POLI803: Maximum Likelihood Estimation

Week 2: Data Cleaning

Ph.D. Student, Sanghoon Park (Univ. of South Carolina)

9/4/2019

이번에는 데이터를 효율적으로, 그리고 체계적으로 전처리하는 방법에 대해서 살펴볼 것이다. 앞서 언급한 바와 같이 전처리 및 데이터 관리를 위해서 `tidyverse` 패키지와 그 패키지에 속하는 다른 패키지들(`tidyverse familiy`), 그리고 함수들을 주로 사용할 것이다.

패키지 불러오기

```
library(tidyverse)   # 데이터 관리 및 전처리를 위한 주요 패키지
library(ezpickr)     # 다른 확장자의 파일을 R로 불러오기 위한 패키지
library(here)        # 현재 작업디렉토리를 R-스크립트가 위치한 디렉토리로 자동설정하는 패키지
library(lubridate)   # 날짜시각 데이터를 원활하게 가공하는 데 특화된 패키지
```

들어가기에 앞서서 간단한 기본 함수들을 다시 리뷰해보자.

```
x <- 10
add_one <- function(x) x + 10
add_one(5) # 결과가 15일까, 20일까?
```

```
## [1] 15
```

자칫하면 위의 함수에서 `x <- 10`으로 x라는 객체에 10을 넣었기 때문에 `10 + 10`이 되어 결과를 20으로 리턴할 것이라고 생각할 수 있다. 하지만 어디까지나 `add_one` 함수가 정의되고 난 이후, x는 `add_one()`의 괄호 안에 들어가는 값으로 재정의되었다. 따라서 5를 `add_one()`에 투입한 순간, 그 함수는 `5 + 10`을 계산하게 된 것이다. 따라서 결과는 15가 된다.

```
new_add_one <- function(x) x + 10 # 그렇다면 이 경우는 어떨까?
new_add_one()
```

```
## Error in new_add_one(): argument "x" is missing, with no default
```

이번에는 `new_add_one()` 함수를 정의하고, x 값을 따로 주지 않고 빈 함수를 작동시켰다. 이때, `new_add_one()`은 주어진 투입값(input)이 없기 때문에, 함수에 요구되는 x를 찾기 위해 함수 안에서 x를 탐색하는 것을 넘어서 한 단계 위에서 x를 찾는다. 바로 처음에 만든 `x <- 10`을 불러오는 것이다. 따라서 `10 + 10`, 20을 출력하게 된다.

함수와 객체의 관계에 대해 다시 한 번 살펴보자.

```
print_hello_world <- function() {
  z <- "Hello, world"
  print(z)
}

print_hello_world()
```

```
## [1] "Hello, world"
```

```
print(z)
```

```
## Error in print(z): object 'z' not found
```

`print_hello_world()` 함수를 작동시키면 함수 내에서 정의된 객체 `z`의 값을 반환한다. 그렇지만 그 `z`는 어디까지나 함수 내에서만 정의된 것이지 R의 글로벌 환경에 저장된 객체는 아니다. 따라서 함수 내에서 정의된 `z`를 출력하라고 명령하면, 오류 코드를 확인하게 된다.

R에서 객체를 제외하고 작동하는 모든 기능들은 '함수'라고 부른다. 그리고 함수는 같은 결과를 다른 방식으로 출력할 수도 있다.

```
5 + 5      # 간단하게 말하자면 이 함수(+)는
```

```
## [1] 10
```

```
`+`(5, 5) # 이런 식으로도 쓸 수 있다.
```

```
## [1] 10
```

또한, 기존에 R에는 내장되지 않았던 함수도 별도로 특정하게 지정하여 만들 수 있다. 그러나 이 경우에는 별도의 패키지로 만들어서 저장해주지 않는 한, R 코드가 작성된 해당 세션에서만 지속되는 함수일 뿐이다.

아래는 문자열과 문자열을 하나로 합치고 있은데, 일반적으로 두 객체를 합칠 때 쓰는 함수인 +는 숫자형 객체 간에만 기능한다. 따라서 문자열끼리 합쳐주는 함수, `paste()`와 동일한 기능을 하는 별도의 함수 기호를 하나 만들어주고자 한다.

```
print("hello" + "world") # 오류메시지를 확인할 수 있다. +는 숫자형 객체들에만 작동한다.
```

```
## Error in "hello" + "world": non-numeric argument to binary operator
```

```
paste("hello ", "world")
```

```
## [1] "hello  world"
```

```
`%+%` <- function(lhs, rhs) paste0(lhs, rhs)
print("hello " %+% "world")
```

```
## [1] "hello world"
```

이제 패키지를 불러오는 작업과 간단한 함수, 그리고 객체의 특성에 대해 리뷰했으니 다음으로 넘어가 보자.

작업 디렉토리 설정하기

아까 불러온 `here` 패키지의 `here()` 함수를 사용할 수 있다. `here()`를 사용하면 자동으로 현재 R-스크립트가 저장된 경로를 확인, 복사한다. `%>%`는 파이프 왼쪽의 기능 이후에 오른쪽의 기능을 적용시키는 지정된 함수로 `tidyverse` 패키지가 가지고 있는 강점 중 하나이다.

이 파이프 함수를 이용하여 우리는 코드를 보다 논리적으로, 그리고 정연하게 작성하여 가독성을 높일 수 있다.

```
here() %>% setwd() # here()로 R-스크립트의 디렉토리를 확인하고 난 다음에
                   # 그 디렉토리로 작업 디렉토리를 설정(set working directory, setwd)하라는
                   # 함수를 작동시킨 것이다.
```

```
## 이렇게 작업 디렉토리를 설정하였다면, 이제 데이터를 불ㄹ오자.: diamond_data를 사용한다.
df <- pick("example_data/diamonds_data.xlsx") # pick() 함수는 ezpickr 패키지에 속해 있다.
## ezpickr는 여러 유형의 자료를 Hadley Wickam이 개발한 신뢰할 수 있는 여러 패키지들의
## 함수와 연동하여 tibble의 형태로 사용할 수 있도록 도와준다.
typeof(df)
```

```
## [1] "list"
```

데이터 전처리 하기(Data Cleaning)

dplyr 패키지를 이용한 데이터 전처리

데이터를 들여다보고, 결측치를 확인하기

```
nrow(df)    # 데이터의 행의 개수를 확인하는 함수이다.
```

```
## [1] 53940
```

```
ncol(df)    # 데이터의 열의 개수를 확인하는 함수이다.
```

```
## [1] 11
```

```
length(df) # 데이터에 속한 관측치의 개수를 확인하는 함수이다.
```

```
## [1] 11
```

```
# 기본적으로 R에 내장된 함수들을 이용하여 변수를 만들고 목록화하기(indexing)
df$index <- 1:nrow(df) # index라는 새로운 변수를 만들고 행의 수로 연번 매기기
head(df)               # 맨 위의 몇 개 행을 보여준다.
```

```
## # A tibble: 6 x 12
##   carat cut   color clarity depth table price     x     y     z sold  index
##   <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <int>
## 1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~     1
## 2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~     2
## 3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~     3
## 4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~     4
## 5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold      5
## 6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~     6
```

```
glimpse(df)            # 데이터의 구조를 깔끔하게 보여주는 함수이다.
```

```
## Observations: 53,940
## Variables: 12
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, ...
## $ cut     <chr> "Ideal", "Premium", "Good", "Premium", "Good", "Very G...
## $ color   <chr> "E", "E", "E", "I", "J", "J", "I", "H", "E", "H", "J",...
## $ clarity <chr> "SI2", "SI1", "VS1", "VS2", "SI2", "VVS2", "VVS1", "SI...
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, ...
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54...
## $ price   <dbl> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339,...
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, ...
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, ...
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, ...
## $ sold    <chr> "unsold", "unsold", "unsold", "unsold", "sold", "unsol...
## $ index   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
```

