Getting and Cleaning Data: Week 3

Video 3-1: Subsetting and sorting

Subsetting – quick review

- Create a data frame with some data
- Scramble the data so that the rows are not in ascending order
- Assign some NA values to var2

```
set.seed(13435)
X <- data.frame("var1"=sample(1:5),"var2"=sample(6:10),"var3"=sample(11:15))
X <- X[sample(1:5),]; X$var2[c(1,3)] = NA
X
```

```
##   var1 var2 var3
## 1    2   NA   15
## 4    1   10   11
## 2    3   NA   12
## 3    5    6   14
## 5    4    9   13
```

```
X[,1] # this returns only the first column for all rows
```

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

```
X[,"var1"] # same but uses name of column instead of index
```

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

```
X[1:2, "var2"] # selects var2 column for only rows 1 and 2
```

```
## [1] NA 10
```

Logical ands and ors

```
X[X$var1 <= 3 & X$var3 > 11,] # rows where var1 less than or equal to 3 and var3
greater than 11
```

```
##   var1 var2 var3
## 1    2   NA   15
## 2    3   NA   12
```

```
X[X$var1 <= 3 | X$var3 > 15,] # rows where var1 less then or equal to 3 *or* var3
greaterh than 15
```

```
##   var1 var2 var3
## 1    2   NA   15
## 4    1   10   11
## 2    3   NA   12
```

Dealing with missing values

The which function gives the TRUE indices for a logical object. If we just used:

```
X[X$var > 8,]
```

then the presence of NAs would mean that no rows were returned. But the which function will ignore NA values.

```
X[X$var > 8,] # no rows because the data has NAs
```

```
## [1] var1 var2 var3
## <0 rows> (or 0-length row.names)
```

```
X[which(X$var2 > 8),] # ignores NAs
```

```
##   var1 var2 var3
## 4    1   10   11
## 5    4    9   13
```

Sorting

The sort function works only on vectors. So you can use it against one column in a data frame, but it will sort and return only that column, not the entire data frame.

```
sort(X$var1) # sorts X$var1 by the var1 column
```

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

```
sort(X$var1,decreasing=TRUE) # same but in descending order
```

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

```
sort(X$var2) # by default, NAs are not returned
```

```
## [1]  6  9 10
```

```
sort(X$var2, na.last=TRUE) # return all NAs at end
```

```
## [1]  6  9 10 NA NA
```

Ordering

The order function works for an entire data frame.

```
X[order(X$var1),] # sort X by the var1 column
```

```
##   var1 var2 var3
## 4    1   10   11
## 1    2   NA   15
## 2    3   NA   12
## 5    4    9   13
## 3    5    6   14
```

```
X[order(X$var1,X$var3),] # sorty first by var1, then by var3
```

```
##   var1 var2 var3
## 4    1   10   11
## 1    2   NA   15
## 2    3   NA   12
## 5    4    9   13
## 3    5    6   14
```

Ordering with plyr

```
library(plyr)
arrange(X,var1) # sort X by var1
```

```
##   var1 var2 var3
## 1    1   10   11
## 2    2   NA   15
## 3    3   NA   12
## 4    4    9   13
## 5    5    6   14
```

```
arrange(X, desc(var1)) # sort X by var1 in descending order
```

```
##   var1 var2 var3
## 1    5    6   14
## 2    4    9   13
## 3    3   NA   12
## 4    2   NA   15
## 5    1   10   11
```

```
arrange(X, desc(var1), var2) # sort X by var1 descending, then var2
```

```
##   var1 var2 var3
## 1    5    6   14
## 2    4    9   13
## 3    3   NA   12
## 4    2   NA   15
## 5    1   10   11
```

Adding rows and columns

Add a new column, var4, to X.

```
X$var4 <- rnorm(5)
X
```

```
##   var1 var2 var3     var4
## 1    2   NA   15  0.18760
## 4    1   10   11  1.78698
## 2    3   NA   12  0.49669
## 3    5    6   14  0.06318
## 5    4    9   13 -0.53613
```

Another method is cbind. The following example adds a column on the right side of X. If X below appeared as the second parameter, then rnorm(5) would be added as the leftmost column rather than the rightmost.

```
Y <- cbind(X, rnorm(5))
Y
```

```
##   var1 var2 var3     var4 rnorm(5)
## 1    2   NA   15  0.18760  0.62578
## 4    1   10   11  1.78698 -2.45084
## 2    3   NA   12  0.49669  0.08909
## 3    5    6   14  0.06318  0.47839
## 5    4    9   13 -0.53613  1.00053
```

There is also an rbind function that works similarly. If the data frame appears as the first parameter, a new row is added to the end of the data frame, otherwise it is added to the top of the data frame.

