StudyR2

PX

3/8/2020

Group Manipulation

```
# apply only works for Matrix, meaning all of the elements must be same type,
# either character, numeric, logical.
# If used on other object like data.frame, it will be converted into matrix first.

# apply(arg1, arg2, arg3)
# arg1: object to work on
# arg2: margin to apply function over
# 1 operate over rows, 2 operate over columns
# arg3: function we want to apply

babyM <- matrix(1:9, nrow = 3)

apply(babyM,1,sum)
```

```
## [1] 12 15 18
```

```
apply(babyM,2,sum)
```

```
## [1]  6 15 24
```

```
babyM[2,1] <- NA
apply(babyM, 1, sum)
```

```
## [1] 12 NA 18
```

```
apply(babyM, 1, sum, na.rm = TRUE)
```

```
## [1] 12 13 18
```

```
# lapply apply a function to each element of a list and return results as a list
babyList <- list(A = matrix(1:9, 3), B =1:5, c = matrix(1:6, 2), D=2)
babyList
```

```
## $A
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $B
## [1] 1 2 3 4 5
##
## $c
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## $D
## [1] 2
```

```
lapply(babyList, sum)
```

```
## $A
## [1] 45
##
## $B
## [1] 15
##
## $c
## [1] 21
##
## $D
## [1] 2
```

```
# mapply
# apply a function to each element of multiple lists.

firstlist <- list(A = matrix(1:12,4), B = matrix(1:16,2), C = 1:7)
secondlist <- list(A = matrix(1:12,4), B = matrix(1:16,4), c =25:15)

mapply(identical,firstlist,secondlist)
```

```
##     A     B     C
##  TRUE FALSE FALSE
```

```
simF <- function(x,y){
  nrow(x)+nrow(y)
}

mapply(simF, firstlist, secondlist)
```

```
## $A
## [1] 8
##
## $B
## [1] 6
##
## $C
## integer(0)
```

```
# Aggregate: sum
# formula consists of a left side and right side separated by tilde(~)
# Left side represents a variable that we want to calculate
# Right side represents one or more varialbes that we want to group the calculation by

require(ggplot2)
```

```
## Loading required package: ggplot2
```

```r
data("diamonds")

#  price should be broken up by cut
aggregate( price ~ cut, diamonds,mean)
```

```
##         cut    price
## 1      Fair 4358.758
## 2      Good 3928.864
## 3 Very Good 3981.760
## 4   Premium 4584.258
## 5     Ideal 3457.542
```

```r
# price grouped by cut and color
aggregate(price ~ cut + color, diamonds, mean, na.rm=TRUE)
```

```
##          cut color    price
## 1       Fair     D 4291.061
## 2       Good     D 3405.382
## 3  Very Good     D 3470.467
## 4    Premium     D 3631.293
## 5      Ideal     D 2629.095
## 6       Fair     E 3682.312
## 7       Good     E 3423.644
## 8  Very Good     E 3214.652
## 9    Premium     E 3538.914
## 10     Ideal     E 2597.550
## 11      Fair     F 3827.003
## 12      Good     F 3495.750
## 13 Very Good     F 3778.820
## 14   Premium     F 4324.890
## 15     Ideal     F 3374.939
## 16      Fair     G 4239.255
## 17      Good     G 4123.482
## 18 Very Good     G 3872.754
## 19   Premium     G 4500.742
## 20     Ideal     G 3720.706
## 21      Fair     H 5135.683
## 22      Good     H 4276.255
## 23 Very Good     H 4535.390
## 24   Premium     H 5216.707
## 25     Ideal     H 3889.335
## 26      Fair     I 4685.446
## 27      Good     I 5078.533
## 28 Very Good     I 5255.880
## 29   Premium     I 5946.181
## 30     Ideal     I 4451.970
## 31      Fair     J 4975.655
## 32      Good     J 4574.173
## 33 Very Good     J 5103.513
## 34   Premium     J 6294.592
## 35     Ideal     J 4918.186
```

```r
# group two variable, use cbind()
aggregate(cbind(price,carat)~cut,diamonds,mean,na.rm=TRUE)
```

```
##         cut    price     carat
## 1      Fair 4358.758 1.0461366
## 2      Good 3928.864 0.8491847
## 3 Very Good 3981.760 0.8063814
## 4   Premium 4584.258 0.8919549
## 5     Ideal 3457.542 0.7028370
```

PLY-R

```