데이터를 좀 더 자세하게 들여다보기

```
df # 이미 ezpickr로 불러와서 자료유형은 tibble이다.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
```

```
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
print(df, n = 12) # tibble 유형의 데이터 df의 첫 12개 행을 보여준다.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## 11 0.3   Good  J     SI1      64      55   339  4.25  4.28  2.73 unso~
## 12 0.23  Ideal J     VS1      62.8    56   340  3.93  3.9   2.46 sold
## # ... with 5.393e+04 more rows, and 1 more variable: index <int>
```

```
df %>% slice(nrow(df) - 5:nrow(df))  # 마지막 5개 행을 제외한 행을 보여준다.
```

```
## # A tibble: 53,935 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  0.72 Prem~ D     SI1      62.7    59  2757  5.69  5.73  3.58 sold
##  2  0.7  Very~ E     VS2      61.2    59  2757  5.69  5.72  3.49 unso~
##  3  0.7  Very~ E     VS2      60.5    59  2757  5.71  5.76  3.47 sold
##  4  0.71 Prem~ F     SI1      59.8    62  2756  5.74  5.73  3.43 unso~
##  5  0.71 Prem~ E     SI1      60.5    55  2756  5.79  5.74  3.49 sold
##  6  0.71 Ideal G     VS1      61.4    56  2756  5.76  5.73  3.53 unso~
##  7  0.79 Prem~ E     SI2      61.4    58  2756  6.03  5.96  3.68 unso~
##  8  0.79 Good  F     SI1      58.1    59  2756  6.06  6.13  3.54 unso~
##  9  0.71 Ideal E     SI1      61.9    56  2756  5.71  5.73  3.54 unso~
## 10  0.79 Ideal I     SI1      61.6    56  2756  5.95  5.97  3.67 unso~
## # ... with 53,925 more rows, and 1 more variable: index <int>
```

```
df %>% slice(nrow(df):12)            # 뒤에서부터 12개 행을 보여준다.
```

```
## # A tibble: 53,929 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  0.75 Ideal D     SI2      62.2    55  2757  5.83  5.87  3.64 unso~
##  2  0.86 Prem~ H     SI2      61      58  2757  6.15  6.12  3.74 unso~
##  3  0.7  Very~ D     SI1      62.8    60  2757  5.66  5.68  3.56 unso~
```

```
##  4  0.72 Good  D     SI1      63.1    55  2757  5.69  5.75  3.61 unso~
##  5  0.72 Ideal D     SI1      60.8    57  2757  5.75  5.76  3.5  unso~
##  6  0.72 Prem~ D     SI1      62.7    59  2757  5.69  5.73  3.58 sold
##  7  0.7  Very~ E     VS2      61.2    59  2757  5.69  5.72  3.49 unso~
##  8  0.7  Very~ E     VS2      60.5    59  2757  5.71  5.76  3.47 sold
##  9  0.71 Prem~ F     SI1      59.8    62  2756  5.74  5.73  3.43 unso~
## 10  0.71 Prem~ E     SI1      60.5    55  2756  5.79  5.74  3.49 sold
## # ... with 53,919 more rows, and 1 more variable: index <int>
```

```
df %>% slice(c(1, 7, 54))            # 원하는 행만을 보여준다.
```

```
## # A tibble: 3 x 12
##   carat cut   color clarity depth table price     x     y     z sold  index
##   <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <int>
## 1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~     1
## 2  0.24 Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~     7
## 3  0.22 Prem~ E     VS2      61.6    58   404  3.93  3.89  2.41 sold     54
```

결측치 확인하기

데이터 안에 결측치(NA, missing data)가 몇 개나 있는지를 확인하는 것은 중요하다. 실제로 통계분석을 수행할 때에는 변수에 결측치가 하나라도 존재하면 최종 모형에서 그 결측치가 속한 행 전체는 분석에서 제외된다. 따라서 분석 모형에 있어서 표본의 수(sample size)라는 측면에서 생각해볼 때, 전체 관측치의 개수만큼이나 결측치가 얼마나 되는 것을 파악하는 것도 중요하다.

```
sum(is.na(df)) # 결측치의 수를 계산하는 함수
```

```
## [1] 0
```

```
df %>% is.na() %>% sum() # 위와 동일한 함수.
```

```
## [1] 0
```

df라는 tibble 을 가지고 is.na(), 즉 df 중 NA인 것들만 골라서 sum()으로 더하라는 코드다.

- 파이프 함수에서 ()의 빈괄호는 앞의 데이터를 그대로 받아넘기는 기능이다.
- 그리고 is.na() 함수는 논리형으로 "()에 들어간 객체에 결측치가 있는가?"를 묻는다.
- 결측치가 있으면 TRUE, 없으면 FALSE로 나올 것이고, R에서 TRUE = 1, FALSE = 0이다.
- 그렇게 나온 TRUE들의 총합을 구하면 df라는 데이터 안의 결측치의 총 개수를 확인할 수 있다.
    - 그렇다면 왜 총합을 구하는 함수 내에 na.rm = T라는 옵션을 사용하지 않은 것일까?
    - 보통 R에서 sum() 함수는 그 객체에 결측치가 하나라도 있으면 전체 계산을 NA로 반환한다.
    - 그러나 이 경우에서는 파이프를 통해서 df에서 결측치/관측치를 각각 1, 0으로 변환시켰기 때문에
    - 더 이상 결측치도 missing data, NA가 아닌 숫자형으로 간주되기 때문에 바로 더할 수 있다.

그렇다면 이제는 각 열마다 (변수마다) 관측치가 몇이나 있는지를 확인해보자.

```
df %>% map_int( function(x) is.na(x) %>% sum() %>% as.integer() )
```

```
##   carat     cut   color clarity   depth   table   price       x       y
##       0       0       0       0       0       0       0       0       0
##       z    sold   index
##       0       0       0
```

```
## df를 map_int 함수로 넘기되, 이 함수는 만약 x라는 객체가 관측치이거든 총합을 구해 그 결과를
## 정수형(integer)로 반환하라는 코드이다.
## 즉, 이 경우 df에 관측치가 있으면 그 관측치의 총합을 더하여 숫자로 바꾼 결과를 출력할 것이다.
## 변수별로 관측치의 개수를 확인할 수 있다.
df %>% map_dbl( function(x) is.na(x) %>% sum() ) # 위와 동일하지만 double 유형의 데이터로 반환한다.
```

```
##   carat     cut   color clarity   depth   table   price       x       y
##       0       0       0       0       0       0       0       0       0
##       z    sold   index
##       0       0       0
```

```
typeof(df %>% map_int( function(x) is.na(x) %>% sum() %>% as.integer() )) # 자료유형 integer
```

```
## [1] "integer"
```

```
typeof(df %>% map_dbl( function(x) is.na(x) %>% sum() ))                  # 자료유형 double
```

```
## [1] "double"
```

```
## 만약 변수명을 따로 보기 싫다면? unname() 함수 추가
df %>% map_int( function(x) is.na(x) %>% sum() %>% as.integer() ) %>% unname()
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0
```

unname() 함수에 대해 조금 더 알아보자.

```
c(one = 1, two = 2, three = 3)          # 벡터 객체는 값에 라벨링을 할 수 있다.
```

```
##   one   two three
##     1     2     3
```

```
unname(c(one = 1, two = 2, three = 3)) # unname()을 사용하면 그 라벨링을 제외한 순수한 요소의
```

```
## [1] 1 2 3
```

```
                                       # 값만을 확인할 수 있다.
```

특정 벡터 내에 결측치의 개수가 몇 개인지를 구하는 코드를 하나의 함수로 다시 만들어서 함수 객체의 형태로 저장하자.

```
get_number_of_missings_in_vector <- function(some_vector) {
  result <- some_vector %>%
    is.na() %>%
    sum() %>%
    as.integer()
  return(result)
}
get_number_of_missings_in_vector(df)
```

```
## [1] 0
```

get_number_of_missings_in_vector() 함수는 아까 위의 함수(주어진 객체의 결측치 확인, 총합 계산, 정수형 반환)를 result라는 객체에 저장하고 반환하라는 코드를 내장하고 있다. 따라서 우리는 앞서와 마찬가지로 df에 결측치가 없다는 0을 반환하게 된다.

