Introduction to Data Extraction: Lab 4-walkthrough

Simon Caton

Fix the row names

```r
rownames(criticResponses) <- c(1:7)
```

Encode the Season attibute in both data.frames

```r
seasons$Season <- factor(seasons$Season, levels = c("Season 1", "Season 2",
            "Season 3", "Season 4", "Season 5", "Season 6", "Season 7"), labels=c(1:7))
criticResponses$Season <- factor(criticResponses$Season, levels = c(1:7), labels = c(1:7))

str(seasons$Season)
```

```
##  Factor w/ 7 levels "1","2","3","4",..: 1 2 3 4 5 6 7
```

Note you can be a little more elegant by replacing:

```r
levels = c("Season 1", "Season 2", "Season 3", "Season 4", "Season 5", "Season 6", "Season 7")
```

With:

```r
levels = paste0("Season ", c(1:7))
levels
```

```
## [1] "Season 1" "Season 2" "Season 3" "Season 4" "Season 5" "Season 6"
## [7] "Season 7"
```

Explore the GoTJson data frame

This one is really on you, but hopefully you at least ran:

```r
str(GoTJson)
```

to see its structure.

Get rid of season 0.

```r
movieDBSeasons <- movieDBSeasons[-1, ]
```

Make the movieDBRatings vector a % score (multiply by 10), and round them

```
movieDBRatings <- round(movieDBRatings * 10)
```

Add averages into the Wikipedia data.frame

In R we can combine data.frames into data.frames. This way would have also worked to merge the two wikipedia tables. However, note that the order is important, as unlike Merge (an insection), this approach is a union of the two data.frames.

```
GoTWikipedia <- data.frame(GoTWikipedia, movieDBRatings)
```

Make two new attributes RT and Critic

```
GoTWikipedia$RT <- as.numeric(substr(GoTWikipedia$Critical.response....Rotten.Tomatoes, 1, 2))
GoTWikipedia$Critic <- as.numeric(substr(GoTWikipedia$Critical.response....Metacritic, 1, 2))
```

Repeat for the other 5 seasons making a factor vector s3gender, . . . , s7gender

```
s3gender <- factor(getGender(GoTJsonSeason3$episodes$guest_stars))
s4gender <- factor(getGender(GoTJsonSeason4$episodes$guest_stars))
s5gender <- factor(getGender(GoTJsonSeason5$episodes$guest_stars))
s6gender <- factor(getGender(GoTJsonSeason6$episodes$guest_stars))
s7gender <- factor(getGender(GoTJsonSeason7$episodes$guest_stars))
```

If you wanted to do this over mulitple lines per season you could, e.g.:

```
s3gender <- getGender(GoTJsonSeason3$episodes$guest_stars)
s3gender <- factor(s3gender)
```

Compute the number of each gender and instantiate attibutes in our main GoT data.frame

Ok, let's do this for just one season to see the idea of how this is being done, then we'll make a loop to do all of them.

```
t <- table(factor(s1gender))
GoTWikipedia1 <- GoTWikipedia[1, ]
GoTWikipedia1$MaleExtras <- t[3]
GoTWikipedia1$FemaleExtras <- t[2]
GoTWikipedia1
```

Ok, so we make a table capturing the number of occurences of each value in our factor. We extract the first row of the data.frame for practice purposes, as we don't want to overwrite the real first row, in case we make a mistake.

We know that 0 is unknown, 1 is female, and 2 is male. Zero is the 1st entry of our table, One is the 2nd entry, and Two is the third entry. Thus we grab the corresponding values and instantiate a new attribute with the corresponding values.

You could repeat this for each season if you so wished, either by repeating the code 6 more times, or by making a function that returns the necessary values.

However, lets look at another way:

```
gender.list <- list(s1gender, s2gender, s3gender, s4gender, s5gender, s6gender, s7gender)
```

This will make a list of 7 vectors (one per Season). Lists are useful, as R has built in functions to operate on lists. One such example is lapply (apply a function to a list). So if we apply a function to the list we have just built above, we can compute the gender variables in fewer lines of code:

```
gender.male <- unlist(lapply(gender.list, FUN = function(x) {
  length(x[x == 2])
  }))
```

We are doing quite a lot in this one line (it's displayed on 3 to aid readability).

First, lapply works by receiving a list (gender.list in this case as input) it then loops over all values in the list, and instantiates the variable *x* with each value in the list (the choice of variable name is irrelevant – *x* is just convenient). So in this case, it loops 7 times, storing one of our gender vectors in *x*. Think of it like a for loop in Java something like *for (Vector v : gender.list)*.

The second part of lapply is the function that we apply to each instance of *x*, in this case we declare a new function as we have before, all our function does is compute the length of a vector. However, we give this function a vector of all the values of *x* that are equal to 2. Or put more simply, our function returns the number of 2s in *x*.

Finally, lapply returns a list (of size 7), where each item in the list is the number of 2s. However, we want a vector as we will add this to our data.frame. The function *unlist* converts a list to a vector (it's not the same as casting).

We're done :) well almost, we need to do the same for female, unknown, and total, and then add them to our data.frame:

```
gender.female <- unlist(lapply(gender.list, FUN = function(x) {
  length(x[x == 1])
  }))

gender.unknown <- unlist(lapply(gender.list, FUN = function(x) {
  length(x[x == 0])
  }))

totalExtras <- unlist(lapply(gender.list, FUN = function(x) {length(x)}))

GoTWikipedia$maleExtras <- gender.male
GoTWikipedia$femaleExtras <- gender.female
GoTWikipedia$unknownGenderExtras <- gender.unknown
GoTWikipedia$totalExtras <- totalExtras
```

Plot Gender against Season

Here we can reuse the melt function again.

```r
data <- data.frame(GoTWikipedia[, c("Season", "maleExtras", "femaleExtras", "unknownGenderExtras")])
data$Season <- as.numeric(data$Season)
Molten <- melt(data, id.vars = "Season")
ggplot(Molten, aes(x = Season, y = value, colour = variable)) + geom_line() +
  scale_x_continuous(breaks=c(1:7)) + ylab("Gender of extras (num)")
```