데이터 만지기

정석오

제 1 절 데이터 입출력

1.1 데이터 입력 (import)

콘솔에서 직접 자료값을 입력하려면 scan() 함수를 사용한다.

```
> x <- scan()
 1: 1
 2: 2
 3: 3
 4:
 Read 3 items
> x
[1] 1 2 3
```

텍스트 파일의 내용을 읽어들여 벡터로 저장할 때에도 scan() 함수를 사용한다.

```
> x <- scan(file = "c:/mydata/data_x.txt")
> y <- matrix(scan("c:/mydata/data_y.txt"), ncol = 3, byrow = TRUE)
```

데이터를 직사각형 형태로 정리한 텍스트 파일의 내용을 읽어들여 데이터프레임으로 저장하려면 read.table() 함수를 사용한다.

```
> x <- read.table(file = "table.txt", header = TRUE, sep = " ")
```

csv 파일을 읽어들여 데이터프레임으로 저장하려면 read.csv() 함수를 사용한다.

```
> x <- read.csv(file = "table.csv")
```

1.2 데이터 내보내기 (export)

`print()` 함수를 이용해 데이터를 콘솔에 출력할 수 있다.

```
> x <- 1:3
> print(x)

[1] 1 2 3
```

벡터로 저장된 데이터를 외부 파일로 내보내려면 'write()' 함수를 사용한다.

```
> x <- seq(from = 0, to = 1, by = 0.1)
> write(x, file = "output.txt")
```

데이터프레임, 행렬 등과 같이 직사각형 표 형태로 저장/작성된 데이터를 텍스트 파일로 내보내려면 `write.table()` 함수를, csv 파일로 내보내려면 `write.csv()` 함수를 이용한다.

```
> x <- matrix(1:20, 4, 5)
> write.table(x, file = "table.txt", row.names = FALSE)
> data(faithful)
> write.csv(faithful, file = "faithful.csv", row.names = FALSE)
```

제 2 절 간단한 데이터 조작

2.1 기준에 따라 데이터 쪼개고 붙이기

2.1.1 `split()` 함수

데이터셋을 기준 변수에 따라 여러 그룹으로 쪼개어서 새로운 데이터셋을 만들어 주는 함수이다. 아래는 `MASS` 패키지의 내장 데이터인 `Cars93`의 `MPG.city` 변수를 가지고 `Origin` 변수값(`USA`, `non-USA`)에 따라 두 개의 그룹으로 쪼개어 두 개의 성분을 갖는 리스트를 만드는 예이다.

```
> library(MASS)    # for the dataset Cars93
> summary(Cars93)

    Manufacturer     Model         Type      Min.Price         Price
 Chevrolet: 8    100    : 1   Compact:16   Min.   : 6.70   Min.   : 7.40
 Ford     : 8    190E   : 1   Large  :11   1st Qu.:10.80   1st Qu.:12.20
 Dodge    : 6    240    : 1   Midsize:22   Median :14.70   Median :17.70
 Mazda    : 5    300E   : 1   Small  :21   Mean   :17.13   Mean   :19.51
 Pontiac  : 5    323    : 1   Sporty :14   3rd Qu.:20.30   3rd Qu.:23.30
 Buick    : 4    535i   : 1   Van    : 9   Max.   :45.40   Max.   :61.90
 (Other)  :57    (Other):87
   Max.Price       MPG.city      MPG.highway                  AirBags
 Min.   : 7.9   Min.   :15.00   Min.   :20.00   Driver & Passenger:16
 1st Qu.:14.7   1st Qu.:18.00   1st Qu.:26.00   Driver only       :43
 Median :19.6   Median :21.00   Median :28.00   None              :34
 Mean   :21.9   Mean   :22.37   Mean   :29.09
 3rd Qu.:25.3   3rd Qu.:25.00   3rd Qu.:31.00
 Max.   :80.0   Max.   :46.00   Max.   :50.00

 DriveTrain  Cylinders    EngineSize      Horsepower         RPM
 4WD  :10   3     : 3   Min.   :1.000   Min.   : 55.0   Min.   :3800
 Front:67   4     :49   1st Qu.:1.800   1st Qu.:103.0   1st Qu.:4800
 Rear :16   5     : 2   Median :2.400   Median :140.0   Median :5200
            6     :31   Mean   :2.668   Mean   :143.8   Mean   :5281
            8     : 7   3rd Qu.:3.300   3rd Qu.:170.0   3rd Qu.:5750
            rotary: 1   Max.   :5.700   Max.   :300.0   Max.   :6500

  Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
 Min.   :1320   No :32          Min.   : 9.20      Min.   :2.000
 1st Qu.:1985   Yes:61          1st Qu.:14.50      1st Qu.:4.000
 Median :2340                   Median :16.40      Median :5.000
 Mean   :2332                   Mean   :16.66      Mean   :5.086
 3rd Qu.:2565                   3rd Qu.:18.80      3rd Qu.:6.000
 Max.   :3755                   Max.   :27.00      Max.   :8.000
```

```
     Length        Wheelbase         Width        Turn.circle
 Min.   :141.0   Min.   : 90.0   Min.   :60.00   Min.   :32.00
 1st Qu.:174.0   1st Qu.: 98.0   1st Qu.:67.00   1st Qu.:37.00
 Median :183.0   Median :103.0   Median :69.00   Median :39.00
 Mean   :183.2   Mean   :103.9   Mean   :69.38   Mean   :38.96
 3rd Qu.:192.0   3rd Qu.:110.0   3rd Qu.:72.00   3rd Qu.:41.00
 Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00

 Rear.seat.room   Luggage.room       Weight         Origin              Make
 Min.   :19.00   Min.   : 6.00   Min.   :1695   USA    :48   Acura Integra: 1
 1st Qu.:26.00   1st Qu.:12.00   1st Qu.:2620   non-USA:45   Acura Legend : 1
 Median :27.50   Median :14.00   Median :3040                Audi 100     : 1
 Mean   :27.83   Mean   :13.89   Mean   :3073                Audi 90      : 1
 3rd Qu.:30.00   3rd Qu.:15.00   3rd Qu.:3525                BMW 535i     : 1
 Max.   :36.00   Max.   :22.00   Max.   :4105                Buick Century: 1
 NA's   :2       NA's   :11                                  (Other)      :87
```

```
> tmp <- split(Cars93$MPG.city, Cars93$Origin)   # grouping by 'Origin'
> str(tmp)                                       # List of 2

List of 2
 $ USA    : int [1:48] 22 19 16 19 16 16 25 25 19 21 ...
 $ non-USA: int [1:45] 25 18 20 19 22 46 30 24 42 24 ...
```

