R Syntax 3: Factor and DataFrame

Pilsung Kang
School of Industrial Management Engineering
Korea University

Handling Factor

Scalar	Vector	List	Matrix	Array	Factor	Data.frame

- Factor
  - ✓ Vector representation for nominal/categorical variables
    - Factor has its levels that are equivalent the number of possible values
- Usage
  - ✓ Categorical variable representation: 1 level for 1 category
  - ✓ Grouping and tagging
- Factor levels must be consistent!

```r
# Part 1-4: Data Handling (Factor)

# Example of a factor
A <- c("Cho","Kim","Kang")
B <- as.factor(A)

print(A)
print(B)

mode(A)
mode(B)

A[1]+A[2]
B[1]+B[2]
```

```
> A[1]+A[2]
Error in A[1] + A[2] : non-numeric argument to binary operator
> B[1]+B[2]
[1] NA
Warning message:
In Ops.factor(B[1], B[2]) :
  요인(factors)에 대하여 의미있는 '+'가 아닙니다.
```

Handling Factor

- Factor
  - ✓ Factor in R is a vector with additional information
    - Additional information is a set of non-redundant values called level
    - The length of a factor is the number of elements, not levels
    - Possible to add a new level
    - A new value with non-existing level is considered as NA

```
Console ~/
> x <- c(5,12,13,12)
> xf <- factor(x)
> xf
[1] 5  12 13 12
Levels: 5 12 13
> str(xf)
 Factor w/ 3 levels "5","12","13": 1 2 3 2
> unclass(xf)
[1] 1 2 3 2
attr(,"levels")
[1] "5"  "12" "13"
> length(xf)
[1] 4
>
```

```
Console ~/
> xff <- factor(x, levels=c(5,12,13,88))
> xff
[1] 5  12 13 12
Levels: 5 12 13 88
> xff[2] <- 88
> xff
[1] 5  88 13 12
Levels: 5 12 13 88
> xff[2] <- 20
Warning message:
In `[<-.factor`(`*tmp*`, 2, value = 20) :
  invalid factor level, NA generated
> xff
[1] 5    <NA> 13   12
Levels: 5 12 13 88
>
```

Handling Factor

- Applying function to a factor
  - ✓ tapply( )
    - Useful to make frequency table with different categories
    - Can be used to more than two factors

```
Console ~/
> ages <- c(25,26,55,37,21,42)
> affils <- c("R","D","D","R","U","D")
> tapply(ages, affils, mean)
 D  R  U
41 31 21
>
```

```
Console ~/
> gender <- c("M", "M", "F", "M", "F", "F")
> age <- c(47,59,21,32,33,24)
> income <- c(55000,88000,32450,76500,123000,45650)
> tmp <- data.frame(gender, age, income)
> tmp$over25 <- ifelse(tmp$age>25,1,0)
> tmp
  gender age income over25
1      M  47  55000      1
2      M  59  88000      1
3      F  21  32450      0
4      M  32  76500      1
5      F  33 123000      1
6      F  24  45650      0
> tapply(tmp$income, list(tmp$gender, tmp$over25), mean)
      0         1
F 39050 123000.00
M    NA  73166.67
>
```

Handling Factor

- Applying function to a factor
  - ✓ split( ) function
    - Used to make groups
    - can be used to more than two factors

```
Console ~/
> split(tmp$income, list(tmp$gender, tmp$over25))
$F.0
[1] 32450 45650

$M.0
numeric(0)

$F.1
[1] 123000

$M.1
[1] 55000 88000 76500
```

