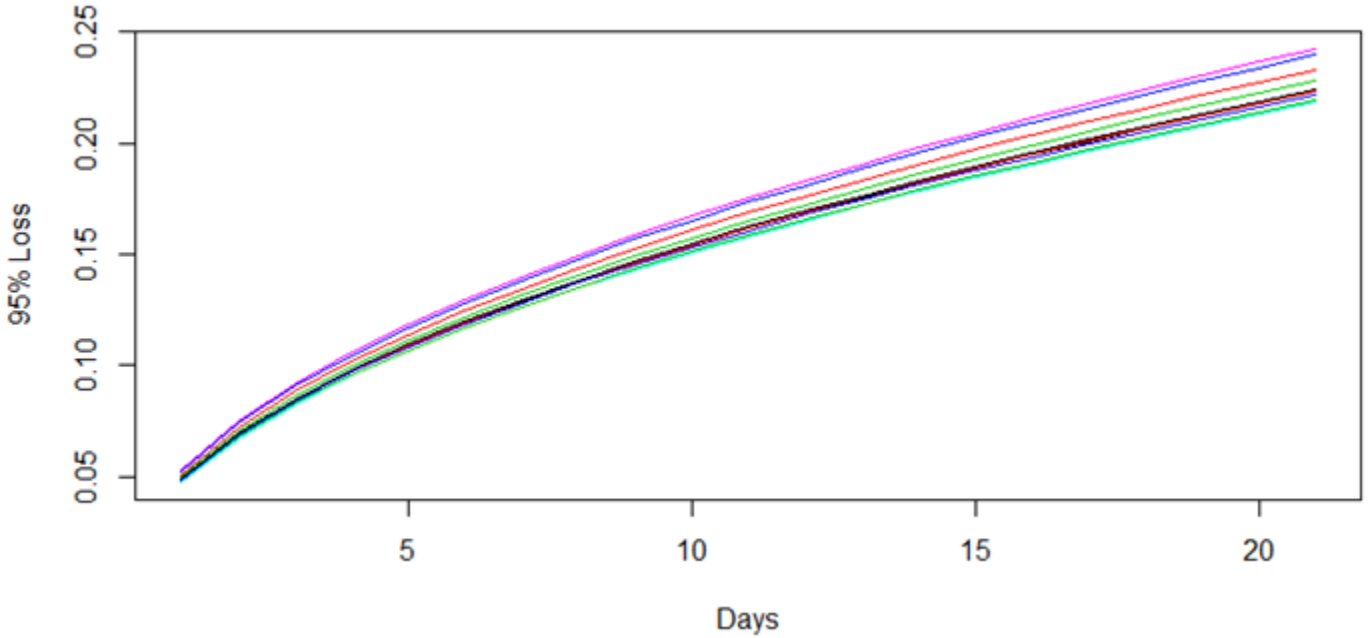
The `eapply` Function

```
> e <- new.env()
> e$a <- rnorm(10)
> e$b <- sample(LETTERS[1:3], 10, TRUE)
> e$c <- tubeData
>
> eapply(e, summary)
$a
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.22100 -0.61980  0.34010  0.04709  0.42680  1.24300

$b
   Length     Class      Mode
       10 character character

$c
     Month           Excess                   Line      Type             WhenOpen      Length
 Min.   : 1.00   Min.   : 1.230   Bakerloo     : 36   DT:252   After 1900 :180   Long   :144
 1st Qu.: 9.75   1st Qu.: 4.987   Central      : 36   SS:108   Before 1900:180   Medium:108
 Median :18.50   Median : 5.730   Circle & Ham: 36                              Short  :108
 Mean   :18.50   Mean   : 5.887   District     : 36
 3rd Qu.:27.25   3rd Qu.: 6.750   Jubilee      : 36
 Max.   :36.00   Max.   :22.250   Metropolitan: 36
                                  (Other)      :144
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `mapply` Function

```
> aVar95 <- function(days, value, mu, sigma) {
+   - (mu - qnorm(.95) * sigma) * value * sqrt(days)
+ }
>
> aVars <- mapply(aVar95, rnorm(10, .03, .001),
+           rnorm(10, .003, .00001), MoreArgs = list(days = 1:21, sigma = 1))
>
> matplot(1:21, aVars, type = "l", lty = 1, ylab = "95% Loss", xlab = "Days")
```



The `rapply` Function

```
> myList
$Pois
[1] 3 1 5 2 4

$Norm
[1]  9.052702 10.739521 10.896779  9.653999  8.217943

$Unif
[1] 0.6790134 0.9032336 0.0255267 0.9890783 0.3028876

> rapply(myList, log)
      Pois1       Pois2       Pois3       Pois4       Pois5       Norm1       Norm2
 1.09861229  0.00000000  1.60943791  0.69314718  1.38629436  2.20306325  2.37393052
      Norm3       Norm4       Norm5       Unif1       Unif2       Unif3       Unif4
 2.38846721  2.26737225  2.10631993 -0.38711439 -0.10177411 -3.66803038 -0.01098181
      Unif5
-1.19439342
> rapply(myList, log, how = "list")
$Pois
[1] 1.0986123 0.0000000 1.6094379 0.6931472 1.3862944