2.1.2 subset() 함수

데이터프레임의 일부 행(row)를 추출할 때는 subset() 함수를 사용한다. 옵션 subset =을 이용해 추출 조건을 지정한다. select = 옵션을 이용하면 추출 대상 변수(열)를 지정할 수 있다. 다음은 내장 데이터 셋인 Cars93에서 지정한 조건에 맞는 데이터를 추출하는 예이다.

```
> subset(Cars93, subset = (MPG.city > 32))

   Manufacturer Model  Type Min.Price Price Max.Price MPG.city MPG.highway
39          Geo Metro Small       6.7   8.4      10.0       46          50
```

```
42        Honda Civic Small       8.4  12.1      15.8       42          46
80       Subaru Justy Small       7.3   8.4       9.5       33          37
83       Suzuki Swift Small       7.3   8.6      10.0       39          43
       AirBags DriveTrain Cylinders EngineSize Horsepower  RPM Rev.per.mile
39        None      Front         3        1.0         55 5700         3755
42 Driver only      Front         4        1.5        102 5900         2650
80        None        4WD         3        1.2         73 5600         2875
83        None      Front         3        1.3         70 6000         3360
   Man.trans.avail Fuel.tank.capacity Passengers Length Wheelbase Width
39             Yes               10.6          4    151        93    63
42             Yes               11.9          4    173       103    67
80             Yes                9.2          4    146        90    60
83             Yes               10.6          4    161        93    63
   Turn.circle Rear.seat.room Luggage.room Weight  Origin         Make
39          34           27.5           10   1695 non-USA    Geo Metro
42          36           28.0           12   2350 non-USA  Honda Civic
80          32           23.5           10   2045 non-USA Subaru Justy
83          34           27.5           10   1965 non-USA Suzuki Swift
```

```
> subset(Cars93, select = Model, subset = (MPG.city > 32))

   Model
39 Metro
42 Civic
80 Justy
83 Swift
```

데이터프레임의 특정 열을 제외시킬 때는 `subset()` 함수의 `select =` 옵션에 `-`을 사용한다.

```
> str(EuStockMarkets)

 Time-Series [1:1860, 1:4] from 1991 to 1999: 1629 1614 1607 1621 1618 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:4] "DAX" "SMI" "CAC" "FTSE"
```

```
> cor(EuStockMarkets)

           DAX       SMI       CAC      FTSE
DAX  1.0000000 0.9911539 0.9662274 0.9751778
SMI  0.9911539 1.0000000 0.9468139 0.9899691
CAC  0.9662274 0.9468139 1.0000000 0.9157265
FTSE 0.9751778 0.9899691 0.9157265 1.0000000

> cor(subset(EuStockMarkets, select = -SMI))  # SMI excluded

           DAX       CAC      FTSE
DAX  1.0000000 0.9662274 0.9751778
CAC  0.9662274 1.0000000 0.9157265
FTSE 0.9751778 0.9157265 1.0000000

> cor(subset(EuStockMarkets, select = -c(SMI, CAC)))  # SMI and CAC excluded

           DAX      FTSE
DAX  1.0000000 0.9751778
FTSE 0.9751778 1.0000000


> head(subset(airquality, Temp > 80, select = c(Ozone, Temp)), n = 10)

   Ozone Temp
29    45   81
35    NA   84
36    NA   85
38    29   82
39    NA   87
40    71   90
41    39   87
42    NA   93
43    NA   92
44    23   82

> subset(airquality, Day == 1, select = -Temp)
```

```
    Ozone Solar.R Wind Month Day
1      41     190  7.4     5   1
32     NA     286  8.6     6   1
62    135     269  4.1     7   1
93     39      83  6.9     8   1
124    96     167  6.9     9   1
```

```
> head(subset(airquality, select = Ozone:Wind), n = 10)
```

```
   Ozone Solar.R Wind
1     41     190  7.4
2     36     118  8.0
3     12     149 12.6
4     18     313 11.5
5     NA      NA 14.3
6     28      NA 14.9
7     23     299  8.6
8     19      99 13.8
9      8      19 20.1
10    NA     194  8.6
```

2.1.3 merge() 함수

지정한 변수를 기준으로 두 데이터프레임을 결합할 때는 merge() 함수를 이용한다. 결합 기준 변수는 by = 옵션에서 지정하는데, 변수명에 ""을 붙여야 함에 유의하자.

```
> a <- data.frame(Name = c("Mary", "Jane", "Alice", "Bianca"),
                  score = c(90, 95, 100, 100))
> b <- data.frame(Name = c("Jane", "Alice", "Ana"),
                  weight = c(70, 55, 60))
> merge(a, b, by = "Name")
```

```
   Name score weight
1 Alice   100     55
2  Jane    95     70
```

```
> merge(a, b, by = "Name", all = TRUE)

    Name score weight
1  Alice   100     55
2    Ana    NA     60
3 Bianca   100     NA
4   Jane    95     70
5   Mary    90     NA
```

2.2 apply 계열 함수

행렬 혹은 데이터프레임의 각 행과 열에 대해 같은 작업을 반복적으로 실행해 새로운 벡터를 생성하려면 apply() 함수를 이용한다. apply()의 두 번째 입력값(MARGIN)을 1로 지정하면 각 행에 대해, 2로 지정하면 각 열에 대해 반복 작업을 실행한다.

```
> a <- matrix(1:20, 4, 5)
> a

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20

> apply(a, 1, mean)    # to every row

[1]  9 10 11 12

> apply(a, 2, mean)    # to every column

[1]  2.5  6.5 10.5 14.5 18.5
```

리스트 객체의 각 성분에 대해 지정한 함수를 적용해 얻은 결과를 새로운 리스트 혹은 벡터로 만들고 싶으면 lapply() 함수, sapply() 함수를 사용한다. lapply는 리스트를, sapply는 벡터를 리턴한다. 다음은 5개의 성분을 갖는 리스트 Hong에 length() 함수를 적용해 각 성분의 벡터 길이를 알려주는 예이다.

```
> Hong <- list(first.name = "Samuel",
               age = 44,
               gender = "M",
               n.child = 2,
               child.gender = c("M", "F"))
> lapply(Hong, length)   # returns a list

$first.name
[1] 1

$age
[1] 1

$gender
[1] 1

$n.child
[1] 1

$child.gender
[1] 2

> sapply(Hong, length)   # returns a vector

  first.name          age       gender      n.child child.gender
           1            1            1            1            2
```

특정 변수에 그룹별로 함수를 적용하려면 `tapply()` 함수를 사용하면 된다. 다음은 `Cars93` 데이터의 `Price` 변수의 평균을 `Origin`별로 계산하는 예제이다.

```
> with(Cars93, tapply(Price, Origin, mean))

     USA  non-USA
18.57292 20.50889
```