# plyr package "split-apply-combine"
# ddply     input: data.frame output:data.frame
# ldply     input: list output: data.frame
# llply     input: list output: list

require(plyr)
```

```
## Loading required package: plyr
```

```
data("baseball")
head(baseball)
```

```
##            id year stint team lg  g  ab  r  h X2b X3b hr rbi sb cs bb so ibb
## 4   ansonca01 1871     1  RC1    25 120 29 39  11   3  0  16  6  2  2  1  NA
## 44  forceda01 1871     1  WS3    32 162 45 45   9   4  0  29  8  0  4  0  NA
## 68  mathebo01 1871     1  FW1    19  89 15 24   3   1  0  10  2  1  2  0  NA
## 99  startjo01 1871     1  NY2    33 161 35 58   5   1  1  34  4  2  3  0  NA
## 102 suttoez01 1871     1  CL1    29 128 35 45   3   7  3  23  3  1  1  0  NA
## 106 whitede01 1871     1  CL1    29 146 40 47   6   5  1  21  2  2  4  1  NA
##     hbp sh sf gidp
## 4    NA NA NA   NA
## 44   NA NA NA   NA
## 68   NA NA NA   NA
## 99   NA NA NA   NA
## 102  NA NA NA   NA
## 106  NA NA NA   NA
```

```
# set year before 1954, sf all zeros
baseball$sf[baseball$year < 1954] <- 0

# check if it works
any(is.na(baseball$sf))
```

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

```
# set NA hbp to 0
baseball$hbp[is.na(baseball$hbp)] <- 0

# check
any(is.na(baseball$hbp))
```

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

```
# only keep players with at least 50 at bats in a season
baseball <- baseball[baseball$ab >= 50,]

# calcualte OBP
baseball$OBP <- with(baseball, (h+bb+hbp)/(ab+bb+hbp+sf))
```

```
tail(baseball)
```

```
##              id year stint team lg   g  ab  r   h X2b X3b hr rbi sb cs  bb  so
## 89499 claytro01 2007     1  TOR AL  69 189 23  48  14   0  1  12  2  1  14  50
## 89502 cirilje01 2007     1  MIN AL  50 153 18  40   9   2  2  21  2  0  15  13
## 89521 bondsba01 2007     1  SFN NL 126 340 75  94  14   0 28  66  5  0 132  54
## 89523 biggicr01 2007     1  HOU NL 141 517 68 130  31   3 10  50  4  3  23 112
## 89530 ausmubr01 2007     1  HOU NL 117 349 38  82  16   3  3  25  6  1  37  74
## 89533  aloumo01 2007     1  NYN NL  87 328 51 112  19   1 13  49  3  0  27  30
##       ibb hbp sh sf gidp       OBP
## 89499   0   1  3  3    8 0.3043478
## 89502   0   1  3  2    9 0.3274854
## 89521  43   3  0  2   13 0.4800839
## 89523   0   3  7  5    5 0.2846715
## 89530   3   6  4  1   11 0.3180662
## 89533   5   2  0  3   13 0.3916667
```

```
# with function, this allows us to sepcify the columns of a data.frame
# without having to specify the data.frame name each time

# build a function

obp <- function(data)
{
  c(OBP = with(data, sum(h+bb+hbp)/sum(ab+bb+hbp+sf)))
}

# use ddply to calculate career OBP for each player
careerOBP <- ddply(baseball, .variables = "id", .fun = obp)
# sort the result by OBPs
careerOBP <- careerOBP[order(careerOBP$OBP,decreasing = TRUE), ]

head(careerOBP,10)
```

```
##             id       OBP
## 1089 willite01 0.4816861
## 875   ruthba01 0.4742209
## 658  mcgrajo01 0.4657478
## 356  gehrilo01 0.4477848
## 85   bondsba01 0.4444622
## 476  hornsro01 0.4339068
## 184   cobbty01 0.4329655
## 327   foxxji01 0.4290509
## 953  speaktr01 0.4283386
## 191  collied01 0.4251246
```

```
# llply

babyList
```

```
## $A
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
##
## $B
## [1] 1 2 3 4 5
##
## $c
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## $D
## [1] 2
```

```
llply(babyList,sum)
```

```
## $A
## [1] 45
##
## $B
## [1] 15
##
## $c
## [1] 21
##
## $D
## [1] 2
```

```
# useful functions in plyr
# each, apply multiple functions to a function

aggregate(price ~ cut, diamonds, each(mean, median))
```

```
##         cut price.mean price.median
## 1      Fair   4358.758     3282.000
## 2      Good   3928.864     3050.500
## 3 Very Good   3981.760     2648.000
## 4   Premium   4584.258     3185.000
## 5     Ideal   3457.542     1810.000
```