$Norm
[1] 2.203063 2.373931 2.388467 2.267372 2.106320

$Unif
[1] -0.38711439 -0.10177411 -3.66803038 -0.01098181 -1.19439342
```

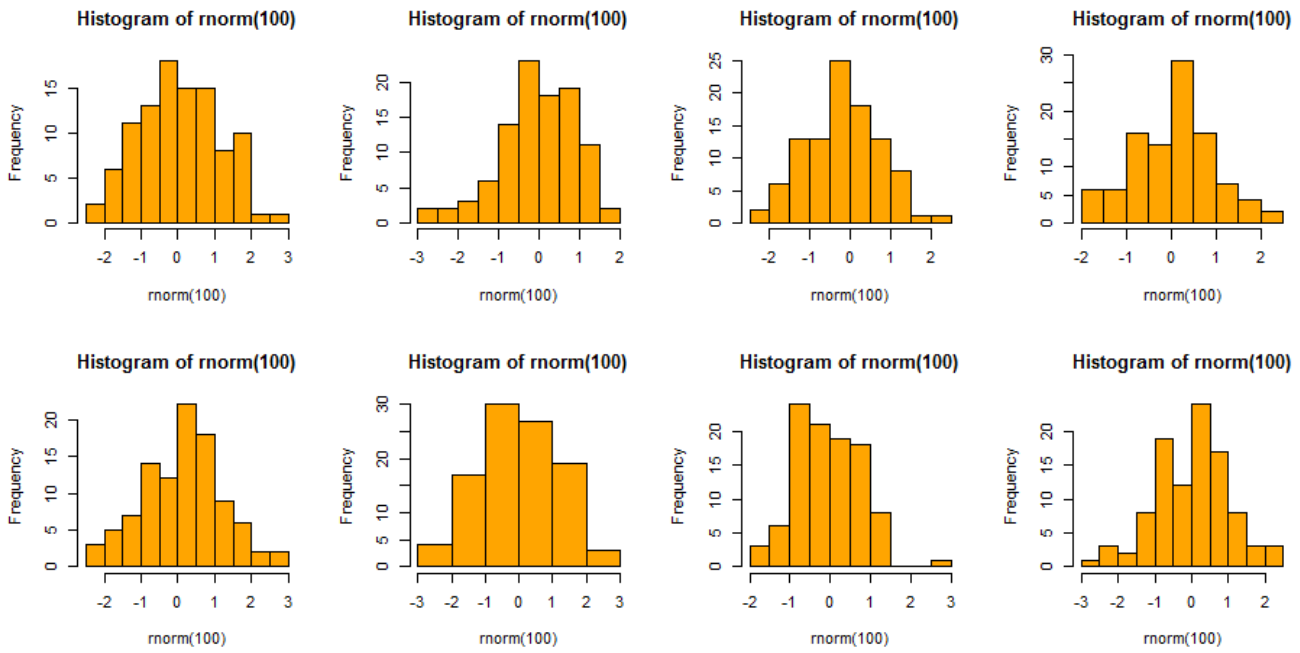
The `vapply` Function

```
> mySummary <- function(vec) {
+   c(mean(vec), median(vec), sd(vec), max(vec), min(vec))
+ }
> sapply(myList, mySummary)
          Pois      Norm      Unif
[1,] 3.000000  9.712189 0.5799479
[2,] 3.000000  9.653999 0.6790134
[3,] 1.581139  1.132447 0.4080014
[4,] 5.000000 10.896779 0.9890783
[5,] 1.000000  8.217943 0.0255267
> vapply(myList, mySummary, c(Mean = 0, Median = 0, SD = 0, Max = 0, Min = 0))
           Pois      Norm      Unif
Mean   3.000000  9.712189 0.5799479
Median 3.000000  9.653999 0.6790134
SD     1.581139  1.132447 0.4080014
Max    5.000000 10.896779 0.9890783
Min    1.000000  8.217943 0.0255267
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The `replicate` Function

```
> replicate(6, rnorm(5))
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
[1,]  2.0769035 -0.26289484 -0.1053884 -1.8673456  0.4678056 -1.9407055
[2,] -0.5713398  0.06148744  1.1788473 -0.8035979 -0.6458064 -0.5358875
[3,] -0.3161910 -0.33996749 -0.8361648 -1.1984086 -0.1525256  0.7135951
[4,]  0.2588852  2.02629406 -0.1389511  0.6295214  0.8562149  1.8601483
[5,] -1.2503121 -1.78676065  1.9432849 -0.8661285  0.1370038 -1.1148261
> par(mfrow = c(2, 4))
> replicate(8, hist(rnorm(100), col = "orange"))
```