Notes and further resources

- R Programming in the data science track
- Andrew Jaffe's lecture notes: http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf

Video 3-2: Summarizing data

Example data set

https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g

```
if(!file.exists("./data")){
  dir.create("./data")
}
if(!file.exists("./data/restaurants.csv")){
  fileUrl <- "https://data.baltimorecity.gov/api/views/k5ry-ef3g/rows.csv?
accessType=DOWNLOAD"
  download.file(fileUrl, destfile="./data/restaurants.csv", method="curl")
}
restData <- read.csv("./data/restaurants.csv")
```

Look at a bit of the data

The head command by default returns the first 6 rows of a data frame. The n argument can adjust the number of rows.

```
head(restData,n=3)
```

```
##    name zipCode neighborhood councilDistrict policeDistrict
## 1   410   21206    Frankford               2   NORTHEASTERN
## 2  1919   21231  Fells Point               1   SOUTHEASTERN
## 3 SAUTE   21224       Canton               1   SOUTHEASTERN
##                          Location.1
## 1 4509 BELAIR ROAD\nBaltimore, MD\n
## 2    1919 FLEET ST\nBaltimore, MD\n
## 3   2844 HUDSON ST\nBaltimore, MD\n
```

The tail command is like the head command, only it shows rows from the end of the data frame.

```
tail(restData,n=3)
```

```
##                  name zipCode  neighborhood councilDistrict
policeDistrict
## 1325 ZINK'S CAF\u0090   21213 Belair-Edison             13
NORTHEASTERN
## 1326     ZISSIMOS BAR   21211       Hampden              7
NORTHERN
## 1327           ZORBAS   21224     Greektown              2
SOUTHEASTERN
##                             Location.1
## 1325 3300 LAWNVIEW AVE\nBaltimore, MD\n
## 1326      1023 36TH ST\nBaltimore, MD\n
## 1327  4710 EASTERN Ave\nBaltimore, MD\n
```

Make summary

The summary command can show different things for different data structures. For a data frame, it will show the counts of factor variables. Quantile values are shown for quantitative variables.

```
summary(restData)
```

```
##                                name          zipCode
neighborhood
##  MCDONALD'S                      :   8   Min.   :-21226   Downtown
:128
##  POPEYES FAMOUS FRIED CHICKEN:   7   1st Qu.: 21202   Fells
Point : 91
##  SUBWAY                          :   6   Median : 21218   Inner
Harbor: 89
##  KENTUCKY FRIED CHICKEN          :   5   Mean   : 21185   Canton
: 81
##  BURGER KING                     :   4   3rd Qu.: 21226   Federal
Hill: 42
##  DUNKIN DONUTS                   :   4   Max.   : 21287   Mount
Vernon: 33
##  (Other)                         :1293                    (Other)
:863
##  councilDistrict      policeDistrict
##  Min.   : 1.00   SOUTHEASTERN:385
##  1st Qu.: 2.00   CENTRAL     :288
##  Median : 9.00   SOUTHERN    :213
##  Mean   : 7.19   NORTHERN    :157
##  3rd Qu.:11.00   NORTHEASTERN: 72
##  Max.   :14.00   EASTERN     : 67
##                  (Other)     :145
##                          Location.1
##  1101 RUSSELL ST\nBaltimore, MD\n:   9
##  201 PRATT ST\nBaltimore, MD\n   :   8
##  2400 BOSTON ST\nBaltimore, MD\n :   8
##  300 LIGHT ST\nBaltimore, MD\n   :   5
##  300 CHARLES ST\nBaltimore, MD\n :   4
##  301 LIGHT ST\nBaltimore, MD\n   :   4
##  (Other)                         :1289
```

More in-depth information

The str function shows the structure behind a data structure.

```
str(restData)
```

```
## 'data.frame':    1327 obs. of  6 variables:
##  $ name           : Factor w/ 1277 levels "1919","19TH
HOLE",..: 9 1 990 3 4 2 6 7 8 5 ...
##  $ zipCode        : int  21206 21231 21224 21211 21223 21218
21205 21211 21205 21231 ...
##  $ neighborhood   : Factor w/ 173 levels
"Abell","Arlington",..: 53 52 18 66 104 33 98 133 98 157 ...
##  $ councilDistrict: int  2 1 1 14 9 14 13 7 13 1 ...
##  $ policeDistrict : Factor w/ 9 levels "CENTRAL","EASTERN",..:
3 6 6 4 8 3 6 4 6 6 ...
##  $ Location.1     : Factor w/ 1210 levels "1000 ALICEANNA
ST\nBaltimore, MD\n",..: 833 324 550 755 484 532 498 525 500 571
...
```