데이터셋에 포함된 모든 변수에 대해 그룹별로 함수를 적용하려면 `by()` 함수를 이용한다. 아래 코드는

Cars93에 대해 Origin별로 summary() 함수를 적용한 예이다.

```
> by(Cars93, Cars93$Origin, summary)

Cars93$Origin: USA
     Manufacturer        Model          Type      Min.Price         Price
 Chevrolet : 8    Achieva   : 1   Compact: 7   Min.   : 6.90   Min.   : 7.40
 Ford      : 8    Aerostar  : 1   Large  :11   1st Qu.:11.40   1st Qu.:13.47
 Dodge     : 6    Astro     : 1   Midsize:10   Median :14.50   Median :16.30
 Pontiac   : 5    Bonneville: 1   Small  : 7   Mean   :16.54   Mean   :18.57
 Buick     : 4    Camaro    : 1   Sporty : 8   3rd Qu.:19.43   3rd Qu.:20.73
 Oldsmobile: 4    Capri     : 1   Van    : 5   Max.   :37.50   Max.   :40.10
 (Other)   :13    (Other)   :42
   Max.Price        MPG.city      MPG.highway                  AirBags
 Min.   : 7.90   Min.   :15.00   Min.   :20.00   Driver & Passenger: 9
 1st Qu.:14.97   1st Qu.:18.00   1st Qu.:26.00   Driver only       :23
 Median :18.40   Median :20.00   Median :28.00   None              :16
 Mean   :20.63   Mean   :20.96   Mean   :28.15
 3rd Qu.:24.50   3rd Qu.:23.00   3rd Qu.:30.00
 Max.   :42.70   Max.   :31.00   Max.   :41.00

 DriveTrain  Cylinders    EngineSize      Horsepower         RPM
 4WD  : 5   3     : 0   Min.   :1.300   Min.   : 63.0   Min.   :3800
 Front:34   4     :22   1st Qu.:2.200   1st Qu.:108.8   1st Qu.:4750
 Rear : 9   5     : 0   Median :3.000   Median :143.5   Median :4900
            6     :20   Mean   :3.067   Mean   :147.5   Mean   :4991
            8     : 6   3rd Qu.:3.800   3rd Qu.:170.0   3rd Qu.:5200
            rotary: 0   Max.   :5.700   Max.   :300.0   Max.   :6500

  Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
 Min.   :1320   No :26          Min.   :10.00      Min.   :2.000
 1st Qu.:1771   Yes:22          1st Qu.:15.47      1st Qu.:5.000
 Median :2035                   Median :16.45      Median :5.000
 Mean   :2119                   Mean   :17.05      Mean   :5.333
```

```
 3rd Qu.:2482                    3rd Qu.:19.05      3rd Qu.:6.000
 Max.   :3285                    Max.   :27.00      Max.   :8.000

    Length        Wheelbase         Width        Turn.circle
 Min.   :141.0   Min.   : 90.0   Min.   :63.00   Min.   :32.00
 1st Qu.:177.0   1st Qu.:101.0   1st Qu.:68.00   1st Qu.:39.00
 Median :188.5   Median :105.0   Median :71.50   Median :41.00
 Mean   :188.3   Mean   :105.7   Mean   :70.96   Mean   :40.48
 3rd Qu.:199.2   3rd Qu.:111.0   3rd Qu.:74.00   3rd Qu.:43.00
 Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00

 Rear.seat.room   Luggage.room       Weight         Origin
 Min.   :19.00   Min.   : 6.00   Min.   :1845   USA    :48
 1st Qu.:26.25   1st Qu.:13.00   1st Qu.:2705   non-USA: 0
 Median :28.00   Median :14.50   Median :3282
 Mean   :28.13   Mean   :14.88   Mean   :3195
 3rd Qu.:30.50   3rd Qu.:17.00   3rd Qu.:3639
 Max.   :36.00   Max.   :22.00   Max.   :4105
 NA's   :1       NA's   :6
               Make
 Buick Century   : 1
 Buick LeSabre   : 1
 Buick Riviera   : 1
 Buick Roadmaster: 1
 Cadillac DeVille: 1
 Cadillac Seville: 1
 (Other)         :42
------------------------------------------------------------
Cars93$Origin: non-USA
     Manufacturer     Model         Type      Min.Price         Price
 Mazda     : 5    100    : 1   Compact: 9   Min.   : 6.70   Min.   : 8.00
 Hyundai   : 4    190E   : 1   Large  : 0   1st Qu.: 9.10   1st Qu.:11.60
 Nissan    : 4    240    : 1   Midsize:12   Median :16.30   Median :19.10
```

```
Toyota    : 4    300E   : 1   Small  :14   Mean   :17.76   Mean   :20.51
Volkswagen: 4    323    : 1   Sporty : 6   3rd Qu.:22.90   3rd Qu.:26.70
Honda     : 3    535i   : 1   Van    : 4   Max.   :45.40   Max.   :61.90
(Other)   :21    (Other):39
  Max.Price        MPG.city      MPG.highway                  AirBags
Min.   : 9.10   Min.   :17.00   Min.   :21.00   Driver & Passenger: 7
1st Qu.:12.90   1st Qu.:19.00   1st Qu.:25.00   Driver only       :20
Median :21.70   Median :22.00   Median :30.00   None              :18
Mean   :23.26   Mean   :23.87   Mean   :30.09
3rd Qu.:28.50   3rd Qu.:26.00   3rd Qu.:33.00
Max.   :80.00   Max.   :46.00   Max.   :50.00

DriveTrain  Cylinders    EngineSize      Horsepower         RPM
4WD  : 5   3     : 3   Min.   :1.000   Min.   : 55.0   Min.   :4500
Front:33   4     :27   1st Qu.:1.600   1st Qu.:102.0   1st Qu.:5400
Rear : 7   5     : 2   Median :2.200   Median :135.0   Median :5600
           6     :11   Mean   :2.242   Mean   :139.9   Mean   :5590
           8     : 1   3rd Qu.:2.800   3rd Qu.:168.0   3rd Qu.:6000
           rotary: 1   Max.   :4.500   Max.   :278.0   Max.   :6500

 Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
Min.   :1955   No : 6          Min.   : 9.20      Min.   :2.000
1st Qu.:2325   Yes:39          1st Qu.:13.20      1st Qu.:4.000
Median :2505                   Median :15.90      Median :5.000
Mean   :2560                   Mean   :16.25      Mean   :4.822
3rd Qu.:2710                   3rd Qu.:18.50      3rd Qu.:5.000
Max.   :3755                   Max.   :22.50      Max.   :7.000

    Length        Wheelbase       Width        Turn.circle    Rear.seat.room
Min.   :146.0   Min.   : 90   Min.   :60.00   Min.   :32.00   Min.   :23.00
1st Qu.:170.0   1st Qu.: 97   1st Qu.:66.00   1st Qu.:36.00   1st Qu.:26.00
Median :180.0   Median :103   Median :67.00   Median :37.00   Median :27.50
Mean   :177.8   Mean   :102   Mean   :67.69   Mean   :37.33   Mean   :27.51
```

```
3rd Qu.:187.0   3rd Qu.:106   3rd Qu.:70.00   3rd Qu.:39.00   3rd Qu.:28.50
Max.   :200.0   Max.   :115   Max.   :74.00   Max.   :43.00   Max.   :35.00
                                                              NA's   :1

 Luggage.room       Weight         Origin                Make
Min.   : 8.00   Min.   :1695   USA     : 0   Acura Integra: 1
1st Qu.:11.00   1st Qu.:2475   non-USA:45   Acura Legend : 1
Median :14.00   Median :2950                Audi 100     : 1
Mean   :12.85   Mean   :2942                Audi 90      : 1
3rd Qu.:14.00   3rd Qu.:3405                BMW 535i     : 1
Max.   :17.00   Max.   :4100                Geo Metro    : 1
NA's   :5                                   (Other)      :39
```

