

2장. R 데이터 구조

벡터, 요인, 날짜, 행렬 및 배열, 데이터 프레임, tibble, 리스트

R 데이터 객체

- 통계 데이터의 유형
 - 양적 데이터(숫자형 데이터)
 - 질적 데이터(범주형 데이터)
 1. 명목형 데이터
 2. 순서형 데이터
- 통계 data set: 데이터가 행과 열의 2차원 형태로 배열된 상태
 - 열: 변수. 하나의 열에는 같은 유형의 데이터만이 올 수 있음
 - 행: 동일한 대상에 대한 여러 변수의 관찰값

- R 데이터 유형

- 숫자형(numeric), 문자형(character), 논리형(logical) 등등

- 변환의 우선순위

문자형 > 숫자형 (실수형 > 정수형) > 논리형

- 유형변환함수 : as.character(), as.numeric(), as.logical() 등 as. 함수

- 다양한 구조의 데이터 객체

- 벡터: 1차원 구조. 구성요소는 모두 동일 유형의 데이터.
- 요인: 1차원 구조. 범주형 데이터를 표현하는 구조.
- 행렬: 2차원 구조. 구성요소는 모두 동일 유형의 데이터
- 배열: 2차원 이상의 구조. 동일 유형의 데이터로 구성.
- 데이터 프레임: 2차원 구조. 통계 데이터 세트에 가장 적합한 구조.
- tibble: 개선된 형태의 데이터 프레임으로서 tidyverse에서 공통적으로 사용.
- 리스트: 가장 포괄적 구조

3

2.1 벡터

- 1차원으로 배열된 구조

- 유형:

- 숫자형(numeric)
 - ▶ 정수형(integer), 실수형(double)
- 문자형(character)
- 논리형(logical)

4

2.1.1 벡터의 기본 특성

- 벡터의 생성: 함수 `c()`

```
> x <- c(TRUE, FALSE, TRUE)
> y1 <- c(1L, 3L, 5L) # L: 숫자를 정수로 지정, 없으면 실수로 지정
> y2 <- c(1.1, 3.5, 10.4)
> z <- c("one", "two", "three")
```

- 벡터의 구성요소: 모두 같은 유형의 데이터

<pre>> typeof(x) [1] "logical" > typeof(y1) [1] "integer"</pre>	<pre>> typeof(y2) [1] "double" > typeof(z) [1] "character"</pre>
---	--

5

- 벡터의 길이(`length`)

- 벡터를 구성하고 있는 요소 개수
- 확인: 함수 `length()`

```
> y1
[1] 1 3 5
> length(y1)
[1] 3
```

- 스칼라(길이가 1인 벡터)의 생성

- 함수 `c()`를 사용하지 않아도 됨

```
> a <- 1; a
[1] 1
```

; 명령문 종료표시
이후 새명령문 가능

```
> a <- 1
> a
[1] 1
```

6

- 다른 유형의 데이터가 뒤섞여 입력된 경우

- 문자형 데이터가 하나라도 포함되면 문자형 벡터가 됨: 문자형이 가장 복잡한 형태의 구조

```
> c(1, "1", TRUE)
[1] "1"    "1"    "TRUE"
```

- 숫자형과 논리형이 함께 있으면 숫자형 벡터가 됨: 논리형이 가장 단순한 형태

▶ 이 경우 TRUE는 1, FALSE는 0으로 변환

```
> c(3, TRUE, FALSE)
[1] 3 1 0
```

7

- 벡터의 구성 요소에 이름 붙이기

- 처음 입력할 때

```
> c(Seoul=9930, Busan=3497, Incheon=2944, Suwon=1194)
Seoul  Busan Incheon Suwon
9930   3497   2944   1194
```

- 이미 생성된 벡터

```
> pop <- c(9930, 3497, 2944, 1194)
> names(pop) <- c("Seoul", "Busan", "Incheon", "Suwon")

> pop
Seoul  Busan Incheon Suwon
9930   3497   2944   1194

> names(pop)
[1] "Seoul" "Busan" "Incheon" "Suwon"
```

8

- 함수 `scan()`에 의한 벡터 생성

- 외부 파일의 입력 및 자판에서 직접 데이터 객체 생성이 가능한 함수
- 직접 생성
 - `scan()`을 실행하면 프롬프트가 ' > ' 기호에서 ' 1: ' 기호로 바뀌는데
이어서 데이터를 직접 입력하거나 또는 복사하고 붙여넣기 가능

- 숫자형 벡터 입력

```
> x <- scan( )
1: 24
2: 35
3: 28 21
4:
Read 4 items

> x
[1] 24 35 28 21
```

'5:' 에서 Enter

9

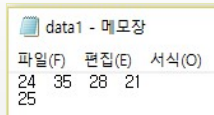
- 문자형 벡터 입력

```
> y <- scan(what="character")
1: Seoul Suwon
3: 'New York'
4:
Read 3 items
> y
[1] "Seoul"      "Suwon"      "New York"
```

- 자료에 인용부호 사용 불필요
- 한 자료가 빈 칸으로 구분된 경우에는 인용부호 사용

10

- 외부 파일 입력



C:\data\data1.txt

```
> z <- scan("C:/Data/data1.txt")
Read 5 items
> z
[1] 24 35 28 21 25
```

11

2.1.2 다양한 형태를 갖는 벡터의 생성

- 벡터에 데이터 추가 및 벡터의 결합
- 일정한 구조를 갖는 벡터의 생성

12

1) 벡터에 데이터 추가 및 벡터들의 결합:

- 함수 `c()`

```
> x <- c(11,12,13,14)
> c(x, 15)           # 벡터에 데이터 추가
[1] 11 12 13 14 15
> y <- c(16,17,18)
> c(x, y)           # 벡터들 결합
[1] 11 12 13 14 16 17 18
```

- 함수 `append()`: 추가되는 스칼라 혹은 벡터의 위치 조절 가능

```
> append(x, 15)
[1] 11 12 13 14 15
> append(x, 15, after=2)
[1] 11 12 15 13 14
> append(x, y)
[1] 11 12 13 14 16 17 18
> append(x, y, after=3)
[1] 11 12 13 16 17 18 14
```

13

2) 일정한 구조를 갖는 벡터의 생성

- 콜론(`:`) 연산자

a:b

- a를 시작점으로 b를 초과하지 않을 때까지 1씩 증가하는 수열
- a > b이면 1씩 감소하는 수열

```
> 1:5
[1] 1 2 3 4 5
> -3:3
[1] -3 -2 -1 0 1 2 3
> 1.5:5.4
[1] 1.5 2.5 3.5 4.5
> 5:0
[1] 5 4 3 2 1 0
```

14

- 함수 seq()에 의한 수열 생성

```
> seq(from=0,to=5)           # seq(0,5) 또는 0:5
[1] 0 1 2 3 4 5
> seq(from=0,to=5,by=2)      # seq(0,5,by=2)
[1] 0 2 4
> seq(from=0,to=5,length=3)  # seq(0,5,len=3)
[1] 0.0 2.5 5.0
> seq(from=0,by=2,length=3)  # seq(0,by=2,len=3)
[1] 0 2 4
```

