

Vector와 Matrix

벡터 (Vector)

- ▶ 하나 이상의 값을 모아둔 집합
 - ▶ 실제 R에서는 하나의 숫자도 하나의 원소를 가진 벡터로 취급
 - ▶ R에서 가장 기초가 되는 자료 객체
- ▶ 벡터의 특징
 - ▶ 벡터는 `c()` 함수, `seq()` 함수, `rep()` 함수 등으로 작성한다
 - ▶ 벡터는 인덱스를 1부터 시작한다
 - ▶ 하나의 벡터에는 하나의 자료형(숫자, 문자, 논리)만 사용할 수 있다
 - ▶ 벡터에서 결측값은 `NA`를 사용한다

벡터

: examples by code

```
> # 1부터 10까지의 numeric 벡터 x
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> x[0] # x의 0번째 인덱스는?
numeric(0)
> x[1] # x의 첫번째 인덱스는? = 1
[1] 1
> length(x) # 벡터 x의 길이
[1] 10
> x[length(x)] # 벡터 x의 마지막 요소
[1] 10
```

```
> # 결측치를 포함한 벡터
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, NA)

> mean(x) # 결측치를 포함한 벡터의 평균 계산
[1] NA

> # 평균 계산시 결측치를 제거(정상)
> mean(x, na.rm = TRUE)
[1] 5.5
```

수열 벡터 생성 함수

: seq, rep

```
seq(from = {시작값}, to = {종료값}, by = {간격})  
rep({벡터}, times = {반복회수}|each = {개별원소 반복횟수})
```

```
> seq(from = 1, to = 10, by = 3) # 1부터 10까지 3씩 증가하는 수열
```

```
[1] 1 4 7 10
```

```
> seq(from = 1, to = 10) # 증가 값이 1일 때는 생략 가능(기본값)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> seq(1, 10) # 시작 값과 종료 값만 있을 때는 from, to 생략 가능
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> rep(1:3, 3) # 1 ~ 3 수열을 3번 반복
```

```
[1] 1 2 3 1 2 3 1 2 3
```

```
> rep(c(1, 3, 5), 2) # 1, 3, 5 원소를 가진 벡터를 2번 반복
```

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

```
> seq(from = 1, to = 10, length.out = 6)
```

```
[1] 1.0 2.8 4.6 6.4 8.2 10.0
```

seq 함수에 length.out 파라미터를 주면, 출력할 수열의 개수로 등분할 수도 있다
rep 함수에 each 파라미터를 주면, 반복할 벡터의 각 요소를 each 개만큼 반복한다

벡터

: Indexing

- ▶ `r`의 인덱스는 1부터 시작
- ▶ 마지막 요소의 값은 `length` 함수를 이용, 마지막 인덱스를 확인 후 참조하면 됨
- ▶ 인덱스의 범위를 벗어난 값의 참조는 `NA`(결측치)를 반환하므로 항상 인덱스의 범위가 유효한지를 확인 후 작업

6	1	3	6	10	5
---	---	---	---	----	---

`vec[5]`

```
> vec = c(6, 1, 3, 6, 10, 5)
> vec[1] # r의 인덱스를 1부터 시작
[1] 6
> vec[7] # 7번 인덱스의 내용
[1] NA
> vec[c(-3, -5)] # 3번, 5번 인덱스의 내용을 제거
[1] 6 1 6 5
> vec[length(vec)] # 가장 마지막 인덱스의 내용
[1] 5
```

벡터

: slicing

- ▶ 벡터의 일부 요소들을 추출하는 작업
- ▶ 추출법: 벡터[인덱스 범위]
 - ▶ `c()` => 개별 인덱스 받기
 - ▶ `{시작값}:{끝값}` => 인덱스의 범위 받기
 - ▶ 특정 컬럼, 혹은 행을 제외하고 추출할 때에는 음수값을 주면 됨

```
> # 1500 ~ 4000의 수열 벡터 (간격: 500)
> incomes <- seq(1500, 4000, by = 500)
> incomes
[1] 1500 2000 2500 3000 3500 4000
> incomes[c(1, 3, 5)] # 1, 3, 5번 인덱스 컬럼값 가져오기
[1] 1500 2500 3500
> incomes[2:4] # 2 ~ 4 범위 인덱스의 내용 가져오기
[1] 2000 2500 3000
> incomes[incomes > 2500] # incomes 내의 2500 초과값
slicing
[1] 3000 3500 4000
> incomes[c(-3, -4)] # 3, 4 컬럼을 제외한 나머지 컬럼
[1] 1500 2000 3500 4000
```

