

금융데이터 분석을 위한 R을 이용한 개발 입문

박찬엽

2017년 7월 20일

대부분의 에러 문제를 해결할 수 있는 기초 자료형에 대한 이해

자료형에 대해 알고 있는 것과 자료형을 파악하는 것은 매우 중요합니다. 함수가 입력 데이터로 사용이 가능한 데이터를 입력 데이터로 지정해 주어야 실행이 되기 때문인데요. 많은 분들이 여기에서 문제가 발생합니다. 특히 작업을 진행하면서 기대하지 않은 자료형으로 중간 결과물이 나오는 경우에는 이후에 생기는 문제에 대해서 이해하기 어려울 수 있습니다.

그렇기 때문에 기본 R에서 제공하는 자료형에 대해서 공부해 두는 것은 대부분의 함수가 사용하는 데이터의 형식을 공부하는 것과 같습니다. 물론 패키지마다 자체 자료형을 정의하기도 합니다. 하지만 지금 기초 자료형을 공부하면서 새로운 자료형을 파악하는 방법을 자연스럽게 배울 수 있을 거라 생각합니다.

자료형을 파악하는 방법은 class 함수를 사용하는 것입니다.

```
x <- 1  
class(1)
```

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

```
class(x)
```

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

```
class("x")
```

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

?class를 입력해서 class 함수에 대해서 파악해 보세요.

위에 class 함수의 결과로 나오는 것이 자료형의 종류입니다. numeric은 숫자를, character는 글자를 뜻합니다. 이제 아래에서 하나하나 확인해 보겠습니다. 기초 자료형에 대해서 [MS의 edX 강의](#), [datacamp](#), [r-tutor](#)의 내용을 참고했습니다.

기초 자료형

논리형 logical

logical(로지컬) 자료형은 기다 아니다 같이 True, False를 표현하기 위해 만들어진 자료형입니다. R에서는 각각 두 가지 표현법이 있습니다.

```
TRUE: class(TRUE)
```

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

```
## [1] "logical"
```

```
F: class(F)
```

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

```
## [1] "logical"
```

언어에 따라 True, False를 채택하는 경우도 있습니다만 R의 경우 위와 같이 전부 대문자인 글자를 logical 자료형으로 인식합니다.

이렇게 class 함수를 이용해서 자료형이 무엇인지 파악하는 것도 중요하지만, 조건문 등을 통해서 일을 자동적으로 수행하게 시키기 위해서는 class 같이 결과값이 다양한 것 보다는 logical 자료형으로 결과가 나오는 것이 앞으로 배우게 될 if 문 등에 조건으로 사용하기 좋습니다. 아래와 같은 is.logical 함수는 괄호안의 데이터가 logical 자료형이 맞는지를 물어보는 함수입니다. 다른 자료형도 대부분 is.자료형이름을 함수로 사용할 수 있습니다.

```
test <- TRUE  
is.logical(test)
```

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

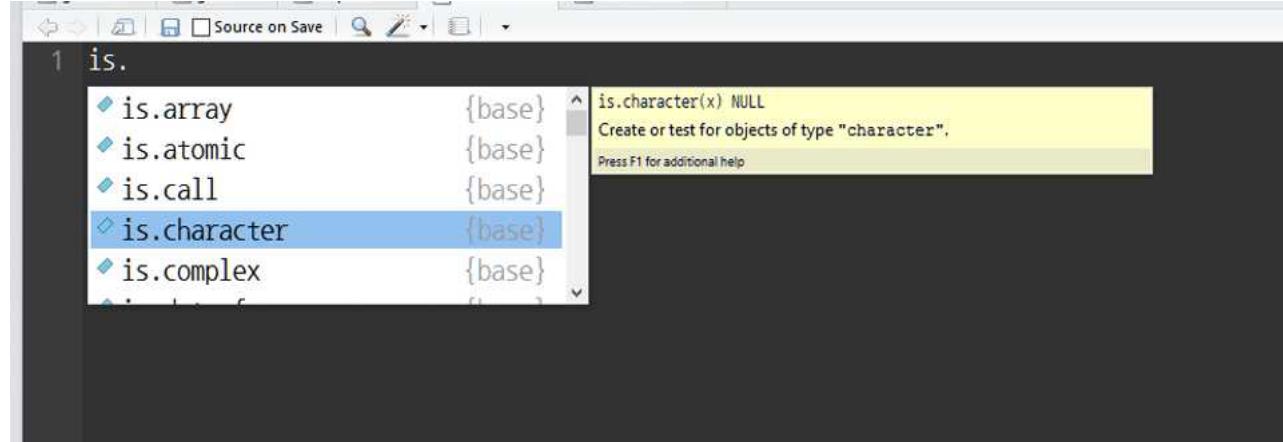
```
is.logical(TRUE)
```

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

```
is.logical("test")
```

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

R Studio에서는 함수의 자동완성 기능을 제공합니다. `is.`을 입력해서 다양한 자료형을 확인하는 명령어를 구경해보세요.



숫자형 numeric

numeric(뉴메릭) 자료형은 그 영어 뜻이 숫자의인 것 같이 숫자를 표현하는 자료형입니다. 물론 복소수 같이 다른 여러 표현들도 사용할 수 있습니다만, R에서 숫자는 보통 numeric 자료형으로 처리합니다.

```
2  
## [1] 2  
class(2)  
## [1] "numeric"  
is.numeric(2)  
## [1] TRUE
```

numeric은 소수점 이하 값을 가지는 숫자도 포함합니다.

```
2.5
```

```
## [1] 2.5
```

```
class(2.5)
```

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

```
is.numeric(2.5)
```

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

숫자를 표현하는 자료형으로 integer도 있습니다. 영어 뜻 그대로 정수를 표현하는 자료형인데요, 소수점이 없는 숫자를 표현할 때 사용합니다. R에서는 integer로 바로 인식시키기 위해서 정수인 숫자 뒤에 L(대문자 엘)을 붙여서 표현합니다.

```
2L
```

```
## [1] 2
```

```
class(2L)
```

```
## [1] "integer"
```

```
is.numeric(2L)
```

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

```
is.integer(2L)
```

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

integer는 numeric에 비해 차지하는 메모리가 적으며 때문에 연산이 좀 더 빠르다는 장점이 있습니다.

글자형 character

character(캐릭터) 자료형은 그 영어 뜻이 문자인 것 같이 글자를 표현하는 자료형입니다. R에서는 변수와 글자를 구분하기 위해서 글자의 앞뒤에 "를 붙이는데요. '도 같은 역할을 합니다. 그래서 "나 '" 사이에 있는 것은 모두 글자, 즉 character로 인식합니다.

```
"2"
```

```
## [1] "2"
```

```
class("2")
```

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

```
is.numeric("2")
```

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

```
is.character("2")
```

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

한글도 됩니다.

```
is.character("한글")
```

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

심지어 " 사이에 있는 ', ' 사이에 있는 "도 글자로 인식합니다. 이걸 이용해서 글자내에 "나 '가 들어가야 할 경우, 안에 들어가지 않는 것을 글자로 인식시키기 위한 표시로 사용하기도 합니다.

```
" " "
```

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

```
is.character(" " ")
```

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

이렇게 다른 자료형의 기반이 되는 기초 자료형 3개에 대해 알아보았습니다. 이 각각은 지금까지 데이터가 한 개인 경우였습니다. 그렇다면 이 자료형들이 모여 이루는 새로운 자료형에 대해서 알아보겠습니다.

