-There’s quizzes and examples at the bottom

Factoring just determines how many groups/categories an object/data-set should have. So if I wanted to split my set into 3 groups, I’d want to factor w/3 levels. If my old object had 12 elements in it, and I wanted to split them evenly I’d need 4 factor elements for each of my 3 factor levels (each of the 3 groups I wanted to create). This can be done with the gl() or factor() functions, which keep the original obejct in-tact only now factorized; OR w/split which breaks up the original object into the desired number of groups(levels).

-factor() allows you to change the label of each level from 1,2,3,etc to whatever you want. See you run\_analysis.r script on my github.

One factor controls levels(grouping) for 1 variable. Eg ‘gender’ could be the factor variable and you’d want to factor it into 2 levels/groups (male/female). ‘Race’ could be another factor variable that maybe has 4 levels (white,black,asian,extraTerrestrial).

It is possible to use split() to split on levels for more than one factor

**lapply:** Loop over a list and evaluate a function on each element

takes three arguments: (1) a list X; (2) either a native OR annoynmous user-defined function; (3) additional arguments to the FUN provided in arg2.

- If X is not a list, it will be coerced to a list using as.list().

-Can take a list of lists and return something like the mean of each internal list

-Returns a list regardless of input class

**sapply**: Same as lapply but try to simplify the result.

-If the result is a list where every element is length 1, then a vector is returned

-If the result is a list where every element is a vector of the same length (> 1), a matrix is returned.

-If it can’t figure things out, a list is returned

**apply:** Apply a function over the margins of an array

-It is most often used to apply a function to the rows or columns of a matrix

-It can be used with general arrays, e.g. taking the average of an array of matrices

Takes 4 arguments: (1) an array X; (2) MARGIN is an integer vector indicating which margins should be “retained”. Think of margins as dimensions. If you want to perform an operation on all columns of an array (‘keep the columns’), you would pass ‘2’ for margin. Pass ‘1’ for margin to perform operations on rows; (3) FUN is a function to be applied to the array X; (4) Additional arguments to the FUN in arg3

To help understand:

rowSums = apply(x, 1, sum)

rowMeans = apply(x, 1, mean)

colSums = apply(x, 2, sum)

colMeans = apply(x, 2, mean)

**tapply**: (short for Table-apply) Apply a function over subsets of a vector. Splits up a vector into little pieces, then applies a function to each of those pieces, then brings the pieces back together and returns them as elements of the output

Arguments: (1)X is a vector; (2)INDEX is a factor variable or a list of factor vars (or else they are coerced to factors); (3)FUN is a function to be applied; (4) ... contains Other Arguments to be passed to the FUN; (4)simplify, should we simplify the result? (T/F).

-X and INDEX can be columns of a data frame as long as INDEX can be coerced into a factor

-How can one calculate the average miles per gallon (mpg) by number of cylinders in the car (cyl)? The cly col can be factorized b/c it’s values are repeating variables (# of cyls)

**tapply(mtcars$mpg, mtcars$cyl, mean)**

**mapply**: Multivariate version of lapply. Whereas lapply takes 1 input, a list, mapply takes many inputs, such as multiple lists or arrays etc.

Arguments: (1) The FUN you want to apply. Weirdly, the number of arguments that the FUN must take must be at least as many as the number of inputs you’re going to apply that FUN to; (2) Moreargs

- Allows you to ‘vectorize’ other functions that don’t support vectorization.

An auxiliary function

**split** (splits objects into smaller pieces) is also useful, particularly in conjunction with lapply.

takes a vector or other objects and splits it into groups determined by a factor or list of factors.

split(x,f,drop=FALSE)

Arguments: (1) x is a vector (or list) or data frame; (2)f is a factor (or coerced to one) or a list of factors; (3)drop indicates whether empty factors levels should be dropped. The interaction() part of the split() call can produce empty levels. setting drop=TRUE will remove all member-less levels from the result that is returned.

-Once you have used split() to split your original object appart into the desired number of factors(groups), you can then use lapply/sapply to perform operations on each of those groups.

