```
# 데이터프레임의 열이 될 벡터를 생성

id <- c('F1', 'F2', 'F3')

name <- c('김가수', '박인호', '고소미')

age <- c(32 , 28 , 22 )

isMarried <- c(TRUE , TRUE , FALSE)


# 데이터프레임 생성

df <- data.frame(id, name, age, isMarried)


# 데이터프레임 출력

df


#------------------------------------------------------------


# 데이터프레임의 구조

str(df)


#------------------------------------------------------------


# 데이터프레임 생성

df <- data.frame(id, name, age, isMarried, stringsAsFactors=FALSE) # 문자인 경우 팩터로 변환할지 여부


# 타입 확인

str(df)
```

```
#------------------------------------------------------------

# 바로 지정해 데이터프레임 만들기
# data.frame(열 이름=벡터, 열 이름=벡터...)
df <- data.frame(name=c("김가수", "박인호"), age=c(32, 28))

# 출력
df

#------------------------------------------------------------
id <- c('F1', 'F2', 'F3')
name <- c('김가수', '박인호', '고소미')
age <- c(32 , 28 , 22 )
isMarried <- c(TRUE , TRUE , FALSE)

# 데이터프레임 생성
df <- data.frame(id, name, age, isMarried)


# 데이터프레임 출력
df

# 2행 3열의 데이터 출력
df[2, 3]
```

```
#------------------------------------------------------------


# 2, 3행 중 2, 4열 출력

df[c(2,3), c(2,4)]


#------------------------------------------------------------


# 2, 3열에 대해 전체 행 출력

df[ , c(2,3)]


# 2, 3행에 대해 전체 열 출력

df[c(2,3), ]


#------------------------------------------------------------


# name과 age 열 출력 - 열 이름으로 접근

df[ , c("name","age")]


#------------------------------------------------------------


# 출력

df


# name이라는 열 이름으로 접근

df$name
```

```
# age라는 열 이름으로 접근

df$age


#-------------------------------------------------------


# df$age 구조 확인 – 숫자 벡터

str(df$age)


# 합계

sum (df$age)


# 2, 3번째 요소 선택

df$age[2:3]


#-------------------------------------------------------


iris


#-------------------------------------------------------


# iris 구조 확인

str(iris)


#-------------------------------------------------------
```

```
# 총 행 수
nrow(iris)

# 총 열 수
ncol(iris)

#------------------------------------------------------------

# head(데이터프레임명, 보고 싶은 행의 수). 행의 수를 입력하지 않으면 기본적으로 6행을 출력
# 앞의 3줄만 봄
head(iris, 3)

# tail(데이터프레임명, 보고 싶은 행의 수). 행의 수를 입력하지 않으면 기본적으로 6행을 출력
# 뒤의 3줄만 봄
tail(iris, 3)

#------------------------------------------------------------

summary(iris)

#------------------------------------------------------------

# Speal.Length의 최솟값 산출 - 열에 접근할 때 기호
min(Sepal.Length)

#------------------------------------------------------------
```

```
# 최솟값

min(iris$Sepal.Length)


# 최댓값

max(iris$Sepal.Length)


# 중간값

median(iris$Sepal.Length)


# 평균

mean(iris$Sepal.Length)


# 4분위수

quantile(iris$Sepal.Length)


#-------------------------------------------------------------


