

# 전산통계학 실습

R 데이터구조

# 벡터

---

- 벡터
  - 동일한 스칼라의 값들을 저장하는 1차원의 배열
  - 다른 프로그래밍 언어에서 사용되는 배열과 동일
  - R 언어에서 가장 기본이 되는 자료 구조
    - 하나의 스칼라는 사실 원소 1개를 가진 벡터
- 원소/요소(element)
  - 벡터에 속한 각각의 스칼라 값들

# 벡터의 생성

```
> v <- c(1, 2, 3)
> v
[1] 1 2 3
> names(v) <- c("A", "B", "C")
> v
A B C
1 2 3
> v2 <- 4:6
> v2
[1] 4 5 6
> v2 <- seq(4, 10, 2)
> v2
[1] 4 6 8 10
> v3 <- rep(1:3, times=2, each=2)
> v3
[1] 1 1 2 2 3 3 1 1 2 2 3 3
```

- 벡터 생성 함수

- > c(...)

- Combine 함수에 저장하고 싶은 원소들을 나열하여 생성한다.

- 원소 이름 지정

- > names(vector)

- 벡터의 각 원소에 이름을 부여
    - 벡터를 이용한 함수 자체에 대해 저장하면 이름이 부여된다.

# 벡터의 생성

```
> v <- c(1, 2, 3)
> v
[1] 1 2 3
> names(v) <- c("A", "B", "C")
> v
  A B C
1 2 3
> v2 <- 4:6
> v2
[1] 4 5 6
> v2 <- seq(4, 10, 2)
> v2
[1] 4 6 8 10
> v3 <- rep(1:3, times=2, each=2)
> v3
[1] 1 1 2 2 3 3 1 1 2 2 3 3
```

- 연속된 숫자 벡터 생성
  - 일련의 숫자 벡터를 생성
    - `> seq(num1, num2, num3)`
      - 숫자 범위 num1 ~ num2 사이를 num3 등분 단위로 나누어 저장
    - `> num1:num2`
      - 숫자 범위 num1 ~ num2 사이의 숫자들을 저장
- 반복된 숫자 벡터 생성
  - 기존 벡터에서 전체 반복 및 각 원소에 대한 반복을 적용한 벡터를 생성하는 방법
    - `> rep(vector, times, each)`
      - 전체 반복 횟수: times
      - 각 원소의 반복 횟수: each

# 벡터의 생성

```
> v <- c(1, TRUE, "String", 4)
> v
[1] "1"      "TRUE"    "String"  "4"
> v2 <- c(NA, 60, 30, 20)
> v2
[1] NA 60 30 20
> v3 <- c(10, NULL, 5, 4)
> v3
[1] 10  5  4
> mean(v2)
[1] NA
> mean(v3)
[1] 6.333333
```

- 벡터 내부의 스칼라
  - 한 벡터에는 한 종류의 스칼라만 들어갈 수 있다.
    - Character > Numeric > Bool 의 우선 순위를 가지게 된다.
    - Character와 Numeric, Bool이 같이 있는 경우 모두 Character가 된다.
    - Numeric과 Bool이 같이 있는 경우 TRUE는 1, FALSE는 0이 된다.
- 특수 자료형 NA와 NULL
  - NA
    - 결측된 에러 값으로 판단된다. 하나의 자리를 차지하면서 다양한 함수의 계산에 영향을 준다.
  - NULL
    - 기존 프로그래밍 언어와 같이 존재 자체가 없는 특별한 객체로써, 자리를 차지하지 않는다.