Quantiles of quantitative variables

The following shows quantile values for 0%, 25%, 50%, 75% and 100%. The na.rm parameter is used to ensure that NA values are ignored. Note however that I tried setting na.rm to FALSE and the results were the same. (Apparently the data contains no NAs.)

```
quantile(restData$councilDistrict, na.rm=T)
```

```
##   0%  25%  50%  75% 100%
##    1    2    9   11   14
```

You can specify the quantiles used.

```
quantile(restData$councilDistrict, probs=c(0.5,0.75,0.9))
```

```
## 50% 75% 90%
##   9  11  12
```

Make table

This creates a table of the frequencies for each zip code. useNA="ifany" will put a count for NA values at the end (otherwise they are ignored and unreported). We can see:

- A zip code with a negative value, probably an entry error
- The zip codes with the highest counts

```
table(restData$zipCode, useNA="ifany")
```

```
##
## -21226  21201  21202  21205  21206  21207  21208  21209  21210
21211
##      1    136    201     27     30      4      1      8     23
41
##  21212  21213  21214  21215  21216  21217  21218  21220  21222
21223
##     28     31     17     54     10     32     69      1      7
56
##  21224  21225  21226  21227  21229  21230  21231  21234  21237
21239
##    199     19     18      4     13    156    127      7      1
3
##  21251  21287
##      2      1
```

You can also make a two-dimensional table:

```
table(restData$councilDistrict, restData$zipCode)
```

```
##
##      -21226 21201 21202 21205 21206 21207 21208 21209 21210
21211 21212
##   1       0     0    37     0     0     0     0     0     0
0     0
##   2       0     0     0     3    27     0     0     0     0
0     0
```

```
##    3        0      0      0      0      0      0      0      0      0
0      0
##    4        0      0      0      0      0      0      0      0      0
0     27
##    5        0      0      0      0      0      3      0      6      0
0      0
##    6        0      0      0      0      0      0      0      1     19
0      0
##    7        0      0      0      0      0      0      0      1      0
27       0
##    8        0      0      0      0      0      1      0      0      0
0      0
##    9        0      1      0      0      0      0      0      0      0
0      0
##    10       1      0      1      0      0      0      0      0      0
0      0
##    11       0    115    139      0      0      0      1      0      0
0      1
##    12       0     20     24      4      0      0      0      0      0
0      0
##    13       0      0      0     20      3      0      0      0      0
0      0
##    14       0      0      0      0      0      0      0      0      4
14       0
##
##       21213 21214 21215 21216 21217 21218 21220 21222 21223
21224 21225
##    1        2      0      0      0      0      0      0      7      0
140      1
##    2        0      0      0      0      0      0      0      0      0
54       0
##    3        2     17      0      0      0      3      0      0      0
0      0
##    4        0      0      0      0      0      0      0      0      0
0      0
##    5        0      0     31      0      0      0      0      0      0
0      0
##    6        0      0     15      1      0      0      0      0      0
0      0
##    7        0      0      6      7     15      6      0      0      0
0      0
##    8        0      0      0      0      0      0      0      0      2
0      0
##    9        0      0      0      2      8      0      0      0     53
0      0
##    10       0      0      0      0      0      0      1      0      0
0     18
##    11       0      0      0      0      9      0      0      0      1
0      0
##    12      13      0      0      0      0     26      0      0      0
0      0
##    13      13      0      1      0      0      0      0      0      0
5      0
##    14       1      0      1      0      0     34      0      0      0
0      0
##
##       21226 21227 21229 21230 21231 21234 21237 21239 21251
21287
##    1        0      0      0      1    124      0      0      0      0
0
##    2        0      0      0      0      0      0      1      0      0
0
```

```
##    3        0       1       0        0       0       7       0       0       2
0
##    4        0       0       0        0       0       0       0       3       0
0
##    5        0       0       0        0       0       0       0       0       0
0
##    6        0       0       0        0       0       0       0       0       0
0
##    7        0       0       0        0       0       0       0       0       0
0
##    8        0       2      13        0       0       0       0       0       0
0
##    9        0       0       0       11       0       0       0       0       0
0
##    10      18       0       0      133       0       0       0       0       0
0
##    11       0       0       0       11       0       0       0       0       0
0
##    12       0       0       0        0       2       0       0       0       0
0
##    13       0       1       0        0       1       0       0       0       0
1
##    14       0       0       0        0       0       0       0       0       0
0
```

Check for missing values

```
sum(is.na(restData$councilDistrict)) # count of NAs in a column
```

```
## [1] 0
```

```
any(is.na(restData$councilDistrict)) # TRUE if there is at least one NA in column
```

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

```
all(restData$councilDistrict > 0) # TRUE if every value in column > 0 (will spot
negative value)
```

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

Row and column sums

colSums and rowSums are more convenient, and also often much faster.

```
colSums(is.na(restData)) # get count of missing values per column
```

```
##             name          zipCode     neighborhood councilDistrict
##                0                0                0                0
##   policeDistrict       Location.1
##                0                0
```

```
all(colSums(is.na(restData))==0) # TRUE if none of the columns have any NAs
```

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

I think that the way to start would be to check all values:

```
any(is.na(restData))
```

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

If this returns TRUE, then check on a per-column or per-row basis.