# 데이터프레임 조회(View의 첫 글자 V는 대문자입니다.)

View(iris)


#-------------------------------------------------------------


# iris에서 Sepal.Length가 7보다 큰 데이터 조회

# (열벡터(select)를 입력하지 않으면 전체 열이 조회됨)

subset(iris, Sepal.Length > 7)
```

```
#------------------------------------------------------------

# Length가 7보다 크고 Length가 6.6보다 같거나 작은 것
subset(iris, Sepal.Length > 7 & Petal.Length <= 6.5)

#------------------------------------------------------------

# Length가 7보다 크거나 Length가 6.6보다 크거나 같은 것
subset(iris, Sepal.Length > 7 | Petal.Length >= 6.5)

#------------------------------------------------------------

# Sepal.Length와 species 열만 보기(열 위치로 지정)
subset(iris, Sepal.Length > 7.2 & Petal.Length <= 6.5, c( 1, 5))

# Sepal.Length와 species 열만 보기(열 이름으로 지정)
subset(iris, Sepal.Length > 7.2 & Petal.Length <= 6.5, c( "Sepal.Length", "Species"))

#------------------------------------------------------------

# Sepal.Length, species 순서로 출력
subset(iris, Sepal.Length > 7.2 & Petal.Length <= 6.5, c( "Sepal.Length", "Species"))

# species, Sepal.Length 순서로 출력
subset(iris, Sepal.Length > 7.2 & Petal.Length <= 6.5, c( "Species", "Sepal.Length"))
```

```
subset(iris, Species == 'setosa')

#-------------------------------------------------------------

# subset 결괏값 구조
str(subset(iris, Species == "setosa"))

# iris에서 Species 항목의 값이 setosa인 데이터 행의 수
nrow(subset (iris, Species == "setosa"))

# Species 항목의 값이 setosa인 데이터의 요약 정보
summary(subset(iris, Species == "setosa"))

#-------------------------------------------------------------

# 입력 항목명을 명시하지 않고 호출하기 – 입력 항목의 순서가 중요
subset(iris, Sepal.Length == 7.2, c("Species", "Sepal.Length"))

# 입력 변수명을 명시하지 않고 호출하기 - 순서가 맞지 않을 경우 오류 발생
subset(Sepal.Length == 7.2, iris, c("Species", "Sepal.Length"))

# 입력 변수명을 명시하고 호출하기
subset(x=iris, subset=(Sepal.Length == 7.2), select=c("Species", "Sepal.Length"))

# 입력 변수명을 명시하고 호출하기 - 순서 바꿔보기
```

```
subset(select=c("Species", "Sepal.Length"), subset=(Sepal.Length == 7.2), x=iris)

#------------------------------------------------------------

# ex_df를 만들 벡터 생성
name <- c('김가수', '박인호', '어만데', '이기성')
age <- c(23, 28, 15, 22)
weight <- c(67, 75, 73, 80)

# ex_df 생성
ex_df <- data.frame(name, age, weight)

#------------------------------------------------------------
# ex_df 조회 - [행, 열]
ex_df

# 별도로 지정하지 않음(전체 출력)
ex_df[ , ]

# 행과 열 이름을 직접 지정
ex_df[c("1","2"), c("name", "weight")]

# 행과 열 위치를 직접 지정(1, 2번째 행, 1, 3번째 열 선택)
ex_df[c(1,2), c(1,3)]

# 보여주고자 하는 위치를 논리 벡터로 직접 지정(선택할 위치에 TRUE)
```

```
ex_df[c(T,T,F,T), c(T,F,T)]

#------------------------------------------------------------

# age 열 조회(열 선택)
ex_df$age

# 나이가 25보다 큰가?(논리연산)
ex_df$age > 25

#------------------------------------------------------------

# 나이가 25보다 큰 항목 추출(열은 모두 출력)
ex_df[ex_df$age > 25, ]

#------------------------------------------------------------

# Longley 데이터프레임의 구조
str(longley)

longley

# 여러 조건 식으로 데이터를 추출
longley[longley$GNP > 200 & longley$Population >= 109 & longley$Year > 1960 & longley$Employed > 50, ]
```

```
#------------------------------------------------------------

# 조건 검색 - 명시적으로 데이터프레임명 지정

longley[longley$GNP > 200 & longley$Population >= 109 & longley$Year > 1960 & longley$Employed > 50, ]