벡터

하나의 기초 자료형만으로 구성된 1차원 데이터 집합

vector(벡터) 자료형은 가장 간단하게 하나의 자료형에 해당하는 데이터들의 1차원적인 집합입니다. vector를 만드는 가장 간단한 함수는 c입니다. ?c를 실행해서 내용을 한 번 읽어주세요.

```
vec_char <- c("a", "b", "c")  
vec_char
```

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

```
class(vec_char)
```

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

```
is.vector(vec_char); is.character(vec_char)
```

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

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

vec_char는 character이기도 하면서 vector입니다. 그럼 이번엔 숫자 벡터를 한번 만들어 보겠습니다.

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

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

```
class(vec_num)
```

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

```
is.vector(vec_num)
```

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

```
is.numeric(vec_num)
```

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

이번엔 vec_num0이 numeric이기도 하면서 vector입니다. 마지막으로 논리 벡터를 만들어 볼까요.

```
vec_logi <- c(T,T,T,F,F,T)
```

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

```
class(vec_logi)
```

```
## [1] "logical"
```

```
is.vector(vec_logi)
```

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

```
is.logical(vec_logi)
```

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

지금까지 살펴본 바에 따르면 vector라는 사실은 class로 파악할 수 없고, is.vector로 확인할 수 있습니다. 사실 파악할 수 없다보다는 파악할 필요가 없습니다. vector는 한 번에 같이 처리할 수 있는 데이터를 묶어둔 것이기 때문에 한 vector로 묶인 데이터들은 모두 자료형이 강제로 같게 만들어 집니다. 아래와 같이 다양한 시도를 해서 강제로 자료형이 통일되게 저장되는 것을 확인해 보세요.

```
vec_test1 <- c(1,2,3,"test")  
vec_test1
```

```
## [1] "1"    "2"    "3"    "test"
```

```
class(vec_test1)
```

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

이렇게 vector는 하나의 기초 자료형만으로 구성된 1차원 데이터 집합인 것을 알게 되었습니다. 그럼 이제 vector에서 일부의 데이터만 불러오는 방법을 알아보겠습니다.

vector에서 일부 데이터 불러오기

데이터를 모아 놓은 것은 좋은 전략이지만 그 중 일부를 자유롭게 사용할 수 있을 때 더욱 빛을 발하는 것 같습니다. vector는 같은 종류의 데이터를 한 줄로 늘어뜨려 놓은 것으로 상상하시면 좋을 것 같은데, 왜냐면 위치로 데이터의 일부를 선택하기 위해서입니다.

컴퓨터 언어에서 무엇을 언급하는 방법은 두 가지가 있습니다. 하나는 임의의 이름으로 지정된 주소값 자체를 언급하는 것이고, 다른 하나는 그 주소값에 이름을 부여해서 그 이름을 부르는 것입니다. 정확한 비유는 아닙니다만, vector에서 특정 위치에 있는 값을 부르는 것은 위치의 주소값이나 지정해둔 이름으로 값을 부르는 것과 비슷합니다. 값의 순서라고 볼 수 있는 주소값을 `index`라고 합니다.

R에서 `index`를 사용하는 방법은 [](대괄호)를 변수뒤에 붙여서 사용합니다.

```
vec_ind <- c("b", "a", "ab", "b", "o", "ab")
vec_ind[1]
```

```
## [1] "b"
```

위에서 지정해둔 이름으로 값을 부르는 방법이 있다고 했습니다. 그러면 우선 vector의 값에 이름을 부여하는 방법을 살펴보겠습니다. 새롭게 살펴볼 명령어는 `names`입니다. `?names`를 실행해서 살펴보세요.

```
blood_type1 <- c(bob = "A", sam = "B", john="O", jim="AB", tom="A")  
blood_type1
```

```
## bob sam john jim tom  
## "A" "B" "O" "AB" "A"
```

```
names(blood_type1)
```

```
## [1] "bob" "sam" "john" "jim" "tom"
```

위의 첫번째 방법은 처음 vector를 만들 때 부터 이름을 짓고 시작하는 것입니다. =를 기준으로 왼쪽이 이름, 오른쪽이 이름에 해당하는 값입니다. 그럼 다른 방법을 보겠습니다.

```
(blood_type2 <- c("A", "B", "O", "AB", "A"))
```

```
## [1] "A" "B" "O" "AB" "A"
```

```
names(blood_type2)
```

```
## NULL
```

```
(blood_type2_name <- c("bob", "sam", "john", "jim", "tom"))
```

```
## [1] "bob" "sam" "john" "jim" "tom"
```

```
names(blood_type2) <- blood_type2_name  
blood_type2
```

```
## bob sam john jim tom  
## "A" "B" "O" "AB" "A"
```

names 함수가 vector의 이름을 보여주는 함수이기도 하면서 값을 지정받을 수 있는 변수의 기능도 합니다. names(blood_type2) 전체가 하나의 변수로 동작하는 것이지요. 그래서 처음 선언한 vector에 이름이 없는 것을 확인하고 새롭게 이름에 해당하는 변수 blood_type2_name을 선언한 후에 names(blood_type2)에 선언해줌으로써 이름을 작성했습니다. 두번째 방법을 보면 언제든 새롭게 이름을 지정해 줄 수 있음을 알 수 있습니다.

원래 내용으로 돌아가서, 그럼 이제 이름으로 그 위치에 해당하는 값을 출력해 보겠습니다.

```
blood_type1 <- c(bob = "A", sam = "B", john="O", jim="AB", tom="A")
blood_type1[bob]
```

조금 이상합니다. Error: object 'bob' not found라고 하면서 동작하질 않네요. Error 메세지는 함수가 작동하는데 문제가 있어서 실행이 되질 않았다는 뜻입니다. (참고로 Warning은 실행은 됬지만 기대하는 결과가 아닐 수 있으니 확인해보라는 뜻입니다.) object는 지금까지 한글로 변수라고 불렀습니다. 그럼 변수인 bob이 없다는 에러가 발생한 겁니다. 글자형에서 R에서는 변수와 글자를 구분하기 위해서 글자의 앞뒤에 "를 붙인다고 했습니다. 그렇다면 blood_type1[bob]에서 bob의 앞뒤에 "표시를 하지 않았기 때문에 R이 변수로 인식한 듯합니다. 그럼 "로 묶어서(앞뒤에 표시해서) 다시 실행해 보겠습니다.

```
blood_type1 <- c(bob = "A", sam = "B", john="O", jim="AB", tom="A")
blood_type1["bob"]
```

```
## bob
## "A"
```

이제 원하는 결과가 나왔습니다. 이런 문법에 대해서는 규칙을 정확히 이해하는 것도 중요하지만 계속 console에 입력해보면서 테스트를 하는 것이 매우 중요합니다. 그렇게 하면 자주 사용하는 것은 익숙해지고, 기억이 흐릿한 것은 확인해가며 코드를 작성할 수 있게 됩니다.