Values with specific characteristics

The %in% operator looks for instances of values in the expression on the right in the expression on the left.

```
table(restData$zipCode %in% c("21212")) # determine number of restaurants in 21212
```

```
##
## FALSE  TRUE
##  1299    28
```

```
table(restData$zipCode %in% c("21212","21213")) # determine num. restaurants in 2
zips
```

```
##
## FALSE  TRUE
##  1268    59
```

```
head(restData[restData$zipCode %in% c("21212", "21213"),]) # return rows in the 2
zips
```

```
##                          name zipCode
neighborhood
## 29           BAY ATLANTIC CLUB   21212
Downtown
## 39                 BERMUDA BAR   21213             Broadway
East
## 92                   ATWATER'S   21212 Chinquapin Park-
Belvedere
## 111 BALTIMORE ESTONIAN SOCIETY   21213        South Clifton
Park
## 187                  CAFE ZEN   21212
Rosebank
## 220        CERIELLO FINE FOODS   21212 Chinquapin Park-
Belvedere
##     councilDistrict policeDistrict
Location.1
## 29               11        CENTRAL    206 REDWOOD
ST\nBaltimore, MD\n
## 39               12        EASTERN    1801 NORTH
AVE\nBaltimore, MD\n
## 92                4       NORTHERN 529 BELVEDERE
AVE\nBaltimore, MD\n
## 111              12        EASTERN    1932 BELAIR
RD\nBaltimore, MD\n
## 187               4       NORTHERN 438 BELVEDERE
AVE\nBaltimore, MD\n
## 220               4       NORTHERN 529 BELVEDERE
AVE\nBaltimore, MD\n
```

Cross tabs

```
data(UCBAdmissions)
DF = as.data.frame(UCBAdmissions)
summary(DF)
```

```
##        Admit       Gender   Dept       Freq
##  Admitted:12   Male  :12   A:4   Min.   :  8
##  Rejected:12   Female:12   B:4   1st Qu.: 80
##                            C:4   Median :170
##                            D:4   Mean   :189
##                            E:4   3rd Qu.:302
##                            F:4   Max.   :512
```

```
xt <- xtabs(Freq ~ Gender + Admit, data=DF)
xt
```

```
##          Admit
## Gender    Admitted Rejected
##   Male        1198     1493
##   Female       557     1278
```

Flat tables

If there are many variables, the output of xtabs can be difficult to examine. Note below that "." in a formula means "all columns except the one to the left of ~".

```
warpbreaks$replicate <- rep(1:9, len=54)
xt <- xtabs(breaks ~ ., data=warpbreaks)
xt
```

```
## , , replicate = 1
##
##     tension
## wool  L  M  H
##    A 26 18 36
##    B 27 42 20
##
## , , replicate = 2
##
##     tension
## wool  L  M  H
##    A 30 21 21
##    B 14 26 21
##
## , , replicate = 3
##
##     tension
## wool  L  M  H
##    A 54 29 24
##    B 29 19 24
##
## , , replicate = 4
##
##     tension
## wool  L  M  H
##    A 25 17 18
##    B 19 16 17
##
## , , replicate = 5
##
##     tension
## wool  L  M  H
##    A 70 12 10
##    B 29 39 13
##
## , , replicate = 6
##
##     tension
## wool  L  M  H
##    A 52 18 43
##    B 31 28 15
##
## , , replicate = 7
##
##     tension
## wool  L  M  H
##    A 51 35 28
##    B 41 21 15
##
## , , replicate = 8
##
##     tension
## wool  L  M  H
##    A 26 30 15
##    B 20 39 16
##
## , , replicate = 9
```

```
##
##      tension
## wool  L  M  H
##    A 67 36 26
##    B 44 29 28
```

So we can use ftable to create “flat” tables.

```
ftable(xt)
```

```
##                replicate  1  2  3  4  5  6  7  8  9
## wool tension
## A    L                   26 30 54 25 70 52 51 26 67
##      M                   18 21 29 17 12 18 35 30 36
##      H                   36 21 24 18 10 43 28 15 26
## B    L                   27 14 29 19 29 31 41 20 44
##      M                   42 26 19 16 39 28 21 39 29
##      H                   20 21 24 17 13 15 15 16 28
```

Size of a data set

```
fakeData <- rnorm(1e5)
object.size(fakeData)
```

```
## 800040 bytes
```

```
print(object.size(fakeData), units="Mb")
```

```
## 0.8 Mb
```

Video 3-3: Creating new variables

Why create new variables?

