전산통계학 실습

07. R 데이터구조

목차

- 벡터
- 행렬

벡터

- 벡터(vector)
  - 동일한 자료형을 가진 스칼라의 값들을 저장하는 1차원의 배열
  - 다른 프로그래밍 언어에서 사용되는 배열과 동일한 개념
  - R 프로그래밍 언어에서 가장 기본이 되는 자료구조
    - 하나의 스칼라는 사실 원소 1개를 가진 벡터

- 원소/요소(element)
  - 벡터에 저장된 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 & 원소 이름 지정
  - > c(...)
    - 벡터를 생성하는 Combine 함수
    - 벡터에 저장하고 싶은 원소들을 파라미터들로 나열하여 생성

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 & 원소 이름 지정
  - > names(vector)
    - 벡터의 원소들에 지정된 이름들을 반환하는 함수
    - 이 반환 함수 자체에 대해 다시 이름 벡터를 저장하여, 원소들의 이름을 지정 가능

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
  - > num1:num2
    - 범위 num1 ~ num2 사이의 일련의 숫자들을 저장

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
  - > seq(num1, num2, num3)
    - 규칙을 가진 일련의 숫자 벡터를 생성
    - num1 부터 num2 까지 num3 씩 증가/감소하는 숫자들을 저장

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
```

- 반복된 숫자 벡터 생성
  - > rep(vector, times, each)
    - 주어진 벡터를 전체 반복과 각 원소에 대한 반복을 적용한 벡터를 반환하는 함수
    - times = 전체 반복 횟수, each = 각 원소의 반복 횟수

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
  - 하나의 벡터에는 한 종류의 스칼라만 저장될 수 있음
  - 동시에 여러 스칼라를 저장하는 경우 상위 스칼라로 저장됨
    - 우선순위: Character > Numeric > Logical
    - Character 와 Numeric, Logical 이 모두 같이 있는 경우 모두 Character 화
    - Numeric 과 Logical 만 같이 있는 경우 TRUE = 1, FALSE = 0

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
```

- 특수 자료형 NA 와 NULL
  - NA
    - 구해지지 못한 결측 값으로 판단
    - 하나의 자리를 차지하기 때문에 다양한 함수의 계산에 영향을 줌
  - NULL
    - 기존 프로그래밍 언어에서 쓰임새와 같이, 존재가 없어 자리를 차지하지 않음

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
  - 벡터의 원소에 대한 접근은 index 혹은 지정된 이름을 통해 가능
  - 다른 프로그래밍 언어와 달리 R 에서의 index 는 1부터 시작

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 예시
  - > v[1]
    - 벡터 v 의 1번째 index 에 존재하는 원소에 접근

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 예시
  - > v[“b”]
    - 벡터 v 의 “b” 이름을 가진 원소에 접근

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 예시
  - > v[2:3]
    - 벡터 v 의 2~3번째 index 에 존재하는 원소들에 접근
    - 즉, 여러 숫자들을 벡터로 만들어 그 index 들에 존재하는 원소를 추출할 수 있음

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
  - 원하는 index 의 원소에 직접 접근하여 해당 값을 변경 가능
  - 새로운 벡터를 생성할 때, 이전 벡터의 내용을 옮기면서 변경 가능
    - 벡터의 추출을 이용한 방법

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 원소 추가
  - 새로운 벡터를 생성하면서, 기존 벡터의 제일 앞/뒤에 값 추가 가능
- 벡터의 범위 추출 / 원소 제거
  - 기존 벡터의 범위 접근을 통해 원하는 범위만 추출 가능
  - 이를 통해 기존 벡터의 제일 앞/뒤의 값들을 제거 가능

벡터의 함수

- 벡터에 사용 가능한 다양한 함수들
  - 벡터 **x <- c(1, 2, 3, 4, 5)** 일 때,

함수	결과	의미
sum(x)	15	원소의 합
cumsum(x)	1 3 6 10 15	원소의 누적합
length(x)	5	벡터의 길이 (원소의 개수)
max(x), min(x)	5 1	벡터 내부 최대값, 최소값
prod(x)	120	원소의 곱
range(x)	1 5	원소 값들이 가지는 범위
sev(x)	5 4 3 2 1	벡터의 역순 벡터
unique(x)	1 2 3 4 5	중복이 제거된 벡터
summary(x)	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) 연산 수행

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 index 에 존재하는 원소끼리 연산 수행
  - 두 벡터의 길이가 배수 관계에 있는 경우
    - 더 작은 벡터가 반복된 뒤 index 에 맞게 연산이 수행됨

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
  - 두 벡터의 길이가 배수 관계에 있지 않은 경우
    - 더 작은 벡터가 배수가 되도록 '가능한' 반복된 뒤 연산이 수행됨
    - 따라서 반복이 일부 잘려서 수행되기 때문에 경고 메시지가 발생됨

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
```

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

벡터의 집합 연산과 비교

```
> 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 비교
  - > identical(x, y)
    - 두 벡터의 일치 확인
  - > element %in% vector
    - 벡터 내부 원소의 존재 여부 확인

벡터의 집합 연산과 비교

```
> 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 비교
  - > vector1 == vector2
    - 벡터 내부 각 index 에 존재하는 원소 간의 일치를 확인하여 반환
    - 두 벡터의 길이 배수 관계 여부에 따라 조정되어 연산이 수행됨

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
  - 주어지는 조건에 대한 만족 여부를 TRUE/FALSE 로 반환하는 함수들
  - > all(...)
    - 벡터 내부의 모든 원소들이 만족하는 경우 TRUE 를 반환
    - 하나라도 성립하지 않는 경우 FALSE 를 반환
  - > any(...)
    - 벡터 내부의 어떤 원소라도 만족하는 경우 TRUE 를 반환
    - 모두 성립하지 않는 경우 FALSE 를 반환

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
  - R 에서는 다양한 조건 및 연산 등이 모두 반환되는 함수콜과 같음
  - 따라서 벡터에서 특정 조건을 만족하는 원소 필터링이 유연함
- 벡터의 필터링 예시
  - 벡터에 조건을 대입하여 만족되는 원소들만을 추출 가능
  - 각 원소 index 에 대해 TRUE/FALSE 를 통한 원소 추출 가능

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
```

