R factors, lists, and matrices

Benjamin Soibam, PhD

CS 5301 – Programming Foundations for Data Analytics

Reference for the slides

- Mainly from Part II of *Hands on Programming with R* by Garrett Grolemund

Factors

- Factors are R's way of storing categorical information, like ethnicity or eye color.
- Think of a factor as something like a gender; it can only have certain values (male or female).
- Pass an atomic vector into the factor function.
- R encodes the data in vector. Factors are stored as integers (the levels) along with their corresponding character values.
- R adds a levels attribute to the integer.

```
>gender <- factor(c("male", "female", "female", "male"))
> gender
[1] male   female female male
Levels: female male
```

Factors

```
>gender <- factor(c("male", "female", "female", "male"))
> gender
[1] male   female female male
Levels: female male

> typeof(gender)
[1] "integer"

> str(gender)
 Factor w/ 2 levels "female","male": 2 1 1 2
```

Dropping Levels for factors

```
> gender <- factor(c("male", "female", "female", "male"))
```

Subset indices 1 and 4 from gender

```
> gender1 <- gender[c(1,4)]
```

gender1 contains “male” only but has two “levels”

```
> gender1
[1] male male
Levels: female male
> levels(gender1)
[1] "female" "male”
```

The user can choose to drop the unused levels

```
> gender1 <- droplevels(gender1)
> gender1
[1] male male
Levels: male
```

Adding levels to factors

Here, gender will have only level

```
> gender <- factor(c("male", "male", "male", "male"))
> gender
[1] male male male male
Levels: male
> str(gender)
 Factor w/ 1 level "male": 1 1 1 1
```

If you try to modify gender with “female” it will give an error since factor

Can only accept something which is one of the levels

```
> gender[2] <- "female"
Warning message:
In `[<-.factor`(`*tmp*`, 2, value = "female") :
  invalid factor level, NA generated
```

Adding levels to factors

The work around is add additional levels to the factor object

```
> levels(gender) <- c(levels(gender),"female")
> levels(gender)
[1] "male"   "female”

> gender[2] <- "female"
> gender
[1] male   female male   male
Levels: male female
```

Therefore, factor is a way of protecting your object from getting modified in a wrong way.

Converting factor to numeric

- Factors are stored as integers (the levels) along with their corresponding character values.
- Converting factor to numeric will simply return the stored levels (or the numbers)

```
>x <- factor(c("4.1","4.1","5.2","5.2"))
>str(x)
 Factor w/ 2 levels "4.1","5.2": 1 1 2 2
> y <- as.numeric(x)
> y
[1] 1 1 2 2
```

- To convert factor to numeric, first convert it to character

```
> y <- as.numeric(as.character(x))
> y
[1] 4.1 4.1 5.2 5.2
```

List

- Lists are like atomic vectors because they group data into a one-dimensional set.
- lists do not group together individual values; **lists group together R objects**, such as atomic vectors and other lists.

```
list1 <- list(100:130, "R", list(TRUE, FALSE))

list1
## [[1]]
## [1] 100 101 102 103 104 105 106 107 108 109 110 111 112
## [14] 113 114 115 116 117 118 119 120 121 122 123 124 125
## [27] 126 127 128 129 130
##
## [[2]]
## [1] "R"
##
## [[3]]
## [[3]][[1]]
## [1] TRUE
##
## [[3]][[2]]
## [1] FALSE
```

First R Object

Second R Object

Third R Object

Accessing entries of a list

```
list1 <- list(100:130, "R", list(TRUE, FALSE))
```

list1[1] **list1[2]** **list1[3]**

To access each sublist, use [ ]