- Often the raw data won't have a value you're looking for
- You will need to transform the data to get the values you need
- Usually you add those values to the data frame you're working with
- Common variables to create
    - Missingness indicators
    - “Cutting up” (binning) quantitative variables
    - Applying transforms

Example data set

The Baltimore restaurant data, same as in video 3-2.

Creating sequences

Sequences can be useful for creating indexes for your data set.

```
s1 <- seq(1,10,by=2) # creates sequence from 1 up to 10 in step size of 2
s1
```

```
## [1] 1 3 5 7 9
```

```
s2 <- seq(1,10,length=32) # creates sequence from 1 to exactly 10, spaced evenly
to create 32 values
s2
```

```
##  [1]  1.000  1.290  1.581  1.871  2.161  2.452  2.742  3.032
3.323  3.613
## [11]  3.903  4.194  4.484  4.774  5.065  5.355  5.645  5.935
6.226  6.516
## [21]  6.806  7.097  7.387  7.677  7.968  8.258  8.548  8.839
9.129  9.419
## [31]  9.710 10.000
```

```
x <- c(1,3,8,25,100); seq(along=x) # creates a sequence for each element in x
```

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

Subsetting variables

```
restData$nearMe <- restData$neighborhood %in% c("Roland Park", "Homeland")
table(restData$nearMe)
```

```
##
## FALSE  TRUE
##  1314    13
```

Creating binary variables

(It seems to me that the following should actually check for restData$zipCode <= 0, or even something like restData$zipCode <= 9999, assuming that zip codes cannot have a first digit of 0. Also, none of these zip code examples handle the “extended” zip codes.)

```
restData$zipWrong <- ifelse(restData$zipCode < 0, TRUE, FALSE) # look for
"negative" zip codes
table(restData$zipWrong, restData$zipCode < 0)
```

```
##
##         FALSE TRUE
##   FALSE  1326    0
##   TRUE      0    1
```

Creating categorical variables

The cut function can be used to “bin” the values of a quantitative variable into a factor.

I find the example rather strange, as if zip codes were numeric values.

```
restData$zipGroups <- cut(restData$zipCode, breaks=quantile(restData$zipCode))
table(restData$zipGroups, restData$zipCode)
```

```
##
##                           -21226 21201 21202 21205 21206 21207
21208 21209
##   (-2.123e+04,2.12e+04]       0   136   201     0     0     0
0     0
##   (2.12e+04,2.122e+04]        0     0     0    27    30     4
1     8
##   (2.122e+04,2.123e+04]       0     0     0     0     0     0
0     0
##   (2.123e+04,2.129e+04]       0     0     0     0     0     0
0     0
##
##                           21210 21211 21212 21213 21214 21215
21216 21217
##   (-2.123e+04,2.12e+04]      0     0     0     0     0     0
0     0
##   (2.12e+04,2.122e+04]      23    41    28    31    17    54
10    32
##   (2.122e+04,2.123e+04]      0     0     0     0     0     0
0     0
##   (2.123e+04,2.129e+04]      0     0     0     0     0     0
0     0
##
##                           21218 21220 21222 21223 21224 21225
21226 21227
##   (-2.123e+04,2.12e+04]      0     0     0     0     0     0
0     0
##   (2.12e+04,2.122e+04]      69     0     0     0     0     0
0     0
##   (2.122e+04,2.123e+04]      0     1     7    56   199    19
0     0
##   (2.123e+04,2.129e+04]      0     0     0     0     0     0
18    4
##
##                           21229 21230 21231 21234 21237 21239
21251 21287
##   (-2.123e+04,2.12e+04]      0     0     0     0     0     0
0     0
##   (2.12e+04,2.122e+04]       0     0     0     0     0     0
0     0
##   (2.122e+04,2.123e+04]      0     0     0     0     0     0
0     0
##   (2.123e+04,2.129e+04]     13   156   127     7     1     3
2     1
```

Easier cutting

```
library(Hmisc)
```

```
## Loading required package: grid
## Loading required package: lattice
## Loading required package: survival
## Loading required package: splines
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
##
## The following objects are masked from 'package:plyr':
##
##     is.discrete, summarize
##
## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
```

```
restData$zipGroups <- cut2(restData$zipCode,g=4) # "bin" zipCode into four groups
table(restData$zipGroups)
```

```
##
## [-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]
##            338             375             300             314
```

Creating factor variables

```
restData$zcf <- factor(restData$zipCode)
restData$zcf[1:10]
```

```
##  [1] 21206 21231 21224 21211 21223 21218 21205 21211 21205
21231
## 32 Levels: -21226 21201 21202 21205 21206 21207 21208 21209 ...
21287
```

```
class(restData$zcf)
```

```
## [1] "factor"
```