# attach - R객체 탐색 경로에 longley 추가 - $기호 없이 열에 접근 가능.

attach(longley)

# "longley $" 사용하지 않고 조건 검색 가능

longley[GNP > 200 & Population >= 109 & Year > 1960 & Employed > 50, ]

# detach - R 객체 탐색 경로에 longley 제거

detach(longley)

# detach 이후에는 "longley $"를 명시하지 않으면 오류 발생

longley[GNP > 200 & Population >= 109 & Year > 1960 & Employed > 50, ]

#------------------------------------------------------------

# 칼럼 두 개 추출 시 데이터 타입 확인

longley[ , c("GNP", "Year")]

str(longley[ , c("GNP", "Year")])

# 칼럼 한 개 추출 시 데이터 타입 확인 - 벡터로 전환됨

longley[ , c("GNP")]

str(longley[ , c("GNP")])
```

```
# drop 옵션 사용 시 데이터 타입 확인
longley[ , c("GNP"), drop=FALSE]
str(longley[ , c("GNP"), drop=FALSE])

#------------------------------------------------------------

# iris 데이터 구조 보기
str(iris)



# subset 이용
subset(iris, Sepal.Length >= 7.6 & Species == "virginica", c ("Species", "Sepal.Length"))



# 데이터프레임 직접 검색
iris[iris$Sepal.Length >= 7.6 & iris$Species == "virginica", c("Species", "Sepal.Length")]

#------------------------------------------------------------

longley

# 만약 sqldf 패키지가 설치돼 있지 않다면 설치
install.packages("sqldf")

# sqldf 패키지 로드
```

```
library(sqldf)

# SQL 사용
sqldf("select GNP, Year, Employed from longley where GNP > 400")

# SQL 사용
sqldf("select Year, sum(GNP) from longley where Year > 1960 group by Year")

#------------------------------------------------------------

# 데이터프레임 확인
ex_df

# 1, 2, 4행 선택
ex_df[c(1,2,4), ]

# 행의 위치를 별도 지정
ex_df[c(4,2,1), ]

#------------------------------------------------------------

# 데이터프레임 확인
ex_df

# 나이
```

```
ex_df$age

# order 함수를 이용해 정렬된 요소의 위치 확인 - order(x..., decreasing, na.last)
# 3, 4, 1, 2번째 요소의 순서로 값이 큼
order(ex_df$age)

# [] 연산자로 정렬 수행 - 나이를 기준으로 오름차순 정렬
ex_df[order(ex_df$age), ]

# [] 연산자로 정렬 수행 - 나이를 기준으로 내림차순 정렬
ex_df[order(ex_df$age, decreasing=T), ]

#------------------------------------------------------------

# 임의 벡터 생성
alpha <- c('A','C','F','B','E','D')
alpha

# order 함수는 정렬된 위치벡터 산출
order(alpha)
# order 함수를 이용해 alpha 벡터 요소 정렬
alpha[order(alpha)]

# 오름차순 정렬 – 값 자체를 정렬
sort(alpha)
```

```
# 내림차순 정렬
sort(alpha, decreasing=T)

#-----------------------------------------------------------

# com_data를 만들 벡터 생성
dept <- c('개발부', '개발부', '개발부', '개발부','영업부', '영업부', '영업부', '영업부')
position <- c('과장', '과장', '차장', '차장','과장', '과장', '차장', '차장')
name <- c('김가윤', '고동산', '박기성', '이소균','황가인', '최유리', '김재석', '유상균')
salary <- c(5400, 5100, 7500, 7300, 4900, 5500, 6000, 6700)
worktime <- c(15, 18, 10, 12, 17, 20, 8, 9)

# com_data 생성
com_data <- data.frame(dept, position, name, salary, worktime)

#-----------------------------------------------------------

com_data

# 부서(dept)별 급여(salary) 평균
aggregate(salary ~ dept, com_data, mean)

# 부서(dept)별 급여(salary) 및 근무시간(worktime) 평균
aggregate(cbind(salary, worktime) ~ dept, com_data, mean)

# 부서(dept) & 직급(position)별 급여(salary) 평균
```

```
aggregate(salary ~ dept + position, com_data, mean)

#------------------------------------------------------------

ex_df