제 3 절 Tidyverse 패키지

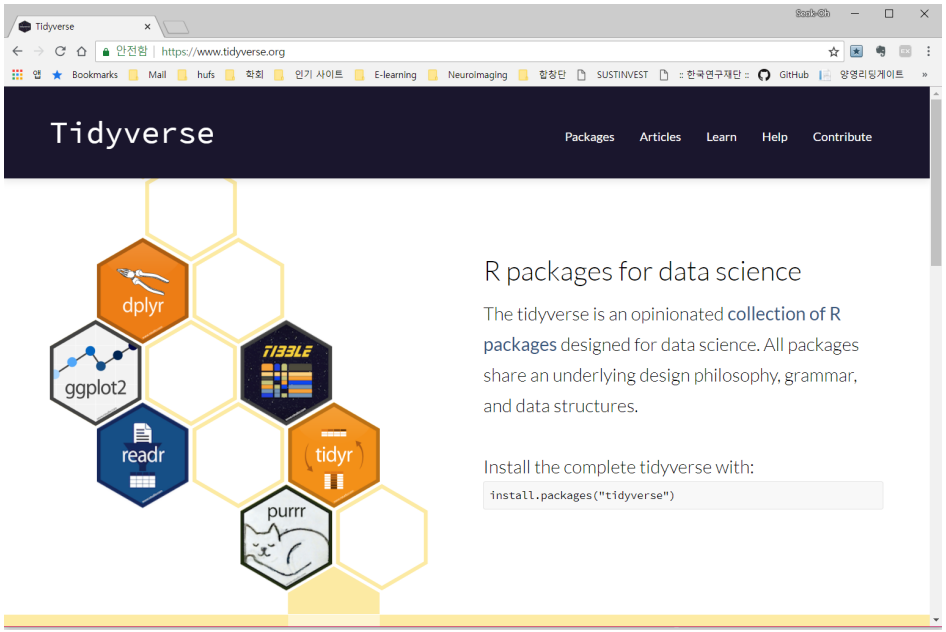


그림 3.1: tidyverse 패키지 https://www.tidyverse.org/

데이터 과학을 위해 디자인된 R-패키지의 모음으로서, 포함되어 있는 모든 패키지는 기본 디자인 철학, 문법 및 데이터 구조를 공유한다. 핵심 패키지는 다음과 같다. `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, `forcats`

```
> #install.packages("tidyverse")
> library(tidyverse)
```

제 4 절 ggplot2를 이용한 데이터시각화

이 절에서 예제 데이터로 사용할 mpg 데이터셋을 로드하자.

```
> data(mpg)
```

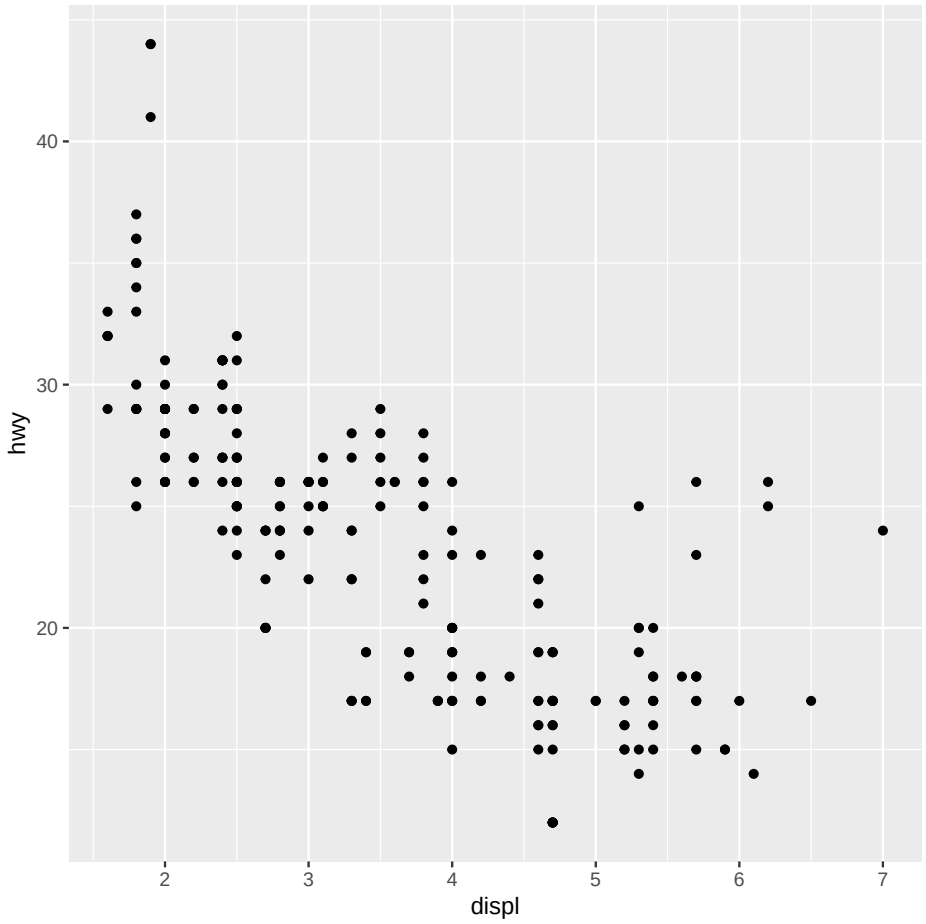
4.1 산점도

ggplot2 패키지에서 산점도를 그리는 기본 방법은 다음과 같다.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

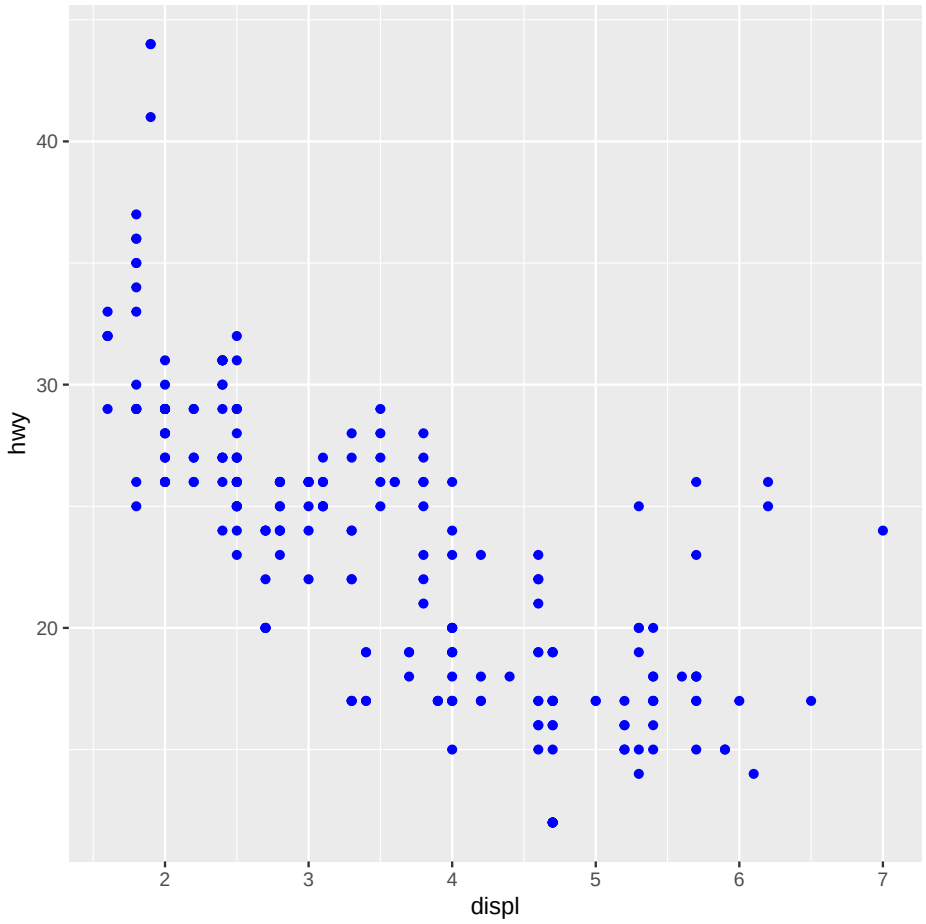
mpg 데이터셋의 배기량(displ)과 고속도로연비(hwy) 간의 산점도를 그려보자.

```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy))
```