- 한 숫자만 입력된 경우: 1을 시작점, 1씩 증가(감소), 지정된 숫자를 끝점

```
> seq(3)
[1] 1 2 3
> seq(-3)
[1] 1 0 -1 -2 -3
```

15

- 예제: 다음의 수열 생성

① 2, 5, 8, 11

```
> seq(from=2, to=11, by=3)
```

② 9, 18, 27, 36, 45

```
> seq(from=9, to=45, by=9)
> seq(from=9, to=45,length=5)
```

③ 1, 3, 5, 7, 2, 4, 6, 8

```
> c(seq(from=1,to=7,by=2), seq(from=2,to=8,by=2))
```

16

- 함수 `rep()`에 의한 반복된 패턴이 있는 데이터 생성

- 옵션 `times`의 활용

```
> rep(1, times=3)           # rep(1,3)과 동일
[1] 1 1 1
> rep(1:3, times=2)         # c(1,2,3) 전체를 2번 반복
[1] 1 2 3 1 2 3
> rep(c("M","F"), times=c(2,3)) # "M"을 2번, "F"를 3번 반복
[1] "M" "M" "F" "F" "F"
```

- `times`에 하나의 숫자 지정: 데이터 전체를 숫자만큼 반복
- `times`에 벡터 지정: 반복 대상 데이터와 일대일 대응

17

- 옵션 `each`의 활용

```
> rep(1:3, each=2)          데이터 각 요소가 each번 반복
[1] 1 1 2 2 3 3
```

- 옵션 `each`와 `times`의 활용

```
> rep(1:3, each=2, times=2)
[1] 1 1 2 2 3 3 1 1 2 2 3 3
```

데이터 각 요소가 each번 반복, 전체를 times번 반복

18

- 옵션 length의 활용

```
> rep(1:3,length=6)
[1] 1 2 3 1 2 3
```

길이가 length가 될 때까지 데이터 전체가 반복

- 옵션 each와 length의 활용

```
> rep(1:3,each=2,length=8)
[1] 1 1 2 2 3 3 1 1
```

각 요소가 each번 반복되는 과정을 길이가 length가 될 때까지 반복

19

2.1.3 문자열을 위한 함수

함수	기능
nchar(x)	문자열 x를 구성하는 문자의 개수
paste(. . . , sep=" ")	문자열들의 결합
substr(x, start, stop)	문자열의 일부분 선택
toupper(x)	영문자 대문자로 변환
tolower(x)	영문자 소문자로 변환
strsplit(x, split)	문자열의 분리
sub(old, new, x)	문자열의 치환
gsub(old, new, x)	문자열의 치환

20

- 함수 nchar(): 문자열을 구성하고 있는 문자 개수

```
> x <- c("Park", "Lee", "Kwon")
> nchar(x)
[1] 4 3 4

> nchar("응용통계학과")
[1] 6
```

21

- 함수 paste(): 문자열의 결합
 - 옵션 sep의 활용

```
> paste("모든", "사람에게는", "통계적", "사고능력이", "필요하다") # 디폴트 sep= " "
[1] "모든 사람은 통계적 사고능력이 필요하다"

> paste("모든", "사람에게는", "통계적", "사고능력이", "필요하다", sep="-")
[1] "모든-사람에게는-통계적-사고능력이-필요하다"

> paste("모든", "사람에게는", "통계적", "사고능력이", "필요하다", sep="")
[1] "모든사람에게는통계적사고능력이필요하다"
```

22

- 입력된 숫자는 문자로 전환되어 문자열과 결합

```
> paste("원주율은", pi, "이다")
[1] "원주율은 3.14159265358979 이다"
```

- 문자형 벡터가 입력되면 대응되는 요소끼리 결합
- 벡터의 길이가 서로 다르면 순환법칙 적용

```
> paste("Stat", 1:3, sep="") # sep="" : Stat 과 숫자 사이 간격 없이 결합
[1] "Stat1" "Stat2" "Stat3"

> paste(c("Stat", "Math"), 1:3, sep="-")
[1] "Stat-1" "Math-2" "Stat-3"
```

23

- 빈칸 없이 문자열 결합:
 - ① 함수 paste()에 옵션 sep="" 사용
 - ② 함수 paste0() 사용

```
> paste0("stat", 1:3)
[1] "stat1" "stat2" "stat3"
```

Paste0: 숫자, not alphabet o

- 문자형 벡터의 문자열을 하나로 결합: 옵션 collapse

예: 벡터 letters와 LETTERS: 각각 26개 영문자의 소문자와 대문자로 구성
26개 영문자를 하나로 결합

```
> paste0(letters, collapse="")
[1] "abcdefghijklmnopqrstuvwxyz"

> paste0(LETTERS, collapse=",")
[1] "A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z"
```

하나의 문자형 벡터 대상: 옵션 sep와는 관련 없음

- sep : 여러 개 문자형 벡터를 결합에 작용
- collapse : 단일 문자형 벡터의 결합에 작용

24

- 함수 `substr()`: 주어진 문자열의 일부분 선택

`substr(x, start, stop)`
 - `start`, `stop`: 정수형 스칼라 또는 벡터(대응되는 숫자끼리 시작점과 끝점 구성)

```
> substr("Statistics", 1, 4)
[1] "Stat"
```

```
> x <- c("응용통계학과", "정보통계학과", "학생회장")
```

```
> substr(x, 3, 6)
[1] "통계학과" "통계학과" "회장"
```

```
> substr(x, c(1,3), c(2,6)) # c(1,3): 시작점 벡터, c(2,6): 끝점 벡터
[1] "응용"      "통계학과"  "학생"
```

시작점과 끝점이 벡터인 경우 필요하다면 순환법칙 적용

25

- 예제: 문자형 벡터 `x`에는 미국의 세 도시와 그 도시가 속한 주 이름이 입력

```
> x <- c("New York, NY", "Ann Arbor, MI", "Chicago, IL")
```

- 세 도시가 속한 주 이름만을 선택하여 출력

```
> substr(x, nchar(x)-1, nchar(x))
[1] "NY" "MI" "IL"
```

26

- 함수 `strsplit()`: 문자열의 분리

옵션 `split`에 지정된 기준으로 분리. 결과는 리스트

- 예: 세 도시의 이름과 주 이름 분리

```
> x <- c("New York, NY", "Ann Arbor, MI", "Chicago, IL")
```

```
> (y <- strsplit(x, split=",")) # > (지정문): 지정문결과 인쇄
```

```
[[1]]  
[1] "New York" " NY"
```

```
[[2]]  
[1] "Ann Arbor" " MI"
```

```
[[3]]  
[1] "Chicago" " IL"
```

```
> unlist(y)  
[1] "New York" " NY" "Ann Arbor" " MI" "Chicago"  
[6] " IL"
```