# ex_df의 값을 수정해 ex_df_m 생성 - 값이 크지 않다면.
ex_df_m <- edit(ex_df)

#------------------------------------------------------------

# ex_df 출력
ex_df

# ex_df_m 출력
ex_df_m

#------------------------------------------------------------

df

# 김가수의 나이는? -> 첫 번째 행의 세 번째 열 데이터
df[1, 3]

# 김가수 나이를 100으로 변경
df[1, 3] <- 100
```

```
# 김가수의 나이는?

df[1, 3]


# 변경된 데이터 확인

df


#-------------------------------------------------------------


# 데이터 조회

subset(iris, Species == "virginica" & Petal.Width >= 2.4)


# 데이터프레임에 조건 검색을 통해 직접 값을 변경

iris[iris$Species == "virginica" & iris$Petal.Width >= 2.4, c("Petal.Length")] <- 1


# 데이터 조회해 반영된 값을 확인

subset(iris, Species == "virginica" & Petal.Width >= 2.4)


#-------------------------------------------------------------


# 첫 6행만 봅니다.

head(iris)


# Sepal.Width의 전체 값을 2배로 만듭니다.

iris$Sepal.Width <- iris$Sepal.Width*2
```

```
# 변경된 값 확인

head(iris)

#-------------------------------------------------------

# iris의 첫 6행 확인

head (iris)

# 열 추가

iris$new_column <- "신규열"

# 추가된 열 확인

head(iris)

#-------------------------------------------------------

# iris의 첫 6행 확인

head(iris)

# new_column 열 삭제 - 대문자 NULL

iris$new_column <- NULL

# 열이 삭제된 것을 확인

head(iris)

# iris에서 1, 2, 3열을 삭제한 결과를 new_iris에 저장
```

```
new_iris <- iris[ , -c(1, 2, 3)]

# 삭제한 결과가 저장된 데이터프레임 조회
head(new_iris)

# 원본 iris는 변경되지 않았음
head(iris)

# iris에서 직접 열 삭제 – Sepal.Length, Sepal.Width, Petal.Length 열 삭제
iris[ , c(1, 2, 3)] <- list(NULL)

# iris의 첫 6행 확인
head(iris)

#------------------------------------------------------------

# NA를 포함하는 벡터 생성 - not available, 계산에는 포함.
ex_na <- c(1, 2, NA, NA, 3)

# NA가 벡터의 요소로 조회됨
print(ex_na)

# sum처리 -> NA도 계산에 포함되기 때문에 결괏값은 NA가 됨(NA + 숫자 = NA)
sum(ex_na)

# NA를 제외하고 계산하려면 별도 옵션을 지정해야 함
```

```
sum(ex_na, na.rm=TRUE)