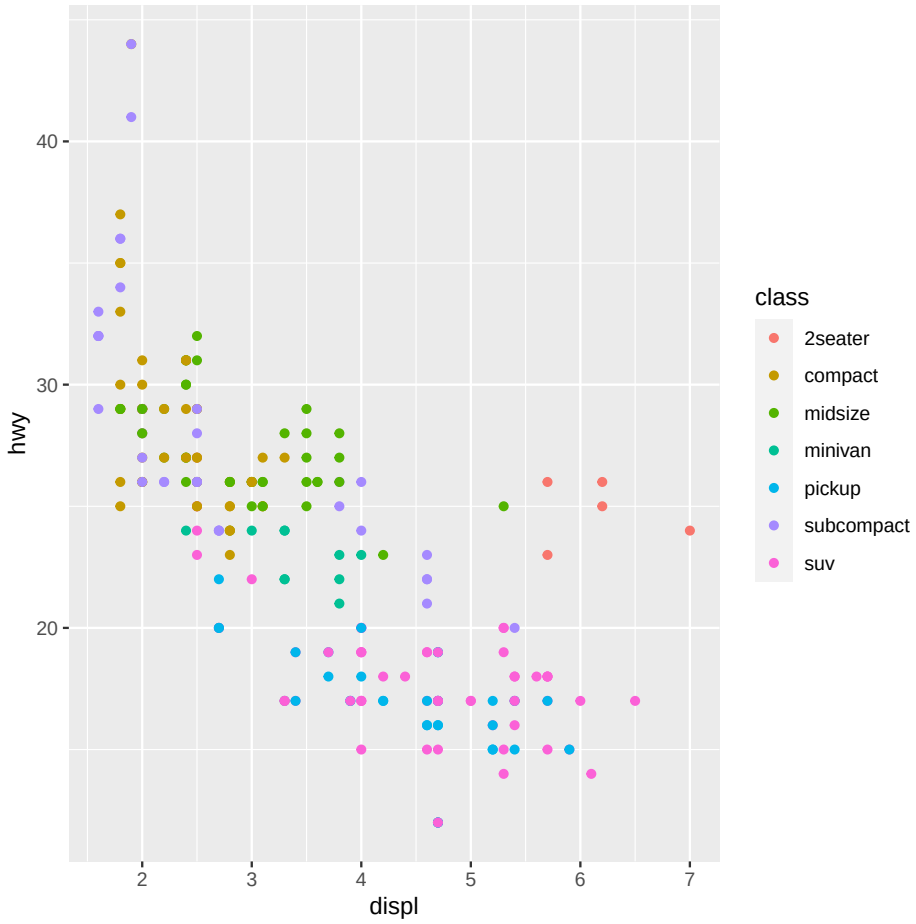


`geom_point()` 함수는 산점도를 그려주는 역할을 한다. 함수 내에 `mapping =` 인수를 이용해 시각화 대상 변수를 지정하는데, `aes()`와 함께 사용한다. `geom_point()` 함수 내의 인수를 조정해 다양한 변형 또는 응용이 가능하다.

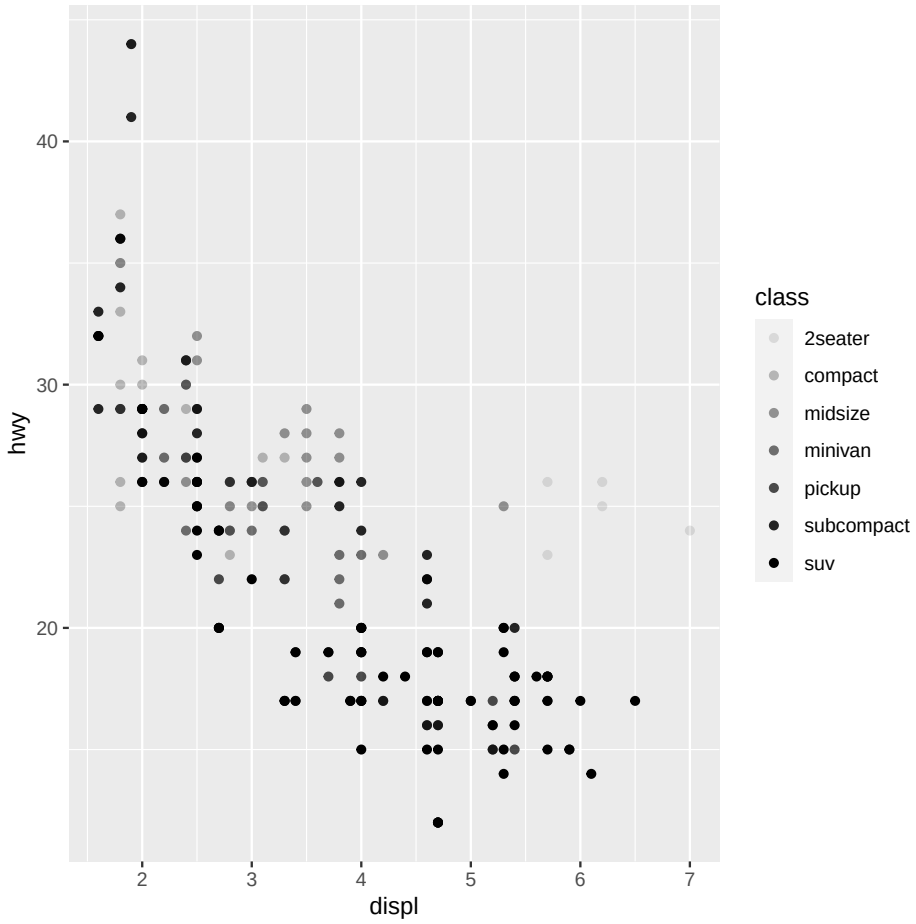
```
> ggplot(data = mpg) +
   geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



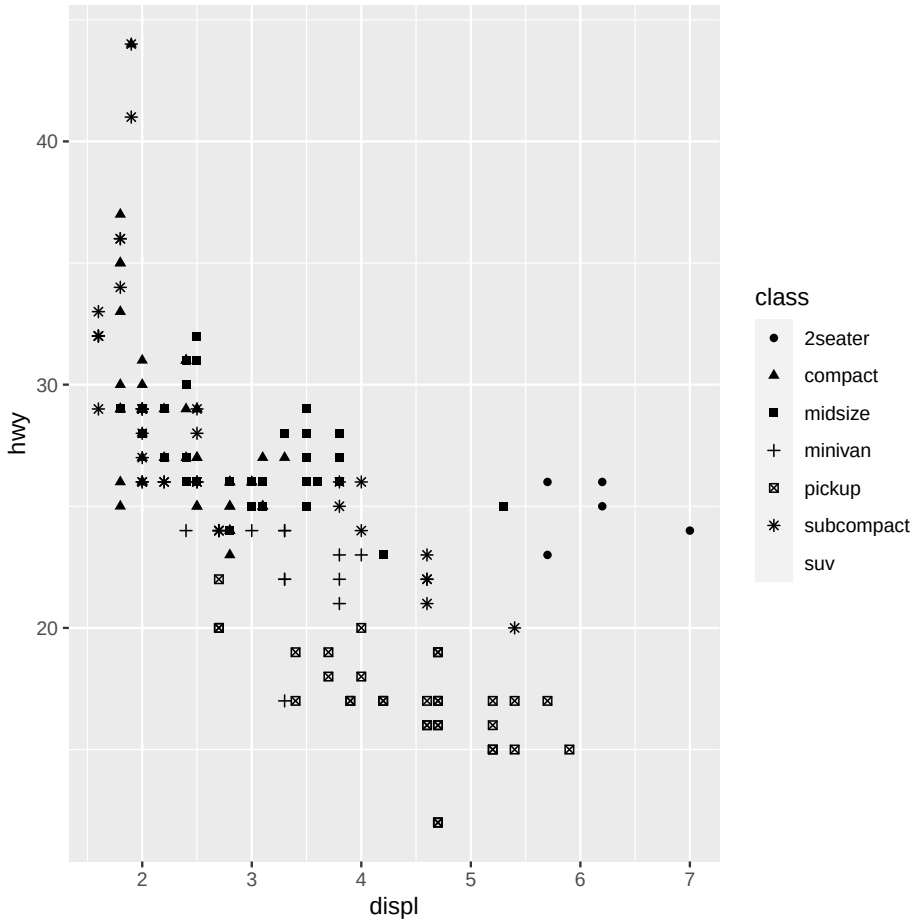