함수 `unlist()`: 리스트를 벡터로 변환

27

- 문자열을 구성하는 모든 문자의 분리

```
> unlist(strsplit("PARK", split=""))  
[1] "P" "A" "R" "K"
```

- 점(.)을 기준으로 문자열 분리하는 경우

- 옵션 `split="."` : 원하는 결과를 얻을 수 없음

```
> unlist(strsplit("a.b.c", split="."))  
[1] "" "" "" "" ""
```

- 옵션 `split="[.]"` 또는 `split="\\"`

```
> unlist(strsplit("a.b.c", split="[.]"))  
[1] "a" "b" "c"
```

- 옵션 `split`에는 정규표현식이 사용됨
- 정규표현식에서 점(.)은 다른 의미가 있음

28

- 함수 toupper(), tolower(): 대(소)문자로 수정

```
> x <- c("park", "lee", "kwon")

> (y <- toupper(x)) # > (지정문) :지정문 결과 인쇄
[1] "PARK" "LEE" "KWON"

> tolower(y)
[1] "park" "lee" "kwon"
```

- 벡터 x의 첫 글자만 대문자로 변환

```
> substr(x,1,1) <- toupper(substr(x,1,1))
> x
[1] "Park" "Lee" "Kwon"
```

29

- 함수 sub(), gsub(): 문자열의 치환

- sub(old, new, 문자열): 문자열의 첫 번째 old만 new로 치환
- gsub(old, new, 문자열): 문자열의 모든 old가 new로 치환

```
> x <- "Park hates stats. He hates math, too."

> sub("hat", "lov", x)
[1] "Park loves stats. He hates math, too."

> gsub("hat", "lov", x)
[1] "Park loves stats. He loves math, too."
```

30

- 예:
 - 문자열 "banana1", "banana2", "banana3" 생성
 - 첫 번째 a를 A로 변경
 - 모든 a를 A로 변경

```
> (y <- paste0("banana", 1:3))
[1] "banana1" "banana2" "banana3"

> sub("a", "A", y)
[1] "bAnana1" "bAnana2" "bAnana3"

> gsub("a", "A", y)
[1] "bAnAnA1" "bAnAnA2" "bAnAnA3"
```

- 문자열의 일부 삭제는 new에 "" 입력

```
> z <- "Everybody cannot do it"

> sub("not", "", z)
[1] "Everybody can do it"
```

31

2.1.4 벡터의 연산

- 벡터와 벡터의 연산은 대응되는 각 구성요소끼리의 연산으로 이루어짐

```
> x <- c(7,8,9,10)
> y <- c(1,2,3,4)

> x+y
[1] 8 10 12 14

> x-y
[1] 6 6 6 6

> x*y
[1] 7 16 27 40

> x/y
[1] 7.0 4.0 3.0 2.5

> x^y
[1] 7 64 729 10000
```

32

- 벡터와 스칼라의 연산도 동일한 개념으로 실행됨

```
> x
[1] 7 8 9 10
> x+3
[1] 10 11 12 13
> x/4
[1] 1.75 2.00 2.25 2.50
> 2^x
[1] 128 256 512 1024
```

- 벡터 단위의 연산은 R의 큰 장점 중 하나
- 벡터 단위의 연산이 가능하지 않는 SAS와 같은 소프트웨어에서 R의 벡터 연산과 동일한 작업을 수행하기 위해서는 루프(loop)에 의한 반복작업이 필요함

33

- 벡터 연산에서 나올 수 있는 특수 문자:

Inf, -Inf : 양 무한대, 음 무한대
NaN : 계산 불능 (Not a Number)

참고) NA : 결측치, Not Available
NULL : 값이 비어있는 상태.
데이터 유형도 없고(NULL), 길이도 없다(0).

```
> c(-1,0,1)/0
[1] -Inf NaN Inf

> sqrt(-1)
[1] NaN
경고메시지(들):
In sqrt(-1) : NaN이 생성되었습니다

> Inf-Inf
[1] NaN
> Inf/Inf
[1] NaN
```

34

- 벡터 연산의 순환법칙

- 벡터와 벡터의 연산은 대응되는 요소끼리의 연산
- 만일 두 벡터의 길이가 달라 일대일 대응이 되지 않는다면 어떤 일이 벌어지겠는가?

```
> c(1,2,3,4,5,6) + c(1,2,3)
[1] 2 4 6 5 7 9
```

- 길이가 짧은 c(1,2,3)을 순환 반복시켜 c(1,2,3,1,2,3)을 만들어 길이를 같게 만든 후 연산 수행
- 벡터와 스칼라의 연산도 동일하게 수행됨.
- 다양한 함수에서도 순환법칙이 적용됨.

35

- 긴 벡터의 길이가 짧은 벡터 길이의 배수가 되지 않는 경우: 반복으로 두 벡터의 길이를 동일하게 만들 수 없음

```
> 1:4 + 1:3
[1] 2 4 6 5
경고메시지(들):
In 1:4 + 1:3 : 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

- 의도적으로 순환법칙을 사용하지 않은 경우에 위와 같은 경고문구를 봤다면 반드시 연산과정을 확인해야 함
- 대부분 잘못된 연산이 수행되었을 것임

36

- 수학 계산 관련 함수

```
> abs(-2)           # 절대값 계산
[1] 2
> sqrt(25)          # 제곱근 계산
[1] 5
> ceiling(3.475)     # 3.475보다 작지 않은 가장 작은 정수
[1] 4
> floor(3.475)       # 3.475보다 크지 않은 가장 큰 정수
[1] 3
> trunc(5.99)        # 소수점 이하 버림
[1] 5
> round(3.475,2)     # 소수 2자리로 반올림
[1] 3.48
> signif(0.00347, 2) # 유효수 2자리로 반올림
[1] 0.0035
```

37

```
> sin(1)             # 삼각 함수
[1] 0.841471
> asin(sin(1))       # 역삼각함수
[1] 1
> log(2,base=2)      # 밑이 2인 로그
[1] 1
> log(10)            # 자연로그
[1] 2.302585
> log10(10)          # 상용로그
[1] 1
> exp(log(10))       # 지수함수. 자연로그의 역함수
[1] 10
```

38

- 기초 통계 관련 함수

<pre>> x <- c(1,2,3,4,50) > mean(x) [1] 12 > median(x) [1] 3 > range(x) [1] 1 50 > IQR(x) [1] 2 > sd(x) [1] 21.27205 > var(x) [1] 452.5</pre>	<pre>> sum(x) [1] 60 > min(x) [1] 1 > max(x) [1] 50 > diff(c(1,2,4,7,11)) [1] 1 2 3 4</pre> <p>- (뒷 값 - 앞 값), not 차이</p>
---	--

39

- 결측값

- 결측값 기호: NA (not available)
- 데이터에 결측값 포함여부 확인: 함수 is.na()

```
> x <- c(1,0,3,5,NA)
> is.na(x)
[1] FALSE FALSE FALSE FALSE TRUE
> sum(is.na(x))
[1] 1
```

- NA는 자신을 포함한 어떤 대상과도 비교되지 않음