# NULL을 포함하는 벡터 생성 - 값이 없다. 계산에 미포함.
ex_null <- c(1, 2, NULL, NULL, 3)

# NULL은 벡터의 요소에 포함되지 않음
print(ex_null)

# sum 계산 -> 문제없음
sum(ex_null)

#------------------------------------------------------------

# 3번째 5번째 요소로 NA를 넣음
ex_cc <- c(1, 2, NA, 4, NA)

# NA가 있는 값은 FALSE 반환 - NA존재 여부 확인
complete.cases(ex_cc)

# 최초 상태 확인 - 데이터가 많아 최초 6개만 확인
head(iris)

# 데이터프레임 변경
iris$Sepal.Length <- 0
iris$Sepal.Width <- 0
iris$Petal.Length <- NULL
```

```
iris$Petal.Width <- NULL

# 값이 0으로 설정되고 칼럼들이 없어짐
head(iris)

# 데이터셋을 다시 로드
data(iris)

# 다시 조회 -> iris는 기본 데이터셋이라 별도 선언 없이 데이터셋명만으로도 로드됨
head(iris)

#------------------------------------------------------------

# iris 열 이름 조회
colnames(iris)

# 벡터 연산 가능 - 3번째 열 이름 바꾸기
colnames(iris)[3] <- "3th_Column"

# iris 열 이름 조회
colnames(iris)

# iris의 첫 6행 조회
head(iris)

# 벡터를 이용해 다수의 열 이름을 동시에 변경. 단, 벡터의 길이는 열의 개수와 같아야 함
```

```
colnames(iris) <- c("꽃받침길이", "꽃받침너비", "꽃잎길이", "꽃잎너비", "종류")

# iris 열 이름 조회
colnames(iris)

# iris의 첫 6행 조회
head(iris)

#------------------------------------------------------------

# 데이터파일 엑셀(시작하세요_데이터분석withR_데이터셋.xlsx)에서 복사 후
# 아래 명령어를 실행 해 데이터 프레임 생성 (2.4.6.2_hotel_rooms 시트)
hotel_rooms <- read.table (file = "clipboard", header=TRUE , sep="₩t", stringsAsFactors=FALSE )

# 호텔 객실 정보
hotel_rooms

# hotel_rooms 구조
str(hotel_rooms)

#------------------------------------------------------------

# 객실 종류 팩터로 변경
hotel_rooms$room_type <- as.factor(hotel_rooms$type)

# 객실 번호 문자로 변경
```

```
hotel_rooms$room_number <- as.character(hotel_rooms$room_number)

# hotel_rooms 구조
str(hotel_rooms)

#------------------------------------------------------------

# hotel_rooms 구조
str(hotel_rooms)

# 방 종류별 평균 가격 – 문자형인 경우 수식 계산을 할 수 없음
aggregate(price ~ room_type, hotel_rooms , mean)

# 가격 숫자로 변경
hotel_rooms$price <- as.numeric(hotel_rooms$price)

# hotel_rooms 구조
str(hotel_rooms)

# 방 종류별 평균 가격
aggregate(price ~ type, hotel_rooms, mean)

#------------------------------------------------------------

# ex_df_age를 만들 벡터 생성
id <- c('F1', 'F2', 'F3', 'F4')
```

```
name <- c('김가인', '박지성', '고아라', '이승철')

age <- c(24, 32, 18, 40)


# ex_df_age 생성

ex_df_age <- data.frame(id, name, age)


# 사용자의 나이가 있는 데이터프레임

ex_df_age


# ex_df_score를 만들 벡터 생성

id <- c('F2', 'F1', 'F4', 'F3')

name <- c('박지성', '김가인', '이승철', '고아라')

age <- c(95, 100, 56, 73)


# ex_df_score 생성

ex_df_score <- data.frame(id, name, age)


# 사용자의 점수가 있는 데이터프레임

ex_df_score


# cbind – 옆으로 단순 병합

cbind(ex_df_age, ex_df_score)


#------------------------------------------------------------


# id(id)와 이름(name) 기준으로 두 데이터프레임 결합
```

```
merge(ex_df_age, ex_df_score, by=c("id", "name"))



# id(id)만으로 두 데이터프레임 결합

merge(ex_df_age, ex_df_score, by=c("id"))

#-------------------------------------------------------

# ex_df_age 조회

ex_df_age

# 행을 추가할 데이터프레임 set 생성 – 열 이름을 동일하게 생성

ex_row_add <- data.frame(id="F5", name = "나지용", age=27)

# 생성된 데이터프레임 확인

ex_row_add

# 두 데이터프레임 행 결합

ex_rbind <- rbind(ex_df_age, ex_row_add)

# 결합 결과 확인

ex_rbind
```