더 확인해봐야 할 것이 있습니다. 지금까지 한 개만 불러오는 것을 했는데, 여러 개를 동시에 해도 되는지를 확인해 보겠습니다.

```
blood_type1 <- c(bob = "A", sam = "B", john="O", jim="AB", tom="A")
blood_type1[c("bob", "john")]
```

```
## bob john
## "A" "O"
```

여러 개도 가능합니다. 여러 개를 지정하기 위해 `c`함수를 함께 사용한 것에 주목해 주세요. `c`로 이름을 묶어서 사용하지 않으면 어떻게 되는지 확인해 보세요.

```
blood_type1 <- c(bob = "A", sam = "B", john="O", jim="AB", tom="A")
blood_type1["bob", "john"]
```

Error in blood_type1["bob", "john"] : incorrect number of dimensions이라는 Error를 보셨으면 맞게 동작하는 것입니다. 메세지를 한번 잘 보겠습니다. `dimension`은 차원이라는 뜻으로 제가 `vector`는 1차원 데이터 집합이라는 표현을 썼었습니다. 그렇다면 차원이 맞지 않게 요청이 되었다는 뜻입니다. R은 []안에서 ,(쉼표)는 차원을 구분하는 것으로 인식합니다. 예를 들어 [, , ,]라고 하면 4차원 데이터를 뜻한다고 할 수 있습니다. 그렇기 때문에 `blood_type1["bob", "john"]`은 저희가 의도하지 않는 방식으로 R이 이해한 것입니다. R이 이해하는 대로 해석하면 첫번째 차원은 이름이 'bob'인 데이터를, 두번째 차원은 이름이 'john'인 데이터를 불러옴이 됩니다. 하지만 우리는 첫번째 차원에서 이름이 'bob'인 데이터와 이름이 'john'인 데이터를 불러옴을 실행하고 싶기 때문에 같은 차원에서 두 개의 이름을 사용했다는 것을 R에게 알려줘야 합니다. 그래서 `c` 함수를 사용해서 이름을 묶어, 여러 데이터를 불러오는 이름을 사용할 때 `vector`를 사용한 것입니다. 이 방법은 차원이 늘어나거나 하더라도 같은 문법을 사용합니다. 그럼 이제 2차원 데이터 집합인 `matrix`를 살펴보겠습니다.

메트릭스

vector의 2차원 확장

`matrix`는 간단하게 1차원의 `vector`를 2차원으로 확장한 것입니다. 여기서 2차원이란 1차원 `vector`가 두 줄이 생겼다는 것이 아니라 행과 열로 2차원을 표현하게 됩니다. `matrix`는 많이 보시는 엑셀 시트와 같은 형태를 띕니다. 차원의 의미가 여러 가지가 가능하겠습니다만, 지금의 [차원](#)은 데이터를 표현하는 축의 갯수라고 이해하시면 좋을 것 같습니다.

소개해야 할 자료형 중에 `array`가 있는데 배열이라는 뜻입니다. `vector`는 1차원 `array`의 특별한 이름이고, `matrix`는 2차원 `array`의 특별한 이름입니다. 3차원 이상 부터는 전부 n차원 `array`라고 합니다.

그래서 vector와 matrix, array는 앞서 vector에서 확인했던 문법적 규칙과 모두 같은 규칙을 따릅니다. 그럼 이제 matrix를 한번 만들어 보겠습니다.

```
matrix(1:6, nrow = 2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
matrix(1:6, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
matrix(1:6, nrow = 2, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

위에서 1:6은 1부터 6까지의 숫자 벡터를 뜻합니다. c(1,2,3,4,5,6)과 같습니다. row는 행을, col은 column의 줄임말로 열을 나타내구요. byrow옵션이 TRUE면 row방향으로 먼저 벡터를 나열하라는 뜻입니다.

`matrix`는 행열의 크기를 지정하기 때문에 지정된 크기보다 작은 양의 데이터를 선언할 때 어떻게 동작할 것인지 정해져 있습니다.

```
matrix(1:3, nrow = 2, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    2    1    3
```

```
matrix(1:4, nrow = 2, ncol = 3)
```

```
## Warning in matrix(1:4, nrow = 2, ncol = 3): 데이터의 길이[4]가 열의 개수[3]
## 의 배수가 되지 않습니다
```

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

첫번째 명령에서는 부족한 갯수만큼 있던 것을 다시 사용해서 자동으로 채워줬네요. 두번째 명령에서도 같은 방식으로 동작한 것 같은데 Warning 메세지가 나왔습니다. 행의 갯수의 약수거나 배수이지 않다는 것을 알려줌으로써 의도적인 행동인지 확인할 수 있게 도와줍니다. 결과를 확인해보면, 4개까지 사용하고 다시 처음부터 1, 2를 사용하면서 행열의 크기만큼 데이터를 채웠습니다. Warning이 나오면 확인해 보면 좋겠네요.

행열의 모양을 배우면서 이제 cbind와 rbind를 확인해 보겠습니다. cbind는 **열 방향으로 묶는다.**라는 의미이구요. rbind는 **행 방향으로 묶는다**는 뜻입니다. 아래 코드로 어떻게 행열의 모양이 달라지는지 확인해 보세요.

```
cbind(1:3, 1:3)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
```

```
rbind(1:3, 1:3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
```

```
m <- matrix(1:6, byrow = TRUE, nrow = 2)
```

```
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
rbind(m, 7:9)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
cbind(m, c(10, 11))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3   10
## [2,]    4    5    6   11
```

vector에서 사용한 names처럼, 행열에 대응하는 함수가 있습니다. rownames, colnames가 그것인데요. ?rownames와 ?colnames로 자세한 내용을 더 확인해 보시기 바랍니다.

```
m <- matrix(1:6, byrow = TRUE, nrow = 2)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
rownames(m)
```

```
## NULL
```

```
rownames(m) <- c("row1", "row2")
m
```

```
##      [,1] [,2] [,3]
## row1    1    2    3
## row2    4    5    6
```

```
colnames(m)
```

```
## NULL
```

```
colnames(m) <- c("col1", "col2", "col3")
m
```

```
##      col1 col2 col3
## row1    1    2    3
## row2    4    5    6
```

vector처럼 처음부터 만들 때 이름을 지정해 줄 수도 있습니다.

```
m <- matrix(1:6, byrow = TRUE, nrow = 2,
            dimnames = list(c("row1", "row2"),
                           c("col1", "col2", "col3")))
m
```

```
##      col1 col2 col3
## row1    1    2    3
## row2    4    5    6
```

matrix도 vector와 같이 한 matrix내의 데이터는 모두 같은 자료형이어야 합니다. 이건 자연스럽다고 생각하는 건지 Warning도 주지 않으니 조심하셔야 합니다.