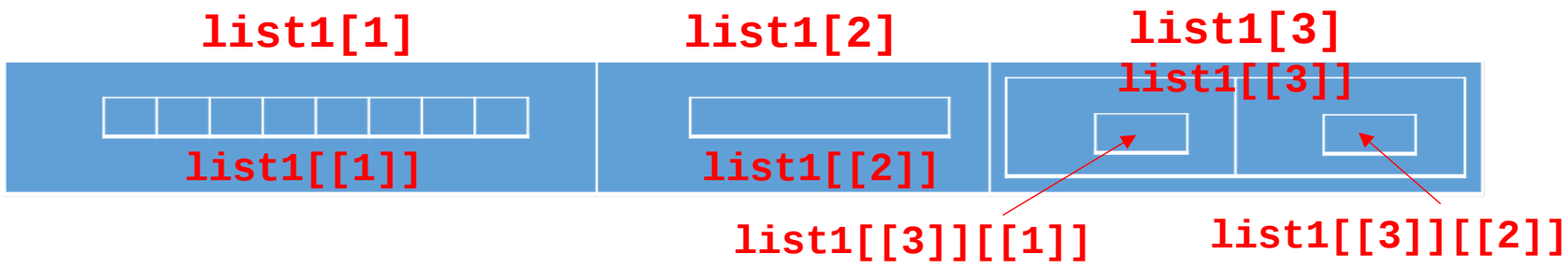
```
> list1[1]
[[1]]
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
[18] 117 118 119 120 121 122 123 124 125 126 127 128 129 130

> list1[2]
[[1]]
[1] "R"

> list1[3]
[[1]]
[[1]][[1]]
[1] TRUE

[[1]][[2]]
[1] FALSE
```

If you use [ ], you extract a list
You don't extract the components of the list

```
> typeof(list1[1])
[1] "list"
> typeof(list1[2])
[1] "list"
> typeof(list1[3])
[1] "list"
```

Accessing entries of a list

```
list1 <- list(100:130, "R", list(TRUE, FALSE))
```



To access components of each sublist, use `[[]]`

```
> list1[[1]]
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
[18] 117 118 119 120 121 122 123 124 125 126 127 128 129 130
> list1[[2]]
[1] "R"
> list1[[3]]
[[1]]
[1] TRUE

[[2]]
[1] FALSE
```

```
> typeof(list1[[1]])
[1] "integer"
> typeof(list1[[2]])
[1] "character"
> typeof(list1[[3]])
[1] "list"
```

Accessing entries of a list

```
list1 <- list(100:130, "R", list(TRUE, FALSE))
```

Look the 3rd item of the list. 3rd item itself is a list. Illustrated below.

```
> typeof(list1[3])
[1] "list"
> typeof(list1[[3]])
[1] "list”

> typeof(list1[[3]][1])
[1] "list"
> typeof(list1[[3]][[1]])
[1] "logical"

> typeof(list1[[3]][2])
[1] "list"
> typeof(list1[[3]][[2]])
[1] "logical"
```

To access the logical values

```
> list1[[3]][[1]]
[1] TRUE
> list1[[3]][[2]]
[1] FALSE
```

Another example of list

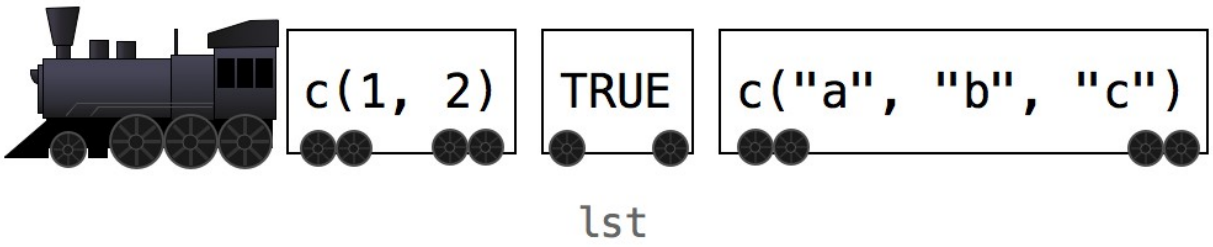
```
>card <- list(“ace”, “hearts”,1)
>card
## [[1]]
## [1] "ace"
##
## [[2]]
## [1] "hearts"
##
## [[3]]
## [1] 1
```