벡터[인덱스 범위]

벡터

: 요소에 이름 붙이기 - names() 함수

▶ 원소에 이름을 붙여준다

▶ ... 라고 설명은 하고 있지만, 실은 name의 목록을 반환하는 것

```
> scores <- c(90, 86, 88) # score 벡터를 생성
```

```
> scores
```

```
[1] 90 86 88
```

```
> names(scores)
```

```
NULL
```

```
> names(scores) <- c("Eng", "Math", "Science") # 각 요소에 이름을 정해줌
```

```
> scores
```

```
Eng    Math Science  
 90     86     88
```

```
names({ 벡터 }) = c("{name1}", "{name2}" ...)
```

벡터 관련 함수들

함수	용법	설명
cor()	cor(x, y)	상관계수
cumsum()	cumsum(x)	누적합
length()	length(x)	길이(요소의 수)
max()	max(x)	최대값
min()	min(x)	최소값
prod()	prod(x)	각 요소의 곱
range()	range(x)	범위
rev()	rev(x)	각 요소의 역순
sd()	sd(x)	표준편차
sum()	sum(x)	총합
var()	var(x)	분산
summary()	summary(x)	요약 통계량
unique()	unique(x)	중복 제거 유일값 벡터

벡터 관련 함수들

: 살짝 맛보기 by code

```
> x <- c(1, 3, 5, 7, 9, 11)
> y <- c(2, 3, 4, 5, 6, 7)
```

```
> cor(x, y) # 상관계수
[1] 1
> cumsum(x) # x의 누적합
[1] 1 4 9 16 25 36
> prod(x) # x 각 요소의 곱
[1] 10395
> range(x) # x의 범위
[1] 1 11
> rev(x) # x의 역순
[1] 11 9 7 5 3 1
> sd(x) # 표준편차
[1] 3.741657
> var(x) # x의 분산
[1] 14
> summary(x) # 요약 통계량
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1.0     3.5     6.0     6.0     8.5    11.0
> quantile(x) # 4분위수
      0%   25%   50%   75%  100%
      1.0   3.5   6.0   8.5  11.0
```

벡터의 연산

- ▶ 벡터는 사칙 연산, 논리 연산이 가능하다
- ~~▶ 연산하는 두 벡터의 길이가 다를 경우, 반복의 규칙에 따라 길이를 맞춘다(주의)~~
- ▶ R의 대부분의 함수는 벡터 연산을 지원한다
 - ▶ `sum` : 합계
 - ▶ `mean` : 평균
 - ▶ `max` : 최대값
 - ▶ `min` : 최소값
 - ▶ `median` : 중앙값
 - ~~▶ `mode` : 최빈값~~

```
> v1 <- c(1, 3, 5)
> v2 <- c(2, 3, 6)
```

```
> v1 + v2 # 벡터의 합
[1] 3 6 11
```

```
> v1 * v2 # 벡터의 곱
[1] 2 9 30
```

```
> v1 / v2 # 벡터의 나눗셈
[1] 0.5000000 1.0000000 0.8333333
```

```
> v1 / 2 # 벡터의 나눗셈 2
[1] 0.5 1.5 2.5
```

```
> v1 == v2 # 벡터의 비교 연산
[1] FALSE TRUE FALSE
```

벡터의 연산

: 응용

- ▶ 벡터의 인덱싱시 TRUE, FALSE 값으로 특정 값을 선택할 수 있게 된다

```
> c = seq(1, 10)
```

```
> c
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> c %% 2 == 0
```

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

```
> c[c %% 2 == 0] # 벡터 내 짝수 값을 인덱스로 선택
```

```
[1] 2 4 6 8 10
```

행렬 (Matrix)

