Introduction to R, Part V

Rebecca C. Steorts, STA 360

Agenda

- Lists
- Dataframes

Lists

Sequence of values, *not* necessarily all of the same type

```
my.distribution <- list("exponential",7,FALSE)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

Most of what you can do with vectors you can also do with lists

Accessing pieces of lists

Can use `[ ]` as with vectors or use `[[ ]]`, but only with a single index `[[ ]]` drops names and structures, `[ ]` does not

```
is.character(my.distribution)
```

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

```
is.character(my.distribution[[1]])
```

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

```
my.distribution[[2]]^2
```

```
## [1] 49
```

What happens if you try `my.distribution[2]^2`? What happens if you try `[[ ]]` on a vector?

Expanding and contracting lists

Add to lists with `c()` (also works with vectors):

```
my.distribution <- c(my.distribution,7)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 7
```

Chop off the end of a list by setting the length to something smaller (also works with vectors):

```
length(my.distribution)
```

```
## [1] 4
```

```
length(my.distribution) <- 3
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

Naming list elements

We can name some or all of the elements of a list

```
names(my.distribution) <- c("family","mean","is.symmetric")
my.distribution
```

```
## $family
## [1] "exponential"
##
## $mean
## [1] 7
##
## $is.symmetric
## [1] FALSE
```

```
my.distribution[["family"]]
```

```
## [1] "exponential"
```

```
my.distribution["family"]
```

```
## $family
## [1] "exponential"
```

Lists have a special short-cut way of using names, `$` (which removes names and structures):

```
my.distribution[["family"]]
```

```
## [1] "exponential"
```

```
my.distribution$family
```

```
## [1] "exponential"
```

Names in lists (cont'd.)

Creating a list with names:

```
another.distribution <- list(family="gaussian",mean=7,sd=1,is.symmetric=TRUE)
```

Adding named elements:

```
my.distribution$was.estimated <- FALSE
my.distribution[["last.updated"]] <- "2011-08-30"
```

Removing a named list element, by assigning it the value `NULL`:

```
my.distribution$was.estimated <- NULL
```

Key-Value pairs

Lists give us a way to store and look up data by *name*, rather than by *position*

A really useful programming concept with many names: **key-value pairs**, **dictionaries**, **associative arrays**, **hashes**

If all our distributions have components named `family`, we can look that up by name, without caring where it is in the list

Dataframes

Dataframe = the classic data table, n rows for cases, p columns for variables

Lots of the really-statistical parts of R presume data frames `penn` from last time was really a dataframe

Not just a matrix because *columns can have different types*

Many matrix functions also work for dataframes (`rowSums()`, `summary()`, `apply()`)

but no matrix multiplying dataframes, even if all columns are numeric

```
a.matrix <- matrix(c(35,8,10,4),nrow=2)
colnames(a.matrix) <- c("v1","v2")
a.matrix
```

```
##      v1 v2
## [1,] 35 10
## [2,]  8  4
```

```
a.matrix[,"v1"]  # Try a.matrix$v1 and see what happens
```

```
## [1] 35  8
```

```
a.data.frame <- data.frame(a.matrix,logicals=c(TRUE,FALSE))
a.data.frame
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
## 2  8  4    FALSE
```

```
a.data.frame$v1
```

```
## [1] 35  8
```

```
a.data.frame[,"v1"]
```

```
## [1] 35  8
```

```
a.data.frame[1,]
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
```

```
colMeans(a.data.frame)
```

```
##       v1       v2 logicals
##     21.5      7.0      0.5
```

Adding rows and columns

We can add rows or columns to an array or data-frame with `rbind()` and `cbind()`, but be careful about forced type conversions

```
rbind(a.data.frame,list(v1=-3,v2=-5,logicals=TRUE))
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
## 2  8  4    FALSE
## 3 -3 -5     TRUE
```

```
rbind(a.data.frame,c(3,4,6))
```

```
##   v1 v2 logicals
## 1 35 10        1
## 2  8  4        0
## 3  3  4        6
```

Back to the Factory Example

Recall from the last module, we had an example with a factory, where a factory makes cars and trucks, using labor and steel

- a car takes 40 hours of labor and 1 ton of steel

- a truck takes 60 hours and 3 tons of steel
- resources: 1600 hours of labor and 70 tons of steel each week

```
factory <- matrix(c(40,1,60,3),nrow=2)
output <- c(10,20)
available <- c(1600,70)
```

Structures of Structures

So far, every list element has been a single data value

List elements can be other data structures, e.g., vectors and matrices:

```
plan <- list(factory=factory, available=available, output=output)
plan$output
```

```
## [1] 10 20
```

Structures of Structures (cont'd.)

List elements can even be other lists which may contain other data structures including other lists which may contain other data structures...

This **recursion** lets us build arbitrarily complicated data structures from the basic ones

Most complicated objects are (usually) lists of data structures

Take-Aways

- Write programs by composing functions to manipulate data
- The basic data types let us represent Booleans, numbers, and characters
- Data structures let us group related values together
- Vectors let us group values of the same type
- Use variable assignment and name components of structures to make data more meaningful
- Matrices act like you'd hope they would
- Lists let us combine different types of data
- Dataframes are hybrids of matrices and lists, for classic tabular data