Levels of factor variables

```
yesno <- sample(c("yes", "no"), size=10, replace=TRUE)
yesnofac <- factor(yesno)
yesnofac # levels default to alphabetical order
```

```
##  [1] no  no  yes yes yes no  yes no  yes no
## Levels: no yes
```

```
yesnofac <- factor(yesno, levels=c("yes", "no")) # specify the levels, and their
order
yesnofac
```

```
##  [1] no  no  yes yes yes no  yes no  yes no
## Levels: yes no
```

```
yesnofac <- factor(yesno)
yesnofac <- relevel(yesnofac, ref="yes") # relevel makes the ref value first,
others pushed down
yesnofac
```

```
##  [1] no  no  yes yes yes no  yes no  yes no
## Levels: yes no
```

```
as.numeric(yesnofac) # returns the level index for each value
```

```
##  [1] 2 2 1 1 1 2 1 2 1 2
```

Cutting produces factor variables

```
library(Hmisc)
restData$zipGroups <- cut2(restData$zipCode,g=4) # "bin" zipCode into four groups
table(restData$zipGroups)
```

```
##
## [-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]
##            338             375             300             314
```

Using the mutate function

The mutate function (from plyr) executives transformations iteratively so that later transformations can use the columns created by earlier transformations (that's from the help for the function). Here, it just adds a column, I think? Note that mutate returns a new data frame based on the one passed in.

```
library(Hmisc); library(plyr)
restData2 <- mutate(restData, zipGroups=cut2(zipCode, g=4))
table(restData2$zipGroups)
```

```
##
## [-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]
##            338             375             300             314
```

Common transforms

- abs(x) absolute value
- sqrt(x) square root
- ceiling(x) round value up to next whole number, so ceiling(3.1) is 4
- floor(x) round value down to next whole number, so floor (3.9) is 3
- round(x, digits=n) rounds to n digits, round(3.457, digits=2) is 3.46
- signif(x, digits=n): signif(3.475,digits=3) is 3.5
- cos(x), sin(x), etc.
- log(x): natural logarithm
- log2(x), log10(x): other common logarithms
- exp(x): exponentiating x

http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf
http://statmethods.net/management/functions.html

Notes and further reading

- Tutorial from developer of plyr: http://plyr.had.co.nz/09-user
- Andrew Jaffe's R notes: http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf

Video 3-4: Reshaping data

The goal is tidy data

1. Each variable forms a column
2. Each obervation forms a row
3. Each table/file stores data about one kind of observation (e.g., people/hospitals)

http://vita.had.co.nz/papers/tidy-data.pdf

Start with reshaping

```
library(reshape2)
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
```

Melting data frames

What the following does is to reshape the data so that we see all of the rows in the original order, but we see only the columns carname, gear and cyl in the first three slots. The fourth column is either “mpg” or “hp”, and the fifth is the corresponding value. The mpg values get listed first (because that was first in the measure.vars parameter) and then all of the “hp” values. So in this case, there will be twice as many rows as in the original data frame; each combination of carname-mpg-cyl will be listed twice.

```
mtcars$carname <- rownames(mtcars)
carMelt <- melt(mtcars, id=c("carname", "gear", "cyl"), measure.vars=c("mpg",
"hp"))
head(carMelt,n=3)
```

```
##          carname gear cyl variable value
## 1      Mazda RX4    4   6      mpg  21.0
## 2  Mazda RX4 Wag    4   6      mpg  21.0
## 3     Datsun 710    4   4      mpg  22.8
```

```
tail(carMelt,n=3)
```

```
##          carname gear cyl variable value
## 62   Ferrari Dino    5   6       hp   175
## 63  Maserati Bora    5   8       hp   335
## 64     Volvo 142E    4   4       hp   109
```

Casting data frames

The following will now show the number of values found for the mpg and hp fields, per distinct value in the cyl field. So when cyl=4, there are 11 mpg values and 11 hp values.

```
cylData <- dcast(carMelt, cyl~variable)
```

```
## Aggregation function missing: defaulting to length
```

```
cylData
```

```
##   cyl mpg hp
## 1   4  11 11
## 2   6   7  7
## 3   8  14 14
```

Of course, we probably want something more useful, like the mean of mpg and hp.

```
cylData <- dcast(carMelt, cyl~variable, mean)
cylData
```

```
##   cyl   mpg     hp
## 1   4 26.66  82.64
## 2   6 19.74 122.29
## 3   8 15.10 209.21
```

Averaging values

I do not see how the following computes averages!

