R컴퓨팅

5강 데이터 구조 II

정보통계학과 장영재 교수

1 행렬

2 배열

1 행렬

1 행렬

행렬은 동일한 형태로 구성된 2차원의 데이터 구조

행의 차원과 열의 차원을 갖고 있으며 벡터와 마찬가지로 하나의 행렬은 수치형, 문자형, 논리형 중 한 가지 형태의 원소만 갖는 점에 유의

※ 행렬의 주요 속성

속성	설명
length	자료의 개수
mode	자료의 형태
dim	행과 열의 개수
dimnames	행과 열의 이름

1 행렬

- 보기 5-1: matrix() 함수를 이용하여 1에서 9까지의 원소를 갖는 3행 3열의 행렬을 생성하고 각 행과 열의 이름을 row 1, row 2, row 3, col 1, col 2, col 3로 지정한 뒤 출력

```
> matr <- matrix(1:9, nrow=3) # 3행 3열 행렬 생성
> dimnames(matr) <- list(paste("row", c(1:3)), paste("col", c(1:3)))
> #행과 열 이름 지정
> matr
col 1 col 2 col 3
row 1 1 4 7
row 2 2 5 8
row 3 3 6 9
```

1 행렬

```
> length(matr)    #원소의 개수
[1] 9
> mode(matr)    #원소의 형태
[1] "numeric"
> dim(matr)    #행과 열의 개수
[1] 3 3
```

1 행렬

1 행렬의 생성

› 행렬은 여러 변수들이 이차원적으로 모여 있는 개체

› 행렬을 생성하는 간단한 몇 가지 기본함수

- matrix() 함수를 이용하여 직접 생성
- cbind() 함수를 이용하여 벡터를 병합
- rbind() 함수를 이용하여 벡터를 병합
- dim() 함수를 이용하여 차원(행과 열의 개수)을 직접 지정

1 행렬

1 행렬의 생성

matrix(data, nrow=, ncol=, byrow=FALSE, dimnames = NULL)

cbind(벡터1, 벡터2, …) 또는 rbind(벡터1, 벡터2, …)

dim(x) <– c(행의 개수, 열의 개수)

1 행렬

1 행렬의 생성

› 보기 5-2: 보기 3-11에서 matrix() 함수를 이용하여 생성된 행렬과 동일한 형태의 행렬을 rbind(), cbind(), dim() 함수를 이용하여 만들기

```
〉 r1 〈– c(1,4,7)     # r1, r2, r3 행 벡터 생성
〉 r2 〈– c(2,5,8)
〉 r3 〈– c(3,6,9)
〉 rbind(r1, r2, r3)     # rbind : 행을 기준으로 결합
   [,1] [,2] [,3]
   r1 1 4 7
   r2 2 5 8
   r3 3 6 9
〉 c1 〈– 1:3      #c1, c2, c3 열 벡터 생성
```

1 행렬

1 행렬의 생성

```
> c2 <- 4:6
> c3 <- 7:9
> cbind(c1, c2, c3)     #cbind : 열을 기준으로 결합
   c1 c2 c3
   [1,] 1 4 7
   [2,] 2 5 8
   [3,] 3 6 9
> m1 <- 1:9
> dim(m1) <- c(3,3)     #dim : 차원을 지정
> m1
   [,1] [,2] [,3]
   [1,] 1 4 7
   [2,] 2 5 8
   [3,] 3 6 9
```

1 행렬

2 행렬의 연산

- 벡터와 마찬가지로 행렬의 경우에도 자료의 삽입, 삭제, 행 또는 열의 연산 등과 관련된 다양한 방법들이 존재

- 몇 가지 주요 함수의 예를 들면 아래와 같음
  - [ ]를 이용하여 행렬의 일부 원소를 추출
  - apply() 함수를 이용한 행 또는 열의 연산
  - sweep() 함수를 이용한 행 또는 열의 연산

1 행렬

2 행렬의 연산

행렬명[원소번호 또는 조건문, …]

apply(행렬, 조건 (1은 행, 2는 열, c(1,2)는 행과 열), FUN, …)

sweep(행렬, 조건 (1은 행, 2는 열, c(1,2)는 행과 열), STATS, FUN = “–”, …)

1 행렬

2 행렬의 연산

› 보기 5-3: 1에서 9까지의 원소를 갖는 행 기준의 3행 3열 행렬을 만들고 특정 행과 열, 조건에 따른 원소 추출들을 실행하고 apply() 와 sweep() 함수를 통한 행 또는 열 연산을 실시

① 원소추출

```
> mat <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3, byrow=T)
> #행 기준 3열의 행렬
> mat
```