```
> x==NA
[1] NA NA NA NA NA
```

== 비교 연산자

40

- NA가 포함된 데이터의 연산결과

```
> x <- c(1,0,3,5,NA)
> mean(x); max(x)
[1] NA
[1] NA
```

연산에서 NA를 제거하는 방법: 옵션 na.rm=TRUE

```
> mean(x, na.rm=TRUE); max(x, na.rm=TRUE)
[1] 2.25
[1] 5
```

41

2.1.5 벡터의 비교

- 벡터의 인덱싱 혹은 벡터의 변환 등에 필수적인 요소
- 비교연산자와 논리연산자

연산자	기능
<	작다
<=	작거나 같다
>	크다
>=	크거나 같다
==	같다
!=	같지 않다
! x	x가 아니다 (NOT)
x y	x 또는 y (OR)
x & y	x 그리고 y (AND)

42

- 벡터 단위로 각 대응되는 요소끼리의 비교가 이루어짐

```
> x <- c(3,8,2)
> y <- c(5,4,2)

> x > y
[1] FALSE TRUE FALSE

> x >= y
[1] FALSE TRUE TRUE

> x < y
[1] TRUE FALSE FALSE

> x <= y
[1] TRUE FALSE TRUE

> x == y
[1] FALSE FALSE TRUE

> x != y
[1] TRUE TRUE FALSE
```

43

- 벡터와 스칼라의 비교는 순환법칙이 적용된 것임

```
> x <- 1:3

> x > 2
[1] FALSE FALSE TRUE

> x < 2
[1] TRUE FALSE FALSE

> x <= 2 | x >= 3
[1] TRUE TRUE TRUE

> x <= 2 & x >= 1
[1] TRUE TRUE FALSE
```

44

- 각 요소끼리의 비교결과보다는 벡터 전체의 비교결과를 원하는 경우

함수 `any()`, `all()`

```
> x <- 1:5
> any(x>=4)
[1] TRUE
> all(x>=4)
[1] FALSE
```

45

- 벡터의 구성요소 중 특정 조건을 만족하는 요소의 개수 혹은 비율

```
> x <- 1:5
> x >= 4
[1] FALSE FALSE FALSE TRUE TRUE
> sum(x>=4)
[1] 2
> mean(x>=4)
[1] 0.4
```

- 논리형 벡터를 숫자형 함수에 적용하면 숫자형 벡터로 전환됨

46

- 연산자 %in% : 벡터의 구성요소 중 특정한 값 포함 여부 확인

- 벡터 x에 2 또는 4가 포함되어 있는지 확인

```
> x <- 1:5
> x %in% c(2,4)
[1] FALSE TRUE FALSE TRUE FALSE
```

- 벡터 x의 구성 요소 하나하나와 %in% 오른쪽에 주어진 값 비교
- x == c(2,4)는 어떤 작업을 수행하는가?

순환반복 작용, x == c(2,4,2,4,2) 와 같은 기능

```
> x <- 1:5
> x == c(2,4)
[1] FALSE FALSE FALSE TRUE FALSE
경고메시지(들):
In x == c(2, 4) : 두 객체의 길이가 서로 배수관계에 있지 않습니다
[1] FALSE TRUE FALSE TRUE FALSE
```

47

2.1.6 벡터의 인덱싱

- 벡터의 인덱싱(Indexing)
 - 벡터의 일부분을 선택하는 작업.
 - x[a]의 형태: 벡터 a는 정수형, 논리형, 문자형(구성요소에 이름이 있는 경우)
- 정수형 벡터에 의한 인덱싱
 - 모두 양수: 지정된 위치의 자료 선택
 - 모두 음수: 지정된 위치의 자료 제외

```
> y <- c(2,4,6,8,10)
> y[c(1,3,5)]
[1] 2 6 10
> y[c(-2,-4)]
[1] 2 6 10
> y[c(2,2,2)] # 같은 위치 반복 지정 가능
[1] 4 4 4
> y[6] # 지정한 위치가 벡터 길이보다 큰 경우
[1] NA
```

48

- 문자형 벡터에 의한 인덱싱
 - 벡터의 구성요소에 이름이 있는 경우에만 적용 가능

```
> pop=c(Seoul=9930,Busan=3497,Inchon=2944,Suwon=1194)
> pop
Seoul  Busan  Inchon  Suwon
9930   3497   2944   1194

> pop[c("Seoul", "Suwon")]
Seoul Suwon
9930  1194
```

49

- 논리형 벡터에 의한 인덱싱
 - TRUE가 있는 위치의 자료만 선택
 - 벡터의 비교에 의한 자료 선택에서 유용하게 사용됨

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

> y[c(TRUE, TRUE, FALSE, FALSE, TRUE)]
[1]  2  4 10

> y>3
[1] FALSE  TRUE  TRUE  TRUE  TRUE

> y[y>3]
[1]  4  6  8 10
```

50

- 조건에 의한 인덱싱

- 벡터 x의 개별 값 중 평균값보다 큰 값 선택

```
> x <- c(80,88,90,93,95,94,99,78,101)

> x >= mean(x)
[1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE TRUE

> x[x >= mean(x)]
[1] 93 95 94 99 101
```

51

- 예제: 벡터 x에서

- 1) 평균으로부터 ± 1 표준편차 안에 있는 관찰값
- 2) 평균으로부터 ± 1 표준편차와 ± 2 표준편차 사이에 있는 관찰값
- 3) 평균으로부터 ± 2 표준편차를 벗어나는 관찰값

$$\bar{x} - s \leq x \leq \bar{x} + s \quad \rightarrow \quad -1 \leq (x - \bar{x})/s \leq 1$$

```
> z <- (x-mean(x))/sd(x)

> x[abs(z) <= 1]
[1] 88 90 93 95 94

> x[abs(z) > 1 & abs(z) <= 2]
[1] 80 99 78 101

> x[abs(z) > 2]
numeric(0)
```

52

2.2 요인(Factor)

- 범주형 데이터만을 위한 구조.
 - 벡터와 같은 1차원 배열.
 - 수준(level): 요인이 취할 수 있는 값

53

2.2.1 요인의 기본 특성

- 명목형 요인의 생성: 함수 factor()

```
> gender <- c("Male", "Female", "Female")
> gender_f <- factor(gender)
> gender
[1] "Male" "Female" "Female"
> gender_f
[1] Male    Female Female
Levels: Female Male
```

- 요인 gender_f의 개별 자료: 인용부호 없음
- Female이 첫 번째, Male이 두 번째 level. 알파벳 순으로 결정
- 수준 (Levels) : 요인 변수에 정의된 값들

54

- 요인과 문자형변수의 차이

```
> as.numeric(gender)
[1] NA NA NA
경고메시지(들):
강제형변환에 의해 생성된 NA 입니다
> as.numeric(gender_f)
[1] 2 1 1
> as.numeric(gender_f) + 1
[1] 3 2 2
> gender_f + 1      # 직접적인 수리적 계산 불가능
[1] NA NA NA
경고메시지(들): In Ops.factor(gender_f, 1) :
요인(factors)에 대하여 의미있는 '+'가 아닙니다.
```

- 요인은 수준이름 으로 표시되지만 내부적으로 알파벳 순서에 따라 정수 값으로 저장된다.
- 요인은 as.numeric() 함수를 통해 숫자화 할 수 있다.
- 이 기능을 이용하여 요인은 다양한 계량적 분석이 가능해진다.
- 문자는 문자열로 표시, 저장되어 계량분석이 제한적이다.