This caclculates sum of the count field per each distinct value in the spray field.

```
head(InsectSprays)
```

```
##    count spray
## 1     10     A
## 2      7     A
## 3     20     A
## 4     14     A
## 5     14     A
## 6     12     A
```

```
tapply(InsectSprays$count,InsectSprays$spray,sum)
```

```
##   A   B   C   D   E   F
## 174 184  25  59  42 200
```

http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/

Another way – split

The next few sections cover the split-apply-combine method.

The split command returns a list. Here, the list is one item per value in the spray field, and each item is a vector containing the values of the count field. The name of each item in the list is the value from the spray field.

```
head(InsectSprays,n=10)
```

```
##    count spray
## 1     10     A
## 2      7     A
## 3     20     A
## 4     14     A
## 5     14     A
## 6     12     A
## 7     10     A
## 8     23     A
## 9     17     A
## 10    20     A
```

```
spIns = split(InsectSprays$count,InsectSprays$spray)
spIns
```

```
## $A
##  [1] 10  7 20 14 14 12 10 23 17 20 14 13
##
## $B
##  [1] 11 17 21 11 16 14 17 17 19 21  7 13
##
## $C
##  [1] 0 1 7 2 3 1 2 1 3 0 1 4
##
## $D
##  [1]  3  5 12  6  4  3  5  5  5  5  2  4
##
## $E
##  [1] 3 5 3 5 3 6 1 1 3 2 6 4
##
## $F
##  [1] 11  9 15 22 15 16 13 10 26 26 24 13
```

Another way – apply

Now we can apply a function to each element in the list. The following produces a list with a sum of the counts for each spray type.

```
sprCount = lapply(spIns,sum)
sprCount
```

```
## $A
## [1] 174
##
## $B
## [1] 184
##
## $C
## [1] 25
##
## $D
## [1] 59
##
## $E
## [1] 42
##
## $F
## [1] 200
```

Another way – combine

The unlist function converts a list into an atomic vector, if possible.

```
unlist(sprCount)
```

```
##   A   B   C   D   E   F
## 174 184  25  59  42 200
```

```
sapply(spIns, sum) # We can combine the apply-combine steps with sapply, which
returns an atomic vector
```

```
##   A   B   C   D   E   F
## 174 184  25  59  42 200
```

Another way – plyr package

The .(spray) notation is apparently equivalent to “spray”. The above will split, apply and combine all in one step.

NOTE: The video shows “summariaze”, which results in an error for me. Some in the forums report that it will work; I had to use “summarise”.

```
ddply(InsectSprays, .(spray),summarise, sum=sum(count))
```

```
##   spray sum
## 1     A 174
## 2     B 184
## 3     C  25
## 4     D  59
## 5     E  42
## 6     F 200
```

Creating a new variable

The video is almost totally incomrehensible about the following. The result is a list of each spray type, and the number of times each spray type is listed, is the number of times it is found in the original data set. The “sum” field for this spray type, in each occurrence, will be the sum of all values for that spray type. How this could possibly be useful to anyone, I don't know.

```
spraySums <- ddply(InsectSprays,.(spray),summarise,sum=ave(count,FUN=sum))
dim(spraySums)
```

```
## [1] 72  2
```

```
head(spraySums)
```

```
##   spray sum
## 1     A 174
## 2     A 174
## 3     A 174
## 4     A 174
## 5     A 174
## 6     A 174
```

More information

- A tutorial from the developer of plyr: http://plyr.had.co.nz/09-user/
- A nice reshape tutorial: http://www.slideshare.net/jeffreybreen/reshaping-data-in-r
- A good plyr primer: http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/
- See also the functions
    - acast: for casting as multidimensional arrays
    - arrange: for faster reording without using order() commands
    - mutate: adding new variable

Video 3-5: Merging data

Peer review experiment data

http://www.plosone.org/article/info:doi/10.1371/journal.pone.0026895

Peer review data

```
if(!file.exists("./data")){
  dir.create("./data")
}
fileUrl1 = "https://dl.dropboxusercontent.com/u/7710864/data/reviews-apr29.csv"
fileUrl2 = "https://dl.dropboxusercontent.com/u/7710864/data/solutions-apr29.csv"
if(!file.exists("./data/reviews.csv")) {
  download.file(fileUrl1,destfile="./data/reviews.csv",method="curl")
}
if(!file.exists("./data/solutions.csv")) {
  download.file(fileUrl2,destfile="./data/solutions.csv",method="curl")
}
reviews = read.csv("./data/reviews.csv")
solutions = read.csv("./data/solutions.csv")
head(reviews,2)
```

```
##   id solution_id reviewer_id      start       stop time_left
accept
## 1  1           3          27 1304095698 1304095758      1754
1
## 2  2           4          22 1304095188 1304095206      2306
1
```

```
head(solutions,2)
```

```
##   id problem_id subject_id      start       stop time_left
answer
## 1  1        156         29 1304095119 1304095169      2343
B
## 2  2        269         25 1304095119 1304095183      2329
C
```

Merging data – merge()

- Merges data frames
- Important parameters: x, y, by, by.x, by.y, all

```
names(reviews)
```

```
## [1] "id"          "solution_id" "reviewer_id" "start"
"stop"
## [6] "time_left"   "accept"
```

```
names(solutions)
```

```
## [1] "id"          "problem_id" "subject_id" "start"      "stop"
## [6] "time_left"  "answer"
```

By default, merge will merge two data frames using all of the column names they have in common.

