Introducing data frames

Mark Andrews

April 5, 2017

Introduction

Most of time, when we are working with data, we work with *data frames*. Data frames can be seen as similar to spreadsheets, i.e. with multiple rows and multiple columns, and each column representing a variable.

We'll start by reading in a csv file as a data frame:

```
Df <- read.csv('../data/LexicalDecision.csv', header=T)
```

Examining the data frame

If you want to see the data frame in a spreadsheet style do *View(Df)*. The following *head* and *tail* commands allow you to see the top and bottom, respectively, of the data frame, and often that's all you need to know how it is laid out.

```
head(Df) # Gives first 6 rows by default
```

```
##   subject    item accuracy latency valence length frequency
## 1       1   alive        1     498    7.25      5     42.54
## 2       1 bandage        1     716    4.54      7      2.53
## 3       1  bright        1     559    7.50      6     55.40
## 4       1 carcass        1     564    3.34      7      1.40
## 5       1   cheer        1     538    8.10      5      7.81
## 6       1   coast        1     463    5.98      5     47.01
```

```
head(Df, 10) # You can ask for as many rows as you like
```

```
##    subject    item accuracy latency valence length frequency
## 1        1   alive        1     498    7.25      5     42.54
## 2        1 bandage        1     716    4.54      7      2.53
## 3        1  bright        1     559    7.50      6     55.40
## 4        1 carcass        1     564    3.34      7      1.40
## 5        1   cheer        1     538    8.10      5      7.81
## 6        1   coast        1     463    5.98      5     47.01
## 7        1  detail        1     486    5.55      6     62.23
## 8        1   devil        1     562    2.21      5     17.32
## 9        1    door        1     541    5.13      4    253.65
## 10       1    evil        1     507    3.23      4     28.83
```

```
tail(Df) # Last 6 rows
```

```
##      subject    item accuracy latency valence length frequency
## 3903     105   trust        1     579    6.68      5    101.98
## 3904     105  useful        1     511    7.14      6    100.71
## 3905     105 vehicle        1     715    6.27      7     42.23
## 3906     105 village        1    1307    5.92      7    113.40
## 3907     105   watch        1     693    5.78      5     95.57
## 3908     105   world        1     602    6.50      5    590.31
```

```r
tail(Df, 8) # Last 8 rows
```

```
##      subject      item accuracy latency valence length frequency
## 3901     105 toothache        0     757    1.98      9      0.98
## 3902     105       toy        1     500    7.00      3      9.77
## 3903     105     trust        1     579    6.68      5    101.98
## 3904     105    useful        1     511    7.14      6    100.71
## 3905     105   vehicle        1     715    6.27      7     42.23
## 3906     105   village        1    1307    5.92      7    113.40
## 3907     105     watch        1     693    5.78      5     95.57
## 3908     105     world        1     602    6.50      5    590.31
```

We can also use the generic functions *str* and *summary* to get a better understanding of the information in the data frame:

```r
str(Df)
```

```
## 'data.frame':    3908 obs. of  7 variables:
##  $ subject  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ item     : Factor w/ 100 levels "alert","alive",..: 2 3 7 8 11 12 17 18 19 21 ...
##  $ accuracy : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ latency  : int  498 716 559 564 538 463 486 562 541 507 ...
##  $ valence  : num  7.25 4.54 7.5 3.34 8.1 5.98 5.55 2.21 5.13 3.23 ...
##  $ length   : int  5 7 6 7 5 5 6 5 4 4 ...
##  $ frequency: num  42.54 2.53 55.4 1.4 7.81 ...
```

```r
summary(Df)
```

```
##     subject            item         accuracy         latency      
##  Min.   :  1.00   alert  :  40   Min.   :0.0000   Min.   :  38.0  
##  1st Qu.: 20.00   beggar :  40   1st Qu.:1.0000   1st Qu.: 458.0  
##  Median : 46.50   brave  :  40   Median :1.0000   Median : 519.0  
##  Mean   : 49.45   breeze :  40   Mean   :0.9803   Mean   : 575.6  
##  3rd Qu.: 77.00   caress :  40   3rd Qu.:1.0000   3rd Qu.: 609.0  
##  Max.   :105.00   charm  :  40   Max.   :1.0000   Max.   :5049.0  
##                   (Other):3668                                    
##     valence          length        frequency     
##  Min.   :1.850   Min.   :3.000   Min.   :  0.33  
##  1st Qu.:3.320   1st Qu.:4.000   1st Qu.:  5.83  
##  Median :5.220   Median :5.000   Median : 16.00  
##  Mean   :5.016   Mean   :5.353   Mean   : 57.31  
##  3rd Qu.:6.770   3rd Qu.:6.000   3rd Qu.: 64.90  
##  Max.   :8.370   Max.   :9.000   Max.   :590.31  
## 
```

We can use *dim* to see the size of the data frame:

```r
dim(Df)
```

```
## [1] 3908    7
```