```
num <- matrix(1:8, ncol = 2)
char <- matrix(LETTERS[1:6], nrow = 4, ncol = 3)
cbind(num, char)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "1"  "5"  "A"  "E"  "C"
## [2,] "2"  "6"  "B"  "F"  "D"
## [3,] "3"  "7"  "C"  "A"  "E"
## [4,] "4"  "8"  "D"  "B"  "F"
```

matrix에서 일부 데이터 불러오기

matrix에서 일부 데이터를 불러오는 방법은 vector와 완전히 같습니다. vector에서 []를 설명하면서 차원에 대해 이야기 했었는데요. matrix는 2차원이기 때문에 중간에 쉼표가 들어가야 합니다.

```
m <- matrix(sample(1:15, 12), nrow = 3)
m
```

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

```
m[1,3]
```

```
## [1] 2
```

```
m[3,2]
```

```
## [1] 11
```

쉼표만 사용하고 어느 하나의 차원을 지정하지 않으면, 모두 불러온 것으로 가정합니다.

```
m[3,]
```

```
## [1] 5 11 9 13
```

```
m[,3]
```

```
## [1] 2 4 9
```

신기하게도, 1차원인 vector의 일부 데이터 불러오기의 문법도 사용할 수 있습니다. m[9]가 어느 위치의 값인지 잘 찾아보시기 바랍니다.

```
m[4]
```

```
## [1] 14
```

```
m[9]
```

```
## [1] 9
```

여전히 vector와 같이 어느 한 차원 내에서 여러개를 사용하기 위해서는 c와 함께 사용합니다.

```
m[2, c(2, 3)]
```

```
## [1] 15 4
```

```
m[c(1, 2), c(2, 3)]
```

```
##      [,1] [,2]
## [1,]    14    2
## [2,]    15    4
```

```
m[c(1, 3), c(1, 3, 4)]
```

```
##      [,1] [,2] [,3]
## [1,]    6    2   10
## [2,]    5    9   13
```

역시 이름을 사용해서 일부 데이터만 불러오는 것도 가능합니다.

```
rownames(m) <- c("r1", "r2", "r3")
colnames(m) <- c("a", "b", "c", "d")
m
```

```
##      a b c d
## r1  6 14 2 10
## r2 12 15 4  8
## r3  5 11 9 13
```

```
m[2,3] : m["r2","c"] : m[2,"c"] : m[3, c("c", "d")]
```

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

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

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

```
##  c d
## 9 13
```

팩터

명목형 변수

R에 대해서 이야기하면서 통계에 대한 이야기가 나오지 않을 수가 없는데요. factor는 categorical data를 표현하기 위해 만들어진 자료형이기 때문입니다. 통계에서 사용하는 자료형은 [여기](#)를 참고하세요. 가장 대표적인 categorical data인 혈액형을 예시로 factor를 사용해 보겠습니다.

```
blood <- c("B", "AB", "O", "A", "O", "O", "A", "B")
str(blood)
```

```
## chr [1:8] "B" "AB" "O" "A" "O" "O" "A" "B"
```

```
blood_factor <- factor(blood)
str(blood_factor)
```

```
## Factor w/ 4 levels "A","AB","B","O": 3 2 4 1 4 4 1 3
```

자료의 상태를 파악하는 함수로 `str`를 사용했습니다. `str`은 변수의 구조가 어떻게 구성되어 있는지를 보여주는 함수로, 데이터를 파악하는데 좋은 방법입니다. 비슷한 다른 방법들로, `summary`, `head` 등이 있습니다. `?str`, `?summary`, `?head`를 입력해서 내용을 확인해 보세요.

`chr [1:8] "B" "AB" "0" "A" "0" "0" "A" "B"`에서 `chr`는 character의 줄임 표현입니다. 그 다음 [1:8]은 데이터가 8개 있다는 뜻입니다. 그 이후에는 8개의 데이터가 모두 출력되었습니다.

`facter`는 character와 다르게 Levels: A AB B 0라는 줄이 추가 되었습니다. 그리고 데이터를 표현할 때 "가 없어졌네요. 우선 그것만으로도 character가 아님을 판단할 수 있습니다. `str(blood_factor)`의 결과도 보겠습니다.

Factor w/ 4 levels "A", "AB", "B", "0": 3 2 4 1 4 4 1 3에서 독특한 부분이 있습니다. 데이터가 숫자로 되어 있네요.

데이터와 비교해 보면 levels "A", "AB", "B", "0"에서 3번째는 "B"입니다. 첫번째 데이터가 "B"인걸 보니 숫자는 levels에서 몇 번째 데이터인지를 뜻하는 것 같습니다. 다른 숫자들도 확인해 보세요.

vector처럼 변수에 선언할 때 처음부터 levels를 지정할 수 도 있습니다. 순서를 지정하는 것도 가능합니다. 처음에 만든 blood_factor와 blood_factor2가 어떻게 다른지 비교해 보세요.

```
blood_factor2 <- factor(blood,
                         levels = c("0", "A", "B", "AB"))
blood_factor2
```

```
## [1] B AB 0 A 0 0 A B
## Levels: 0 A B AB
```

```
str(blood_factor2)
```

```
## Factor w/ 4 levels "0","A","B","AB": 3 4 1 2 1 1 2 3
```

```
str(blood_factor)
```

```
## Factor w/ 4 levels "A","AB","B","0": 3 2 4 1 4 4 1 3
```

names처럼 levels도 중간에 바꿀 수 있습니다. labels라는 것도 있는데, 이것은 데이터의 값을 뜻합니다. 어떻게 달라지는지 확인해 보세요.

```
blood <- c("B", "AB", "O", "A", "O", "A", "B")
blood_factor <- factor(blood)
blood_factor
```

```
## [1] B AB O A O A B
## Levels: A AB B O
```

```
levels(blood_factor) <- c("BT_A", "BT_AB", "BT_B", "BT_O")
blood_factor
```

```
## [1] BT_B BT_AB BT_O BT_A BT_O BT_O BT_A BT_B
## Levels: BT_A BT_AB BT_B BT_O
```

```
factor(blood, labels = c("BT_A", "BT_AB", "BT_B", "BT_O"))
```

```
## [1] BT_B BT_AB BT_O BT_A BT_O BT_O BT_A BT_B
## Levels: BT_A BT_AB BT_B BT_O
```

factor는 명목형 변수여서 크기를 비교할 수 없습니다.

```
blood <- c("B", "AB", "O", "A", "O", "A", "B")
blood_factor <- factor(blood)
blood_factor[1] < blood_factor[2]
```

```
## Warning in Ops.factor(blood_factor[1], blood_factor[2]): '<' not meaningful
## for factors
```

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

하지만 크기를 비교할 수 있는 형태로 선언할 수 도 있습니다.

```
tshirt <- c("M", "L", "S", "S", "L", "M", "L", "M")
tshirt_factor <- factor(tshirt, ordered = TRUE,
                        levels = c("S", "M", "L"))
tshirt_factor
```

```
## [1] M L S S L M L M
## Levels: S < M < L
```

```
tshirt_factor[1] < tshirt_factor[2]
```

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

리스트

다른 종류의 자료형들이 함께

지금까지 전부 같은 자료형의 데이터가 모여 있는 형태의 자료형을 알아봤습니다. 같은 자료형으로 구성하는 것은 계산의 효율등 분명한 장점도 있지만, 그것을 사용하는 사람이 이해하기 어렵거나 불편할 때가 있습니다. 특히 다양한 자료형으로 구성된 데이터일 때 하나의 변수로 관리하기 위해서는 `list`를 사용해 합니다. 아래 `c`와 `list`로 만든 데이터를 비교해 보겠습니다.