Handling Dataframe

Scalar	Vector	List	Matrix	Array	Factor	Data.frame

- Dataframe
  - ✓ A table with rows and columns
  - ✓ Regarded as a special case of list
  - ✓ Can have different modes for different columns
  - ✓ Elements in a column must have the same modes
  - ✓ Columns can have names

```r
# Part 1-5: Data Handling (DataFrame) -------

# Example of data frame
A <- c(1,2,3)
B <- c("a","b","c")
C <- data.frame(A,B)
C
C[[1]]
C[[2]]
C[1,2]
C$B[2]

C <- data.frame(A,B, stringsAsFactors=FALSE)
C
C[[1]]
C[[2]]
C[1,2]
C$B[2]
```

Handling Dataframe

- Creating and accessing Dataframe
  - ✓ Use data.frame( ) function to create a dataframe
  - ✓ Three ways to access a certain element

```
Console ~/
> kids <- c("Jack", "Jill")
> ages <- c(12,10)
> d <- data.frame(kids, ages, stringsAsFactors=FALSE)
> d
  kids ages
1 Jack   12
2 Jill   10
>
```

```
Console ~/
> d[[1]]
[1] "Jack" "Jill"
> class(d[[1]])
[1] "character"
> d$kids
[1] "Jack" "Jill"
> class(d$kids)
[1] "character"
> d[,1]
[1] "Jack" "Jill"
> class(d[,1])
[1] "character"
> d[1]
  kids
1 Jack
2 Jill
> class(d[1])
[1] "data.frame"
>
```

Handling Dataframe

- Extracting and filtering a subset of dataframe
  - ✓ Same as matrix
- Combine dataframe
  - ✓ Same as matrix

```
Console ~/
> Exam
  Exam1 Exam2 Quiz
1   2.0   3.3  4.0
2   3.3   2.0  3.7
3   4.0   4.0  4.0
4   2.3   0.0  3.3
5   2.3   1.0  3.3
6   3.3   3.7  4.0
> Exam[2:5,]
  Exam1 Exam2 Quiz
2   3.3     2  3.7
3   4.0     4  4.0
4   2.3     0  3.3
5   2.3     1  3.3
> Exam[2:5,2]
[1] 2 4 0 1
> Exam[2:5,2, drop=FALSE]
  Exam2
2     2
3     4
4     0
5     1
>
```

```
Console ~/
> Exam[Exam$Exam1 > 3,]
  Exam1 Exam2 Quiz
2   3.3   2.0  3.7
3   4.0   4.0  4.0
6   3.3   3.7  4.0
> rbind(d,list("Laura",19))
   kids ages
1  Jack   12
2  Jill   10
3 Laura   19
>
```

Handling Dataframe

- Merge dataframes
  - ✓ If there are more than on sources of data tables in a database
  - ✓ Inner/outer/left/right joins are possible

```
> merge(dfA, dfB) # default: inner join
   kids ages state
1  Jill   10    NY
2 Laura   19    CA
> merge(dfA, dfB, all = TRUE) # outer join
   kids ages state
1 Alice   NA    MA
2  Jack   12  <NA>
3  Jill   10    NY
4 Laura   19    CA
> merge(dfA, dfB, all.x = TRUE) # left join
   kids ages state
1  Jack   12  <NA>
2  Jill   10    NY
3 Laura   19    CA
> merge(dfA, dfB, all.y = TRUE) # right join
   kids ages state
1 Alice   NA    MA
2  Jill   10    NY
3 Laura   19    CA
```

Handling Dataframe

- Merge dataframes
  - ✓ If different data frames have different column name strategy, explicitly state the column names to use

```
firstname <- c("Alice","Jill", "Laura")
state <- c("MA", "NY", "CA")
dfC <- data.frame(firstname, state, stringsAsFactors=FALSE)
dfC

merge(dfA, dfC, by.x="kids", by.y="firstname")
```

```
> dfC <- data.frame(firstname, state, stringsAsFactors=FALSE)
> dfC
  firstname state
1     Alice    MA
2      Jill    NY
3     Laura    CA
> merge(dfA, dfC, by.x="kids", by.y="firstname")
   kids ages state
1  Jill   10    NY
2 Laura   19    CA
```


ANY
questions?