```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



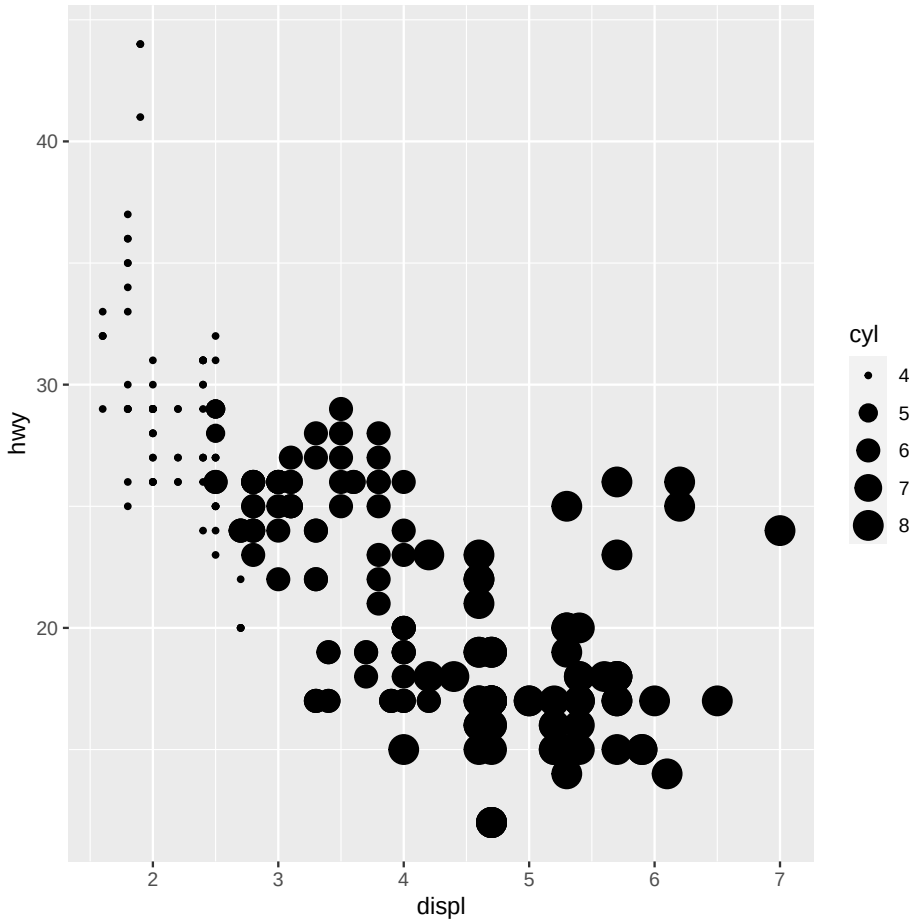

```
> ggplot(data = mpg) +
   geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```



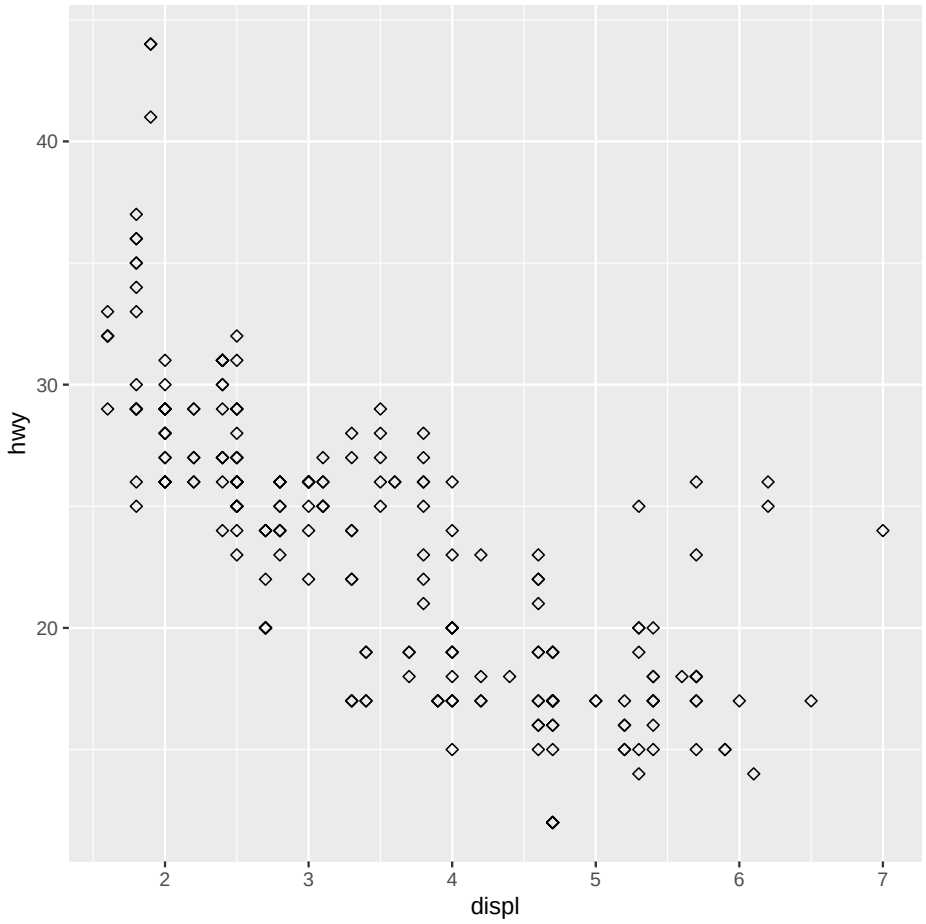

```
> ggplot(data = mpg) +
   geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```




```
> ggplot(data = mpg) +
   geom_point(mapping = aes(x = displ, y = hwy, size = cyl))
```

