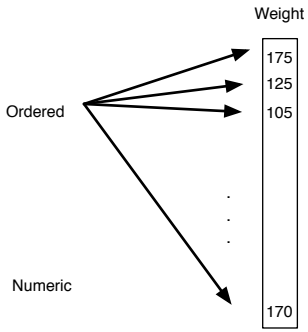
Lists and Matrices

AND the Apply Family of Functions

load(url("http://www.stat.berkeley.edu/users/nolan/data/anExampleList.rda"))

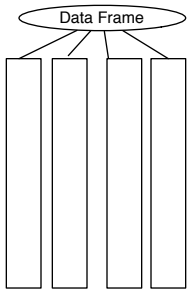
Vector

- *Ordered* container of literals
- Elements must be *same type*



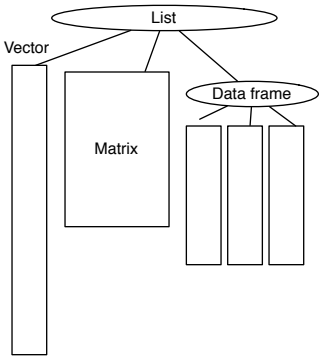
Data Frame

- *Ordered* container of vectors
- Vectors must all be the *same length*
- Vectors can be *different types*

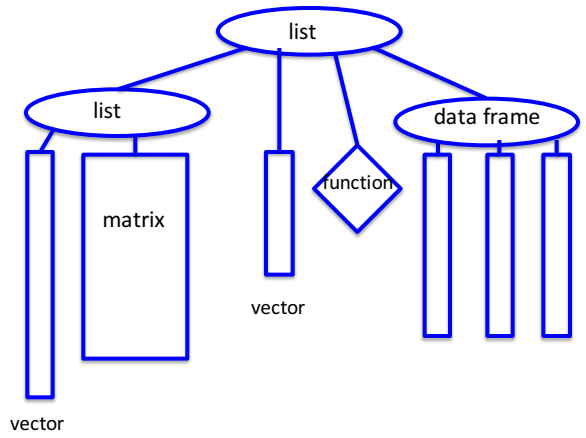


List

- Ordered collection of arbitrary objects
- Each element can be either a list, data frame, vector, matrix, …
- Data frames are a special kind of *list* where all elements are vectors of the same length



An Example List



Let's inspect this List

Description of Structure: `str()`

```
List of 4
 $ listToo:List of 2
  ..$ aVec: num [1:4] 1 3 5 7
  ..$ aMat: num [1:3, 1:2] 10 14 18 12 16 20
 $ aVex   : chr [1:7] "a" "b" "c" "d" ...
 $ aFunc  :function (x)
  ..- attr(*, "srcref")=Class 'srcref'  atomic
[1:8] 1 140 1 159 140 159 1 1
  .. .. ..- attr(*, "srcfile")=Classes
'srcfilecopy', 'srcfile' <environment:
0x10f2b2b10>
 $ aDF    :'data.frame': 4 obs. of  3 variables:
  ..$ id    : num [1:4] 101 102 103 104
  ..$ height: num [1:4] 60 72 66 70
  ..$ sex   : Factor w/ 2 levels "f","m": 1 2 1 2
```


```
List of 4
 $ listToo:List of 2
  ..$ aVec: num [1:4] 1 3 5 7
  ..$ aMat: num [1:3, 1:2] 10 14 18 12 16 20
 $ aVex   : chr [1:7] "a" "b" "c" "d" ...
 $ aFunc  :function (x)
  ..- attr(*, "srcref")=Class 'srcref'  atomic
[1:8] 1 140 1 159 140 159 1 1
  .. .. ..- attr(*, "srcfile")=Classes
'srcfilecopy', 'srcfile' <environment:
0x10f2b2b10>
 $ aDF    :'data.frame': 4 obs. of  3 variables:
  ..$ id    : num [1:4] 101 102 103 104
  ..$ height: num [1:4] 60 72 66 70
  ..$ sex   : Factor w/ 2 levels "f","m": 1 2 1 2
```

Taking a Subset of a List

Ways to subset

- 5 types of subsetting work here too
  - Name, position, exclusion, logical, all
- **$**-sign notation accesses one element, if the elements have names
- Special subsetting for lists with **[[ ]]** – double square brackets - to access 1 element
- Take a subset of a subset with consecutive square brackets

5 ways to subset

`aList["listToo"]`	returns a list with 1 element (listToo), i.e. a list with 1 element
`aList[ c(3, 2)]`	returns a list with 2 elements (3rd, 2nd)
`aList[ -(1:3) ]`	Returns a list with 4th element