앞서 말했다시피 데이터프레임과 같은 형식의 자료에 NA가 있으면, R은 결측치를 제외할 때, 측치가 속한 행을 아예 삭제해버린다.

```
df %>% na.omit()
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
```

```
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

중요한 점은 R은 단지 함수적인 프로그래밍 언어이기 때문에 그 결측치를 제거한 이후에 별도로 저장해주지 않으면 다시 불러오는 객체 df는 결측치가 제거되지 않은 원래의 형태로 다시 불려오게 된다. 즉, `new_df <- df %>% na.omit()`와 같은 식으로 재저장 해주어야만 결측치가 제거된 데이터를 가지게 된다.

데이터에 결측치가 있을 때, 그 결측치들을 제외하게 되면 어떻게 되는지 한 번 아래의 예제 코드로 확인해보자.

```
example <- matrix(c(1, 2, 3, NA), nrow = 2, byrow = T) %>%
  as.data.frame()
example %>% na.omit()
```

```
##   V1 V2
## 1  1  2
```

```
example  # 결측치를 제거한 example 데이터를 재저장 하지 않았기 때문에 결측치가 그대로 남아있다.
```

```
##   V1 V2
## 1  1  2
## 2  3 NA
```

```
example <- example %>% na.omit()
example  # 결측치가 제거되어 있는 것을 확인할 수 있다.
```

```
##   V1 V2
## 1  1  2
```

한 가지 강조할 것은 모든 dplyr 함수는 (아마 거의 모든 tidyverse 함수들은) 첫 번째로 데이터를 지정하여 파이프로 넘기면 그 이후의 파이프들은 맨 처음의 데이터를 그대로 가지고 후속 함수들을 그 데이터에 순차적으로 적용하는 과정을 거치게 된다.

dplyr패키지의 주요 함수들

dplyr::filter(): 필터 함수

dplyr::filter() 함수는 () 안에 설정하는 조건문에 따라서 관측치를 필터링한다. 이때, 조건문은 논리형 연산자로 기능하는데, 조건에 따라 투입값이 참(TRUE)인지 거짓(FALSE)인지 반환한다. R에는 dplyr 패키지 말고도 다른 filter() 함수를 가지고 있기 때문에 ::의 로딩 함수를 가지고 dplyr 패키지의 filter() 함수를 직접 불러오는 것이 확실하다.

```
## df 데이터의 cut이라는 변수가 Ideal이라는 값을 가질 경우만 보여주어라.
df %>% dplyr::filter(cut == "Ideal")
```

```
## # A tibble: 21,551 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46 sold
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71 sold
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68 unso~
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78 sold
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75 sold
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76 unso~
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44 unso~
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72 unso~
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63 sold
## # ... with 21,541 more rows, and 1 more variable: index <int>
```

```
df %>% dplyr::filter(cut %in% c("Ideal", "Premium")) # 조건을 여러 개 걸 수도 있다.
```

```
## # A tibble: 35,342 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  4 0.23  Ideal J     VS1      62.8    56   340  3.93  3.9   2.46 sold
##  5 0.22  Prem~ F     SI1      60.4    61   342  3.88  3.84  2.33 unso~
##  6 0.31  Ideal J     SI2      62.2    54   344  4.35  4.37  2.71 sold
##  7 0.2   Prem~ E     SI2      60.2    62   345  3.79  3.75  2.27 sold
##  8 0.32  Prem~ E     I1       60.9    58   345  4.38  4.42  2.68 sold
##  9 0.3   Ideal I     SI2      62      54   348  4.31  4.34  2.68 unso~
## 10 0.24  Prem~ I     VS1      62.5    57   355  3.97  3.94  2.47 unso~
## # ... with 35,332 more rows, and 1 more variable: index <int>
```

```
## %in% 함수는 우측에 지정한 객체가 좌측에 포함되어 있냐는 것을 묻는 논리형의 기능을 수행한다.

## %in% 함수를 자세히 알아보자.
names_ <- c("Sara", "Robert", "James") # names_라는 객체에 세 이름이 있을 때,
"James" %in% names_                    # names_안에 James라는 이름이 있으면?
```

```
## [1] TRUE
```

dplyr::select(): 선택 함수

dplyr::select() 함수는 데이터 안에서 특정한 변수만을 선택하고자 할 때 사용할 수 있다. 데이터를 관리하고 전처리를 할 때 굉장히 유용하게 사용할 수 있는 함수이다. 예를 들어, World Development Indicators에서 전체 변수 중 필요한 변수만을 선택하여 새로운 데이터로 재지정할 수 있는 것이다.

```
df %>% select(carat, color, x)            # df 데이터 중 carat ,color, x 변수만 뽑아내라.
```

```
## # A tibble: 53,940 x 3
##    carat color     x
##    <dbl> <chr> <dbl>
##  1 0.23  E      3.95
##  2 0.21  E      3.89
##  3 0.23  E      4.05
##  4 0.290 I      4.2
##  5 0.31  J      4.34
##  6 0.24  J      3.94
##  7 0.24  I      3.95
##  8 0.26  H      4.07
##  9 0.22  E      3.87
## 10 0.23  H      4
## # ... with 53,930 more rows
```

```
df %>% select(-carat, -color, x)          # carat과 color를 제외한 나머지 변수들만 뽑아내라.
```

```
## # A tibble: 53,940 x 10
##    cut       clarity depth table price     x     y     z sold   index
##    <chr>     <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>  <int>
##  1 Ideal     SI2      61.5    55   326  3.95  3.98  2.43 unsold     1
##  2 Premium   SI1      59.8    61   326  3.89  3.84  2.31 unsold     2
##  3 Good      VS1      56.9    65   327  4.05  4.07  2.31 unsold     3
```

```
##  4 Premium   VS2      62.4    58   334  4.2   4.23  2.63 unsold     4
##  5 Good      SI2      63.3    58   335  4.34  4.35  2.75 sold       5
##  6 Very Good VVS2     62.8    57   336  3.94  3.96  2.48 unsold     6
##  7 Very Good VVS1     62.3    57   336  3.95  3.98  2.47 unsold     7
##  8 Very Good SI1      61.9    55   337  4.07  4.11  2.53 unsold     8
##  9 Fair      VS2      65.1    61   337  3.87  3.78  2.49 sold       9
## 10 Very Good VS1      59.4    61   338  4     4.05  2.39 sold      10
## # ... with 53,930 more rows
```

```
df %>% select(depth, price, everything()) # 변수의 순서 정리: depth, price, 나머지는 그대로.
```

```
## # A tibble: 53,940 x 12
##    depth price carat cut   color clarity table     x     y     z sold
##    <dbl> <dbl> <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <chr>
##  1  61.5   326 0.23  Ideal E     SI2        55  3.95  3.98  2.43 unso~
##  2  59.8   326 0.21  Prem~ E     SI1        61  3.89  3.84  2.31 unso~
##  3  56.9   327 0.23  Good  E     VS1        65  4.05  4.07  2.31 unso~
##  4  62.4   334 0.290 Prem~ I     VS2        58  4.2   4.23  2.63 unso~
##  5  63.3   335 0.31  Good  J     SI2        58  4.34  4.35  2.75 sold
##  6  62.8   336 0.24  Very~ J     VVS2       57  3.94  3.96  2.48 unso~
##  7  62.3   336 0.24  Very~ I     VVS1       57  3.95  3.98  2.47 unso~
##  8  61.9   337 0.26  Very~ H     SI1        55  4.07  4.11  2.53 unso~
##  9  65.1   337 0.22  Fair  E     VS2        61  3.87  3.78  2.49 sold
## 10  59.4   338 0.23  Very~ H     VS1        61  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
df %>% select_if(is.character)            # 문자열인 변수들만 남겨라.
```