```
c("Rsome times", 190, 5)
```

```
## [1] "Rsome times"    "190"      "5"
```

```
list("Rsome times", 190, 5)
```

```
## [[1]]
## [1] "Rsome times"
##
## [[2]]
## [1] 190
##
## [[3]]
## [1] 5
```

list 역시 is.list 함수가 있네요. c로 만들어진 데이터는 자료형을 강제로 전부 character로 바꾸었습니다. list는 데이터가 주욱 늘어지고, 숫자들은 숫자 데이터로 표현된 것 같습니다. 이 때 처음 보는 것이 있는데요. [[]]입니다. 이것은 vector가 하나의 자료형으로만 이루어져야 하는 점에 착안하여, vector를 여러 개 합쳐 list를 만듬으로써 문제를 해결했습니다. 그렇기 때문에 일부의 데이터를 불러오는 []과 비슷하지만 합쳐져 있는 그 내부의 vector를 불러오는 [[]] 문법이 만들어 지게 되었습니다. []과 [[]]를 실험해 보세요.

```
song <- list("Rsome times", 190, 5)
song[3]
```

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

```
class(song[3])
```

```
## [1] "list"
```

```
song[[3]]
```

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

```
class(song[[3]])
```

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

list도 다른 자료형들 처럼 names를 사용합니다.

```
song <- list("Rsome times", 190, 5)
names(song) <- c("title", "duration", "track")
song
```

```
## $title
## [1] "Rsome times"
##
## $duration
## [1] 190
##
## $track
## [1] 5
```

이번엔 [[]] 위치에 \$title 같이 \$ + names의 문법이 나타났습니다. 이것이 list의 이름을 짓는 방법입니다. 이름으로 데이터를 부르는 방법은 더 복잡한 모양의 데이터를 만들고 실습해보기로 하고, list가 가진 다른 특성을 확인해 보겠습니다.

```
song <- list(title = "Rsome times",
             duration = 190,
             track = 5)
str(song)
```

```
## List of 3
## $ title  : chr "Rsome times"
## $ duration: num 190
## $ track   : num 5
```

```
similar_song <- list(title = "R you on time?",
                      duration = 230)

song <- list(title = "Rsome times",
             duration = 190, track = 5,
             similar = similar_song)

str(song)
```

```
## List of 4
## $ title  : chr "Rsome times"
## $ duration: num 190
## $ track   : num 5
## $ similar :List of 2
##   ..$ title  : chr "R you on time?"
##   ..$ duration: num 230
```

list 안에 list가 들어가 있네요 str로 확인해 보니 list 안에 list가 ..\$로 들여쓰기 되어 표현되어 있습니다. 데이터 표현도 List of 2라고 str(song)의 맨 윗줄(List of 4)과 같은 모양이 보입니다. 이것처럼 list는 대부분의 자료형을 요소로 가질 수 있습니다.

list는 인터넷에서 많이 사용하는 자료형인 JSON에 대응됩니다. 과거에는 XML도 사용했다고 알고 있는데, 최근에는 많은 곳에서 JSON으로 사용하고 있습니다. JSON에 대해서는 [이곳](#)의 예제를 참고하세요

```
library(N2H4)
url<-“http://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=100&oid=437&aid=0000152054”
tem<-getComment(url)
```

```
## [1] "success : TRUE"
```

```
str(tem)
```

```
## 'data.frame': 1 obs. of 7 variables:
## $ success: logi TRUE
## $ code   : chr "1000"
## $ message: chr "요청을 성공적으로 처리하였습니다."
## $ lang   : chr "ko"
## $ country: chr "KR"
## $ result :'data.frame': 1 obs. of 11 variables:
##   ..$ listStatus    : chr "current"
##   ..$ sort          : chr "FAVORITE"
##   ..$ graph         :'data.frame': 1 obs. of 3 variables:
##     ..$ gender:'data.frame': 1 obs. of 2 variables:
##       ..$ male : int 75
##       ..$ female: int 25
##   ..$ old   :List of 1
##     ..$ :data.frame': 5 obs. of 5 variables:
##       ..$ age : chr "10" "20" "30" "40" ...
##       ..$ value: int 2 18 33 24 22
##       ..$ type : chr "exact" "exact" "exact" "exact" ...
##       ..$ min  : logi TRUE FALSE FALSE FALSE FALSE
##       ..$ max  : logi FALSE FALSE TRUE FALSE FALSE
##       ..$ empty : logi FALSE
##     ..$ count      :'data.frame': 1 obs. of 8 variables:
##       ..$ comment   : int 1038
##       ..$ reply     : int 480
##       ..$ exposeCount: int 1110
##       ..$ delCommentByUser: int 0
##       ..$ delCommentByMon : int 4
##       ..$ blindCommentByUser: int 0
##       ..$ blindReplyByUser: int 0
##       ..$ total     : int 1518
##     ..$ config      :'data.frame': 1 obs. of 76 variables:
##       ..$ useSnsLogin        : logi TRUE
##       ..$ lineFeedOn          : logi FALSE
##       ..$ useBest             : logi FALSE
##       ..$ useVote              : logi TRUE
##       ..$ useVoteSelf          : logi FALSE
##       ..$ useVoteGoodOnly      : logi FALSE
##       ..$ useReport            : logi TRUE
##       ..$ useCommonReport      : logi FALSE
```

```
## ...$ useSort : logi TRUE
## ...$ sortTypes :List of 1
## ...$ : chr "FAVORITE" "NEW" "RELATIVE"
## ...$ defaultSort : chr "FAVORITE"
## ...$ useByteLength : logi FALSE
## ...$ useReply : logi TRUE
## ...$ useAutoRefresh : logi FALSE
## ...$ min : int 1
## ...$ max : int 300
## ...$ useSticker : logi FALSE
## ...$ stickerOnly : logi FALSE
## ...$ stickerSupportedCategories : logi NA
## ...$ stickerDefaultCategory : logi NA
## ...$ stickerCategory : logi NA
## ...$ stickerContentsUrl : logi NA
## ...$ stickerKeyUrl : logi NA
## ...$ stickerTabUrl : logi NA
## ...$ stickerType : logi NA
## ...$ stickerText : logi FALSE
## ...$ stickerMarketUrl : logi NA
## ...$ stickerMobileResize : logi FALSE
## ...$ useProfile : logi FALSE
## ...$ profileEmptyImage : logi NA
## ...$ displayMaskedUserId : logi FALSE
## ...$ useManager : logi FALSE
## ...$ managerDelete : logi FALSE
## ...$ managerBlock : logi FALSE
## ...$ managerNotice : logi FALSE
## ...$ contentsManagerIcon : logi FALSE
## ...$ secretComment : logi FALSE
## ...$ exposureConfig : logi FALSE
## ...$ deleteAllAfterBlock : logi FALSE
## ...$ useFold : logi TRUE
## ...$ replyPreviewCount : int 0
## ...$ maxImageUploadCount : int 0
## ...$ maxImageUploadFileSize : int 0
## ...$ imageAutoRotate : logi FALSE
## ...$ autoRefreshTime : int 0
## ...$ autoRefreshDefaultOff : logi FALSE
## ...$ autoRefreshChat : logi FALSE
## ...$ maxChatFPS : int 0
## ...$ commentModify : logi FALSE
## ...$ useCommentModify : logi FALSE
## ...$ useViewAll : logi TRUE
## ...$ useSnsComment : logi TRUE
## ...$ snsCommentDefaultOn : logi TRUE
## ...$ useUserLevel : logi FALSE
## ...$ useUserBlind : logi FALSE
## ...$ maxUserBlindCount : int 0
## ...$ useImageComment : logi FALSE
## ...$ useUrlLink : logi FALSE
## ...$ useCommentListIncludeDelete: logi FALSE
## ...$ useStats : logi TRUE
## ...$ useGpopCache : logi FALSE
## ...$ useEnterSubmit : logi FALSE
## ...$ useListReverse : logi FALSE
## ...$ useTranslation : logi FALSE
## ...$ useDeletedList : logi TRUE
## ...$ useAudio : logi FALSE
## ...$ combinedDeletedList : logi TRUE
## ...$ displayDeletedList : logi FALSE
## ...$ photoInfraUploadDomain : logi NA
## ...$ photoInfraSelectDomainHttp : logi NA
```