1 행렬

2 행렬의 연산

```
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
> mat[1,]       #행렬 mat의 1행의 값
[1] 1 2 3
> mat[,3]      #행렬 mat의 3열의 값
[1] 3 6 9
> mat[mat[,3] > 4, 2]      # 3열에서 4보다 큰 행의 값 중 2열의 모든 값
[1] 5 8
> mat[2,3]       # 2행 3열의 값 추출
[1] 6
```

1 행렬

2 행렬의 연산

② apply() 함수

```
〉 Height 〈– c(140,155,142,175)        # Height 벡터 생성
〉 size.1 〈– matrix(c(130,26,110,24,118,25,112,25), ncol=2, byrow=T,
  + dimnames=list(c("Lee", "Kim", "Park", "Choi"), c("Weight", "Waist")))
〉 # size.1 행렬 생성하고 이름을 부여
〉 size 〈– cbind(size.1, Height)  # size.1 행렬과 Height 벡터의 열 기준 결합
〉 size
Weight Waist Height
Lee 130 26 140
Kim 110 24 155
Park 118 25 142
Choi 112 25 175
```

1 행렬

2 행렬의 연산

```
〉 colmean 〈– apply(size, 2, mean)     # 2:열의 평균값을 계산
〉 colmean
Weight Waist Height
117.5 25.0 153.0
〉 rowmean 〈– apply(size, 1, mean)     # 1 : 행의 평균값을 계산
〉 rowmean
Lee Kim Park Choi
98.66667 96.33333 95.00000 104.00000
〉 colvar 〈– apply(size, 2, var)      # 2 : 열의 분산값을 계산
〉 colvar
Weight Waist Height
81.0000000 0.6666667 259.3333333
```

1 행렬

2 행렬의 연산

```
> rowvar <- apply(size, 1, var)      # 1 : 행의 분산값을 계산
> rowvar
Lee Kim Park Choi
3985.333 4430.333 3819.000 5673.000
```

1 행렬

2 행렬의 연산

③ sweep 함수 (기본연산은 “-”로 지정되어 있음)

```
〉 sweep(size, 2, colmean)     # size 각 열의 값과 colmean의 차
Weight Waist Height
Lee 12.5 1 –13
Kim –7.5 –1 2
Park 0.5 0 –11
Choi –5.5 0 22
〉 sweep(size, 1, rowmean)     # size 각 행의 값과 rowmean의 차
Weight Waist Height
Lee 31.33333 –72.66667 41.33333
Kim 13.66667 –72.33333 58.66667
Park 23.00000 –70.00000 47.00000
Choi 8.00000 –79.00000 71.00000
```

1 행렬

2 행렬의 연산

```
〉 sweep(size, 1, c(1,2,3,4), "+")     # size 각 행의 값에 c(1,2,3,4)값을 더해줌
Weight Waist Height
Lee 131 27 141
Kim 112 26 157
Park 121 28 145
Choi 116 29 179
〉 sweep(size, 1, c(1,2,3,4), "-")     # size 각 행의 값에서 c(1,2,3,4)값을 빼줌
Weight Waist Height
Lee 129 25 139
Kim 108 22 153
Park 115 22 139
Choi 108 21 171
```

1 행렬

2 행렬의 연산

› 보기 5-4: 1에서 4까지의 값을 갖는 열 기준 행렬 m1과 5에서 8까지의 값을 갖는 열 기준의 행렬 m2를 생성하고 두 행렬의 곱, m1의 전치행렬 및 m1의 역행렬을 구하기

```
〉 m1 〈– matrix(1:4, nrow=2)    # 1~4까지 2행 2열의 행렬 생성
〉 m1
[,1] [,2]
[1,] 1 3
[2,] 2 4
〉 m2 〈– matrix(5:8, nrow=2)    # 5~8까지 2행 2열의 행렬 생성
```

1 행렬

2 행렬의 연산

```
> m2
[,1] [,2]
[1,] 5 7
[2,] 6 8
> m1%*%m2              #m1과 m2 행렬의 곱셈
[,1] [,2]
[1,] 23 31
[2,] 34 46
> solve(m1)            #m1행렬의 역 행렬 생성
[,1] [,2]
[1,] –2 1.5
[2,] 1 –0.5
```

1 행렬

2 행렬의 연산

```
> t(m1)          #m1행렬의 전치행렬 생성
[,1] [,2]
[1,] 1 2
[2,] 3 4
```

2 배열

2 배열

배열(Array)은 행렬을 2차원 이상으로 확장시킨 객체를 의미

2차원 구조로 이루어진 행렬도 일종의 배열이라고 할 수 있으며
일반적으로는 3차원 이상의 데이터 객체를 배열이라고 함

※ 배열의 주요 속성

속성	설명
length	자료의 개수
mode	자료의 형태
dim	각 차원 벡터의 크기
dimnames	각 차원 리스트의 이름

2 배열

- 보기 5-5: 다음은 배열을 생성하는데 사용되는 array() 함수를 이용하여 1에서 18까지의 원소를 갖는 3행 3열의 행렬 2개를 생성하는 예제

```
> arr <- array(1:24, c(3,3,2))          #1~18까지의 자료 생성
> dimnames(arr) <- list(paste("row", c(1:3)),paste("col",
+ c(1:3)),paste("ar", c(1:2)))          # 각 차원의 이름을 지정
> arr
, , ar 1
col 1 col 2 col 3
row 1 1 4 7
row 2 2 5 8
row 3 3 6 9
```