55

- 옵션 labels의 활용

- 수준의 이름 변경

```
> x <- c(1, 3, 2, 2, 1, 4)
> factor(x)
[1] 1 3 2 2 1 4
Levels: 1 2 3 4
> factor(x, labels=c("A", "B", "C", "D"))
[1] A C B B A D
Levels: A B C D
```

- 수준 병합

```
> factor(x, labels=c("A","A","B","B"))
[1] A B A A A B
Levels: A B
```

56

- 요인의 유형 및 속성

- 정수형. Female=1, Male=2로 입력되어 있음
- class 속성: 객체의 속성중 하나.
 - 요인
 - 객체 지향 프로그램에서 중요한 역할

```
> typeof(gender_f) #데이터 유형표시
[1] "integer"
> class(gender_f)   #class 속성표시
[1] "factor"
```

```
> summary(gender) #문자형: 속성분석
  Length      Class      Mode
      3 character character

> summary(gender_f) #요인형: 빈도분석
  Female   Male
      2     1
```

generic 함수의 적용 예

입력되는 객체의 class 속성에 따라 다른 분석 실시

57

- 순서형 요인의 생성: 순서형은 table, bar chart, 모델링 등에서 효과적
 - 함수 factor()에 옵션 order=TRUE 추가
 - level의 순서를 조절하고자 하는 경우, 옵션 level에서 지정
 - 결과적으로 storage에 배정되는 숫자 변경

```
> income <- c("Low", "Medium", "High", "Medium")
> factor(income) # High 1, Low 2, Medium 3 배정
[1] Low      Medium High      Medium
Levels: High Low Medium
> as.numeric(factor(income))
[1] 2 3 1 3

> factor(income, order=TRUE) # High 1, Low 2, Medium 3 배정
[1] Low      Medium High      Medium
Levels: High < Low < Medium
> as.numeric(factor(income,order=T))
[1] 2 3 1 3

> factor(income, order=TRUE,level=c("Low", "Medium", "High"))
[1] Low      Medium High      Medium
Levels: Low < Medium < High
> as.numeric(factor(income,order=TRUE,level=c("Low", "Medium", "High")))
[1] 1 2 3 2

# Low 1, Medium 2 High 3 배정
```

58

2.2.2 숫자형 벡터를 요인으로 변환

- 연속형 변수 → 범주형 변수로 변환
 - 변환방법 1: 논리형 벡터 이용 (생략)
 - 변환방법 2: 함수 `cut()` 이용
- 예제: 숫자형 벡터 `x`를 '90 이상', '90 미만 80 이상', '80 미만'의 세 등분으로 구분하고 각각 A, B, C의 값을 갖는 요인으로 변환

```
> x <- c(80,88,90,93,95,94,100,78,65)
```

59

- 함수 `cut()`에 의한 방법

```
> cat.x <- cut(x, breaks=c(0,80,90,100), include.lowest=TRUE,  
               right=FALSE, labels=c("C","B","A"))  
> cat.x  
[1] B B A A A A A C C  
Levels: C B A
```

`breaks`: 구간의 최소값, 최대값을 포함한 구간 설정 벡터

`right`: 구간이 $(a < x \leq b)$ 이면 TRUE, $(a \leq x < b)$ 이면 FALSE

`include.lowest`: **구간**의 최소값(`right=TRUE` 일때) 또는 최대값(`right=FALSE` 일때)과 같은 관찰값도 변환에 포함시킬지 여부

- `include.lowest=T` $\min \leq x \leq b$ (`right=TRUE`)

$a \leq x \leq \max$ (`right=FALSE`)

`labels`: 수준(level)의 라벨 지정

60

- 함수 cut()으로 순서형 요인 생성: 옵션 ordered_result=TRUE

```
> cut(x, breaks=c(0,80,90,100), include.lowest=TRUE,  
      right=FALSE, labels=c("C","B","A"), ordered_result=TRUE)  
[1] B B A A A A A C C  
Levels: C < B < A
```

61

2.3 날짜

- 시간의 흐름에 따라 데이터 얻는 경우, 중요한 변수

- 생성: 문자형 벡터에 함수 as.Date() 적용
- 문자형 벡터의 디폴트 형태: yyyy-mm-dd

```
> x <- as.Date(c("2017-01-01", "2018-01-01"))  
> x  
[1] "2017-01-01" "2018-01-01"
```

- 유형: 숫자형. 1970년 1월 1일부터의 날수

```
> typeof(x)  
[1] "double"  
  
> x[2]-x[1]  
Time difference of 365 days
```

62

- 일정한 간격의 날짜 생성: 함수 seq()
 - 동일한 간격의 날짜 생성: 옵션 by에 숫자 지정

```
> s1 <- as.Date("2018-03-01")
> e1 <- as.Date("2018-03-31")
> seq(from=s1, to=e1, by=7)
[1] "2018-03-01" "2018-03-08" "2018-03-15" "2018-03-22"
[5] "2018-03-29"
```

- 증가 폭을 주 단위 혹은 월 단위로 조절 가능

```
> seq(from=s1, by="week", length=5)
[1] "2018-03-01" "2018-03-08" "2018-03-15" "2018-03-22"
[5] "2018-03-29"
> seq(from=s1, by="month", length=5)
[1] "2018-03-01" "2018-04-01" "2018-05-01" "2018-06-01"
[5] "2018-07-01"
> seq(from=s1, by="year", length=5)
[1] "2018-03-01" "2019-03-01" "2020-03-01" "2021-03-01"
[5] "2022-03-01"
```

63

2.4 행렬 및 배열

- 행렬(2차원 구조) 및 배열(2차원 이상의 구조)
 - 1차원 구조인 벡터에 dim 속성이 추가된 형태
 - 구성요소는 같은 유형의 데이터(숫자형, 문자형, 논리형)

64

2.4.1 행렬과 배열의 기본 특성

- 행렬의 생성 1: `matrix()`
 - 행과 열의 개수 `nrow=` 또는 `ncol=`로 지정(둘 중 하나)

```
> x <- matrix(1:12, nrow=3)
> x
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- 자료가 열 단위로 입력

```
> y <- matrix(1:12, nrow=3, byrow=TRUE)
> y
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