- 필터링을 이용한 벡터 변경
  - 주어진 조건에 부합하는 벡터 내부 원소들의 값들을 변경할 수 있음

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 관련 함수들
  - > subset(vector, condition)
    - 기존 벡터의 필터링은 NA 를 걸러내지 못하고 같이 추출됨
    - subset 함수는 NA 또한 제외하고 조건에 맞는 값들만 반환

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 관련 함수들
  - > which(condition)
    - 벡터 내부에서 조건에 맞는 값들의 위치 index 를 반환
    - which 함수의 결과를 벡터에 다시 적용하면 subset 함수와 같은 결과가 수행됨

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 관련 함수들
  - > ifelse(condition, true, false)
    - 주어진 조건에 따라 TRUE/FALSE 에 해당되는 원소 값들을 일괄적으로 변경 가능

행렬

- 행렬
  - 여러 행과 열에 속한 각 스칼라 값들이 모인 자료구조
  - 행렬은 행과 열의 개수를 정한 '하나의 벡터'와 동일한 개념
    - 형태는 2차원 배열이지만 하나의 벡터이므로 하나의 자료형만 저장 가능
    - 행렬 전체가 하나의 벡터이며, 각 행과 열 또한 하나의 벡터 형태를 가짐
  - 여러 변수(열)에 대한 관측치(행)를 표현하는데 유리한 자료구조

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 함
  - nrow와 ncol 중 하나만 지정하는 경우 반대 값은 자동계산

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
  - 이름 지정 벡터의 길이는 각각 nrow, ncol과 동일해야 함
  - > rownames(...)
    - 입력된 행렬의 행에 지정된 이름 벡터 반환, 혹은 저장하여 새로 지정
  - > colnames(...)
    - 입력된 행렬의 열에 지정된 이름 벡터 반환, 혹은 저장하여 새로 지정

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
```

- 행렬의 각 행과 열에 대한 접근
  - 행렬의 각 행과 열은 벡터
  - > matrix[row,]
    - 입력되는 index 를 가지는 행의 벡터 반환
  - > matrix[,col]
    - 입력되는 index 를 가지는 열의 벡터 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
```

- 행렬의 내부 원소에 대한 접근
  - > matrix[index]
    - 행렬은 하나의 벡터이므로 벡터 접근처럼 index 로 접근 가능
  - > matrix[row, col]
    - 입력한 행과 열의 index 에 해당되는 원소 반환
    - 혹은 index 대신 행과 열에 지정된 이름을 통하여 접근도 가능

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 고정됨
  - 다만 함수를 통해 행/열 추가 가능
- 행렬의 행/열 추가 함수
  - > rbind(matrix, newrow)
    - 행을 추가하는 함수
    - 파라미터 위치에 따라 앞/뒤에 새로운 행 추가
  - > cbind(matrix, newcolumn)
    - 열을 추가하는 함수
    - 파라미터 위치에 따라 앞/뒤에 새로운 열 추가

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 제거
  - 원하는 행 및 열을 제거하고 싶은 경우 해당 행이나 열의 index 범위를 통하여 행렬을 일부 추출하고 다시 저장하여 제거

행렬의 함수

함수	의미
M + x	행렬 M의 모든 각 원소에서 x를 더하기 (사칙연산 동일)
M1 + M2	행렬 M1과 행렬 M2의 합 (element-wise) (사칙연산 동일)
M1 %*% M2	행렬 M1과 행렬 M2의 곱 (행렬곱)
t(M)	행렬 M의 전치행렬
solve(M1, M2)	행렬 M1 * 행렬 X = 행렬 M2를 만족하는 행렬 X 반환
row(M)	행렬 M의 각 원소의 row index 값을 가지는 행렬 반환
col(M)	행렬 M의 각 원소의 column index 값을 가지는 행렬 반환
nrow(M)	행렬 M의 행의 개수
ncol(M)	행렬 M의 열의 개수
rowSums(M)	행렬 M의 각 행의 합 반환
colSums(M)	행렬 M의 각 열의 합 반환

과제5

- 문제 1.
  - MASS 패키지의 Cars93 데이터를 사용한다.
    - 아래 예시와 같이 Cars93 내부의 Price 컬럼을 새로운 변수벡터로 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 15.0 을 '초과'하는 원소들이 몇 개 있는지 구한다.
    - 조건을 이용한 벡터의 필터링과 원소의 개수 구하기를 응용하여 구할 수 있다.

과제5

- 문제 2.
  - 대학교 한 반의 80명 학생들의 음주 및 흡연 여부를 조사하였다.
  - 아래 주어진 그림과 같은 행렬(2개)을 생성하여 출력해 본다.
    - 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)
  - 결과 출력 화면 (.PDF)
    - 터미널 캡처(이미지)를 Word 혹은 HWP 에 붙여넣어 PDF 로 변환

- 제출 방법
  - 위 목록의 파일들을 하나로 압축
  - 아래 서식으로 압축파일 이름 지정
  - 블랙보드를 통해 제출

- 제출 서식 (XX = 주차번호 ex. 01, 02, ...)
  - 파일 이름: 전산통계학_실습과제_XX주차_학번_이름.zip