# 벡터의 접근

```
> v <- 1:3
> names(v) <- c("a", "b", "c")
> v
a b c
1 2 3
> v[1]
a
1
> v["b"]
b
2
> v[2:3]
b c
2 3
> length(v)
[1] 3
```

- 벡터의 접근

- 벡터는 index를 이용하거나, 원소의 이름을 이용하여 벡터의 값에 직접 접근할 수 있다.
- 다른 프로그래밍 언어와 달리 R에서의 index는 1부터 시작

- 벡터의 접근 예시

- > v[1]
  - 벡터 v의 index 1의 값 접근
- > v["b"]
  - 벡터 v의 "b" 이름을 가진 값 접근
- > v[2:3]
  - 벡터 v의 index 2~3의 값 접근
- > length(v)
  - 벡터 v 내부 원소의 개수 (벡터 길이)

# 벡터의 변경

```
> v <- c(33, 42, 8, 99)
> v
[1] 33 42 8 99
> v[3] <- 52
> v
[1] 33 42 52 99
> v <- c(v[1], 62, v[3:4])
> v
[1] 33 62 52 99
> v <- c(100, v)
> v
[1] 100 33 62 52 99
> v <- v[1:4]
> v
[1] 100 33 62 52
```

- 벡터의 변경
  - 벡터의 원하는 index에 직접 접근하여 해당 값을 변경할 수 있다.
  - 혹은 새로운 벡터를 생성하면서 이전 벡터의 내용을 옮기면서 변경할 수 있다.
- 벡터의 원소 추가
  - 새로운 벡터를 생성하고 저장하여 기존 벡터의 제일 앞/뒤에 값을 추가할 수 있다.
- 벡터의 범위 추출 / 원소 제거
  - 기존 벡터의 범위 접근을 통해 원하는 범위만 추출할 수 있다.
  - 이를 통해 기존 벡터의 제일 앞/뒤의 값들을 제거할 수 있다.

# 벡터의 함수

- 벡터 관련 다양한 함수
  - 예시) 벡터 `x <- c(1, 2, 3, 4, 5)` 일 때,

함수	결과	의미
<code>sum(x)</code>	15	원소의 합
<code>cumsum(x)</code>	1 3 6 10 15	원소의 누적합
<code>length(x)</code>	5	벡터의 길이 (원소의 개수)
<code>max(x), min(x)</code>	5 1	벡터 내부 최대값, 최소값
<code>prod(x)</code>	120	원소의 곱
<code>range(x)</code>	1 5	원소가 가지는 범위
<code>rev(x)</code>	5 4 3 2 1	벡터의 역순 벡터
<code>unique(x)</code>	1 2 3 4 5	중복이 제거된 벡터
<code>summary(x)</code>	Min:1, Median:3, Mean:3, ...	요약된 통계량



# 벡터의 연산

```
> x <- 1:3
> y <- 4:6
> x + 3
[1] 4 5 6
> x + y
[1] 5 7 9
> x * y
[1] 4 10 18
> y <- 4:7
> x + y
[1] 5 7 9 8
경고메시지(들):
In x + y : 두 객체의 길이가 서로 배수관계에 있지 않습니다
> x * y
[1] 4 10 18 7
경고메시지(들):
In x * y : 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

- 벡터와 스칼라의 연산
  - 모든 원소 각각에 대하여 (element-wise) 수행된다.

# 벡터의 연산

```
> x <- 1:3
> y <- 4:6
> x + 3
[1] 4 5 6
> x + y
[1] 5 7 9
> x * y
[1] 4 10 18
> y <- 4:7
> x + y
[1] 5 7 9 8
경고메시지(들):
In x + y : 두 객체의 길이가 서로 배수관계에 있지 않습니다
> x * y
[1] 4 10 18 7
경고메시지(들):
In x * y : 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

## • 벡터와 벡터의 연산

- 서로 같은 index에 존재하는 원소 간 연산이 수행된다.
- 벡터의 길이가 배수 관계에 있을 경우
  - 더 작은 벡터가 반복하여 연산이 수행된다.
- 벡터의 길이가 배수 관계에 있지 않은 경우
  - 더 작은 벡터가 배수가 되도록 '가능한' 반복되어 연산이 수행된다 (재사용).