A common idiom is split followed by an lapply.

**> lapply(split(x, f), mean)** #split x into the number of groups specified by f; then use lapply to get the mean of each group.

-SEE the example at bottom to split a data frame!

-SEE other example to split levels on more than one factor using interaction()! The str() wrapper around split seems necessary when doing this: str(split(x,list(factor1,factor2)))

**str():** compact display of the internal structure of an R-object. It’s a diagnostic function and similar to ‘summary’. Especially good for displaying abbreviated contents of nested lists. You get roughly one line per basic object.

EXAMPLES

LAPPLY:

#Take list of lists, compute mean for each internal list

x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))

lapply(x, mean)

## $a

## [1] 2.5

## $b

## [1] 0.5261

## $c

## [1] 1.421

## $d

## [1] 4.927

DOING W/sapply

# Get a vector back instead of a list w/elements all of length 1

sapply(x, mean)

a b c d

2.50000000 0.06082667 1.46708277 5.07474950

# x is a list. runif generates random variables, the first arg for runif is the number of random variables to generate. runif is applied to first element in x (which is 1), so it makes 1 random variable. The runif is applied to second element and makes 2 random variables. etc

x <- 1:4

> lapply(x, runif, min = 0, max = 10) #min/max set lower/upper limits for RUNIF not lapply

[[1]]

[1] 3.302142

[[2]]

[1] 6.848960 7.195282

[[3]]

[1] 3.5031416 0.8465707 9.7421014

[[4]]

[1] 1.195114 3.594027 2.930794 2.766946

USE ANNONYMOUS FUNCTIONS

-Just like lambdas except instead of ‘lambda x:’ you use ‘function(x)’

#Extract first column from a list of matrices

>x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))

>lapply(x, function(efc) efc[,1]) #**e**xtract **f**irst **c**olumn. Meaningless annoymous placeholder

#elt is an annoymous function that I’ve defined. It takes x as it’s input and returns elt[,1], or the first column for each item in x.

APPLY EXAMPLES

Quantiles of the rows of a matrix.

# Get the 25th and 75th percentile of every of a matrix

> x <- matrix(rnorm(200), 20, 10) # make a 20rX10c matrix populated with random numbers

> apply(x, 1, quantile, probs = c(0.25, 0.75))

# for each row, 2 numbers will be returned, the 25th percentile and 75th percentile

# will return a matrix w/2 rows (25% & 75%). The number of columns in the returned matrix will be the number of rows from the input matrix

TAPPLY EXAMPLES

Take Group means

**x <- c(rnorm(10), runif(10), rnorm(10, 1))** #make values of from 3 different components

> **f <- gl(3, 10)** #create factors w/3 different levels and repeat each level 10 times; remember

# you need to have f be the same length as x.

> f

[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3

[24] 3 3 3 3 3 3 3

Levels: 1 2 3

> **tapply(x, f, mean)** # take the mean of each of the 3 groups in x.

1 2 3

0.1144464 0.5163468 1.2463678

> **tapply(x, f, range)** # take the range of each of the 3 groups

$‘1‘

[1] -1.097309 2.694970

$‘2‘

[1] 0.09479023 0.79107293

$‘3‘

[1] 0.4717443 2.5887025

SPLITTING A DATA FRAME

**library(datasets)**

> head(airquality)

Ozone Solar.R Wind Temp Month Day

1 41 190 7.4 67 5 1

2 36 118 8.0 72 5 2

3 12 149 12.6 74 5 3

4 18 313 11.5 62 5 4

5 NA NA 14.3 56 5 5

6 28 NA 14.9 66 5 6

# get the mean of each environment variable for every month

> **s <- split(airquality, airquality$Month)** #airqual already contains numerics that can be

#coerced into factor variables (months are numbered 1-12)

**> lapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")]))**

$‘5‘ #means for month 5

Ozone Solar.R Wind

NA NA 11.62258

$‘6‘ #means for month 6

Ozone Solar.R Wind

NA 190.16667 10.26667

$‘7‘

Ozone Solar.R Wind

NA 216.483871 8.941935

#the output from above can be cleaned up by simplifying with sapply() and removing NAs by setting a flag