```
## # A tibble: 53,940 x 4
##    cut       color clarity sold
##    <chr>     <chr> <chr>   <chr>
##  1 Ideal     E     SI2     unsold
##  2 Premium   E     SI1     unsold
##  3 Good      E     VS1     unsold
##  4 Premium   I     VS2     unsold
##  5 Good      J     SI2     sold
##  6 Very Good J     VVS2    unsold
##  7 Very Good I     VVS1    unsold
##  8 Very Good H     SI1     unsold
##  9 Fair      E     VS2     sold
## 10 Very Good H     VS1     sold
## # ... with 53,930 more rows
```

위의 예제에서 눈여겨볼 만한 것은 바로 세 번째 `select()` 함수 내에서 작동하는 `everything()`함수와 `select_if()`라는 변형 함수이다.

- 만약 `everything()` 함수가 없었다면 변수들의 이름을 줄줄이 나열해야 해서 `select()` 함수의 효용이 많이 떨어졌을 것이다.
- 그리고 `select_if()`는 조건문을 반영할 수 있다.
- `-`를 통해서 변수를 제외하는 여집합적 구성도 가능하다(`carat`과 `color`만 제외하는 것처럼)

`select()` 함수를 조금 더 자세하게 알아보자.

```
## 인덱싱 기능을 이용하여 열번(number of columns)을 이용해 select()를 활용해보자.
df %>% select(1:3)          # 첫 번째부터 세 번째 변수만을 선택해 뽑아내라.
```

```
## # A tibble: 53,940 x 3
##    carat cut       color
```

```
##    <dbl> <chr>     <chr>
##  1 0.23  Ideal     E
##  2 0.21  Premium   E
##  3 0.23  Good      E
##  4 0.290 Premium   I
##  5 0.31  Good      J
##  6 0.24  Very Good J
##  7 0.24  Very Good I
##  8 0.26  Very Good H
##  9 0.22  Fair      E
## 10 0.23  Very Good H
## # ... with 53,930 more rows
```

```
df %>% select(-c(1, 2, 6)) # 첫 번째, 두 번째, 여섯 번째 변수를 제외하고 나머지를 뽑아내라.
```

```
## # A tibble: 53,940 x 9
##    color clarity depth price     x     y     z sold   index
##    <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>  <int>
##  1 E     SI2      61.5   326  3.95  3.98  2.43 unsold     1
##  2 E     SI1      59.8   326  3.89  3.84  2.31 unsold     2
##  3 E     VS1      56.9   327  4.05  4.07  2.31 unsold     3
##  4 I     VS2      62.4   334  4.2   4.23  2.63 unsold     4
##  5 J     SI2      63.3   335  4.34  4.35  2.75 sold       5
##  6 J     VVS2     62.8   336  3.94  3.96  2.48 unsold     6
##  7 I     VVS1     62.3   336  3.95  3.98  2.47 unsold     7
##  8 H     SI1      61.9   337  4.07  4.11  2.53 unsold     8
##  9 E     VS2      65.1   337  3.87  3.78  2.49 sold       9
## 10 H     VS1      59.4   338  4     4.05  2.39 sold      10
## # ... with 53,930 more rows
```

```
## select() 함수는 범용성이 높다. 변수명이 어떤 글자로 시작하는지, 끝나는지,
## 혹은 어떤 글자를 포함하는지에 따라서도 select()를 적용하여 변수를 뽑아낼 수 있다.
df %>% select(starts_with("c"))
```

```
## # A tibble: 53,940 x 4
##    carat cut       color clarity
##    <dbl> <chr>     <chr> <chr>
##  1 0.23  Ideal     E     SI2
##  2 0.21  Premium   E     SI1
##  3 0.23  Good      E     VS1
##  4 0.290 Premium   I     VS2
##  5 0.31  Good      J     SI2
##  6 0.24  Very Good J     VVS2
##  7 0.24  Very Good I     VVS1
##  8 0.26  Very Good H     SI1
##  9 0.22  Fair      E     VS2
## 10 0.23  Very Good H     VS1
## # ... with 53,930 more rows
```

```
df %>% select(ends_with("y"))
```

```
## # A tibble: 53,940 x 2
##    clarity     y
##    <chr>   <dbl>
##  1 SI2      3.98
##  2 SI1      3.84
```

```
##  3 VS1      4.07
##  4 VS2      4.23
##  5 SI2      4.35
##  6 VVS2     3.96
##  7 VVS1     3.98
##  8 SI1      4.11
##  9 VS2      3.78
## 10 VS1      4.05
## # ... with 53,930 more rows
```

```
df %>% select(contains("olo"))
```

```
## # A tibble: 53,940 x 1
##    color
##    <chr>
##  1 E
##  2 E
##  3 E
##  4 I
##  5 J
##  6 J
##  7 I
##  8 H
##  9 E
## 10 H
## # ... with 53,930 more rows
```

```
## 마찬가지로 제외하는 함수 (-)도 적용된다.
df %>% select(-contains("olo"))
```

```
## # A tibble: 53,940 x 11
##    carat cut       clarity depth table price     x     y     z sold   index
##    <dbl> <chr>     <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>  <int>
##  1 0.23  Ideal     SI2      61.5    55   326  3.95  3.98  2.43 unsold     1
##  2 0.21  Premium   SI1      59.8    61   326  3.89  3.84  2.31 unsold     2
##  3 0.23  Good      VS1      56.9    65   327  4.05  4.07  2.31 unsold     3
##  4 0.290 Premium   VS2      62.4    58   334  4.2   4.23  2.63 unsold     4
##  5 0.31  Good      SI2      63.3    58   335  4.34  4.35  2.75 sold       5
##  6 0.24  Very Good VVS2     62.8    57   336  3.94  3.96  2.48 unsold     6
##  7 0.24  Very Good VVS1     62.3    57   336  3.95  3.98  2.47 unsold     7
##  8 0.26  Very Good SI1      61.9    55   337  4.07  4.11  2.53 unsold     8
##  9 0.22  Fair      VS2      65.1    61   337  3.87  3.78  2.49 sold       9
## 10 0.23  Very Good VS1      59.4    61   338  4     4.05  2.39 sold      10
## # ... with 53,930 more rows
```

`select()` 함수를 이용해서 변수를 선택-추출해내는 것 외에도 변수 이름을 변경하는 것도 가능하다. 단, 이때 `everything()` 함수를 지정해주는 것을 잊어는 안 된다. 왜냐하면 `everything()` 없이 변수명만 바꿔버리면 `select()`는 바꾼 그 변수들만을 출력하고 나머지 변수들은 제외해버리기 때문이다.

물론 그렇게 `select()` 함수를 적용하고 별도로 저장하지 않으면 `df` 자체에는 변화가 없기 때문에 다시 `everything()`을 추가해서 코드를 작동시키고 다른 객체로 저장하면 된다.

- 그러면 바뀐 변수 + 바꾸지 않은 다른 변수들이 `new_df` 라던지 다른 객체 이름으로 저장될 것이다.
- 그리고 이때, 바뀐 함수들이 먼저 오고 그 다음으로 다른 변수들이 순서대로 붙게 된다.