- 옵션 `byrow=TRUE` 로
행 단위로 입력

65

- 행렬 생성 2: `cbind()`, `rbind()`
 - `cbind()`: 벡터들을 열 단위로 묶어 행렬 생성
 - `rbind()`: 벡터들을 행 단위로 묶어 행렬 생성

```
> x1 <- 1:3
> x2 <- 4:6

> (A <- cbind(x1,x2))
      x1 x2
[1,]  1  4
[2,]  2  5
[3,]  3  6

> (B <- rbind(x1,x2))
      [,1] [,2] [,3]
x1      1    2    3
x2      4    5    6
```

66

- 기존의 행렬에 행 또는 열 추가

```
> cbind(A, x3=7:9)
      x1 x2 x3
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9

> rbind(A, 7:8)
      x1 x2
[1,]  1  4
[2,]  2  5
[3,]  3  6
[4,]  7  8
```

67

- 결합 대상이 되는 벡터의 길이가 다른 경우: 순환법칙 적용

```
> x1 <- 1:4
> x2 <- 5:6
> x3 <- 7

> cbind(x1,x2,x3)
      x1 x2 x3
[1,]  1  5  7
[2,]  2  6  7
[3,]  3  5  7
[4,]  4  6  7
```

68

- 행렬의 생성 3: dim()

```
> x <- 1:12
> dim(x) <- c(3,4)
> x
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	4	7	10
[2,]	2	5	8	11
[3,]	3	6	9	12

69

- 행렬의 행과 열에 이름 붙이기

- 함수 rownames()와 colnames()

```
> x
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	4	7	10
[2,]	2	5	8	11
[3,]	3	6	9	12

```
> rownames(x) <- c("one", "two", "three")
> colnames(x) <- c("a", "b", "c", "d")

> x
```

	a	b	c	d
one	1	4	7	10
two	2	5	8	11
three	3	6	9	12

70

- 행렬의 길이 확인

```
> x
      a b c d
one   1 4 7 10
two   2 5 8 11
three 3 6 9 12

> length(x)
[1] 12

> nrow(x); ncol(x)
[1] 3
[1] 4

> dim(x)
[1] 3 4
```

71

- 배열의 생성

- 함수 `array()`: 각 차원에 대한 정의를 함수 `c()`로 지정

```
> xyz <- array(1:24, c(4, 3, 2))
> xyz
, , 1
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2
      [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

72

- 배열의 각 차원에 이름 부여

- 함수 `dimnames()`: 리스트로 할당

```
> dimnames(xyz) <- list(X=c("x1", "x2", "x3", "x4"),
                        Y=c("y1", "y2", "y3"),
                        Z=c("z1", "z2"))

> xyz
, , Z = z1

      Y
X    y1 y2 y3
x1   1  5  9
x2   2  6 10
x3   3  7 11
x4   4  8 12

, , Z = z2

      Y
X    y1 y2 y3
x1  13 17 21
x2  14 18 22
x3  15 19 23
x4  16 20 24
```

73

- 행렬과 배열의 인덱싱

- 행렬 인덱싱

`x[i,j]`: 행렬 `x`의 i 번째 행, j 번째 열의 요소

`x[i,]`: 행렬 `x`의 i 번째 행 전체 # `[i,빈칸]` 의미로 이해

`x[,j]`: 행렬 `x`의 j 번째 열 전체

<pre>> x [,1] [,2] [,3] [,4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12 > x[2,3] # 2번째 행, 3번째 열 [1] 8 > x[1,] # 1번째 행 [1] 1 4 7 10</pre>	<pre>> x[,2] # 2번째 열 [1] 4 5 6 > x[1:2,] [,1] [,2] [,3] [,4] [1,] 1 4 7 10 [2,] 2 5 8 11</pre>
---	---

74

- 배열 인덱싱
 - 차원 수만큼의 첨자 필요

```
> xyz <- array(1:24, c(4, 3, 2))

> xyz[,1,1]
[1] 1 2 3 4

> xyz[, ,1]
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
```

75

2.4.2 행렬의 연산

- 선형대수 문제뿐만이 아니라 다양한 통계분석에서도 종종 사용됨

연산자 및 함수	기능	A, B : 행렬 x, b : 벡터 k : 스칼라
+ - * / ^ A %*% B	행렬을 구성하는 숫자 각각에 적용 행렬 A와 B의 곱하기	
colMeans(A) colSums(A)	행렬 A 각 열의 평균값으로 구성된 벡터 행렬 A 각 열의 합으로 구성된 벡터	
diag(A) diag(x) diag(k)	행렬 A의 대각선 원소로 구성된 벡터 벡터 x를 대각선 원소로 하는 대각행렬 k * k 단위행렬	
eigen(A)	행렬 A의 고유값, 고유벡터로 구성된 리스트	
rowMeans(A) rowSums(A)	행렬 A 각 행의 평균값으로 구성된 벡터 행렬 A 각 행의 합으로 구성된 벡터	
solve(A) solve(A, b)	행렬 A의 역행렬 연립방정식 Ax=b의 해	
t(A)	행렬 A의 전치 (A^T)	

76

```

> A <- matrix(1:4, nrow=2, byrow=T)
> B <- matrix(5:8, nrow=2, byrow=T)
> A
      [,1] [,2]
[1,]    1    2
[2,]    3    4
> B
      [,1] [,2]
[1,]    5    6
[2,]    7    8
> A * B
      [,1] [,2]
[1,]    5   12
[2,]   21   32
> A %*% B
      [,1] [,2]
[1,]   19   22
[2,]   43   50
> t(A)
      [,1] [,2]
[1,]    1    3
[2,]    2    4
> colMeans(A)
[1] 2 3
> rowSums(A)
[1] 3 7
> diag(A)
[1] 1 4
> x <- c(10,20); diag(x)
      [,1] [,2]
[1,]   10    0
[2,]    0   20
> diag(2)
      [,1] [,2]
[1,]    1    0
[2,]    0    1

```

77

```

> A
      [,1] [,2]
[1,]    1    2
[2,]    3    4
> solve(A)
      [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> solve(A)%*%A
      [,1] [,2]
[1,]    1 4.440892e-16
[2,]    0 1.000000e+00
> b <- c(5,6); solve(A,b)
[1] -4.0  4.5

```

$$\begin{aligned} x + 2y &= 5 \\ 3x + 4y &= 6 \end{aligned}$$

78

2.5 데이터 프레임

- 특성
 - 행렬과 같은 2차원 구조
 - 하나의 열에는 같은 유형의 자료
 - 각각의 열은 서로 다른 유형의 자료가 올 수 있음
 - 통계 데이터 세트에 적합한 구조

79

2.5.1 데이터 프레임의 생성

- 데이터 프레임의 생성: `data.frame()`

```
> df1 <- data.frame(x=c(2,4,6), y=c("a","b","c"))
> df1
  x y
1 2 a
2 4 b
3 6 c
```

- 벡터 `x`와 `y`가 `df1`의 변수
- 행 번호 자동 생성

- 함수 `str()`: 데이터 객체의 구조 확인

```
> str(df1)
'data.frame': 3 obs. of 2 variables:
 $ x: num  2 4 6
 $ y: chr  "a" "b" "c"
```

80

- 함수 data.frame()의 특성

- 입력되는 벡터의 길이가 같아야 함