```
## ...$ photoInfraSelectDomainHttps: logi NA
## ...$ pcTempThumbnailType : chr "ff80_80"
## ...$ mobileTempThumbnailType : chr "ff50_50"
## ...$ realnameVerificationBlock : logi FALSE
## ...$ realnameVerificationMessage: logi NA
## ...$ useBlindByReport : logi TRUE
## ...$ exposureConfig:'data.frame': 1 obs. of 2 variables:
## ...$ reason: logi NA
## ...$ status: chr "COMMENT_ON"
## ...$ bestList :List of 1
## ...$ : list()
## ...$ notice : 'data.frame': 1 obs. of 4 variables:
## ...$ noticeNo: int 12
## ...$ title : chr "6/22 밤10시부터 댓글접기/이력공개 오픈"
## ...$ content : chr "안녕하세요, 네이버 뉴스입니다. Wr<br>Wr<br>하루에도 수 많은 의견들이 교환되는 뉴스 댓글 공간을 더 투명하게 서비스" | __truncated__
## ...$ regTime : chr "2017-06-22T22:51:57+0900"
## ...$ userInfo :'data.frame': 1 obs. of 0 variables
## ...$ pageModel :'data.frame': 1 obs. of 16 variables:
## ...$ page : int 1
## ...$ pageSize : int 10
## ...$ indexSize : int 10
## ...$ startRow : int 1
## ...$ endRow : int 10
## ...$ totalRows : int 1038
## ...$ startIndex : int 0
## ...$ totalPages : int 104
## ...$ firstPage : int 1
## ...$ prevPage : int 0
## ...$ nextPage : int 2
## ...$ lastPage : int 10
## ...$ current : logi NA
## ...$ moveToLastPage: logi FALSE
## ...$ moveToComment : logi FALSE
## ...$ moveToLastPrev: logi FALSE
## ...$ commentList :List of 1
## ...$ : 'data.frame': 10 obs. of 68 variables:
## ...$ ticket : chr "news" "news" "news" "news" ...
## ...$ objectId : chr "news437,0000152054" "news437,0000152054" "news437,0000152054" "news437,0000152054" ...
## ...$ categoryId : chr "*" "*" "*" ...
## ...$ templateId : chr "default" "view" "view_politics" "default" ...
## ...$ commentNo : int 895990872 895990402 895993542 895994142 895997652 895994952 895998652 895990312 895993642 896006322
## ...$ parentCommentNo : int 895990872 895990402 895993542 895994142 895997652 895994952 895998652 895990312 895993642 896006322
## ...$ replyLevel : int 1 1 1 1 1 1 1 1 1 1
## ...$ replyCount : int 46 28 25 6 0 1 3 4 0 0
## ...$ replyAllCount : int 0 0 0 0 0 0 0 0 0 0
## ...$ replyPreviewNo : logi NA NA NA NA NA NA ...
## ...$ replyList : logi NA NA NA NA NA NA ...
## ...$ imageCount : int 0 0 0 0 0 0 0 0 0 0
## ...$ imageList : logi NA NA NA NA NA NA ...
## ...$ imagePathList : logi NA NA NA NA NA NA ...
## ...$ imageWidthList : logi NA NA NA NA NA NA ...
## ...$ imageHeightList : logi NA NA NA NA NA NA ...
## ...$ commentType : chr "txt" "txt" "txt" "txt" ...
## ...$ stickerId : logi NA NA NA NA NA NA ...
## ...$ sticker : logi NA NA NA NA NA NA ...
## ...$ sortValue : num 1.49e+12 1.49e+12 1.49e+12 1.49e+12 1.49e+12 ...
## ...$ contents : chr "국회의원에게 권력을 나눠주느니 차라리 대통령중심제 해라. 각당 원내대표들이 돌아가며 총리 해먹는 꼴 못본다" "4년 중임제가 답이다" "의원내각제는 진짜로 박지원 상왕 만들자"
## ...$ userIdNo : chr "oCJq" "5qcIZ" "9oKyX" "6U92Q" ...
## ...$ lang : chr "ko" "ko" "ko" "ko" ...
## ...$ country : chr "KR" "KR" "KR" "KR" ...
## ...$ idType : chr "naver" "naver" "naver" "naver" ...
## ...$ idProvider : chr "naver" "naver" "naver" "naver" ...
## ...$ userName : chr "ssan****" "zamp****" "tjto****" "hb12****" ...
```