```
df %>% select(new_depth = depth, new_color = color, everything())  # 새 변수 + 기존 변수
```

```
## # A tibble: 53,940 x 12
```

```
##    new_depth new_color carat cut   clarity table price     x     y     z
##        <dbl> <chr>     <dbl> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1      61.5 E         0.23  Ideal SI2        55   326  3.95  3.98  2.43
##  2      59.8 E         0.21  Prem~ SI1        61   326  3.89  3.84  2.31
##  3      56.9 E         0.23  Good  VS1        65   327  4.05  4.07  2.31
##  4      62.4 I         0.290 Prem~ VS2        58   334  4.2   4.23  2.63
##  5      63.3 J         0.31  Good  SI2        58   335  4.34  4.35  2.75
##  6      62.8 J         0.24  Very~ VVS2       57   336  3.94  3.96  2.48
##  7      62.3 I         0.24  Very~ VVS1       57   336  3.95  3.98  2.47
##  8      61.9 H         0.26  Very~ SI1        55   337  4.07  4.11  2.53
##  9      65.1 E         0.22  Fair  VS2        61   337  3.87  3.78  2.49
## 10      59.4 H         0.23  Very~ VS1        61   338  4     4.05  2.39
## # ... with 53,930 more rows, and 2 more variables: sold <chr>, index <int>
```

```
df %>% select(everything(), new_depth = depth, new_color = color)  # 기존 변수 + 새 변수
```

```
## # A tibble: 53,940 x 12
##    carat cut   new_color clarity new_depth table price     x     y     z
##    <dbl> <chr> <chr>     <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 0.23  Ideal E         SI2          61.5    55   326  3.95  3.98  2.43
##  2 0.21  Prem~ E         SI1          59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good  E         VS1          56.9    65   327  4.05  4.07  2.31
##  4 0.290 Prem~ I         VS2          62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good  J         SI2          63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very~ J         VVS2         62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very~ I         VVS1         62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very~ H         SI1          61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair  E         VS2          65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very~ H         VS1          59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows, and 2 more variables: sold <chr>, index <int>
```

```
## select()로도 변수명을 바꿀 수 있지만, rename()을 이용하면 굳이 everything() 안쓰고도
## 간단하게 할 수 있다. 역시 편법은 쓰는 게 아니다.
df %>% rename(new_depth = depth, new_color = color)
```

```
## # A tibble: 53,940 x 12
##    carat cut   new_color clarity new_depth table price     x     y     z
##    <dbl> <chr> <chr>     <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 0.23  Ideal E         SI2          61.5    55   326  3.95  3.98  2.43
##  2 0.21  Prem~ E         SI1          59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good  E         VS1          56.9    65   327  4.05  4.07  2.31
##  4 0.290 Prem~ I         VS2          62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good  J         SI2          63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very~ J         VVS2         62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very~ I         VVS1         62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very~ H         SI1          61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair  E         VS2          65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very~ H         VS1          59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows, and 2 more variables: sold <chr>, index <int>
```

```
## 한 번에 모든 변수들의 이름을 일괄적으로 변경하기
df %>% rename_all(function(x) str_c(x, "_new"))
```

```
## # A tibble: 53,940 x 12
##    carat_new cut_new color_new clarity_new depth_new table_new price_new
##        <dbl> <chr>   <chr>     <chr>           <dbl>     <dbl>     <dbl>
```

```
##  1     0.23  Ideal   E         SI2              61.5        55       326
##  2     0.21  Premium E         SI1              59.8        61       326
##  3     0.23  Good    E         VS1              56.9        65       327
##  4     0.290 Premium I         VS2              62.4        58       334
##  5     0.31  Good    J         SI2              63.3        58       335
##  6     0.24  Very G~ J         VVS2             62.8        57       336
##  7     0.24  Very G~ I         VVS1             62.3        57       336
##  8     0.26  Very G~ H         SI1              61.9        55       337
##  9     0.22  Fair    E         VS2              65.1        61       337
## 10     0.23  Very G~ H         VS1              59.4        61       338
## # ... with 53,930 more rows, and 5 more variables: x_new <dbl>,
## #   y_new <dbl>, z_new <dbl>, sold_new <chr>, index_new <int>
```

```
df %>% rename_all(function(x) str_to_upper(x))
```

```
## # A tibble: 53,940 x 12
##    CARAT CUT   COLOR CLARITY DEPTH TABLE PRICE     X     Y     Z SOLD
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: INDEX <int>
```

```
## 특정한 조건을 가진 변수들만 이름을 변경하기rename only certain variables
df %>% rename_at(              # rename_at()으로 조건을 특정한다.
  vars( starts_with("c") ),    # 변수들을 대상으로 하되, "c"로 변수명이 시작하는
  function(x) str_to_upper(x) # 어떻게 바꾼다? 모두 변수명을 대문자로 바꾼다.
)
```

```
## # A tibble: 53,940 x 12
##    CARAT CUT   COLOR CLARITY depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
## rename_if() 함수를 이용하면 특정 조건을 충족하는 변수들의 이름을 변경할 수 있다.
df %>% rename_if( is.numeric, str_to_upper ) # 숫자형인 변수들의 이름을 대문자로 바꿔라.
```

```
## # A tibble: 53,940 x 12
##    CARAT cut   color clarity DEPTH TABLE PRICE     X     Y     Z sold
```

```
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: INDEX <int>
```

```
## 이 경우 문자형 값을 가지는 color, cut 등은 변수명이 바뀌지 않는 것을 확인할 수 있다.
```

dplyr::mutate(): 변수 조작 함수

dplyr::mutate()는 데이터 전처리 및 관리에서 가장 요긴하게 쓰일 함수이다. 이 함수는 새로운 변수를 만들거나 기존 변수에 조작을 가할 때 사용한다.

```
df %>% mutate(carat_multiplied = carat * 10) # 기존 carat 변수에 10배가 된 값을 가진
```

```
## # A tibble: 53,940 x 13
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 2 more variables: index <int>,
## #   carat_multiplied <dbl>
```

```
                                        # 새로운 변수 carat_multiplied를 만들어라.
df %>% mutate(carat = carat * 10)       # 기존 carat 변수의 값에 10배를 곱하라.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1   2.3 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2   2.1 Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3   2.3 Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4   2.9 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5   3.1 Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6   2.4 Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7   2.4 Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8   2.6 Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9   2.2 Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10   2.3 Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
## 첫 번째 코딩과 두 번째 코딩의 차이점은 첫 번째 코딩은 기존 변수를 이용해
## 새 변수를 만든 것이고, 두 번째 코딩은 기존 변수 자체의 값을 바꾸어 버린 것이다.
```

하나의 mutate() 함수 내부에 여러 줄의 멀티코드를 통해서 순서대로 변수를 조작할 수 있다.

```
df %>% select(carat) %>%
  mutate(
    caret_times_2 = carat * 2,
    caret_times_2_times_2 = caret_times_2 * 2,
    caret_times_2_times_2_times_3 = caret_times_2_times_2 * 3
  )
```

```
## # A tibble: 53,940 x 4
##    carat caret_times_2 caret_times_2_times_2 caret_times_2_times_2_times_3
##    <dbl>         <dbl>                 <dbl>                         <dbl>
##  1 0.23          0.46                   0.92                          2.76
##  2 0.21          0.42                   0.84                          2.52
##  3 0.23          0.46                   0.92                          2.76
##  4 0.290         0.580                  1.16                          3.48
##  5 0.31          0.62                   1.24                          3.72
##  6 0.24          0.48                   0.96                          2.88
##  7 0.24          0.48                   0.96                          2.88
##  8 0.26          0.52                   1.04                          3.12
##  9 0.22          0.44                   0.88                          2.64
## 10 0.23          0.46                   0.92                          2.76
## # ... with 53,930 more rows
```

```
## mutate() 함수 역시 _at, _all, _if의 세부함수를 가진다.
df %>% mutate_if(is.character, factor)  # 변수가 문자형이거든 요인형으로 바꾸어라.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <fct> <fct> <fct>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
df %>% mutate_at(vars( color, clarity ), factor) # color, clarity 변수를 요인형으로 바꾸어라.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <fct> <fct>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43 unso~
##  2 0.21  Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31 unso~
##  3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31 unso~
##  4 0.290 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63 unso~
##  5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75 sold
##  6 0.24  Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48 unso~
```