# 벡터의 집합 연산과 비교

```
> x <- 1:3
> y <- 1:3
> union(x,y)
[1] 1 2 3
> x <- 1:3
> y <- 3:5
> union(x,y)
[1] 1 2 3 4 5
> intersect(x,y)
[1] 3
> setdiff(x,y)
[1] 1 2
```

```
>
> x <- 1:3
> y <- 1:3
> identical(x, y)
[1] TRUE
> 3 %in% x
[1] TRUE
> 4 %in% x
[1] FALSE
> x == y
[1] TRUE TRUE TRUE
> y <- 1:4
> x == y
[1] TRUE TRUE TRUE FALSE
경고메시지 (들) :
In x == y : 두 객체의 길이가 서로 배수관계에 있지 않습니다
> y <- 3:5
> x == y
[1] FALSE FALSE FALSE
```

## • 벡터의 집합 연산

- > union(x, y)
  - 두 벡터의 합집합
- > intersect(x, y)
  - 두 벡터의 교집합
- > setdiff(x, y)
  - 두 벡터의 차집합

## • 벡터의 비교

- > identical(x, y)
  - 두 벡터의 일치 확인
- > element %in% vector
  - 벡터 내부 원소의 존재 여부 확인
- > vector1 == vector2
  - 벡터 내부 각 원소 간의 일치 확인

# 벡터의 비교

```
> x <- 1:10
> any(x > 8)
[1] TRUE
> all(x > 8)
[1] FALSE
> any(x > 10)
[1] FALSE
> all(x > 1)
[1] FALSE
> all(x >= 1)
[1] TRUE
```

- 벡터의 원소 조건 비교

- > all(...)
  - 주어진 조건에 대해 벡터의 각 원소가 모두 만족하는지에 따라 TRUE/FALSE를 반환한다.
- > any(...)
  - 주어진 조건에 대해 벡터의 각 원소 중 만족하는 것이 존재하는지의 여부에 따라 TRUE/FALSE를 반환한다.

# 벡터의 필터링

```
> x <- c(5, 3, 2, 8, -7, 6)
> y <- x[x > 2]
> y
[1] 5 3 8 6
> z <- x[x*x > 4]
> z
[1] 5 3 8 -7 6
> y2 <- x[c(TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)]
> y2
[1] 5 3 2 6
> x
[1] 5 3 2 8 -7 6
> x[x < 2] <- 0
> x
[1] 5 3 2 8 0 6
```

## • 벡터의 필터링

- 다양한 조건 및 연산 등도 함수이기 때문에, 벡터에서 특정 조건을 만족하는 원소들을 필터링할 수 있다.
- 기존 벡터에서 index 접근과 같은 문법을 통해 조건을 지정하여, 만족하는 원소들만 따로 추출할 수 있다.
- 각 원소에 대한 접근의 TRUE/FALSE를 통해 값을 추출할지 말지 결정할 수 있다.

# 벡터의 필터링

```
> x <- c(5, 3, 2, 8, -7, 6)
> y <- x[x > 2]
> y
[1] 5 3 8 6
> z <- x[x*x > 4]
> z
[1] 5 3 8 -7 6
> y2 <- x[c(TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)]
> y2
[1] 5 3 2 6
> x
[1] 5 3 2 8 -7 6
> x[x < 2] <- 0
> x
[1] 5 3 2 8 0 6
```

- 조건에 따른 벡터 변경
  - 조건에 부합하는 벡터 내부의 값을 변경할 수도 있다.
  - 조건을 적용하여 벡터를 추출하는 방법을 이용하여 실제 추출된 벡터에 대하여 새로운 값을 지정(변경)하고 다시 저장한다.

# 벡터의 필터링

```
> x <- c(7, NA, 12, 3:4)
> x
[1] 7 NA 12 3 4
> x[x > 4]
[1] 7 NA 12
> subset(x, x>4)
[1] 7 12
>
> which(x>4)
[1] 1 3
> x <- 1:10
> y <- ifelse(x > 5, 10, 1)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> y
[1] 1 1 1 1 1 10 10 10 10 10
```

## • 벡터의 필터링

- > subset(vector, condition)
  - 기존 벡터의 필터링에서는 NA 값을 걸러내지 못하고 항상 필터링 된다.
  - subset 함수는 NA 또한 제외하고 조건에 맞는 값들만 반환한다.