```
## ... .$.userProfileImage: chr "http://profile.phinf.naver.net/47661/c2300c203d789c4761cb7256042c54e1434afc037bc9e510593d610b89108090.jpg" "http://profile.phinf.naver.net/27148/f77ca3b0a86973c6273"
## ... .$.profileType : chr "naver" "naver" "naver" "naver" ...
## ... .$.modTime : chr "2017-04-12T21:00:55+0900" "2017-04-12T21:00:21+0900" "2017-04-12T21:04:44+0900" "2017-04-12T21:05:26+0900" ...
## ... .$.modTimeGmt : chr "2017-04-12T12:00:55+0000" "2017-04-12T12:00:21+0000" "2017-04-12T12:04:44+0000" "2017-04-12T12:05:26+0000" ...
## ... .$.regTime : chr "2017-04-12T21:00:55+0900" "2017-04-12T21:00:21+0900" "2017-04-12T21:04:44+0900" "2017-04-12T21:05:26+0900" ...
## ... .$.regTimeGmt : chr "2017-04-12T12:00:55+0000" "2017-04-12T12:00:21+0000" "2017-04-12T12:04:44+0000" "2017-04-12T12:05:26+0000" ...
## ... .$.sympathyCount : int 2360 1907 1914 645 470 428 446 404 262 237
## ... .$.antipathyCount : int 70 100 152 37 5 10 39 34 4 5
## ... .$.userBlind : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.hideReplyButton : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.status : int 0 0 0 0 0 0 0 0 0 0
## ... .$.mine : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.best : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.mentions : logi NA NA NA NA NA NA ...
## ... .$.userStatus : int 0 0 0 0 0 0 0 0 0 0
## ... .$.categoryImage : logi NA NA NA NA NA NA ...
## ... .$.open : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.levelCode : logi NA NA NA NA NA NA ...
## ... .$.sympathy : logi FALSE FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.antipathy : logi FALSE FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.snsList : logi NA NA NA NA NA NA ...
## ... .$.metaInfo : logi NA NA NA NA NA NA ...
## ... .$.extension : logi NA NA NA NA NA NA ...
## ... .$.audioInfoList : logi NA NA NA NA NA NA ...
## ... .$.translation : logi NA NA NA NA NA NA ...
## ... .$.report : logi NA NA NA NA NA NA ...
## ... .$.visible : logi TRUE TRUE TRUE TRUE TRUE TRUE ...
## ... .$.manager : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.deleted : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.blindReport : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.shardTableNo : int 144 144 144 144 144 144 144 144 144 144
## ... .$.blind : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.secret : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.expose : logi TRUE TRUE TRUE TRUE TRUE TRUE ...
## ... .$.profileUserId : logi NA NA NA NA NA NA ...
## ... .$.containText : logi TRUE TRUE TRUE TRUE TRUE TRUE ...
## ... .$.maskedUserId : chr "ssan****" "zamp****" "tjto****" "hb12****" ...
## ... .$.maskedUserName : chr "ss****" "za****" "tj****" "hb****" ...
## ... .$.validateBanWords: logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.exposeByCountry : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## ... .$.virtual : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## $ date : chr "2017-07-20T03:38:50+0000"
```

JSON을 전문적으로 다루기 위한 패키지도 있으니 확인해 보세요.

데이터프레임

사람이 이해하기 쉬운 자료형

다양한 형태의 자료형을 묶어서 사용하는 list를 만들고 보니, 반대로 너무 자유도가 높아서 사용하기 어려운 문제들이 나타났습니다. 그렇다 보니 조금 제약사항을 만들어 보기로 합니다. 그렇게 해서 탄생한게 data.frame입니다. 기존에 알고 계시는 엑셀 시트나 설문조사를 정리한 표 같은 걸 생각하시면 도움이 되실 겁니다. data.frame은 list와 달리 vector를 각 열로 유지하여 합치는 방식을 사용했습니다. 그래서 하나의 열 내에서는 데이터가 vector와 같이 자료형이 모두 같아야 하고, 하나의 행에서는 자료형이 여러 개가 가능합니다.

```
name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
age <- c(28, 30, 21, 39, 35)
child <- c(FALSE, TRUE, TRUE, FALSE, TRUE)
df <- data.frame(name, age, child)
df
```

```
##   name age child
## 1 Anne  28 FALSE
## 2 Pete  30  TRUE
## 3 Frank 21  TRUE
## 4 Julia 39 FALSE
## 5 Cath  35  TRUE
```

물론 names도 사용합니다. names는 열의 이름을 뜻합니다. 똑같이 처음부터 지정해 줄수 있구요. 지정하지 않으면 임의로 data.frame이 선정해서 가지고 있습니다.

```
names(df)
```

```
## [1] "name"  "age"   "child"
```

```
names(df) <- c("Name", "Age", "Child")
df
```

```
##   Name Age Child
## 1 Anne 28 FALSE
## 2 Pete 30 TRUE
## 3 Frank 21 TRUE
## 4 Julia 39 FALSE
## 5 Cath 35 TRUE
```

```
(df <- data.frame(Name = name, Age = age, Child = child))
```

```
##   Name Age Child
## 1 Anne 28 FALSE
## 2 Pete 30 TRUE
## 3 Frank 21 TRUE
## 4 Julia 39 FALSE
## 5 Cath 35 TRUE
```

`data.frame`은 각 열의 데이터 갯수가 맞지 않으면 만들어 지지 않습니다.

```
> data.frame(name[-1], age, child)
## Error in data.frame(name[-1], age, child) :
##   arguments imply differing number of rows: 4, 5
```

그리고 글자를 모두 `factor`를 기본 값으로 지정합니다. 그래서 `stringsAsFactors` 옵션을 `FALSE`로 해줘야 `character`로 데이터를 만들 수 있습니다. `stringsAsFactors`는 `options`에서도 지정해서 사용할 수 있습니다.

```
df <- data.frame(name, age, child,
                  stringsAsFactors = FALSE)
str(df)
```

```
## 'data.frame': 5 obs. of 3 variables:
## $ name : chr "Anne" "Pete" "Frank" "Julia" ...
## $ age : num 28 30 21 39 35
## $ child: logi FALSE TRUE TRUE FALSE TRUE
```

list와 같이 위치값이나 이름으로 데이터의 일부를 불러올 수 있습니다.

```
name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
age <- c(28, 30, 21, 39, 35)
child <- c(FALSE, TRUE, TRUE, FALSE, TRUE)
people <- data.frame(name, age, child, stringsAsFactors = FALSE)
people
```

```
##   name age child
## 1 Anne  28 FALSE
## 2 Pete  30  TRUE
## 3 Frank 21  TRUE
## 4 Julia 39 FALSE
## 5 Cath  35  TRUE
```

```
people[3,2]
```

```
## [1] 21
```

```
people[3,"age"]
```

```
## [1] 21
```

```
people
```

```
##   name age child
## 1 Anne 28 FALSE
## 2 Pete 30 TRUE
## 3 Frank 21 TRUE
## 4 Julia 39 FALSE
## 5 Cath 35 TRUE
```

```
people[3,]
```

```
##   name age child
## 3 Frank 21 TRUE
```

```
people[, "age"]
```

```
## [1] 28 30 21 39 35
```

```
people[c(3, 5), c("age", "child")]
```

```
##   age child
## 3 21 TRUE
## 5 35 TRUE
```

```
people
```

```
##   name age child
## 1 Anne 28 FALSE
## 2 Pete 30 TRUE
## 3 Frank 21 TRUE
## 4 Julia 39 FALSE
## 5 Cath 35 TRUE
```

```
people[2]
```

```
##   age
## 1 28
## 2 30
## 3 21
## 4 39
## 5 35
```

```
people$age
```

```
## [1] 28 30 21 39 35
```

```
people[["age"]]
```

```
## [1] 28 30 21 39 35
```

`data.frame`은 데이터를 추가하는 여러 가지 방법을 지원합니다. 이전에 한번씩 본 방식이니 테스트해보세요.