$-sign and [[ ]]

`aList$listToo`	returns the list listToo
`aList[[2]]`	Returns the vector aVec

Subset of a Subset

```
aList$listToo$aVec
```

returns the vector

```
[1] 1  3  5  7
```

```
aList[[1]]$aVec
```

Returns the vector

```
[1] 1  3  5  7
```

```
aList[[1]][[1]]
```

Ditto

```
aList$listToo[[1]]
```

Ditto

Subset of a Subset

```
aList$aVec[1:3]
```

returns the vector

```
[1] "a" "b" "c"
```

```
aList[[4]]$id
```

Returns the vector

```
[1] 101 102 103 104
```

```
aList[[4]][1:2, 2:3]
```

Returns the data frame

```
 height sex
1    60   f
2    72   m
```

Subset of a Subset

```
aList[[3]](2:3)
```

A. returns the vector

```
[1] "a" "b" "c"
```

B. Returns an error

C. Returns 13

D. Returns the data frame

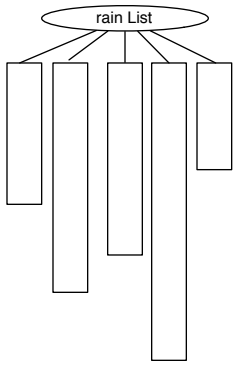
```
 height sex
1    60   f
2    72   m
3    66   f
4    70   m
```

E. None of the above

Rainfall in Colorado Front Range

Rainfall

- Daily rainfall collected at 5 weather stations
- **rain** is a list of length 5
  - One element for each station
  - Each element is a numeric vector of rain measurements
  - Stations not in operation for the same length of time




```
load(url("http://www.stat.berkeley.edu/users/nolan/data/rainfallCO.rda"))
```

```
> class(rain)
[1] "list"
> length(rain)
[1] 5
> names(rain)
[1] "st050183" "st050263" "st050712" "st050843" "st050945"
```

Or

```
> str(rain)
List of 5
 $ st050183: atomic [1:9878] 0 10 11 1 0 0 0 0 0 0 ...
  ..- attr(*, "Csingle")= logi TRUE
 $ st050263: atomic [1:6751] 0 0 0 0 0 0 0 0 0 0 ...
  ..- attr(*, "Csingle")= logi TRUE
 $ st050712: atomic [1:3959] 0 0 16 78 42 0 0 0 0 0 ...
  ..- attr(*, "Csingle")= logi TRUE
 $ st050843: atomic [1:11122] 0 5 7 14 2 0 0 0 0 0 ...
  ..- attr(*, "Csingle")= logi TRUE
 $ st050945: atomic [1:3692] 0 0 1 0 26 0 0 0 0 0 ...
  ..- attr(*, "Csingle")= logi TRUE
```

Accessing a Station's Data

- We can index 1 station by name, using $.

```
> class(rain$st050183)
[1] "numeric"
> length(rain$st050183)
[1] 9878
> head(rain$st050183)
[1]  0 10 11  1  0  0
```

- Or by [[ ]] with position or name

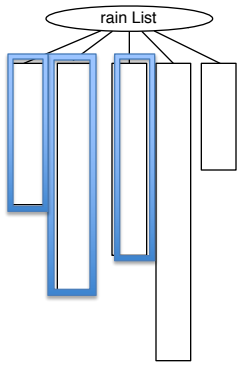
```
> class(rain[["st050945"]])
[1] "numeric"
> length(rain[["st050945"]])
[1] 3692
> head(rain[[5]])
[1]  0  0  1  0 26  0
```

Apply Functions

- Sometimes we want an operation to be applied to each element of a list, to each vector in a data frame – e.g., that maximum rainfall at each weather station
- R provides the *apply* mechanism to do this.
- There are several apply functions:
  - sapply(), lapply(), mapply() for lists and data frames
  - apply() for matrices
  - tapply() for "tables", i.e., ragged arrays as vectors

Rainfall

- **Apply** the mean function to each element of **`rain`**
- Finds: average precipitation of first station,
- Second station,
- Third station,
- Etc.



lapply() and sapply()

- The **lapply** and **sapply** both apply a specified function to each element of a list (remember data frames are special types of lists).
- The former returns a list object and the latter a vector when possible.