```
# idata.frame, creates a reference to a data.frame
# so that subsetting is much faster and more memory efficient.

system.time(dlply(baseball,"id",nrow))
```

```
##    user  system elapsed
##   0.072   0.000   0.072
```

```
iBaseball <-idata.frame(baseball)

system.time(dlply(iBaseball,"id",nrow))-system.time(dlply(baseball,"id",nrow))
```

```
##    user  system elapsed
##   0.016   0.003   0.019
```

Data Table

```
# using plyr can be very slow.
# data.table extends data.frame
```

```r
# data.table have an index like databases
# fast access values, group by operation and joins

require(data.table)
```

```
## Loading required package: data.table
```

```r
theDT <- data.table(A = 1:10, B = letters[1:10], C = LETTERS[11:20], D = rep(c("One", "Two", "Three"),le

theDT
```

```
##       A B C     D
##  1:  1 a K   One
##  2:  2 b L   Two
##  3:  3 c M Three
##  4:  4 d N   One
##  5:  5 e O   Two
##  6:  6 f P Three
##  7:  7 g Q   One
##  8:  8 h R   Two
##  9:  9 i S Three
## 10: 10 j T   One
```

```r
class(theDT$B)
```

```
## [1] "character"
```

```r
# by  default data.frame turns character data into factors
# while data.table does not
theDT[theDT$A <=3, ]
```

```
##    A B C     D
## 1: 1 a K   One
## 2: 2 b L   Two
## 3: 3 c M Three
```

```r
# the big difference between data.table and data.frame
# to access columns
# column should be specified as a list instead of a  character vector

theDT[, list(A,D, C)]
```

```
##       A     D C
##  1:  1   One K
##  2:  2   Two L
##  3:  3 Three M
##  4:  4   One N
##  5:  5   Two O
##  6:  6 Three P
##  7:  7   One Q
##  8:  8   Two R
##  9:  9 Three S
## 10: 10   One T
```

```r
theDT[, A]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
theDT[,list(A)]
```

```
##       A
##  1:  1
##  2:  2
##  3:  3
##  4:  4
##  5:  5
##  6:  6
##  7:  7
##  8:  8
##  9:  9
## 10: 10
```

```
# if we must specify the column names as characters
# set with to false in the argument
theDT[,"B", with=FALSE]
```

```
##     B
##  1: a
##  2: b
##  3: c
##  4: d
##  5: e
##  6: f
##  7: g
##  8: h
##  9: i
## 10: j
```

```
theDT[, c("A","B"), with=FALSE]
```

```
##       A B
##  1:  1 a
##  2:  2 b
##  3:  3 c
##  4:  4 d
##  5:  5 e
##  6:  6 f
##  7:  7 g
##  8:  8 h
##  9:  9 i
## 10: 10 j
```

```
# show tables
tables()
```

```
##     NAME NROW NCOL MB    COLS KEY
## 1: theDT   10    4  0 A,B,C,D
## Total: 0MB
```

```
class(theDT)
```

```
## [1] "data.table" "data.frame"
```

```
# The Key is used to index the data.table
# speed some calculation
```

```r
# setup the key
setkey(theDT,D)
theDT
```

```
##      A B C     D
##  1:  1 a K   One
##  2:  4 d N   One
##  3:  7 g Q   One
##  4: 10 j T   One
##  5:  3 c M Three
##  6:  6 f P Three
##  7:  9 i S Three
##  8:  2 b L   Two
##  9:  5 e O   Two
## 10:  8 h R   Two
```

```r
# the data have been reordered according to column D
# sorted alphabatically
tables()
```

```
##     NAME NROW NCOL MB    COLS KEY
## 1: theDT   10    4  0 A,B,C,D   D
## Total: 0MB
```

```r
head(theDT)
```

```
##     A B C     D
## 1:  1 a K   One
## 2:  4 d N   One
## 3:  7 g Q   One
## 4: 10 j T   One
## 5:  3 c M Three
## 6:  6 f P Three
```