```
height <- c(163, 177, 163, 162, 157)
people$height <- height
people[["height"]] <- height
people
```

```
##   name age child height
## 1 Anne  28 FALSE  163
## 2 Pete  30  TRUE  177
## 3 Frank 21  TRUE  163
## 4 Julia 39 FALSE  162
## 5 Cath  35  TRUE  157
```

```
weight <- c(74, 63, 68, 55, 56)
cbind(people, weight)
```

```
##   name age child height weight
## 1 Anne  28 FALSE  163    74
## 2 Pete  30  TRUE  177    63
## 3 Frank 21  TRUE  163    68
## 4 Julia 39 FALSE  162    55
## 5 Cath  35  TRUE  157    56
```

```
> tom <- data.frame("Tom", 37, FALSE, 183)
> rbind(people, tom)
## Error in match.names(clabs, names(xi)) :
##   names do not match previous names
```

```
tom <- data.frame(name = "Tom", age = 37,
                   child = FALSE, height = 183)
rbind(people, tom)
```

```
##   name age child height
## 1 Anne  28 FALSE  163
## 2 Pete  30 TRUE  177
## 3 Frank 21 TRUE  163
## 4 Julia 39 FALSE  162
## 5 Cath  35 TRUE  157
## 6 Tom   37 FALSE  183
```

순서에 대해 작업하기

순서는 `data.frame`뿐만 아니라 다른 자료형에서도 그대로 적용되는 내용입니다. 여기서는 `sort`와 `order`에 대해서 알아보겠습니다.

```
sort(people$age)
```

```
## [1] 21 28 30 35 39
```

```
ranks <- order(people$age)  
ranks
```

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

```
people$age
```

```
## [1] 28 30 21 39 35
```

```
ranks <- order(people$age)  
ranks
```

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

```
people[ranks, ]
```

```
##   name age child height  
## 3 Frank 21  TRUE  163  
## 1 Anne  28 FALSE  163  
## 2 Pete  30  TRUE  177  
## 5 Cath  35  TRUE  157  
## 4 Julia 39 FALSE  162
```

```
people[order(people$age, decreasing = TRUE), ]
```

```
##   name age child height  
## 4 Julia 39 FALSE  162  
## 5 Cath  35  TRUE  157  
## 2 Pete  30  TRUE  177  
## 1 Anne  28 FALSE  163  
## 3 Frank 21  TRUE  163
```

`sort`는 순서가 있는 데이터(지금의 예시로는 나이)를 오름차순으로 정렬해줍니다. 글자라면 알파벳순으로 정렬해 줄 것입니다. 한 번 실험해 보세요.

`order`는 그 위치에 있는 데이터가 전체 데이터 내에서 몇 번째에 위치하는지를 알려줍니다. `sort`가 정렬이 끝난 결과를 보여주는 것이라면 `order`는 그 데이터의 순서값 자체를 보여주는 것이죠. 그렇기 때문에 `order`는 []의 행부분 조건과 결합하여 많이 사용됩니다.

날짜

날짜는 어느 곳에서든 다루기 어려운 데이터입니다. 불규칙적인 윤달이라던지 하는 여러 가지 문제로 실제로 사용할 때 많은 문제가 있는데요, R에서는 여러 형태의 날짜를 표현하는 데이터를 Date라는 자료형으로 관리하고 있습니다. 사람들이 관행적으로 사용하는 형태의 character 날짜를 Date형으로 바꿔보겠습니다.

```
(dates <- c("2016/01/01", "2016/02/01", "2016/03/01", "2016/04/01", "2016/05/01", "2016/06/01"))
```

```
## [1] "2016/01/01" "2016/02/01" "2016/03/01" "2016/04/01" "2016/05/01"  
## [6] "2016/06/01"
```

```
class(dates)
```

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

```
(trans.dates <- as.Date(dates))
```

```
## [1] "2016-01-01" "2016-02-01" "2016-03-01" "2016-04-01" "2016-05-01"  
## [6] "2016-06-01"
```

```
class(trans.dates)
```

```
## [1] "Date"
```

Date형 일때의 장점은 계산이 가능하다는 것입니다.

```
trans.dates[3]-trans.dates[1]
```

```
## Time difference of 60 days
```

다른 모양의 character도 되는지 한번 보겠습니다.

```
dates2 <- c("2016-01-01", "2016-02-01", "2016-03-01", "2016-04-01", "2016-05-01", "2016-06-01")
trans.dates2 <- as.Date(dates2)
trans.dates2
```

```
## [1] "2016-01-01" "2016-02-01" "2016-03-01" "2016-04-01" "2016-05-01"
## [6] "2016-06-01"
```

```
class(trans.dates2)
```

```
## [1] "Date"
```

어려운 모양은 인식하지 못합니다. 그래서 일부러 양식을 알려주면 R이 고칠 수 있는데요.

```
> as.Date("2016년 4월 5일")
## Error in charToDate(x) : 문자열이 표준서식을 따르지 않습니다
```

```
as.Date("2016년 4월 5일", format="%Y년 %m월 %d일")
```

```
## [1] "2016-04-05"
```

양식은 저도 다 외우지 못하고 매번 검색해서 사용합니다.

<http://www.stat.berkeley.edu/classes/s133/dates.html>

하지만 year의 y(4자 Y, 2자 y), month의 m, day의 d로 생각하시면 우선 해결되고 나머지는 위에 링크를 참고해 보시면 좋을 것 같습니다.

```
as.Date(34519, origin="1900-01-01")
```

```
## [1] "1994-07-06"
```

시간은 POSIXct와 POSIXlt 두 가지로 준비되어 있습니다. 분석에 적용하는데 있어 특별히 구분해서 사용하지 않아서, 하나를 선택하셨다면 일관되게 한 가지만 계속 사용하시면 좋을 것 같습니다.

```
# 시간 자료형  
as.POSIXct("2017-04-12 12:00:00")
```

```
## [1] "2017-04-12 12:00:00 KST"
```

```
as.POSIXlt("2017-04-12 12:00:00")
```

```
## [1] "2017-04-12 12:00:00 KST"
```

날짜와 시간을 다루는 패키지로 유명한 lubridate가 있습니다. 아래 여러 코드의 실행결과를 보시면 그 유연한 기능에 감탄하시게 될 겁니다.

```
library(lubridate)

## 
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
## 
##     date
```

```
ymd("2017-05-05")
```

```
## [1] "2017-05-05"
```

```
ymd("170505")
```

```
## [1] "2017-05-05"
```

```
ymd("20170505")
```

```
## [1] "2017-05-05"
```

```
dmy("17-05-15")
```

```
## [1] "2015-05-17"
```

```
ymd("2017년 5월 5일")
```

```
## [1] "2017-05-05"
```

```
dmy("5일5월2017년")
```

```
## [1] "2017-05-05"
```

```
dates <- c("2017-05-05", "170505", "20170505", "17-05-15", "2017년 5월 5일")  
ymd(dates)
```

```
## [1] "2017-05-05" "2017-05-05" "2017-05-05" "2017-05-15" "2017-05-05"
```

```
(data <- ymd("2017-05-05"))
```

```
## [1] "2017-05-05"
```

```
year(data)
```

```
## [1] 2017
```

```
month(data)
```

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

```
month(data, label=T)
```

```
## [1] May
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

```
week(data)
```

```
## [1] 18
```

```
yday(data)
```

```
## [1] 125
```

```
mday(data)
```

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

```
wday(data)
```

```
## [1] 6
```

```
wday(data, label=T)
```

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

```
## Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```