```
##  7 0.24  Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47 unso~
##  8 0.26  Very~ H     SI1      61.9    55   337  4.07  4.11  2.53 unso~
##  9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49 sold
## 10 0.23  Very~ H     VS1      59.4    61   338  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
df %>% mutate_all(as.character) # 모든 변수들을 문자형으로 바꾸어라.
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price x     y     z     sold
##    <chr> <chr> <chr> <chr>   <chr> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 0.23  Ideal E     SI2     61.5  55    326   3.95  3.98  2.43  unso~
##  2 0.21  Prem~ E     SI1     59.8  61    326   3.89  3.84  2.31  unso~
##  3 0.23  Good  E     VS1     56.9  65    327   4.05  4.07  2.31  unso~
##  4 0.29  Prem~ I     VS2     62.4  58    334   4.2   4.23  2.63  unso~
##  5 0.31  Good  J     SI2     63.3  58    335   4.34  4.35  2.75  sold
##  6 0.24  Very~ J     VVS2    62.8  57    336   3.94  3.96  2.48  unso~
##  7 0.24  Very~ I     VVS1    62.3  57    336   3.95  3.98  2.47  unso~
##  8 0.26  Very~ H     SI1     61.9  55    337   4.07  4.11  2.53  unso~
##  9 0.22  Fair  E     VS2     65.1  61    337   3.87  3.78  2.49  sold
## 10 0.23  Very~ H     VS1     59.4  61    338   4     4.05  2.39  sold
## # ... with 53,930 more rows, and 1 more variable: index <chr>
```

다른 dplyr 패키지의 유용한 함수들

arrange(): 변수의 값을 정렬할 때 쓰는 함수이다.

df 데이터에서 price라는 함수 + 나머지 다른 함수로 순서를 재정리하고, 그 다음에 price를 기준으로 변수를 정렬해보도록 하겠다. arrange()의 디폴트 값은 오름차순이다.

내림차순으로 바꾸고 싶으면 desc() 함수를 사용하면 된다.

```
df %>% select(price, everything()) %>% arrange(price) # price 기준으로 오름차순 정렬
```

```
## # A tibble: 53,940 x 12
##    price carat cut   color clarity depth table     x     y     z sold
##    <dbl> <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1   326 0.23  Ideal E     SI2      61.5    55  3.95  3.98  2.43 unso~
##  2   326 0.21  Prem~ E     SI1      59.8    61  3.89  3.84  2.31 unso~
##  3   327 0.23  Good  E     VS1      56.9    65  4.05  4.07  2.31 unso~
##  4   334 0.290 Prem~ I     VS2      62.4    58  4.2   4.23  2.63 unso~
##  5   335 0.31  Good  J     SI2      63.3    58  4.34  4.35  2.75 sold
##  6   336 0.24  Very~ J     VVS2     62.8    57  3.94  3.96  2.48 unso~
##  7   336 0.24  Very~ I     VVS1     62.3    57  3.95  3.98  2.47 unso~
##  8   337 0.26  Very~ H     SI1      61.9    55  4.07  4.11  2.53 unso~
##  9   337 0.22  Fair  E     VS2      65.1    61  3.87  3.78  2.49 sold
## 10   338 0.23  Very~ H     VS1      59.4    61  4     4.05  2.39 sold
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
df %>% select(price, everything()) %>% arrange(desc(price)) # price 기준 내림차순 ㅈㅇ렬
```

```
## # A tibble: 53,940 x 12
##    price carat cut   color clarity depth table     x     y     z sold
##    <dbl> <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 18823  2.29 Prem~ I     VS2      60.8    60  8.5   8.47  5.16 sold
##  2 18818  2    Very~ G     SI1      63.5    56  7.9   7.97  5.04 unso~
```

```
##  3 18806  1.51 Ideal G     IF       61.7    55  7.37  7.41  4.56 sold
##  4 18804  2.07 Ideal G     SI2      62.5    55  8.2   8.13  5.11 unso~
##  5 18803  2    Very~ H     SI1      62.8    57  7.95  8     5.01 sold
##  6 18797  2.29 Prem~ I     SI1      61.8    59  8.52  8.45  5.24 sold
##  7 18795  2.04 Prem~ H     SI1      58.1    60  8.37  8.28  4.84 sold
##  8 18795  2    Prem~ I     VS1      60.8    59  8.13  8.02  4.91 sold
##  9 18791  1.71 Prem~ F     VS2      62.3    59  7.57  7.53  4.7  unso~
## 10 18791  2.15 Ideal G     SI2      62.6    54  8.29  8.35  5.21 unso~
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
df %>% arrange(color, cut, desc(price))  # color, cut을 맨 앞으로 빼고 전체 변수는
```

```
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  2.02 Fair  D     SI1      65      55 16386  7.94  7.84  5.13 sold
##  2  2.01 Fair  D     SI2      66.9    57 16086  7.87  7.76  5.23 unso~
##  3  3.4  Fair  D     I1       66.8    52 15964  9.42  9.34  6.27 sold
##  4  2.01 Fair  D     SI2      59.4    66 15627  8.2   8.17  4.86 unso~
##  5  2    Fair  D     SI1      64.8    58 15540  7.98  7.89  5.14 sold
##  6  1.51 Fair  D     VS2      66      57 15152  7.17  7.08  4.7  unso~
##  7  2.07 Fair  D     SI2      64.5    60 13016  7.99  7.88  5.12 sold
##  8  1.5  Fair  D     VS2      65.4    53 12606  7.12  7.1   4.65 sold
##  9  1    Fair  D     VVS1     56.7    68 10752  6.66  6.64  3.77 unso~
## 10  1    Fair  D     VVS2     61.1    57 10562  6.37  6.3   3.87 unso~
## # ... with 53,930 more rows, and 1 more variable: index <int>
```

```
                                          # price 기준으로 내림차순 정렬
## 바로 위의 코딩은 관심있는 주요 변수를 맨 앞으로 빼고 주요 변수들이 다른 변수(price)의
## 크기에 따라 어떻게 나타나는지를 파악할 수 있게 해주는 코드이다.
## select() 사용하지 않고 바로 arrange()를 적용하였다.
## 예를 들어, 정치체제(민주주의/비민주주의) 변수를 앞으로 빼고 정렬 기준을 GDPpc 로 하는 등
## 응용이 가능하다.
```

group_by(): 집단별 묶음

group_by()를 쓰면 함수 내의 같은 변수값별로 묶인 결과를 보여준다. 숫자형, 문자형 모두 적용된다. 즉, 만약 group_by(price)로 하면 변수들이 같은 가격별로 묶여서 보일 것이고, 아래와 같이 group_by(cut)을 한다면 다이아몬드 컷팅 유형별로 분류해서 보여준다.

유의할 점은 먼저 group_by()를 지정해주고 그 이후에 다른 함수를 사용하면 집단별로 묶인 상태에서 그 함수들이 순차적으로 적용된다는 점이다. group_by()를 사용했을 때와 그렇지 않을 때를 구분해보자. 내림차순된 가격 변수를 기준으로 첫째 행과 둘째 행, 즉 가장 비싼 가격과 두 번째로 비싼 가격만을 잘라서(slice(1 : 2)) 보여주라는 명령어이다.