```
> data.frame(x=c(2,4), y=c("a","b","c"))
Error in data.frame(x = c(2, 4), y = c("a", "b", "c")) :
  arguments imply differing number of rows: 2, 3
```

- 길이가 1인 벡터: 순환법칙 적용

```
> data.frame(x=1, y=c("a","b","c"))
  x y
1 1 a
2 1 b
3 1 c
```

81

- 데이터 프레임의 행과 열 이름, 변수 개수 확인

- 열(변수) 이름: 함수 colnames(), names()
- 행 이름: 함수 rownames()
- 변수 개수: length()

```
> df1
  x y
1 2 a
2 4 b
3 6 c

> colnames(df1)
[1] "x" "y"
> names(df1)
[1] "x" "y"

> rownames(df1)
[1] "1" "2" "3"
> length(df1)
[1] 2
```

82

2.5.2 데이터 프레임의 인덱싱

- 데이터 프레임의 인덱싱 1: 리스트에 적용되는 방식
 - 열(변수) 선택.
 - `df[[a]]` 또는 `df[a]`의 형식: 벡터 `a`는 숫자형 혹은 문자형
 - `df[[a]]` : 한 변수의 선택. 결과는 벡터
 - `df[a]` : 하나 또는 그 이상의 변수 선택. 결과는 데이터 프레임

<pre>> df1 x y 1 2 a 2 4 b 3 6 c > typeof(df1) [1] "list" > df1[1] x 1 2 2 4 3 6</pre>	<pre>> df1[[1]] [1] 2 4 6 > df1["x"] x 1 2 2 4 3 6 > df1[["x"]] [1] 2 4 6</pre>
---	--

83

- 데이터 프레임의 변수 선택
 - 벡터 형태로 선택하는 것이 일반적
 - `df[[a]]`의 형태가 더 많이 사용됨
 - 조금 더 편한 방법: `$` 기호 사용

```
> df1[["x"]]
[1] 2 4 6

> df1$x
[1] 2 4 6
```

데이터 프레임 이름\$변수 이름

84

- 데이터 프레임의 인덱싱 2: 행렬에 적용되는 방법
 - `df[i, j]`의 형태
 - 선택된 변수가 하나이면 결과는 벡터
하나 이상이면 결과는 데이터 프레임

```
> df1[c(1,2),1]
[1] 2 4

> df1[c(1,2),]
  x y
1 2 a
2 4 b
```

85

2.5.3 함수 `with()`

- 데이터 프레임을 대상으로 하는 통계 분석
 - 데이터 프레임의 개별 변수를 벡터 형태로 선택하여 분석 진행
 - 인덱싱 기법에 의한 변수 선택: 매우 번거로운 방법
- 함수 `with()` : 편하게 데이터 프레임에 접근하는 방법

86

- 함수 with()의 사용법

- 일반적인 사용 형태: with(데이터 프레임, R 명령문)
 - with() 안에서는 지정된 데이터 프레임의 변수를 인덱싱 없이 사용 가능
- 예제: 데이터프레임 airquality
 - 미국 뉴욕시의 공기 질과 관련된 데이터
 - 변수 Temp의 표준화: $(x - \bar{x})/s$

```
> z.Temp <- (Temp-mean(Temp))/sd(Temp)
Error: object 'Temp' not found

> z.Temp <- (airquality$Temp-mean(airquality$Temp))/sd(airquality$Temp)

> z.Temp <- with(airquality, (Temp-mean(Temp))/sd(Temp))
```

87

2.6 tibble: 개선된 형태의 데이터 프레임

- tibble: tidyverse에 속한 패키지들이 공통적으로 사용하는 데이터 프레임
- 전통적인 데이터 프레임에 몇 가지 기능을 추가하여 사용하기 더 편리한 형태를 취하고 있음
- 패키지 tibble: core tidyverse에 속한 패키지

```
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.3.0 --
√ ggplot2 3.3.1      √ purrr  0.3.4
√ tibble  3.0.1      √ dplyr  1.0.0
√ tidyr   1.1.0      √ stringr 1.4.0
√ readr   1.3.1      √ forcats 0.5.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

88

2.6.1 tibble의 생성

- 기존의 데이터 프레임을 tibble로 전환

- 함수 `as_tibble()`

```
> as_tibble(cars)
# A tibble: 50 x 2
  speed dist
  <dbl> <dbl>
1     4     2
2     4    10
3     7     4
4     7    22
5     8    16
6     9    10
7    10    18
8    10    26
9    10    34
10   11    17
# ... with 40 more rows
```

← 유형 표시

89

- 개별 벡터를 이용한 tibble의 생성

- 함수 `tibble()`

```
> tibble(x=1:3,y=x+1,z=1)
# A tibble: 3 x 3
      x     y     z
  <int> <dbl> <dbl>
1     1     2     1
2     2     3     1
3     3     4     1
```

- 길이가 1인 스칼라만 순환법칙 적용
(만일 `z=1:2`가 입력되면?)

- 함께 입력되는 변수를 이용한 다른
변수의 생성 가능

- 열(변수) 단위로 입력

- 함수 `tibble()` vs `data.frame()`

- 함께 입력된 변수를 이용하여 다른 변수를 만드는 기능

```
> data.frame(x=1:3,y=x+1)
Error in data.frame(x = 1:3, y = x + 1) : object 'x'
not found
```

- 함수 `data.frame()`에서는 가능하지 않음

90

- 행 단위로 입력하여 tibble 생성

- 함수 tribble()

```
> tribble(
  ~x, ~y,
  1, "a",
  2, "b",
  3, "c"
)
# A tibble: 3 x 2
  x y
<dbl> <chr>
1     1 a
2     2 b
3     3 c
```

- 첫 줄: 변수 이름은 '~'로 시작

- 각 자료는 콤마로 구분

91

2.6.2 tibble과 전통적 데이터 프레임의 비교

- 주된 차이점
 - 데이터 프레임의 출력 방식
 - 인덱싱 방법
 - Row names를 다루는 방식

92

● 출력 방식의 차이

- 전통적 데이터 프레임: 가능한 모든 자료를 화면에 출력. 대규모 자료의 경우 내용 확인에 어려움

```
> data(Cars93, package="MASS")
> Cars93
```