- ▶ 요소가 2차원 직사각형 레이아웃으로 배열된 것
- ▶ R에서 행렬은 모양이 다른 벡터로 취급
 - ▶ 벡터의 명령어가 그대로 적용
 - ▶ 벡터의 특징을 그대로 지님
- ▶ 논리형, 캐릭터형 매트릭스도 가능하지만 주로 수학적 계산에 사용되는 숫자 요소를 포함하는 행렬을 사용함

$$A = \begin{bmatrix} -5 & 1 & -3 \\ 6 & 0 & 2 \\ 2 & 6 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 & 4 & 5 \\ -8 & 10 & 3 \\ -2 & -3 & -9 \end{bmatrix}$$

$$A + B = \begin{bmatrix} -3 & 5 & 2 \\ -2 & 10 & 5 \\ 0 & 3 & -8 \end{bmatrix}$$

$$A - B = \begin{bmatrix} -7 & -3 & -8 \\ 14 & -10 & -1 \\ 4 & 9 & 10 \end{bmatrix}$$

행렬

: 행렬의 생성

- ▶ 벡터를 원본 데이터로 이용,
`matrix()` 함수를 이용하여 생성

```
matrix({벡터},  
       ncol = {컬럼수},  
       nrow = {행수},  
       byrow = {행기준 여부})
```

```
> matrix(1:10, ncol = 2, byrow = TRUE)
```

```
      [,1] [,2]  
[1,]     1     2  
[2,]     3     4  
[3,]     5     6  
[4,]     7     8  
[5,]     9    10
```

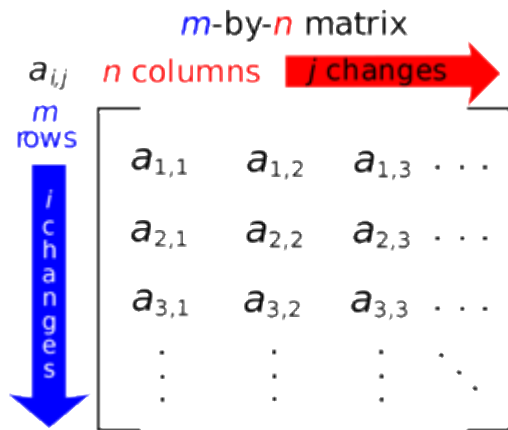
```
> matrix(1:10, nrow = 2)
```

```
      [,1] [,2] [,3] [,4] [,5]  
[1,]     1     3     5     7     9  
[2,]     2     4     6     8    10
```

행렬

: indexing

- ▶ Row 인덱스와 Col 인덱스를 이용,
요소를 인덱싱할 수 있음



```
> mat = matrix(1:10, ncol = 2, byrow = T)
> mat
      [,1] [,2]
[1,]     1     2
[2,]     3     4
[3,]     5     6
[4,]     7     8
[5,]     9    10
> mat[3, 2]
[1] 6
```

`matrix[{row_num}, {column_num}]`

행렬

: naming

- ▶ rownames : 행 이름
- ▶ colnames : 열 이름
- ▶ 사실상은, 벡터와 마찬가지로
열 이름, 행 이름을 돌려주는 함수

```
> mat
```

	[,1]	[,2]
[1,]	1	2
[2,]	3	4
[3,]	5	6
[4,]	7	8
[5,]	9	10

```
> colnames(mat) <- paste0("col", c(1, 2))
```

```
> rownames(mat) <- paste0("row", 1:5)
```

```
> mat
```

	col1	col2
row1	1	2
row2	3	4
row3	5	6
row4	7	8
row5	9	10

행렬

: slicing

- ▶ 행 인덱스 범위와 열 인덱스 범위를 이용 원하는 부분을 얻어올 수 있음
- ▶ 범위가 생략되면 전체를 의미
 - ▶ 예) `mat[, 1:2]` : 행 전체, 열 1~2

```
> mat
```

	[,1]	[,2]
[1,]	1	2
[2,]	3	4
[3,]	5	6
[4,]	7	8
[5,]	9	10

```
> mat[2:3, 2] # 2~3행, 2열
```

```
[1] 4 6
```

```
> mat[4:5, 1:2] # 4~5행, 1~2열
```

	[,1]	[,2]
[1,]	7	8
[2,]	9	10

```
> mat[, 2] # 전체 행, 2열
```

```
[1] 2 4 6 8 10
```