```
df %>% group_by(cut) %>%      # df의 cut 변수 유형별로 묶은 것이다.
  arrange(desc(price)) %>%    # 그렇게 묶인 데이터가 파이프로 넘어가고, 가격 기준 내림차순 정렬
  slice(1 : 2)                # 컷팅 유형별 + 가격 기준 내림차순 중 첫 두 행만 보여주라는 코드
```

```
## # A tibble: 10 x 12
## # Groups:   cut [5]
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  2.01 Fair  G     SI1      70.6    64 18574  7.43  6.64  4.69 sold
##  2  2.02 Fair  H     VS2      64.5    57 18565  8     7.95  5.14 sold
##  3  2.8  Good  G     SI2      63.8    58 18788  8.9   8.85  0    sold
##  4  2.07 Good  I     VS2      61.8    61 18707  8.12  8.16  5.03 sold
```

```
##  5  1.51 Ideal G     IF       61.7    55 18806  7.37  7.41  4.56 sold
##  6  2.07 Ideal G     SI2      62.5    55 18804  8.2   8.13  5.11 unso~
##  7  2.29 Prem~ I     VS2      60.8    60 18823  8.5   8.47  5.16 sold
##  8  2.29 Prem~ I     SI1      61.8    59 18797  8.52  8.45  5.24 sold
##  9  2    Very~ G     SI1      63.5    56 18818  7.9   7.97  5.04 unso~
## 10  2    Very~ H     SI1      62.8    57 18803  7.95  8     5.01 sold
## # ... with 1 more variable: index <int>
```

```
## group_by()를 사용하지 않았을 때와 비교해보자.
df %>% arrange(desc(price)) %>% slice(1:2)
```

```
## # A tibble: 2 x 12
##   carat cut   color clarity depth table price     x     y     z sold  index
##   <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <int>
## 1  2.29 Prem~ I     VS2      60.8    60 18823   8.5  8.47  5.16 sold  27750
## 2  2    Very~ G     SI1      63.5    56 18818   7.9  7.97  5.04 unso~ 27749
```

```
## 이 경우는 전체 df 데이터에서 가격 순으로 1, 2위의 값만 갖게 된다.
## cut 변수는 반영되지 않는다.
```

한 가지 유의해야할 점은 티블 유형에 group_by()를 적용할 경우 그 결과가 일반 티블과는 다른 특성을 가지게 된다는 것이다.

```
df_group <- df %>% group_by(cut) %>%
  arrange(desc(price)) %>%
  slice(1:2)
class(df_group)
```

```
## [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

```
df_group
```

```
## # A tibble: 10 x 12
## # Groups:   cut [5]
##    carat cut   color clarity depth table price     x     y     z sold
##    <dbl> <chr> <chr> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  2.01 Fair  G     SI1      70.6    64 18574  7.43  6.64  4.69 sold
##  2  2.02 Fair  H     VS2      64.5    57 18565  8     7.95  5.14 sold
##  3  2.8  Good  G     SI2      63.8    58 18788  8.9   8.85  0    sold
##  4  2.07 Good  I     VS2      61.8    61 18707  8.12  8.16  5.03 sold
##  5  1.51 Ideal G     IF       61.7    55 18806  7.37  7.41  4.56 sold
##  6  2.07 Ideal G     SI2      62.5    55 18804  8.2   8.13  5.11 unso~
##  7  2.29 Prem~ I     VS2      60.8    60 18823  8.5   8.47  5.16 sold
##  8  2.29 Prem~ I     SI1      61.8    59 18797  8.52  8.45  5.24 sold
##  9  2    Very~ G     SI1      63.5    56 18818  7.9   7.97  5.04 unso~
## 10  2    Very~ H     SI1      62.8    57 18803  7.95  8     5.01 sold
## # ... with 1 more variable: index <int>
```

보면 "grouped_df"라는 특성이 추가된 것을 확인할 수 있다. 티블과 group_by()를 함께 쓸 때는 ungroup() 함수를 같이 사용할 것을 추천하는데, 이는 다음과 같은 이유에서이다.

1. 미리 언급한 바와 같이 grouped_라는 속성이 생김으로써, group_by()가 야기할 수 있는 잠재적인 오류를 피하기 위해서다.
2. ungroup() 함수를 이용하여 파이프 함수로 구성된 코드가 group_by() 함수가 적용된 것임을 명시적으로 줄 수 있다. 따라서 우리는 ungroup() 함수가 코드에 포함되어 있다면 해당 티블이 그룹핑된 결과일 수 있다고 바로 알 수 있다. 단적으로 코드의 가독성과 명확성이 좋아진다.
3. 글로벌 환경을 .Rdata 객체로 저장하여 불러오거나 할 때, group_by() 해놓고 ungroup() 안하면 기록은 남아있지 않는데 해당 티블에 grouped_ 속성이 남아 추후 분석에 어려움이 있을 수 있다.

```
## 따라서 ungroup()을 이용하여 일반적인 티블로 다시 바꿔준다.
df_group <- df %>% group_by(cut) %>%
  arrange(desc(price)) %>%
  slice(1:2) %>%
  ungroup() # 원래의 티블로 돌아와!
class(df_group)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

또, dplyr 패키지는 count() 함수도 제공한다. 이 함수는 데이터의 특정 변수값에 기초해 그 집단 수를 세어 준다. 보통 분류형 변수에 많이 사용되지만 숫자형도 적용된다. 얘를 들어 1부터 2만에 이르는 범주를 가지는 변수가 총 50만개의 관측치를 가지고 있다고 할 때, 1의 값은 몇 개, 15는 몇 개, 2만은 몇 개와 같은 식으로 범주화를 시켜주는 것이다.

```
df %>% count(cut)
```

```
## # A tibble: 5 x 2
##   cut           n
##   <chr>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Ideal     21551
## 4 Premium   13791
## 5 Very Good 12082
```

```
## count()함수를 group_by() 함수로 바꾸어서 표현하면 아래와 같다.
df %>% group_by(cut) %>% summarise(n = n())
```

```
## # A tibble: 5 x 2
##   cut           n
##   <chr>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Ideal     21551
## 4 Premium   13791
## 5 Very Good 12082
```

```
## group_by() 함수는 summarise() 함수와 결합될 경우 다양한 응용이 가능하다.
## 여기서 summarise는 총계를 구하라는 것이 아니라 데이터를 요약정리해서 보여줄 수 있는
## 여러 함수들을 통칭하는 것이다.
df %>%
  group_by(cut) %>%
  summarise(price_mean = mean(price)) # 컷팅 유형별로 평균 가격을 계산하라.
```

```
## # A tibble: 5 x 2
##   cut       price_mean
##   <chr>          <dbl>
## 1 Fair           4359.
## 2 Good           3929.
## 3 Ideal          3458.
## 4 Premium        4584.
## 5 Very Good      3982.
```

마찬가지로 summarise() 함수도 _if, _at, _all과 같은 세부유형으로 분류하여 사용할 수 있다.

```
## cut, x, y, z 변수만 df 티블에서 뽑아내어 cut 유형별로 그룹화. 그리고 각 컷팅유형 별로
## x, y, z의 평균을 구하라.
df %>%
```

```
  select(cut, x, y, z) %>%
  group_by(cut) %>%
  summarise_all(mean, na.rm = T)
```

```
## # A tibble: 5 x 4
##   cut           x     y     z
##   <chr>     <dbl> <dbl> <dbl>
## 1 Fair       6.25  6.18  3.98
## 2 Good       5.84  5.85  3.64
## 3 Ideal      5.51  5.52  3.40
## 4 Premium    5.97  5.94  3.65
## 5 Very Good  5.74  5.77  3.56
```

```
## 평균에 더하여 중앙값, 최소값, 최대값도 구해보자.
df %>% group_by(cut) %>%
  summarise_at(             # 특정한 변수인 x, y, z를 대상으로
    vars(x, y, z),          # list 뒤의 함수들을 적용하라.
    list(mean, median, min, max),
    na.rm = T)              # mean 등은 데이터에 결측치가 있으면 결측치를 반환하므로
```

```
## # A tibble: 5 x 13
##   cut     x_fn1 y_fn1 z_fn1 x_fn2 y_fn2 z_fn2 x_fn3 y_fn3 z_fn3 x_fn4 y_fn4
##   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Fair     6.25  6.18  3.98  6.18  6.1   3.97     0     0     0 10.7  10.5
## 2 Good     5.84  5.85  3.64  5.98  5.99  3.7      0     0     0  9.44  9.38
## 3 Ideal    5.51  5.52  3.40  5.25  5.26  3.23     0     0     0  9.65 31.8
## 4 Premium  5.97  5.94  3.65  6.11  6.06  3.72     0     0     0 10.1  58.9
## 5 Very G~  5.74  5.77  3.56  5.74  5.77  3.56     0     0     0 10.0   9.94
## # ... with 1 more variable: z_fn4 <dbl>
```

```
                          # 결측치 제거(remove na)가 TRUE이도록 설정한다.