# 벡터의 필터링

```
> x <- c(7, NA, 12, 3:4)
> x
[1] 7 NA 12 3 4
> x[x > 4]
[1] 7 NA 12
> subset(x, x>4)
[1] 7 12
>
```

```
> which(x>4)
[1] 1 3
> x <- 1:10
> y <- ifelse(x > 5, 10, 1)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> y
[1] 1 1 1 1 1 10 10 10 10 10
```

## • 벡터의 필터링

- > which(condition)
  - 벡터 내부에서 조건에 맞는 값들의 위치를 반환한다.
- > ifelse(condition, true, false)
  - 조건이 TRUE인 경우와 FALSE인 경우에 따라 주어지는 벡터의 값들을 변경할 수 있다.



# 행렬

- 행렬

- 여러 행과 여러 열에 속하는 각 값들이 모인 자료구조
- 행과 열의 개수를 정한 ‘하나의 벡터’와 동일
  - 형태는 2차원 배열이지만 하나의 벡터이므로 하나의 자료형만 저장 가능
  - 행렬 전체가 하나의 벡터이며, 각 행과 열 또한 하나의 벡터 형태
- 여러 변수 간의 관계에 따른 수치, 결과 등을 표현할 때 유리

# 행렬의 생성

```
> v <- 1:9
> m <- matrix(v, nrow=3, byrow=TRUE)
> rownames(m) <- c("a", "b", "c")
> colnames(m) <- c("x", "y", "z")
> m
  x y z
a 1 2 3
b 4 5 6
c 7 8 9
> rownames(m)
[1] "a" "b" "c"
> colnames(m)
[1] "x" "y" "z"
```

## • 행렬의 생성

- `> matrix(...)`
  - `nrow` = 행의 개수
  - `ncol` = 열의 개수
  - `byrow` = TRUE/FALSE (TRUE = 행 방향, FALSE = 열 방향)
- 행렬을 만드는 벡터의 길이가 (`nrow * ncol`) 로 표현되어야 한다.
- `nrow`와 `ncol` 중 하나만 지정하는 경우 반대 값은 자동계산 된다.

# 행렬의 생성

```
> v <- 1:9
> m <- matrix(v, nrow=3, byrow=TRUE)
> rownames(m) <- c("a", "b", "c")
> colnames(m) <- c("x", "y", "z")
> m
  x y z
a 1 2 3
b 4 5 6
c 7 8 9
> rownames(m)
[1] "a" "b" "c"
> colnames(m)
[1] "x" "y" "z"
```

- 행과 열의 이름 지정

- 지정하는 벡터의 길이가 nrow, ncol과 동일해야 한다.
- > rownames(...)
  - 벡터를 저장하여 행의 이름을 지정, 또는 행의 이름 반환
- > colnames(...)
  - 벡터를 저장하여 열의 이름을 지정, 또는 열의 이름 반환

# 행렬의 접근

```
> x
  c d
a 1 3
b 2 4
> x[1,1]
[1] 1
> x[2,]
 c d
2 4
> x[,1]
a b
1 2
> x["a", "c"]
[1] 1
```

- 행과 열로의 접근
  - 행렬의 각 행과 열은 벡터
  - `> matrix[row,]`
    - 입력한 행의 벡터 반환
  - `> matrix[,col]`
    - 입력한 열의 벡터 반환
- 행렬 내부 값으로의 접근
  - `> matrix[index]`
    - 행렬은 벡터와 같기 때문에 하나의 벡터로 생각하여 벡터의 접근과 같이 index로 접근 가능
  - `> matrix[row, col]`
    - 입력한 행과 열에 해당되는 값 반환
  - 행과 열의 index 대신 지정된 이름을 통하여 접근할 수도 있다.