Mean rainfall at each station

```
> lapply(rain, mean)
$st050183
[1] 6.631707

$st050263
[1] 3.798993

$st050712
[1] 5.102299

$st050843
[1] 6.084607

$st050945
[1] 4.549296
```

```
> sapply(rain, mean)
st050183 st050263 st050712
6.631707 3.798993 5.102299

st050843 st050945
6.084607 4.549296
```

Notice that lapply returns a *list* and sapply returns a *vector* (when it can)

Additional arguments

```
> lapply(rain, mean, na.rm = TRUE,
        trim = 0.1)
$st050183
[1] 2.393978

$st050263
[1] 0.9875949

$st050712
[1] 0.7895235

$st050843
[1] 1.238481

$st050945
[1] 0.7366283
```

> args(lapply)
function (x, FUN, **...**)

- x takes the list object
- FUN is the function to apply to each element in X
- **...** allows any number of arguments to be passed to FUN

Format

lapply(a_list, a_function, arg1 = xx, arg2 = yy)

The List to apply the function to each element

The function to apply to each element in the list

Additional, named, arguments passed to the function

Rainfall data

- Maximum rainfall at each station

```
sapply(rain, max)
```

- 99th percentile of rainfall at each station

```
sapply(rain, quantile, probs = 0.99)
```

Find Proportion of Rainy Days

For one station:

```
stat1 = rain[[1]]
sum(stat1 > 0) / length(stat1)
```

How can we incorporate this sequence of functions into one call to an apply function?

Advanced

For any one station:

```
sum(stat1 > 0) / length(stat1)
```

We want to apply this composition of functions to each element of rain.

```
sapply(rain, sum(? > 0) / length(?))
```

We can create our own function to do this:

```
sapply(rain, function(x) sum(x > 0) / length(x))
```

Find Proportion of Rainy Days

We can break the expression up into smaller pieces and put it all back together

```
numDays = sapply(rain, length)
rainDays = lapply(rain, '>', 0)
numRainyDays = sapply(rainDays, sum)
numRainyDays / numDay
```

This is dissatisfying –

- many steps for a simple calculation
- many intermediate copies of the list/vector

Matrices and Arrays

Matrices and Arrays

- Rectangular collection of elements
- Dimensions are two, three, or more
- Homogeneous primitive elements (e.g., all numeric or all character)

- You can create a matrix in R using the `matrix` function.
- By default, matrices in R are assigned by *column-major* order.
- You can assign them by *row-major* order by setting the `byrow` argument to TRUE. Note that the first argument to `matrix` is a vector, so all elements must be of the same type (numeric, character, or logical).

```
> m = matrix(1:6, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m = matrix(1:6, nrow = 2, byrow = TRUE)
 m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- Assign names to the rows and columns of a matrix:

```
> rownames(m) = letters[1:2]
> colnames(m) = letters[3:5]

> m
  c d e
a 1 2 3
b 4 5 6
```

- Find the dimensions of a matrix:

```
> dim(m)
[1] 2 3

> nrow(m)
[1] 2

> ncol(m)
[1] 3
```

- To index elements of a matrix, use the same five methods of indexing we covered for vectors, but with the first index for rows and the second for columns.
- What will each line return?

```
> m
  c d e
a 1 3 5
b 2 4 6
> m[-1, 2]     # Exclusion & position
> m["a",]      # Row by name, all cols
> m[, c(TRUE, TRUE, FALSE)] #logical cols
```

Apply Functions for Matrices

apply()

- **apply(m, 1, sum)** for the matrix x, the sum function is applied across the columns so that the row dimension (i.e. dim 1) is preserved.

```
> m
   c d e
a  1 2 3
b  4 5 6
> apply(m, 1, sum)
 a  b
 6 15
```

apply()

- **apply(x, 2, sum)** for the matrix x, the sum function is applied down the rows so that the column dimension (i.e. dim 2) is preserved.

```
> m
   c d e
a  1 2 3
b  4 5 6
> apply(m, 2, sum)
 c d e
 5 7 9
```

Arrays – matrices in higher dimensions

```
> x = array(1:24, c(4, 3, 2))
> x
, , 1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2
     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

- The integers 1, 2, … , 24 are arranged in a 3-dimensional array
- The array has:
  - 4 rows
  - 3 columns
  - 2 panels

```
> x[1:2, 3, 2]
[1] 21 22

> x[ , 2, 1]
[1] 5 6 7 8

> x[3:4, c(3, 1), 1]
     [,1] [,2]
[1,]   11    3
[2,]   12    4
```

Arrays – matrices in higher dimensions

```
> x = array(1:24, c(4, 3, 2))
> x
, , 1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2
     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

```
> apply(x, 1:2, sum)
     [,1] [,2] [,3]
[1,]   14   22   30
[2,]   16   24   32
[3,]   18   26   34
[4,]   20   28   36
```

```
> apply(x, 1, median)
[1] 11 12 13 14
```