The following merges reviews and solutions using reviews$solution_id and solutions.id. The all=TRUE is like a SQL outer join; if a value appears in a column in one data frame but not in the other, then a row is added for that value, and all of the columns from the non-matching data frame will have NA. (I think.)

Note that the "id" field shown below is from the reviews data frame. The id field from solutions is not shown, because it is the same as reviews$solution_id (it was merged).

```
mergedData = merge(reviews,solutions,by.x="solution_id",by.y="id",all=TRUE)
head(mergedData)
```

```
##   solution_id id reviewer_id    start.x     stop.x time_left.x
accept
## 1           1  4          26 1304095267 1304095423        2089
1
## 2           2  6          29 1304095471 1304095513        1999
1
## 3           3  1          27 1304095698 1304095758        1754
1
## 4           4  2          22 1304095188 1304095206        2306
1
## 5           5  3          28 1304095276 1304095320        2192
1
## 6           6 16          22 1304095303 1304095471        2041
1
##   problem_id subject_id    start.y     stop.y time_left.y
answer
## 1        156         29 1304095119 1304095169        2343
B
## 2        269         25 1304095119 1304095183        2329
C
## 3         34         22 1304095127 1304095146        2366
C
## 4         19         23 1304095127 1304095150        2362
D
## 5        605         26 1304095127 1304095167        2345
A
## 6        384         27 1304095131 1304095270        2242
C
```

Default – merge all common column names

The intersect function can show which column names the data frames have in common. If we use the default merge, then the merge will be performed on all of these fields. But merging on start, stop and time_left probably doesn't make sense.

Note that NA below is from the first data frame (here, that is reviews) and is from the second.

```
intersect(names(solutions), names(reviews))
```

```
## [1] "id"        "start"     "stop"      "time_left"
```

```
mergedData2 = merge(reviews, solutions, all=TRUE)
head(mergedData2)
```

```
##   id      start       stop time_left solution_id reviewer_id
accept
## 1  1 1304095119 1304095169      2343          NA          NA
NA
## 2  1 1304095698 1304095758      1754           3          27
1
## 3  2 1304095119 1304095183      2329          NA          NA
NA
## 4  2 1304095188 1304095206      2306           4          22
1
## 5  3 1304095127 1304095146      2366          NA          NA
NA
## 6  3 1304095276 1304095320      2192           5          28
1
##   problem_id subject_id answer
## 1        156         29      B
## 2         NA         NA   <NA>
## 3        269         25      C
## 4         NA         NA   <NA>
## 5         34         22      C
## 6         NA         NA   <NA>
```

Using join in the plyr package

Faster, but less full-featured–defaults to left join, can only merge using common names. See help file for more.

So this example has two data frames with the same column name (id).

```
df1 = data.frame(id=sample(1:10),x=rnorm(10))
df2 = data.frame(id=sample(1:10),y=rnorm(10))
arrange(join(df1,df2),id) # merge and then sort
```

```
## Joining by: id
```

```
##    id        x       y
## 1   1  2.15231  1.9963
## 2   2 -1.05017 -1.8964
## 3   3  0.67459 -0.7388
## 4   4 -0.02148  0.9996
## 5   5  0.91804  2.6998
## 6   6 -0.30831  1.7473
## 7   7  0.46534  1.0070
## 8   8  1.48010 -1.0179
## 9   9 -0.29983  0.2366
## 10 10 -0.33971 -0.3394
```

If you have multiple data frames

The join_all (plyr) can join a list of data frames into a single data frame.

```
df1 = data.frame(id=sample(1:10),x=rnorm(10))
df2 = data.frame(id=sample(1:10),y=rnorm(10))
df3 = data.frame(id=sample(1:10),z=rnorm(10))
dfList = list(df1,df2,df3)
join_all(dflist)
```

```
## Error: object 'dflist' not found
```

More on merging data

- The quick R data merging page: http://www.statmethods.net/management/merging.html
- plyr information: http://plyr.had.co.nz/
- Types of joins: http://en.wikipedia.org/wiki/Join_(SQL))