> **sapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")],**

**na.rm = TRUE))**

5 6 7 8 9

Ozone 23.61538 29.44444 59.115385 59.961538 31.44828

Solar.R 181.29630 190.16667 216.483871 171.857143 167.43333

Wind 11.62258 10.26667 8.941935 8.793548 10.18000

SPLIT LEVELS ON MORE THAN 1 FACTOR

> **x <- rnorm(10)**

**> f1 <- gl(2, 5)** # 5 levels in the first factor

**> f2 <- gl(5, 2)** # 2 levels in the second factor. So 2\*5=10 total combinations of levels

> f1

[1] 1 1 1 1 1 2 2 2 2 2

Levels: 1 2

> f2

[1] 1 1 2 2 3 3 4 4 5 5

Levels: 1 2 3 4 5

#be sure to NOTE that level 1 in f2 != level 1 in f1. The variables/numbers here are annoymous. Remember you could assign different names to them for clarity.

**> interaction(f1, f2)**

[1] 1.1 1.1 1.2 1.2 1.3 2.3 2.4 2.4 2.5 2.5

10 Levels: 1.1 2.1 1.2 2.2 1.3 2.3 1.4 ... 2.5

# 10 different levels, each level with one or more members. Here think of a level as ‘female.asian’. Members would be how many asain females exist in your set. interaction() can create EMPTY levels, for example it’s possible to have ‘female.black’ but maybe you don’t have any members in your set.

#Interaction() doesn’t have to be used, you can just use split(), which calls interaction() interanlly

**str(split(x, list(f1, f2)))**

List of 10

$ 1.1: num [1:2] -0.378 0.445

$ 2.1: num(0)

$ 1.2: num [1:2] 1.4066 0.0166

$ 2.2: num(0)

$ 1.3: num -0.355

$ 2.3: num 0.315

$ 1.4: num(0)

$ 2.4: num [1:2] -0.907 0.723

$ 1.5: num(0)

$ 2.5: num [1:2] 0.732 0.360

#use drop=TRUE to tidy-up the result

**> str(split(x, list(f1, f2), drop = TRUE))**

List of 6

$ 1.1: num [1:2] -0.378 0.445

$ 1.2: num [1:2] 1.4066 0.0166

$ 1.3: num -0.355

$ 2.3: num 0.315

$ 2.4: num [1:2] -0.907 0.723

$ 2.5: num [1:2] 0.732 0.360

**Quiz challenges and answers:**

1.

library(datasets)

data(iris)

In this dataset, what is the mean of 'Sepal.Length' for the species *virginica*?

I solved this 2 ways:

-using split and lapply

> iris$Species <- factor(iris$Species, labels = c("setosa", "versicolor", "virginica"))

> s <- split(iris,iris$Species)

> lapply(s, function(x) colMeans(x[, c("Sepal.Length","Sepal.Width")]))

# using this setup, colMeans needs 2 columns

-using data.tables

a <- dt[,mean(Sepal.Length), by = Species]

OR to just see the one species

p <- dt[grep("virginica",Species),mean(Sepal.Length) ]

# data.frame style

> q <- dt[grep("virginica", iris$Species),mean(Sepal.Length) ]

2.what R code returns a vector of the means of the variables 'Sepal.Length', 'Sepal.Width', 'Petal.Length', and 'Petal.Width'?

**apply(iris[, 1:4], 2, mean)**

3.How can one calculate the average miles per gallon (mpg) by number of cylinders in the car (cyl)?

data(mtcars)

#data.table way

> dt <- data.table(mtcars)

> dt[,mean(mpg),by=cyl]

#Using tapply

tapply(mtcars$mpg, mtcars$cyl, mean)

4. what is the absolute difference between the average horsepower of 4-cylinder cars and the average horsepower of 8-cylinder cars?

#data.table method

> horse4 <- dt[cyl == 4, mean(hp)]

> horse8 <- dt[cyl == 8, mean(hp)]

> horse8 - horse4