The “plyr” package

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

The “plyr” package

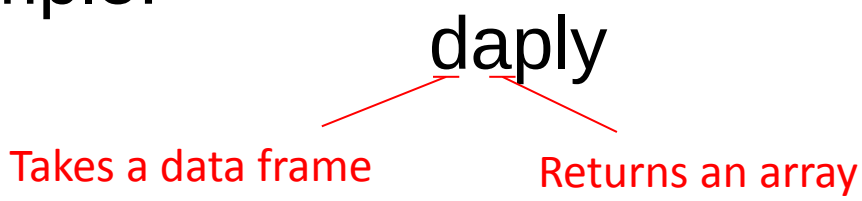
Overview

- Written and maintained by Hadley Wickham
- Widely used in the R community:
  - 158 other packages depend/import/suggest plyr
  - Most downloaded package from the Rstudio cran mirror last month (19,546 downloads)
- Consists of tools for splitting data, applying a function to each part and then combining the results

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Common plyr functions

- Named according to the data structure they split up and the data structure they return.
- Available data structures are:
  - data frame, array, list, multiple inputs, repeat multiple times, _ nothing
- For example:



Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Some plyr alternatives to base R

- Provides an alternative to using the apply family of functions (covered so far today):

Base function	plyr function
apply	aaply/alply
lapply	llply
sapply	laply
tapply	n/a
by	dlply
aggregate	ddply + colwise

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Simple lapply vs sapply example with plyr

```
> myList
$Pois
 [1] 4 4 5 3 3 1 3 2 1 2

$Norm
[1] -0.32033487  0.43315902 -0.09740632 -0.38475806 -0.07488000

$Unif
[1] 1.013038 5.436154 6.745381 5.676055 6.879456
```

plyr

```
> llply(myList, mean)
$Pois
[1] 2.8

$Norm
[1] -0.08884405

$Unif
[1] 5.150017

> laply(myList, mean)
[1]  2.80000000 -0.08884405  5.15001680
```

base

```
> lapply(myList, mean)
$Pois
[1] 2.8

$Norm
[1] -0.08884405

$Unif
[1] 5.150017

> sapply(myList, mean)
      Pois        Norm        Unif
2.80000000 -0.08884405  5.15001680
```


Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Why use plyr?

- Consistent function names make it easier to know which apply-type function is required
- Fast and memory efficient
  - Uses parallelisation through the foreach package
- Additional “helper” functions included:
  - arrange
  - mutate
  - summarise
  - join
  - match_df
  - colwise
  - rename
  - round_any
  - count

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

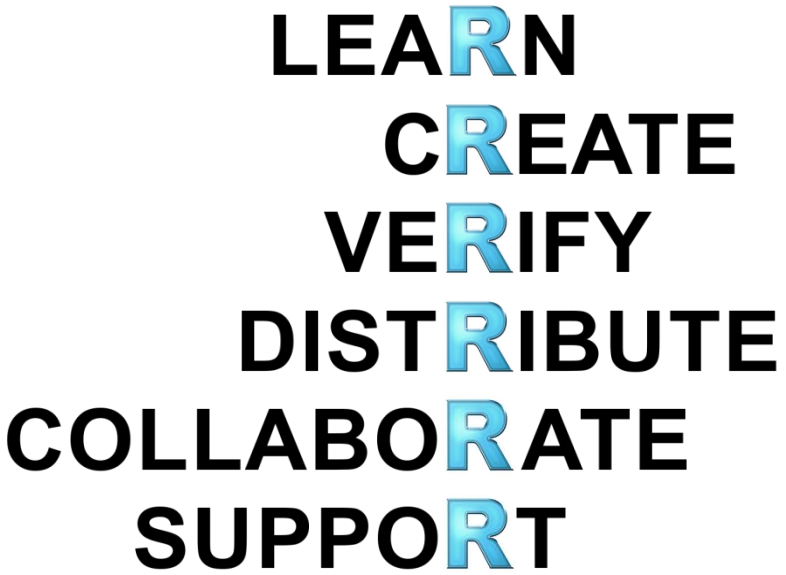
Summary

Richard Pugh, Commercial Director
rpugh@mango-solutions.com

Summary

- This was a quick overview of the apply family of functions
- The key functions are **`apply`**, **`sapply`**, **`lapply`** and **`aggregate`**
- The “plyr” package functions can be used as an alternative to these functions

Richard Pugh, Commercial Director
rpugh@mango-solutions.com



MANGOSOLUTIONS

Richard Pugh, Commercial Director
rpugh@mango-solutions.com