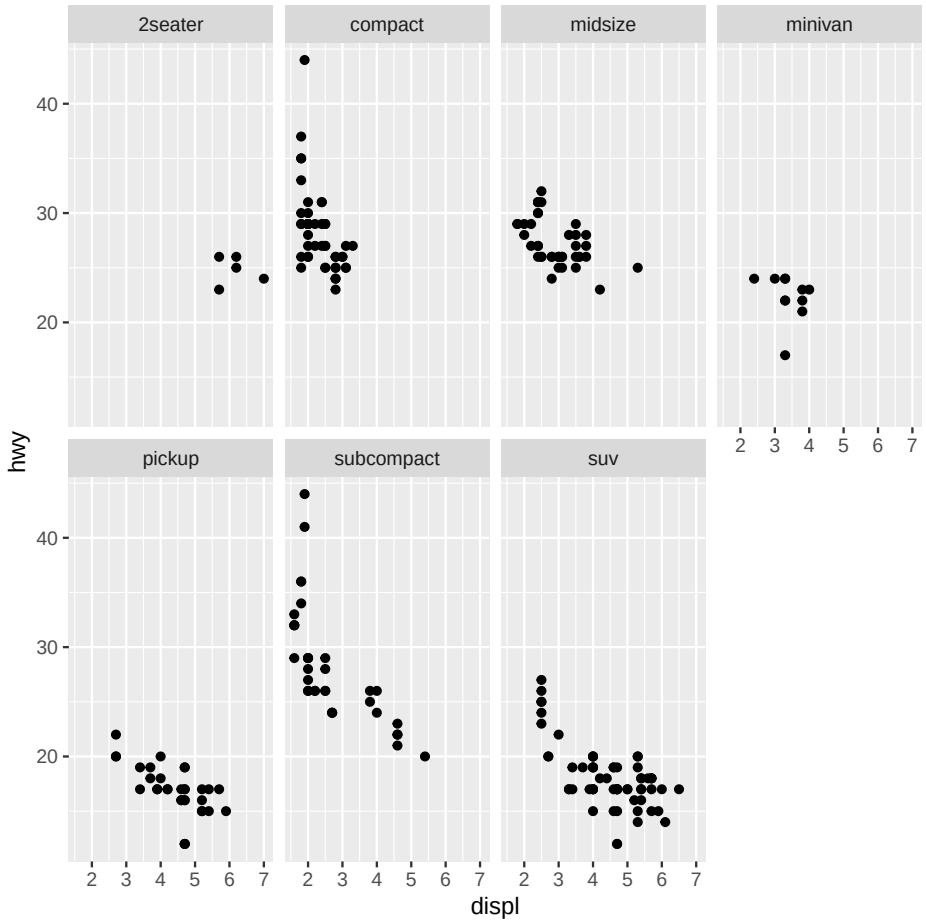



```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy), shape = 5)
```

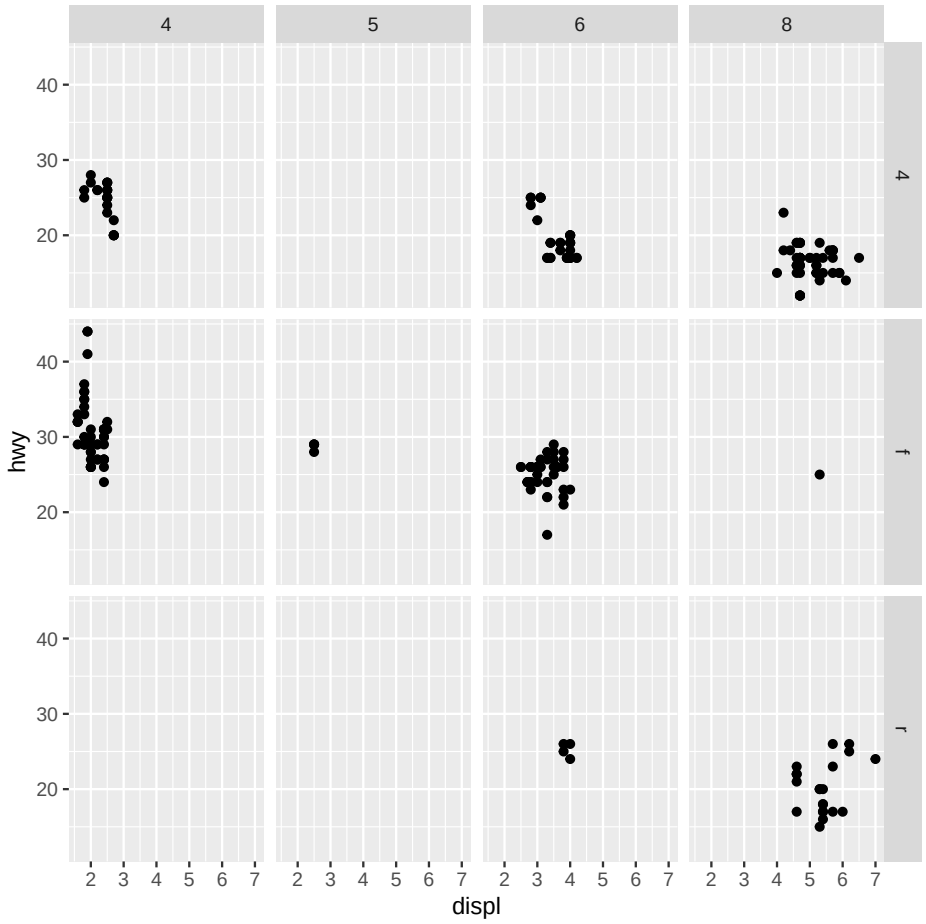


`facet_wrap()` 함수를 사용하면 지정한 범주형 변수의 값별로 산점도를 따로 그릴 수 있다. 두 개의 범주형 변수값의 조합별로 산점도를 따로 그리려면 `facet_grid()` 함수를 사용하면 된다.

```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy)) +
    facet_wrap(~ class, nrow = 2)
```