`matrix[{행 인덱스 범위}, {열 인덱스 범위}]`

행렬의 연산

- ▶ 벡터와 마찬가지로 사칙연산 (+, -, *, /)이 가능
- ▶ 선형대수에서 사용하는 행렬과 행렬 사이의 곱은 %%%을 이용

```
> x <- matrix(1:4, ncol = 2, nrow = 2, byrow = TRUE)
> y <- matrix(1:4, ncol = 2, nrow = 2, byrow = FALSE)
> x
      [,1] [,2]
[1,]     1     2
[2,]     3     4
> y
      [,1] [,2]
[1,]     1     3
[2,]     2     4
>
> x + y
      [,1] [,2]
[1,]     2     5
[2,]     5     8
> x * y
      [,1] [,2]
[1,]     1     6
[2,]     6    16
> x %%% y # 선형대수에서의 행렬간 곱
      [,1] [,2]
[1,]     5    11
[2,]    11    25
```

행렬의 연산

: 주요 함수들

함수	설명
nrow()	행렬의 행 수
ncol()	행렬의 열 수

함수	설명
cbind()	행렬 컬럼 결합
rbind()	행렬 행 결합
colMeans()	열의 평균값 벡터
rowMeans()	행의 평균값 벡터
colSums()	열의 합계 벡터
rowSums()	행의 합계 벡터
t()	행렬의 전치

```
> x
      [,1] [,2]
[1,]     1     2
[2,]     3     4
> colSums(x)
[1] 4 6
> colMeans(x)
[1] 2 3
```

```
> z <- matrix(1:10, ncol = 5)
> z
      [,1] [,2] [,3] [,4] [,5]
[1,]     1     3     5     7     9
[2,]     2     4     6     8    10
> t(z) # 행렬의 전치
      [,1] [,2]
[1,]     1     2
[2,]     3     4
[3,]     5     6
[4,]     7     8
[5,]     9    10
```

```
> cbind(x, z)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]     1     2     1     3     5     7     9
[2,]     3     4     2     4     6     8    10
> rbind(x, z)
```

Error in rbind(x, z) :
number of columns of matrices must match (see arg 2)

행렬의 연산

: apply

▶ matrix의 각 행, 혹은 각 열에 원하는 함수를 적용해 값을 반환 받고자 할 때 사용

- ▶ margin == 1 : 행 기준 적용
- ▶ margin == 2 : 열 기준 적용

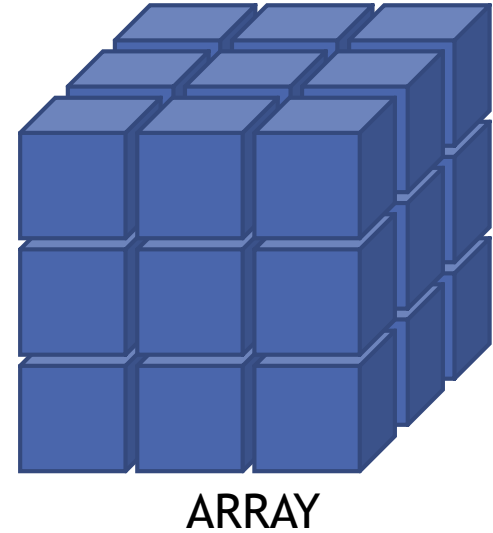
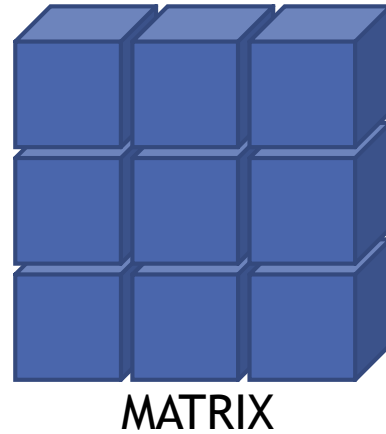
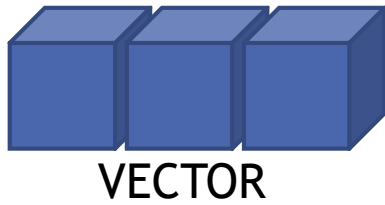
```
> mat = matrix(c(80, 90, 70, 65, 75, 90), ncol = 2)
> mat
      [,1] [,2]
[1,]   80   65
[2,]   90   75
[3,]   70   90
> apply(mat, 1, FUN = median) # 행 기준으로 median
[1] 72.5 82.5 80.0
> apply(mat, MARGIN = 2, median) # 열 기준 median
[1] 80 75
```

```
apply({데이터}, margin = {적용기준}, FUN = {함수})
```

Array in R

배열(Array)

- ▶ 2차원 이상의 데이터를 담을 수 있는 객체
- ▶ 벡터, 매트릭스와 마찬가지로 한 가지 형식의 데이터만 담을 수 있음
- ▶ 3차원 배열이라면 여러 개의 매트릭스가 겹쳐 있는 형태를 상상
- ▶ 4차원 이상은?