- **card[[1]]** is a character string
- **card[1]** is a list

Another Example of list with names

- Lets make a list

```
lst <- list(numbers = c(1, 2), logical = TRUE, strings = c("a",
"b", "c"))
lst
## $numbers
## [1] 1 2
## $logical
## [1] TRUE
## $strings
## [1] "a" "b" "c"
```



Accessing entries of a list with names

```
> lst <- list(numbers = c(1, 2), logical = TRUE, strings =
c("a", "b", "c"))
> str(lst)
List of 3
 $ numbers: num [1:2] 1 2
 $ logical: logi TRUE
 $ strings: chr [1:3] "a" "b" "c“
```

Access the entries of each sublist by using $ sign

```
> lst$numbers
[1] 1 2
> lst$logical
[1] TRUE
> lst$strings
[1] "a" "b" "c"
```

If $ is used you extract the components of each sublist. In this case each is a **vector**

```
> typeof(lst$numbers)
[1] "double"
> typeof(lst$logical)
[1] "logical"
> typeof(lst$strings)
[1] "character"
```

Modify a sublist in R

```
> lst <- list(numbers = c(1, 2), logical = TRUE, strings =
c("a", "b", "c"))
```

Modifying the first item in “numbers”

```
> lst$numbers[1] <- 3
> lst$numbers
[1] 3 2
> lst <- list(numbers = c(1, 2), logical = TRUE, strings =
c("a", "b", "c"))
```

Modifying “numbers”

```
> lst$numbers <- c(0,1,3,-9)
> lst$numbers
[1]  0  1  3 -9
```

Replacing the first item by character vector

```
> lst$numbers <- c("this","hello","are")
> lst$numbers
[1] "this"  "hello" "are"
```

Adding a sublist

```
> lst <- list(numbers = c(1, 2), logical = TRUE, strings =
c("a", "b", "c"))
```

Adding a sublist with name “score”. Use `[[]]` to add a vector

```
> lst[["score"]] <- c(100,80,50)
> lst
$numbers
[1] 1 2

$logical
[1] TRUE

$strings
[1] "a" "b" "c"

$score
[1] 100  80  50
```

Removing/deleting a sublist

```
> lst <- list(numbers = c(1, 2), logical = TRUE, strings =
c("a", "b", "c"),score=c(100,80,50))
```

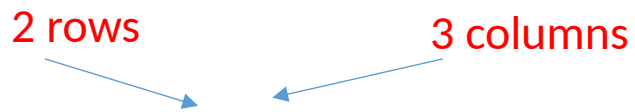
Removing the “score” sublist. Use NULL

```
> lst[["score"]] <- NULL
> lst
$numbers
[1] 1 2

$logical
[1] TRUE

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

Matrices

- You can convert a vector into a matrix (**2D array**). It's a way of reorganizing your data

```
> die <- c(1,2,3,4,5,6)
```




```
> dim(die) <- c(2,3) # organize the data columnwise
> die
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>dim(die) # checking the dimensions of the matrix
[1] 2 3
>nrow(die) # checking the number of rows of the matrix
[1] 2
>ncol(die) # checking the number of columns of the matrix
[1] 3
```

Directly creating matrices

```
> die <- c(1,2,3,4,5,6)
> m <- matrix(die,nrow=2)# or m <-matrix(c(1,2,3,4,5,6),nrow=2)
```

a vector

number of rows in the desired matrix

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Explore matrix by typing **?matrix**

Arrange row wise

```
> m1 <- matrix(die,nrow=2,byrow=TRUE)
> m1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Diagonal matrices

- Construct a diagonal matrix

```
> diag(c(1,-1,0))
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0   -1    0
[3,]    0    0    0
```

- Extract the diagonal elements of a matrix

```
> x <- matrix(seq(from=1,to=9,by=1),3,3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> diag(x)
[1] 1 5 9
```