```r
# add more choices to select data
theDT["One",]
```

```
##     A B C   D
## 1:  1 a K One
## 2:  4 d N One
## 3:  7 g Q One
## 4: 10 j T One
```

```r
theDT[c("One","Three"),]
```

```
##     A B C     D
## 1:  1 a K   One
## 2:  4 d N   One
## 3:  7 g Q   One
## 4: 10 j T   One
## 5:  3 c M Three
## 6:  6 f P Three
## 7:  9 i S Three
```

```r
diamondsDT <- data.table(diamonds)

# set key for more than one column
```

```
setkey(diamondsDT, cut, color)

# access some rows, the special function J

diamondsDT[J("Ideal", "E"),]
```

```
##       carat   cut color clarity depth table price    x    y    z
##    1:  0.23 Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
##    2:  0.26 Ideal     E    VVS2  62.9    58   554 4.02 4.06 2.54
##    3:  0.70 Ideal     E     SI1  62.5    57  2757 5.70 5.72 3.57
##    4:  0.59 Ideal     E    VVS2  62.0    55  2761 5.38 5.43 3.35
##    5:  0.74 Ideal     E     SI2  62.2    56  2761 5.80 5.84 3.62
##   ---
## 3899:  0.70 Ideal     E     SI1  61.7    55  2745 5.71 5.74 3.53
## 3900:  0.51 Ideal     E    VVS1  61.9    54  2745 5.17 5.11 3.18
## 3901:  0.56 Ideal     E    VVS1  62.1    56  2750 5.28 5.29 3.28
## 3902:  0.77 Ideal     E     SI2  62.1    56  2753 5.84 5.86 3.63
## 3903:  0.71 Ideal     E     SI1  61.9    56  2756 5.71 5.73 3.54
```

```
diamondsDT[J("Ideal",c("E", "D"))]
```

```
##       carat   cut color clarity depth table price    x    y    z
##    1:  0.23 Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
##    2:  0.26 Ideal     E    VVS2  62.9    58   554 4.02 4.06 2.54
##    3:  0.70 Ideal     E     SI1  62.5    57  2757 5.70 5.72 3.57
##    4:  0.59 Ideal     E    VVS2  62.0    55  2761 5.38 5.43 3.35
##    5:  0.74 Ideal     E     SI2  62.2    56  2761 5.80 5.84 3.62
##   ---
## 6733:  0.51 Ideal     D    VVS2  61.7    56  2742 5.16 5.14 3.18
## 6734:  0.51 Ideal     D    VVS2  61.3    57  2742 5.17 5.14 3.16
## 6735:  0.81 Ideal     D     SI1  61.5    57  2748 6.00 6.03 3.70
## 6736:  0.72 Ideal     D     SI1  60.8    57  2757 5.75 5.76 3.50
## 6737:  0.75 Ideal     D     SI2  62.2    55  2757 5.83 5.87 3.64
```

```
# data.table aggregation

# the default aggregation function will be slower in data.table

# Let's have one comparison

aggregate(price ~ cut, diamonds, mean)
```

```
##         cut    price
## 1      Fair 4358.758
## 2      Good 3928.864
## 3 Very Good 3981.760
## 4   Premium 4584.258
## 5     Ideal 3457.542
```

```
# in data.table

diamondsDT[, mean(price), by = cut] # notice that name becomes V1
```

```
##          cut       V1
## 1:      Fair 4358.758
## 2:      Good 3928.864
```

```
## 3: Very Good 3981.760
## 4:   Premium 4584.258
## 5:     Ideal 3457.542
```

```
# to fix the name issue

diamondsDT[,list(price = mean(price)), by = cut]
```

```
##          cut    price
## 1:      Fair 4358.758
## 2:      Good 3928.864
## 3: Very Good 3981.760
## 4:   Premium 4584.258
## 5:     Ideal 3457.542
```

```
# more column

diamondsDT[,list(price = mean(price)), by = list(cut, color)]
```

```