# 행렬의 변경

```
> x <- 1:4
> m <- matrix(x, nrow=2)
> m
      [,1] [,2]
[1,]    1    3
[2,]    2    4
> m <- cbind(m, c(5,6))
> m
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m <- rbind(m, c(7,8,9))
> m
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
[3,]    7    8    9
> m <- m[1:2,1:2]
> m
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

## • 행렬의 행/열 추가

- 행렬은 생성 시에 크기가 고정되어 변경할 수 없으나, 함수를 통하여 행이나 열을 추가할 수 있다.
- > rbind(matrix, newrow)
  - 행을 추가하는 함수, matrix를 앞에 쓰면 새로운 행이 1행으로 추가되고, 뒤에 쓰면 마지막 행으로 추가된다.
- > cbind(matrix, newcolumn)
  - 열을 추가하는 함수, matrix를 앞에 쓰면 새로운 열이 1열로 추가되고, 뒤에 쓰면 마지막 열로 추가된다.

## • 행렬의 행/열 제거

- 원하는 행 및 열을 제거하고 싶은 경우 해당 행이나 열의 index 범위를 통하여 새롭게 자르고 저장하면 된다.

# 행렬의 함수

함수	의미
$M + x$	행렬 $M$ 의 모든 각 원소에서 $x$ 를 더하기 (사칙연산 동일)
$M1 + M2$	행렬 $M1$ 과 행렬 $M2$ 의 합 (element-wise) (사칙연산 동일)
$M1 \%*\% M2$	행렬 $M1$ 과 행렬 $M2$ 의 곱 (행렬곱)
$t(M)$	행렬 $M$ 의 전치행렬
$solve(M1, M2)$	행렬 $M1 * \text{행렬 } X = \text{행렬 } M2$ 를 만족하는 행렬 $X$ 반환
$row(M)$	행렬 $M$ 의 각 원소의 row 값 행렬 반환
$col(M)$	행렬 $M$ 의 각 원소의 column 값 행렬 반환
$nrow(M)$	행렬 $M$ 의 행의 개수
$ncol(M)$	행렬 $M$ 의 열의 개수
$rowSums(M)$	행렬 $M$ 의 각 행의 합 반환
$colSums(M)$	행렬 $M$ 의 각 열의 합 반환

# 과제5

- 문제 1.

- MASS 패키지의 Cars93 데이터를 사용한다.
  - Cars93 내부의 Price 컬럼을 새로운 변수로 저장한다.

```
> x <- Cars93$Price
> x
[1] 15.9 33.9 29.1 37.7 30.0 15.7 20.8 23.7 26.3 34.7 40.1
[34] 15.9 14.0 19.9 20.2 20.9  8.4 12.5 19.8 12.1 17.5  8.0
[67] 21.5 13.5 16.3 19.5 20.7 14.4  9.0 11.1 17.7 18.5 24.4
```

- 새로운 벡터에 대해 원소의 개수, 평균, 총합을 구한다.
- 벡터의 원소들 중 15.0을 초과하는 원소들이 몇 개 있는지 구한다.
  - 조건을 이용한 벡터의 필터링과 원소의 개수 구하기를 통해 구할 수 있다.


# 과제5

- 문제 2.

- 대학교 한 반의 80명 학생들의 음주 및 흡연 여부를 조사하였다.
- 아래 주어진 그림과 같은 행렬(2개)을 생성하여 출력한다.
  - 단, “sum” 행과 열은 기존 2x2 행렬에서 각각 함수들을 이용하여 추가되도록 한다.

```
> m
```

	drink	non-drink
smoke	25	17
non-smoke	15	23



```
> m2
```

	drink	non-drink	sum
smoke	25	17	42
non-smoke	15	23	38
sum	40	40	80



# 과제 제출

---

- 제출 목록
  - 작성한 코드 파일(.R)
  - 결과를 출력한 터미널 캡처(이미지)를 Word 혹은 HWP에 붙여 넣어 PDF 파일로 변환
- 제출 방법
  - 목록의 파일들을 압축
  - 아래 서식으로 압축파일 이름 지정
  - 이메일 제목 또한 지정
- 제출 서식 (X=과제번호)
  - 파일 이름: 통계학실습\_과제X\_학번\_이름.zip
  - 이메일 제목: 통계학실습\_과제X\_학번\_이름
- 제출 이메일
  - [gtsk623@gmail.com](mailto:gtsk623@gmail.com)