Accessing matrix elements

- How to access individual items

```
>m[rowindex,colindex]

> m[1,1] # first row, first column
[1] 1
> m[1,2] # first row, second column
[1] 3
> m[1,3] # first row, third column
[1] 5
> m[2,2] # second row, second column
[1] 4
```

Accessing matrix elements

- How to access sub-matrices

```
> m[1,1:3] # first row and cols = 1,2,3
[1] 1 3 5
> m[,1:3] # all rows and col = 1,2,3
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[,1:2] # all rows and first 2 columns
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Simple Matrix operations

Element wise matrix multiplication

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> m*m1
     [,1] [,2] [,3]
[1,]    1    6   15
[2,]    8   20   36
```

Computing Column & Row Sums

```
> M <- matrix(seq(from=1,to=10,by=1),nrow=2,byrow=TRUE)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Use ColSums and rowSums

```
> colSums(M)
[1]  7  9 11 13 15
> rowSums(M)
[1] 15 40
```

An indirect way of computing sum of each row is:

```
> sum(M[1,])
[1] 15
> sum(M[2,])
[1] 40
```

Simple Matrix operations

- Transpose of a Matrix [flips the matrix]

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m_transpose <- t(m)
> m_transpose
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> dim(m)
[1] 2 3
> dim(m_transpose)
[1] 3 2
```

Horizontal concatenation

```
> m1 <- matrix(c(2,3,-2,1,2,2,4,6,-4),3,3)
> m2 <- matrix(c(1,3,2,1,4,2),3,2)
> m1
     [,1] [,2] [,3]
[1,]    2    1    4
[2,]    3    2    6
[3,]   -2    2   -4
> m2
     [,1] [,2]
[1,]    1    1
[2,]    3    4
[3,]    2    2
```

Use cbind to concatenate. Make sure the number of rows for the 2 matrices are same.

```
> M_column_bind <- cbind(m1,m2)
> M_column_bind
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    1    4    1    1
[2,]    3    2    6    3    4
[3,]   -2    2   -4    2    2
```

Vertical concatenation

```
> m1 <- matrix(c(2,3,-2,1,2,2),3,2)
> m2 <- matrix(c(1,3,2,1,4,2),3,2)
> m1
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
> m2
     [,1] [,2]
[1,]    1    1
[2,]    3    4
[3,]    2    2
```

Use rbind to concatenate. Make sure the number of columns for the 2 matrices are same.

```
> M_row_bind <- rbind(m1,m2)
> M_row_bind
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
[4,]    1    1
[5,]    3    4
[6,]    2    2
```

Modifying matrices

- Modifying one item in a matrix

```
> m1 <- matrix(c(2,3,-2,1,2,2),3,2)
> m1
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
> m1[1,2] <- 0 # modify row=1, column = 2
> m1
     [,1] [,2]
[1,]    2    0
[2,]    3    2
[3,]   -2    2
```

Modifying matrices

- Modifying submatrix

```
> m1 <- matrix(c(2,3,-2,1,2,2),3,2)
> m1
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
```

Modify the matrix to look like

```
     [,1] [,2]
[1,]    5    9
[2,]    3    2
[3,]    0   10
```

```
> m1[c(1,3),c(1,2)] <- matrix(c(5,0,9,10),nrow=2)
> m1
     [,1] [,2]
[1,]    5    9
[2,]    3    2
[3,]    0   10
```

Modifying matrices

- Modifying submatrix

```
> m1 <- matrix(c(2,3,-2,1,2,2),3,2)
> m1
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
```

Modify the matrix to look like

```
     [,1] [,2]
[1,]    5    9
[2,]    3    2
[3,]    0    2
```

```
> m1[c(1,3),c(1)] <- c(5,0) # modify the first row
> m1[1,2] <- 9 # modify only one item
     [,1] [,2]
[1,]    5    9
[2,]    3    2
[3,]    0    2
```