2 배열

```
, , ar 2
col 1 col 2 col 3
row 1 10 13 16
row 2 11 14 17
row 3 12 15 18
〉length(arr)       # 자료의 개수 확인
[1] 18
〉mode(arr)        # 자료의 형태 확인
[1] "numeric"
〉dim(arr)       # 각 차원 벡터의 크기
[1] 3 3 2
```

2 배열

```
> dimnames(arr)      #각 차원 리스트의 이름
[[1]]
[1] "row 1" "row 2" "row 3"
[[2]]
[1] "col 1" "col 2" "col 3"
[[3]]
[1] "ar 1" "ar 2"
```

2 배열

1 배열의 생성

- 배열은 행렬의 확장으로 기본적인 성질이 행렬과 유사
- 배열을 생성하기 위한 대표적인 함수로는 아래와 같은 함수들이 있음

- array() 함수를 이용하여 직접 자료를 입력하는 방법
- dim() 함수를 이용하여 차원을 직접 지정

```
array(data, dim= c(행의 개수, 열의 개수, 행렬의 개수, …),
dim.names = NULL)
```

```
dim(x) <– c(행의 개수, 열의 개수, 행렬의 개수)
```

2 배열

1 배열의 생성

- 보기 5-6: ① array() 함수를 이용하여 1에서 6까지의 자료로 1차원 및 2차원 배열 을 생성해 보고 1에서 8까지 값으로 3차원 배열을 만들기 ② dim() 함수를 이용하여 1~24까지 자료로 3행 4열 행렬 2개를 생성

```
〉 array(1:6)        #1~6의 자료로 1차원 배열 생성
[1] 1 2 3 4 5 6
〉 array(1:6, c(2,3))          #1~6의 자료로 2차원 배열 생성
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

2 배열

1 배열의 생성

```
〉 array(1:8, c(2,2,2))        #1~8의 자료로 3차원 배열 생성
, , 1
[,1] [,2]
[1,] 1 3
[2,] 2 4
, , 2
[,1] [,2]
[1,] 5 7
[2,] 6 8
〉 arr 〈– c(1:24)     #1~24의 자료 생성
〉 dim(arr) 〈– c(3,4,2)      # dim() 함수를 이용하여 3행 4열의 행렬 2개 생성
```

2 배열

1 배열의 생성

```
> arr
, , 1
[,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12
, , 2
[,1] [,2] [,3] [,4]
[1,] 13 16 19 22
[2,] 14 17 20 23
[3,] 15 18 21 24
```

2 배열

2 배열의 연산

- 보기 5-7: array() 함수를 이용하여 생성된 배열 간에는 아래와 같이 각 원소간 덧셈과 곱셈, 두 배열 원소들의 곱의 합과 같은 연산 및 원소의 추출이 가능

```
> ary1 <- array(1:8, dim = c(2,2,2))
> ary2 <- array(8:1, dim = c(2,2,2))
> ary1
, , 1
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

2 배열

2 배열의 연산

```
, , 2
[,1] [,2]
[1,] 5 7
[2,] 6 8
〉 ary2
, , 1
[,1] [,2]
[1,] 8 6
[2,] 7 5
, , 2
[,1] [,2]
[1,] 4 2
[2,] 3 1
```

```
〉 ary1 + ary2    # 배열의 덧셈
, , 1
[,1] [,2]
[1,] 9 9
[2,] 9 9
, , 2
[,1] [,2]
[1,] 9 9
[2,] 9 9
〉 ary1 * ary2    # 배열의 곱셈
```

2 배열

2 배열의 연산

```
, , 1
[,1] [,2]
[1,] 8 18
[2,] 14 20
, , 2
[,1] [,2]
[1,] 20 14
[2,] 18 8
> #두 배열, 원소들의 곱의 합
> ary1 %*% ary2
[,1]
[1,] 120
```

```
> sum(ary1 * ary2)
[1] 120
> ary1[,,1]  # 배열 원소의 추출/삭제
[,1] [,2]
[1,] 1 3
[2,] 2 4
> ary1[1,1,]
[1] 1 5
> ary1[1,,-2]
[1] 1 3
```

R컴퓨팅