##           cut color    price
##  1:      Fair     D 4291.061
##  2:      Fair     E 3682.312
##  3:      Fair     F 3827.003
##  4:      Fair     G 4239.255
##  5:      Fair     H 5135.683
##  6:      Fair     I 4685.446
##  7:      Fair     J 4975.655
##  8:      Good     D 3405.382
##  9:      Good     E 3423.644
## 10:      Good     F 3495.750
## 11:      Good     G 4123.482
## 12:      Good     H 4276.255
## 13:      Good     I 5078.533
## 14:      Good     J 4574.173
## 15: Very Good     D 3470.467
## 16: Very Good     E 3214.652
## 17: Very Good     F 3778.820
## 18: Very Good     G 3872.754
## 19: Very Good     H 4535.390
## 20: Very Good     I 5255.880
## 21: Very Good     J 5103.513
## 22:   Premium     D 3631.293
## 23:   Premium     E 3538.914
## 24:   Premium     F 4324.890
## 25:   Premium     G 4500.742
## 26:   Premium     H 5216.707
## 27:   Premium     I 5946.181
## 28:   Premium     J 6294.592
## 29:     Ideal     D 2629.095
## 30:     Ideal     E 2597.550
## 31:     Ideal     F 3374.939
## 32:     Ideal     G 3720.706
## 33:     Ideal     H 3889.335
## 34:     Ideal     I 4451.970
## 35:     Ideal     J 4918.186
##           cut color    price
```

```r
# more aggregate

diamondsDT[,list(price = mean(price), carat = mean(carat), caratsum = sum(carat)), by = list(cut, color
```

```
##           cut color    price     carat caratsum
##  1:      Fair     D 4291.061 0.9201227   149.98
##  2:      Fair     E 3682.312 0.8566071   191.88
##  3:      Fair     F 3827.003 0.9047115   282.27
##  4:      Fair     G 4239.255 1.0238217   321.48
##  5:      Fair     H 5135.683 1.2191749   369.41
##  6:      Fair     I 4685.446 1.1980571   209.66
##  7:      Fair     J 4975.655 1.3411765   159.60
##  8:      Good     D 3405.382 0.7445166   492.87
##  9:      Good     E 3423.644 0.7451340   695.21
## 10:      Good     F 3495.750 0.7759296   705.32
## 11:      Good     G 4123.482 0.8508955   741.13
## 12:      Good     H 4276.255 0.9147293   642.14
## 13:      Good     I 5078.533 1.0572222   551.87
## 14:      Good     J 4574.173 1.0995440   337.56
## 15: Very Good     D 3470.467 0.6964243  1053.69
## 16: Very Good     E 3214.652 0.6763167  1623.16
## 17: Very Good     F 3778.820 0.7409612  1603.44
## 18: Very Good     G 3872.754 0.7667986  1762.87
## 19: Very Good     H 4535.390 0.9159485  1670.69
## 20: Very Good     I 5255.880 1.0469518  1260.53
## 21: Very Good     J 5103.513 1.1332153   768.32
## 22:   Premium     D 3631.293 0.7215471  1156.64
## 23:   Premium     E 3538.914 0.7177450  1677.37
## 24:   Premium     F 4324.890 0.8270356  1927.82
## 25:   Premium     G 4500.742 0.8414877  2460.51
## 26:   Premium     H 5216.707 1.0164492  2398.82
## 27:   Premium     I 5946.181 1.1449370  1634.97
## 28:   Premium     J 6294.592 1.2930941  1044.82
## 29:     Ideal     D 2629.095 0.5657657  1603.38
## 30:     Ideal     E 2597.550 0.5784012  2257.50
## 31:     Ideal     F 3374.939 0.6558285  2509.20
## 32:     Ideal     G 3720.706 0.7007146  3422.29
## 33:     Ideal     H 3889.335 0.7995249  2490.52
## 34:     Ideal     I 4451.970 0.9130291  1910.97
## 35:     Ideal     J 4918.186 1.0635937   952.98
##           cut color    price     carat caratsum
```