## 컷팅 유형별로 그룹화한 다음에 숫자형 변수들일 경우에만 평균을 계산하라.
df %>% group_by(cut) %>%
  summarise_if(is.numeric, mean, na.rm = T)
```

```
## # A tibble: 5 x 9
##   cut       carat depth table price     x     y     z  index
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 Fair      1.05   64.0  59.1 4359.  6.25  6.18  3.98 24147.
## 2 Good      0.849  62.4  58.7 3929.  5.84  5.85  3.64 24775.
## 3 Ideal     0.703  61.7  56.0 3458.  5.51  5.52  3.40 29048.
## 4 Premium   0.892  61.3  58.7 4584.  5.97  5.94  3.65 25600.
## 5 Very Good 0.806  61.8  58.0 3982.  5.74  5.77  3.56 26097.
```

```
## 동일한 코드이지만 표현식이 조금 다르다.
df %>% group_by(cut) %>%
  summarise_if(is.numeric, function(x) mean(x, na.rm = T))
```

```
## # A tibble: 5 x 9
##   cut       carat depth table price     x     y     z  index
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 Fair      1.05   64.0  59.1 4359.  6.25  6.18  3.98 24147.
## 2 Good      0.849  62.4  58.7 3929.  5.84  5.85  3.64 24775.
## 3 Ideal     0.703  61.7  56.0 3458.  5.51  5.52  3.40 29048.
## 4 Premium   0.892  61.3  58.7 4584.  5.97  5.94  3.65 25600.
```

```
## 5 Very Good 0.806  61.8  58.0 3982.  5.74  5.77  3.56 26097.
```

```
df %>% group_by(cut) %>%
  summarise_if(is.numeric, ~ mean(., na.rm = T))
```

```
## # A tibble: 5 x 9
##   cut       carat depth table price     x     y     z  index
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 Fair      1.05   64.0  59.1 4359.  6.25  6.18  3.98 24147.
## 2 Good      0.849  62.4  58.7 3929.  5.84  5.85  3.64 24775.
## 3 Ideal     0.703  61.7  56.0 3458.  5.51  5.52  3.40 29048.
## 4 Premium   0.892  61.3  58.7 4584.  5.97  5.94  3.65 25600.
## 5 Very Good 0.806  61.8  58.0 3982.  5.74  5.77  3.56 26097.
```

데이터 결합하기(Join data)

연구를 진행하다보면 하나로 분석에 필요한 모든 변수가 포함된 데이터를 만나기란 하늘에 별 따기라는 것을 알 수 있다. 따라서 서로 다른 소스에서 필요한 변수들을 추출해 하나의 데이터로 구성하는, 데이터 결합 작업이 중요하다. 보통은 머징(merging)이라고도 많이 한다.

일단 미국의 주 이름 객체 반복추출(replacement)가 가능하도록 설정하고 총 250개의 관측치를 가지는 표본을 만들어보자. 변수명이 `state_name`인 티블을 하나 만들었다. 250의 관측치들은 미국의 각 주 이름이 중복되어 존재한다.

- `sample()` 함수의 `replace = T` 옵션은 상자 안에서 공을 꺼낼 때, 한 번 꺼낸 공을 다시 집어넣고 다시 꺼낼 수 있다는 것을 의미한다.
- 이렇게 반복추출된 `state_name` 변수는 미국 각 주의 이름이 무작위로 반복추출되어 총 250개의 관측치를 가지게 된다.

```
states_df <- tibble(state_name = sample(state.name, 250, replace = T))
states_df %>% count(state_name)
```

```
## # A tibble: 50 x 2
##    state_name      n
##    <chr>       <int>
##  1 Alabama         3
##  2 Alaska          6
##  3 Arizona         4
##  4 Arkansas        6
##  5 California      5
##  6 Colorado        3
##  7 Connecticut     7
##  8 Delaware        4
##  9 Florida         3
## 10 Georgia         6
## # ... with 40 more rows
```

이번에는 `state_name`와 미국 주 이름의 약자를 의미하는 `state_abb` 변수를 만들어보자. 즉, `states_table`은 미국의 50개 주의 이름과 약자의 두 변수를 가지고 있는 티블이다.

```
states_table <- tibble(
  state_name = state.name, state_abb = state.abb
)
head(states_table)
```

```
## # A tibble: 6 x 2
##   state_name state_abb
##   <chr>      <chr>
## 1 Alabama    AL
```

```
## 2 Alaska     AK
## 3 Arizona    AZ
## 4 Arkansas   AR
## 5 California CA
## 6 Colorado   CO
```

자, 이제 250개의 관측치를 갖는 state_df 티블과 50개의 관측치 값을 갖는 states_table 티블을 결합해보자. 기준은 left_join()이므로 states_df가 된다. 따라서 우리는 states_df의 모든 관측치를 유지한 채로 states_table의 관측치를 옮겨 붙일 것이다.

```
left_join(states_df, states_table) %>% print( n = 10 )
```

```
## # A tibble: 250 x 2
##    state_name     state_abb
##    <chr>          <chr>
##  1 Mississippi    MS
##  2 Virginia       VA
##  3 South Carolina SC
##  4 Indiana        IN
##  5 South Carolina SC
##  6 Pennsylvania   PA
##  7 Missouri       MO
##  8 Washington     WA
##  9 Wyoming        WY
## 10 North Dakota   ND
## # ... with 240 more rows
```

이게 가능한 이유는 두 티블 사이에 공통의 변수, state_name이 존재하기 때문이다. 이 경우는 자동으로 묶였지만 어떤 변수를 기준으로 그룹화할 것인지 지정해줄 수도 있다.

```
left_join(states_df, states_table, by = 'state_name') %>%
  print( n = 10 )
```

```
## # A tibble: 250 x 2
##    state_name     state_abb
##    <chr>          <chr>
##  1 Mississippi    MS
##  2 Virginia       VA
##  3 South Carolina SC
##  4 Indiana        IN
##  5 South Carolina SC
##  6 Pennsylvania   PA
##  7 Missouri       MO
##  8 Washington     WA
##  9 Wyoming        WY
## 10 North Dakota   ND
## # ... with 240 more rows
```

이외에도 right_join(), inner_join(), full_join(), 그리고 anti_join()과 같은 함수로 결합할 수도 있다. 자세한 내용은 tidyverse 패키지 중 결합(join)에 관한 내용에서 살펴볼 수 있다. 결합, 머징에 관한 내용은 추후 더 구체적으로 다루어볼 것이다.

일단 예시로 anti_join() 함수가 어떻게 쓰이는지 보자. anti_join()은 대개 텍스트 분석에서 사용된다.

```
text_df <- tibble(
  text = c('the fox is brown and the dog is black and the rabbit is white')
)
library(tidytext)
```

```
text_df <- text_df %>%
  unnest_tokens(word, text) # text를 어절로 분해
text_df
```

```
## # A tibble: 14 x 1
##    word
##    <chr>
##  1 the
##  2 fox
##  3 is
##  4 brown
##  5 and
##  6 the
##  7 dog
##  8 is
##  9 black
## 10 and
## 11 the
## 12 rabbit
## 13 is
## 14 white
```

```
text_df %>% anti_join(tidytext::stop_words, by = 'word') # 특정 어절은 제외하고 단어만.
```

```
## # A tibble: 6 x 1
##   word
##   <chr>
## 1 fox
## 2 brown
## 3 dog
## 4 black
## 5 rabbit
## 6 white
```