Base R Objects

Base R is great for data

Base R offers really awesome objects for analysis.

You can do nearly all of your tabular data anaysis using Base R.

With vecotorized operations you can you can program tabular data operations quickly.

Most common Base R objects

- Vector
- Matrix
- List
- Data Frame

Vectors

A collection of values of a single type.

Typed - Named - Indexed

```r
vec <- c("A" = 1.1, "B" = 1.2, "C" = 1.3)
class(vec)
```

```
## [1] "numeric"
```

```r
vec["C"]
```

```
##   C
## 1.3
```

```r
vec[3]
```

```
##   C
## 1.3
```

Vector main types

numeric

```
vec <- c(1.1, 1.2, 1.3)
```

character

```
vec <- c("This", "That", "Other")
```

logical - can be abbreviated with `T` or `F`

```
vec <- c(TRUE, FALSE, FALSE)
```

Other vector types

These come up, but are for special cases.

integer

```
vec <- c(1L, 2L, 3L)
```

factor - these are catagorical values

```
fac <- factor(c("This", "That", "Other"))
```

Vector subsetting

You subset vectors with other vectors using the '[' notation.

```
vec <- c(1, 2, 3, 4, 5)
```

Subsetting a vector with a logical vector.

```
bool <- vec > 3
bool
```

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

```
vec[bool]
```

```
## [1] 4 5
```

. . . or simply . . .

```
vec[vec > 3]
```

```
## [1] 4 5
```

What is happening here?

We are already seeing functional programming in action.

'`vec > 3`' is a vectorized operation. It can also be written as: `'>'(vec, 3)`

The function '>' takes two vectors as an inputs and checks each element from each vector to see if the first is greater than the second. **Looping over the vectors happens internally**.

```
'>'(vec, 3)
```

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

The '`vec > 3`' notation is what we call **syntax sugar**. It is imbedded styling that makes reading and writing code easier.

Vectors are any length

A single number, string, or bool is a vector of length 1.

The ‘c()’ function in r concatenates vectors of length 0 or more.

```r
vec1 <- "This"
class(vec1); length(vec1)
```

```
## [1] "character"
```

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

```r
vec2 <- c(vec1, c("That", "Other"), c())
class(vec2); length(vec2)
```

```
## [1] "character"
```

```
## [1] 3
```

Matrices

These are essentially 2 dimentional vectors.

```
vec <- 1:16
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```
mat <- matrix(vec, nrow = 4)
mat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Lists

Lists are containers for storing r objects of multiple types and of varying lengths.

Much like vectors each element has an index and optionally a name.

```r
myList <- list(nums  = 1:5,
               chars = LETTERS[1:8],
               bools = c(TRUE, FALSE))
myList
```

```
## $nums
## [1] 1 2 3 4 5
##
## $chars
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
##
## $bools
## [1]  TRUE FALSE
```

List subsetting

Lists are subsetted in a simular way to vectors, but there are some slight differences.

This will return the first element of the list, but not the elements contents.

```
myList[1]
```

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

List subsetting

This comes from Hadley Wickham courtesy of Residence Inn. . .

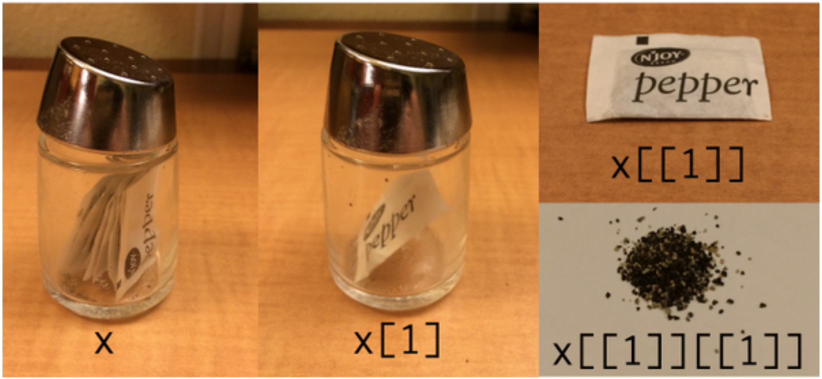

Figure 1: x <- list(pepper1, pepper2, pepper3)

List subsetting

So, using ‘myList’...

```
myList <- list(1:5, LETTERS[1:8], c(TRUE, FALSE))
myList[2]
```

```
## [[1]]
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

```
myList[[2]]
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

```
myList[[2]][[2]]
```

```
## [1] "B"
```

Data frames

These are the primary tabular data structure in R.

Data frames are essentially lists that follow a couple of rules that make them behave like tables. Named list elements in data frames can been thought about as columns.

1. ~~Each column (list item) must be a vector.~~
2. Each column (list item) must be the same length.

Because each vector is the same length, rows can have names. Row names default from 1 to the number of rows in the data frame.

Think of this like a spreadsheet that follows very strict rules on how to stucture your data.

Data frames subsetting

Data frames can be interacted with much like lists.

```
myDf <- data.frame(nums  = 1:5,
                   chars = LETTERS[1:5],
                   bools = c(T, T, T, F, F),
                   stringsAsFactors = F)
myDf
```

```
##   nums chars bools
## 1    1     A  TRUE
## 2    2     B  TRUE
## 3    3     C  TRUE
## 4    4     D FALSE
## 5    5     E FALSE
```

Data frame subsetting

```r
myDf[2]
```

```
##   chars
## 1     A
## 2     B
## 3     C
## 4     D
## 5     E
```

```r
myDf[[2]]
```

```
## [1] "A" "B" "C" "D" "E"
```

```r
myDf[[2]][[2]]
```

```
## [1] "B"
```

Data frame subsetting

It is very common to interact with data frames by column name.

```
myDf$chars
```

```
## [1] "A" "B" "C" "D" "E"
```

```
myDf$nums
```

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

Data frame subsetting

You can also use the '[' notation.

'[' notation has two dimensions, '`[rows, columns]`'.

```
myDf[1, ] # first row
```

```
##   nums chars bools
## 1    1     A  TRUE
```

```
myDf[ ,1] # first column
```

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

```
myDf[1,1] # first row, first column
```

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

Data frame subsetting

Just like vectors and lists, data frames can be subsetted by name, index, or bool.

```r
myDf$nums > 3
```

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

```r
myDf[myDf$nums > 3,]
```

```
##   nums chars bools
## 4    4     D FALSE
## 5    5     E FALSE
```

```r
myDf[myDf$bool,]
```

```
##   nums chars bools
## 1    1     A  TRUE
## 2    2     B  TRUE
## 3    3     C  TRUE
```

Adding columns to data frames

Adding new data to data frames is intuitive. Replacing columns is the same.

```
myDf$newData <- round(runif(5, 1, 100))
myDf
```

```
##   nums chars bools newData
## 1    1     A  TRUE      27
## 2    2     B  TRUE      38
## 3    3     C  TRUE      58
## 4    4     D FALSE      91
## 5    5     E FALSE      21
```

You can also add/replace by index.

Removing columns from data frames

Removing columns is easy too.

```r
myDf[c("nums", "bools")] <- NULL
myDf
```

```
##   chars newData
## 1     A      27
## 2     B      38
## 3     C      58
## 4     D      91
## 5     E      21
```

There are actually several ways to remove columns from a data frame. This is just one.

Review

Vectors are of a single type.

Lists can store vectors or other objects of different types and lengths.

Data frames are lists with rules so that they behave like tables.

'[' notation is everywhere. You can use it to subset single or multiple dimensions using logical, numeric, or character vectors.