- Tibble: 처음 10개 케이스만 출력. 화면의 크기에 따라 출력되는 변수의 개수 조절. 한 화면에서 자료의 특성 파악 용이

```
> as_tibble(Cars93)
```

```
# A tibble: 93 x 27
  Manufacturer Model      Type Min.Price Price Max.Price MPG.city
  <fct>          <fct>    <fct>   <dbl> <dbl>   <dbl>   <int>
1 Acura          Integra Small    12.9  15.9    18.8    25
2 Acura          Legend  Midsi~   29.2  33.9    38.7    18
3 Audi           90      Compa~   25.9  29.1    32.3    20
4 Audi           100     Midsi~   30.8  37.7    44.6    19
5 BMW            535i     Midsi~   23.7  30     36.2    22
6 Buick           Century Midsi~   14.2  15.7    17.3    22
7 Buick           LeSabre Large    19.9  20.8    21.7    19
8 Buick           Roadmas~ Large    22.6  23.7    24.9    16
9 Buick           Riviera  Midsi~   26.3  26.3    26.3    19
10 Cadillac       Deville Large    33    34.7    36.3    16
# ... with 83 more rows, and 20 more variables: MPG.highway <int>,
#   AirBags <fct>, DriveTrain <fct>, Cylinders <fct>,
#   EngineSize <dbl>, Horsepower <int>, RPM <int>,
#   Rev.per.mile <int>, Man.trans.avail <fct>,
#   Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>,
#   Rear.seat.room <dbl>, Luggage.room <int>, Weight <int>,
#   Origin <fct>, Make <fct>
```

- 변수 이름과 더불어 변수의 유형을 함께 표시

- 더 많은 자료 확인

```
print(tbl, n=20, width=Inf)
```

tbl: tibble 객체

93

● Row names 처리 방식의 차이

- 전통적 데이터 프레임: 자료 출력 시 row name 함께 출력
- Tibble: 생략

```
> head(mtcars)
```

```
      mpg  cyl  disp  hp  drat    wt  qsec  vs  am  gear  carb
Mazda RX4         21.0   6  160 110  3.90 2.620 16.46 0   1    4    4
Mazda RX4 Wag     21.0   6  160 110  3.90 2.875 17.02 0   1    4    4
Datsun 710        22.8   4  108  93  3.85 2.320 18.61 1   1    4    1
Hornet 4 Drive    21.4   6  258 110  3.08 3.215 19.44 1   0    3    1
Hornet Sportabout 18.7   8  360 175  3.15 3.440 17.02 0   0    3    2
Valiant           18.1   6  225 105  2.76 3.460 20.22 1   0    3    1
```

```
> mtcars_t <- as_tibble(mtcars)
> print(mtcars_t, n=6)
```

```
# A tibble: 32 x 11
  mpg  cyl  disp  hp  drat    wt  qsec  vs  am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21    6   160   110  3.9    2.62 16.5    0    1    4    4
2  21    6   160   110  3.9    2.88 17.0    0    1    4    4
3  22.8   4   108    93  3.85    2.32 18.6    1    1    4    1
4  21.4   6   258   110  3.08    3.22 19.4    1    0    3    1
5  18.7   8   360   175  3.15    3.44 17.0    0    0    3    2
6  18.1   6   225   105  2.76    3.46 20.2    1    0    3    1
# ... with 26 more rows
```

94

- 생략된 row name: 변수로 전환

- 함수 rownames_to_column()

```
> mtcars_d <- rownames_to_column(mtcars, var="rowname")
> mtcars_t <- as_tibble(mtcars_d)
> mtcars_t
```

```
# A tibble: 32 x 12
  rowname   mpg   cyl  disp    hp  drat    wt   qsec    vs  am
  <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4      21     6   160   110   3.9   2.62  16.5     0     1
2 Mazda RX4 W~   21     6   160   110   3.9   2.88  17.0     0     1
3 Datsun 710     22.8    4   108    93   3.85   2.32  18.6     1     1
4 Hornet 4 Dr~   21.4    6   258   110   3.08   3.22  19.4     1     0
5 Hornet Spor~   18.7    8   360   175   3.15   3.44  17.0     0     0
6 Valiant        18.1    6   225   105   2.76   3.46  20.2     1     0
7 Duster 360     14.3    8   360   245   3.21   3.57  15.8     0     0
8 Merc 240D      24.4    4   147    62   3.69   3.19   20      1     0
9 Merc 230       22.8    4   141    95   3.92   3.15  22.9     1     0
10 Merc 280      19.2    6   168   123   3.92   3.44  18.3     1     0
# ... with 22 more rows, and 2 more variables: gear <dbl>, carb <dbl>
```

95

- 인덱싱 방법의 차이

- 기호 '\$'을 이용하는 경우: 변수 이름의 부분 매칭 허용 여부

전통적 데이터 프레임: 부분 매칭 허용

```
> df1 <- data.frame(xyz=1:3, abc=letters[1:3])
> df1$x
[1] 1 2 3
```

tibble: 부분 매칭 불허

```
> tb1 <- as_tibble(df1)

> tb1$x
NULL
Warning message:
Unknown or uninitialised column: 'x'.

> tb1$xyz
[1] 1 2 3
```

96

- 행렬 방식의 인덱싱 결과

전통적인 데이터 프레임: 선택되는 변수의 개수에 따라 벡터 혹은 데이터 프레임

tibble: 선택되는 변수의 개수와 관계 없이 항상 tibble 유지

```
> mtcars[, 1:2]
> mtcars[, 1]

> mtcars_t[, 1:2]
> mtcars_t[, 1]
> mtcars_t[1, 1]
```

97

2.7 리스트(list)

- 구조의 특성
 - 가장 포괄적인 구조
 - 구성요소: 벡터, 배열, 데이터 프레임, 함수, 다른 리스트
 - 서로 다른 유형의 객체를 한데 묶은 또 다른 객체

98

- 리스트 생성: list()

```
> x <- list(a=c("one","two","three"), b=1:3, c=list(-1,-5),
            d=data.frame(x1=c("s1","s2"),x2=1:2))
> x
$a
[1] "one" "two" "three"

$b
[1] 1 2 3

$c
$c[[1]]
[1] -1

$c[[2]]
[1] -5

$d
  x1 x2
1 s1  1
2 s2  2
```

99

- 리스트의 인덱싱
 - list[[a]] 또는 list[a]의 형태
 - list[a]: 결과는 리스트
 - list[[a]] 또는 list\$a: 해당되는 구성요소의 객체 구조

```
> x[1]
$a
[1] "one" "two" "three"

> x[[1]]
[1] "one" "two" "three"

> str(x[1])
list of 1
 $ a: chr [1:3] "one" "two" "three"

> str(x[[1]])
chr [1:3] "one" "two" "three"
```

100

- 리스트 x의 4번째 요소를 데이터 프레임 형태로 선택

```
> x[[4]]
  x1 x2
1 s1  1
2 s2  2
```

```
> x$d
  x1 x2
1 s1  1
2 s2  2
```

- 리스트 x의 4번째 요소의 두 번째 열을 벡터 형태로 선택

```
> x[[4]][[2]]
[1] 1 2
> x$d$x2
[1] 1 2
```

- 리스트 x의 4번째 요소의 두 번째 열을 데이터 프레임 형태로 선택

```
> x[[4]][2]
  x2
1  1
2  2
```

101

- R에서 리스트의 활용

- 산만하게 흩어져 있는 정보를 간편하게 묶을 수 있음
- 많은 R 함수들의 수행 결과가 리스트의 형태로 출력.

그 중 원하는 결과를 리스트의 인덱싱 기법으로 선택하여 사용

102