Subsetting the data frame

We can slice the data frame by rows, and by columns, and by both simultaneously, to create new data frames that are subsets of the original. Here are some examples:

```
Df[1:10,] # Rows 1 to 10, all cols
```

```
##    subject    item accuracy latency valence length frequency
## 1        1   alive        1     498    7.25      5     42.54
## 2        1 bandage        1     716    4.54      7      2.53
## 3        1  bright        1     559    7.50      6     55.40
## 4        1 carcass        1     564    3.34      7      1.40
## 5        1   cheer        1     538    8.10      5      7.81
## 6        1   coast        1     463    5.98      5     47.01
## 7        1  detail        1     486    5.55      6     62.23
## 8        1   devil        1     562    2.21      5     17.32
## 9        1    door        1     541    5.13      4    253.65
## 10       1    evil        1     507    3.23      4     28.83
```

```
Df[10:20, c(1, 2)] # Rows 10 to 20, cols 1 and 2
```

```
##    subject    item
## 10       1    evil
## 11       1    face
## 12       1     fat
## 13       1    foul
## 14       1   glass
## 15       1 grenade
## 16       1  hatred
## 17       1    heal
## 18       1  kettle
## 19       1    kick
## 20       1    kind
```

```
Df[1:10, c('subject', 'valence')] # Rows 1 to 10, cols 'subject' and 'valence'
```

```
##    subject valence
## 1        1    7.25
## 2        1    4.54
## 3        1    7.50
## 4        1    3.34
## 5        1    8.10
## 6        1    5.98
## 7        1    5.55
## 8        1    2.21
## 9        1    5.13
## 10       1    3.23
```

We can also use the *subset* command to subset the data frame in more interestings ways:

```
Df.new <- subset(Df, latency > 2000) # Only rows where latency takes value greater than 2000
# Return rows where responses are accurate and latency is less than 2000
Df.new <- subset(Df, accuracy == 1 & latency < 2000)
```

Getting and changing variable (column) names

This will return the names of the columns

```
(original.col.names <- names(Df) )
```

```
## [1] "subject"   "item"      "accuracy"  "latency"   "valence"   "length"
## [7] "frequency"
```

and so you could do the following:

```
names(Df)[2] <- 'words' # Rename name of second column
names(Df)[c(2, 3)] <- c('words', 'correct') # Rename names of second and third column
names(Df) <- original.col.names
```

Adding/deleting variables

We can create a new variable (column) simply as follows. This creates a new variable called *loglatency*, which is the logarithm of the latency variable.

```
Df$loglatency <- log(Df$latency)
```

We can delete this, or any other, with

```
Df$loglatency <- NULL
```

As some further example, we could create a new binary variable that indicates if the latency variable is fast, where fast is defined as anything less than 500.

```
Df$fast.rt <- Df$latency < 500
```

and we could then do

```
sum(Df$fast.rt)
```

```
## [1] 1652
```

to see that 1652 of the 3908 reaction times are fast, according to this definition.

To create more interesting categorical variables from continuous ones, we can use the *cut* command. For example, this will cut the *valence* variable into three categories and create a new variable named *valence.category*:

```
# values in [0, 3) are labelled "negative"
# values in [3, 6) are labelled "neutral"
# values in [6, 10) are labelled "positive"
Df$valence.category <- cut(Df$valence,
                             breaks = c(0, 3, 6, 10),
                             labels = c('negative', 'neutral', 'positive'))
```

Aggregations over variables

Often, we want to group observations according to certain categories and apply functions to these grouped data. For example, in this data frame, we might like to group the observations according to the valence category just created and then calculate the mean values of these groups:

```
aggregate(latency ~ valence.category, data=Df, mean)
```

```
##   valence.category  latency
## 1         negative 585.1196
## 2          neutral 578.9767
## 3         positive 565.6504
```

As another example, we could get the mean accuracy and latency by valence category

```
aggregate(cbind(accuracy, latency) ~ valence.category, data=Df, mean)
```

```
##   valence.category  accuracy  latency
## 1         negative 0.9687500 585.1196
## 2          neutral 0.9788274 578.9767
## 3         positive 0.9887218 565.6504
```

Combining and merging data frames

For these examples, we'll first read in some new data sets:

```
lexicon.A <- read.csv('../data/lexiconA.csv', header=T)
lexicon.B <- read.csv('../data/lexiconB.csv', header=T)
lexicon.C <- read.csv('../data/lexiconC.csv', header=T)
behav.data <- read.csv('../data/data.csv', header=T)
```

The data frames *lexicon.A* and *lexicon.C* have the same column names and so we can stack them on top of each other:

```
rbind(lexicon.A, lexicon.C)
```

```
##      word length  pos
## 1     dog      3 noun
## 2    walk      4 verb
## 3   happy      5  adj
## 4 quickly      7  adv
## 5  dragon      6 noun
## 6     cat      3 noun
## 7   mouse      5 noun
```

The data frames *lexicon.A* and *behav.data* have the same number of rows, so we can stack them side by side:

```
cbind(lexicon.A, behav.data)
```

```
##      word length  pos reaction.time accuracy
## 1     dog      3 noun           200        1
## 2    walk      4 verb           300        0
## 3   happy      5  adj           450        1
## 4 quickly      7  adv           500        0
## 5  dragon      6 noun           345        1
```

A more interesting case is where we want to merge values from two data frames according to common variables:

```
merge(lexicon.A, lexicon.B)
```

```
##      word length  pos valence
## 1     dog      3 noun       3
## 2  dragon      6 noun       1
## 3   happy      5  adj       7
## 4 quickly      7  adv       4
## 5    walk      4 verb       3
```