

Module 10

Lists and functions

Andrew Jaffe
Instructor

Review of Week Thus Far

- Reading data into R `{read.table() }`
- Subsetting vectors `{[ind]}` and data frames `{[row,col]}`
- Creating logical tests for variables in your dataset
- Creating new variables
 - Binary
 - Categorical
 - Transforming, e.g. `log()`, `exp()`, `sqrt()`
- Summarizing variables
 - Basic statistics, e.g. `mean()`, `sum()`, `sd()`
 - One variable by levels of another variable: `tapply()`
 - Basic exploratory plots

You should feel comfortable doing most of the above

Data

- We will be using multiple data sets in this lecture:
 - Salary, Monument, Circulator, and Restaurant from OpenBaltimore:
<https://data.baltimorecity.gov/browse?limitTo=datasets>
 - Gap Minder - very interesting way of viewing longitudinal data
 - Data is here - <http://www.gapminder.org/data/>
 - http://spreadsheets.google.com/pub?key=rMsQHawTObBb6_U2ESjKXYw&output=xls

Lists

- One other data type that is the most generic are **lists**.
- Can be created using `list()`
- Can hold vectors, strings, matrices, models, list of other list, lists upon lists!
- Can reference data using `$` (if the elements are named), or using `[]`, or `[[[]]`

```
> mylist <- list(letters=c("A", "b", "c"), numbers=1:3, matrix(1:25, ncol=5))
```

List Structure

```
> head(mylist)
```

```
$letters
```

```
[1] "A" "b" "c"
```

```
$numbers
```

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

```
[[3]]
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

List referencing

```
> mylist[1] # returns a list
```

```
$letters  
[1] "A" "b" "c"
```

```
> mylist["letters"] # returns a list
```

```
$letters  
[1] "A" "b" "c"
```

List referencing

```
> mylist[[1]] # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

```
> mylist$letters # returns vector
```

```
[1] "A" "b" "c"
```

```
> mylist[["letters"]] # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

List referencing

You can also select multiple lists with the single brackets.

```
> mylist[1:2] # returns a list
```

```
$letters
```

```
[1] "A" "b" "c"
```

```
$numbers
```

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


List referencing

You can also select down several levels of a list at once

```
> mylist$letters[1]
```

```
[1] "A"
```

```
> mylist[[2]][1]
```

```
[1] 1
```

```
> mylist[[3]][1:2,1:2]
```

```
      [,1] [,2]  
[1,]    1    6  
[2,]    2    7
```

Splitting Data Frames

The `split()` function is useful for splitting `data.frames`

"`split` divides the data in the vector `x` into the groups defined by `f`. The replacement forms replace values corresponding to such a division. `unsplit` reverses the effect of `split`."

```
> dayList = split(circ,circ$day)
```

Splitting Data Frames

Here is a good chance to introduce `lapply`, which performs a function within each list element:

```
> # head(dayList)
> lapply(dayList, head, n=2)
```

```
$Friday
  day      date orangeBoardings orangeAlightings orangeAverage
5  Friday 01/15/2010          1645             1643          1644.0
12 Friday 01/22/2010          1401             1388          1394.5
  purpleBoardings purpleAlightings purpleAverage greenBoardings
5                NA                NA                NA                NA
12               NA                NA                NA                NA
  greenAlightings greenAverage bannerBoardings bannerAlightings
5                NA                NA                NA                NA
12               NA                NA                NA                NA
  bannerAverage daily
5                NA 1644.0
12               NA 1394.5

$Monday
  day      date orangeBoardings orangeAlightings orangeAverage
1  Monday 01/11/2010           877             1027           952.0
8  Monday 01/18/2010           999             1000           999.5
  purpleBoardings purpleAlightings purpleAverage greenBoardings
1                NA                NA                NA                NA
8                NA                NA                NA                NA
  greenAlightings greenAverage bannerBoardings bannerAlightings
1                NA                NA                NA                NA
```

```
> # head(dayList)
> lapply(dayList, dim)
```

```
$Friday
[1] 146 15
```

```
$Monday
[1] 147 15
```

```
$Saturday
[1] 146 15
```

```
$Sunday
[1] 146 15
```

```
$Thursday
[1] 146 15
```

```
$Tuesday
[1] 147 15
```

```
$Wednesday
[1] 147 15
```

Writing your own functions

This is a brief introduction - we will cover more on Friday. The syntax is:

```
functionName = function(inputs) {  
< function body >  
return(value)  
}
```

Then you would run the 4 lines of the code, which adds it to your workspace.

Writing your own functions

Here we will write a function that returns the second element of a vector:

```
> return2 = function(x) {  
+   return(x[2])  
+ }  
> return2(c(1,4,5,76))
```

```
[1] 4
```

Writing your own functions

Note that your function will automatically return the last line of code run:

```
> return2a = function(x) {  
+   x[2]  
+ }  
> return2a(c(1,4,5,76))
```

```
[1] 4
```

And if your function is really one line or evaluation, like here, you do not need the curly brackets, and you can put everything on one line:

```
> return2b = function(x) x[2]  
> return2b(c(1,4,5,76))
```

```
[1] 4
```

Writing your own functions

Also note that functions can take multiple inputs. Maybe you want users to select which element to extract

```
> return2c = function(x,n) x[n]  
> return2c(c(1,4,5,76), 3)
```

```
[1] 5
```


Writing a simple function

Let's write a function, `sqdif`, that:

1. takes two numbers `x` and `y` with default values of 2 and 3.
2. takes the difference
3. squares this difference
4. then returns the final value

Writing a simple function

```
> sqdif <- function(x=2,y=3){  
+   (x-y)^2  
+ }  
>  
> sqdif()
```

```
[1] 1
```

```
> sqdif(x=10,y=5)
```

```
[1] 25
```

```
> sqdif(10,5)
```

```
[1] 25
```

Writing your own functions

Try to write a function called `top()` that takes a `matrix` or `data.frame`, and returns the first `n` rows and columns, with the default value of `n=5`.

Writing your own functions

Try to write a function called `top()` that takes a `matrix` or `data.frame`, and returns the first `n` rows and columns

```
> top = function(mat,n=5) mat[1:n,1:n]
> my.mat = matrix(1:1000,nr=100)
> top(my.mat) #note that we are using the default value for n here
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1  101  201  301  401
[2,]    2  102  202  302  402
[3,]    3  103  203  303  403
[4,]    4  104  204  304  404
[5,]    5  105  205  305  405
```

Custom functions in `apply`

You can use any function you want in `apply` statements. For example, from our split Circulator data

```
> lapply(dayList, top, n = 2)
```

```
$Friday
  day      date
5 Friday 01/15/2010
12 Friday 01/22/2010

$Monday
  day      date
1 Monday 01/11/2010
8 Monday 01/18/2010

$Saturday
  day      date
6 Saturday 01/16/2010
13 Saturday 01/23/2010

$Sunday
  day      date
7 Sunday 01/17/2010
14 Sunday 01/24/2010

$Thursday
  day      date
4 Thursday 01/14/2010
```

Custom functions in `apply`

You can also designate functions "on the fly"

```
> lapply(dayList, function(x) x[1:2,1:2])
```

```
$Friday
  day      date
5 Friday 01/15/2010
12 Friday 01/22/2010

$Monday
  day      date
1 Monday 01/11/2010
8 Monday 01/18/2010

$Saturday
  day      date
6 Saturday 01/16/2010
13 Saturday 01/23/2010

$Sunday
  day      date
7 Sunday 01/17/2010
14 Sunday 01/24/2010

$Thursday
  day      date
4 Thursday 01/14/2010
11 Thursday 01/21/2010
```

Simple apply

`sapply()` is a user-friendly version and wrapper of `lapply` by default returning a vector, matrix, or array

```
> sapply(dayList, dim)
```

	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
[1,]	146	147	146	146	146	147	147
[2,]	15	15	15	15	15	15	15

```
> sapply(circ, class)
```

day	date	orangeBoardings	orangeAlightings
"character"	"character"	"integer"	"integer"
orangeAverage	purpleBoardings	purpleAlightings	purpleAverage
"numeric"	"integer"	"integer"	"numeric"
greenBoardings	greenAlightings	greenAverage	bannerBoardings
"integer"	"integer"	"numeric"	"integer"
bannerAlightings	bannerAverage	daily	
"integer"	"numeric"	"numeric"	

```
> myList = list(a=1:10, b=c(2,4,5), c = c("a","b","c"),  
+             d = factor(c("boy","girl","girl")))  
> tmp = lapply(myList,function(x) x[1])  
> tmp
```

```
$a  
[1] 1  
  
$b  
[1] 2  
  
$c  
[1] "a"  
  
$d  
[1] boy  
Levels: boy girl
```

```
> sapply(tmp, class)
```

a	b	c	d
"integer"	"numeric"	"character"	"factor"


```
> sapply(myList, function(x) x[1])
```

```
  a   b   c   d  
"1" "2" "a" "1"
```

```
> sapply(myList, function(x) as.character(x[1]))
```

```
  a     b     c     d  
"1"  "2"  "a" "boy"
```