배열

: 배열의 생성

- ▶ 벡터를 원본 데이터로 하여
array 함수를 이용하여 생성

```
array({벡터},  
      dim = {차원 정의},  
      dimnames = {이름 목록})
```

```
> # 3행 3열을 가진 2개의 매트릭스 생성
```

```
> vec <- 1:9
```

```
> vec
```

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

```
> arr <- array(vec, dim = c(3, 3, 2))
```

```
> arr
```

```
, , 1
```

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

```
, , 2
```

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

배열

: naming

- ▶ `dimnames` 파라미터를 이용, 각 차원의 열과 행, 그리고 매트릭스의 이름을 부여할 수 있음
- ▶ `dimnames`는 배열을 생성하는 시점에서 부여할 수도 있지만, `dimnames` 함수를 이용, 생성 이후에 부여할 수도 있음

```
> dimnames(arr)
```

```
NULL
```

```
> arr.names.col <- c("COL1", "COL2", "COL3")  
> arr.names.row <- c("ROW1", "ROW2", "ROW3")  
> arr.names.matrix <- c("MATRIX1", "MATRIX2")  
> dimnames(arr) <- list(arr.names.row,  
arr.names.col, arr.names.matrix)
```

```
> arr
```

```
, , MATRIX1
```

	COL1	COL2	COL3
ROW1	1	4	7
ROW2	2	5	8
ROW3	3	6	9

```
, , MATRIX2
```

	COL1	COL2	COL3
ROW1	1	4	7
ROW2	2	5	8
ROW3	3	6	9

배열

: indexing

- ▶ 각 차원의 인덱스를 조합,
배열상의 특정 요소에 접근할 수 있다.
- ▶ 3차원 배열이라면
행 인덱스, 열 인덱스, 매트릭스 인덱스
순으로 접근한다고 생각

```
> # 배열의 2열 2행 2번 매트릭스 요소 값을 조회  
> arr[2, 2, 2]  
[1] 5  
> # 배열의 3열 1행 1번 매트릭스 요소 값을 조회  
> arr[3, 1, 1]  
[1] 3
```


배열

: slicing

- ▶ 각 차원의 인덱스 범위를 부여하면
배열 내 원하는 부분의 내용을 추출할 수 있음
- ▶ 범위가 생략되면 해당 부분 전체를 의미

> `arr[, , 2]` # 2번 매트릭스 전체

	COL1	COL2	COL3
ROW1	1	4	7
ROW2	2	5	8
ROW3	3	6	9

> `arr[2:3, 1:2, 1]` # 1번 매트릭스의 2~3행, 1~2열

	COL1	COL2
ROW2	2	5
ROW3	3	6

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

> # 1번 매트릭스의 1, 3번 컬럼, 2행 제외한 나머지

	COL1	COL3
ROW1	1	7
ROW3	3	9

> `arr[3, , 1]` # 1번 매트릭스의 3행 전체

COL1	COL2	COL3
3	6	9

배열

: apply

- ▶ 각 행, 각 열, 각 매트릭스에 원하는 함수를 적용
값을 반환 받을 수 있음

- ▶ margin 값을 이용, 적용할 데이터셋을 결정

- ▶ margin == 1 : 행 방향의 적용
- ▶ margin == 2 : 열 방향의 적용
- ▶ margin == 3 : 각 매트릭스를 계산

```
> arr.names.row <- c("ROW1", "ROW2", "ROW3")
> arr.names.col <- c("COL1", "COL2", "COL3")
> arr.names.matrix <- c("MATRIX1", "MATRIX2")
>
> arr <- array(1:18,
+             dim = c(3, 3, 2),
+             dimnames = list(arr.names.row,
+                             arr.names.col,
+                             arr.names.matrix))
> apply(arr, c(1), sum) # 행 방향으로 sum 함수 적용
ROW1 ROW2 ROW3
  51   57   63
> apply(arr, c(3), mean) # 개별 매트릭스에 평균 함수 적용
MATRIX1 MATRIX2
      5      14
```

`apply({데이터}, margin = {적용기준}, FUN = {함수})`