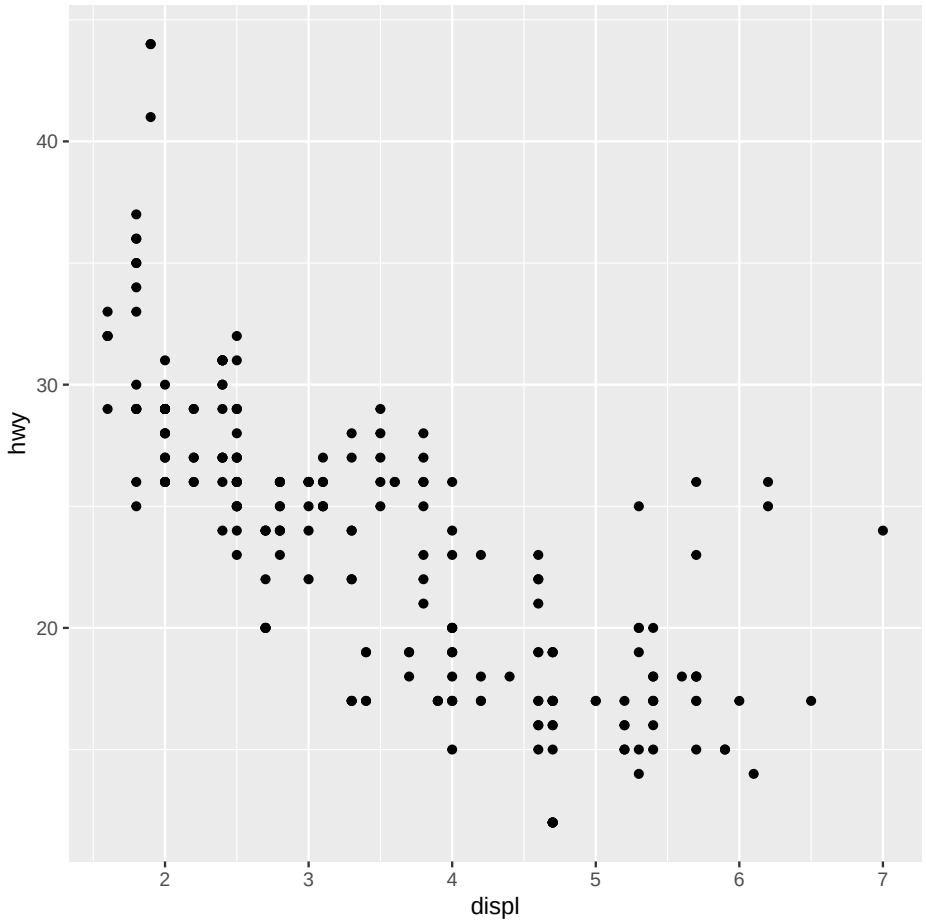

```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy)) +
    facet_grid(drv ~ cyl)
```



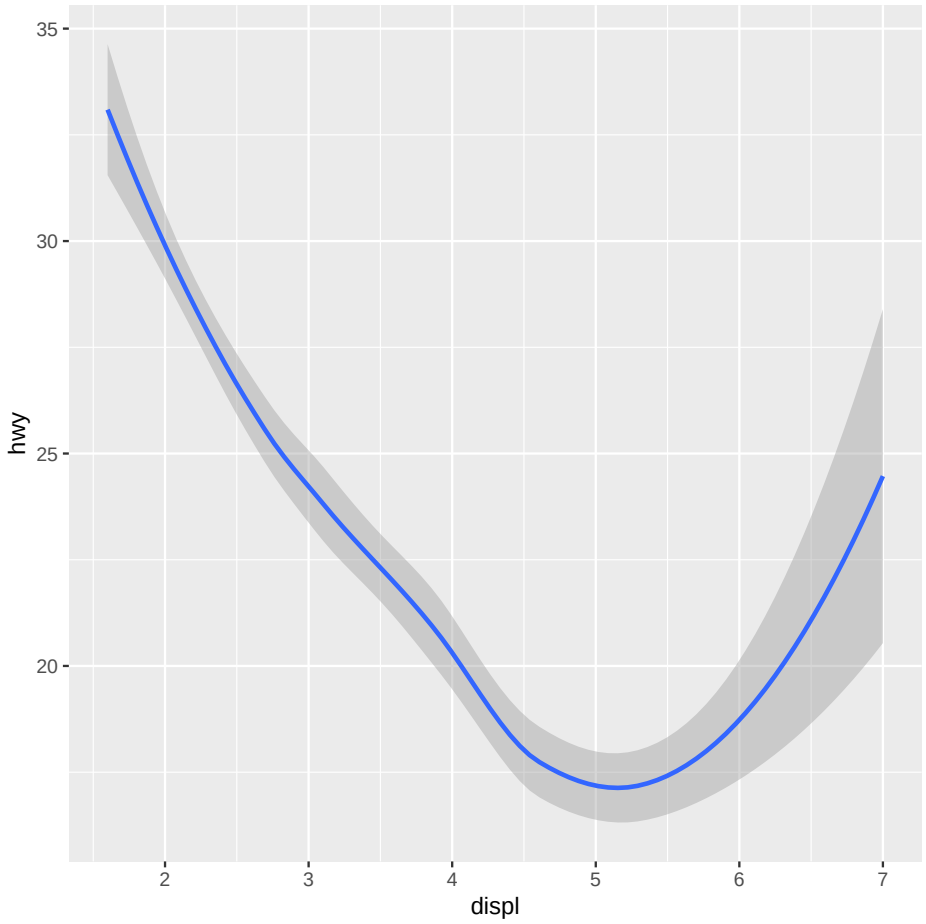
4.2 Geometric objects

geom_ 계열 함수는 데이터를 시각화할 방법(또는 결과적으로 생성되는 객체의 종류)을 지정해준다.

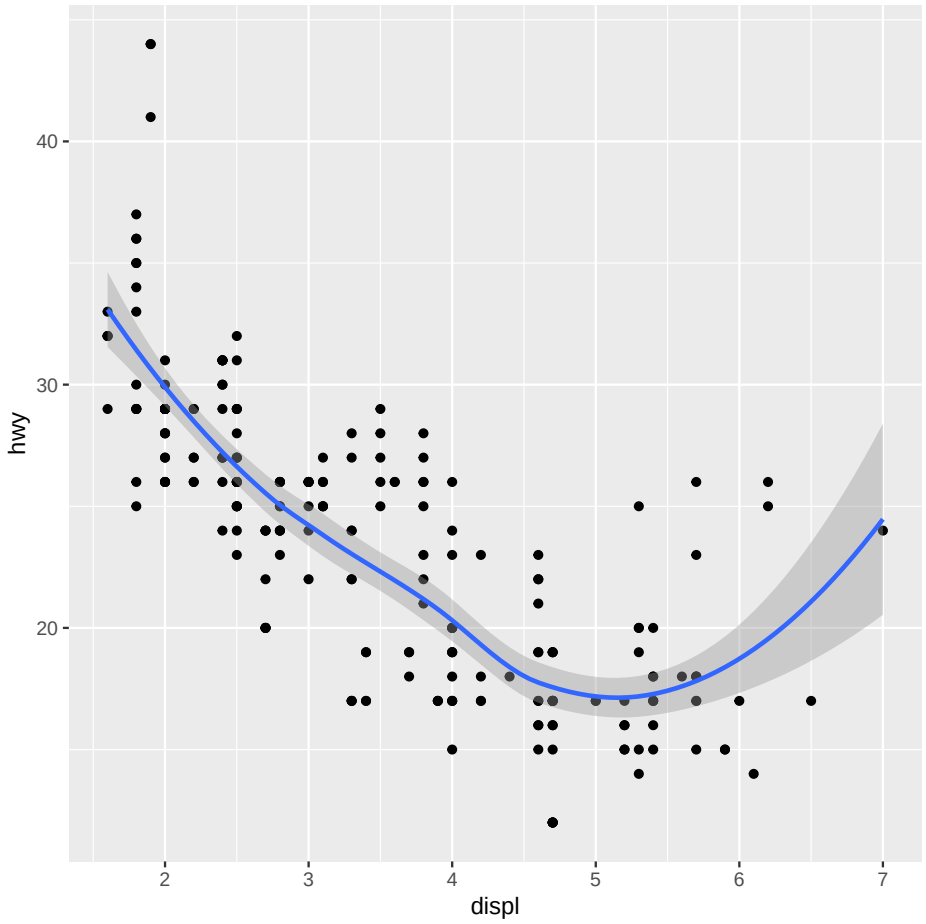
```
> ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy))
```




```
> ggplot(data = mpg) +
   geom_smooth(mapping = aes(x = displ, y = hwy))
```

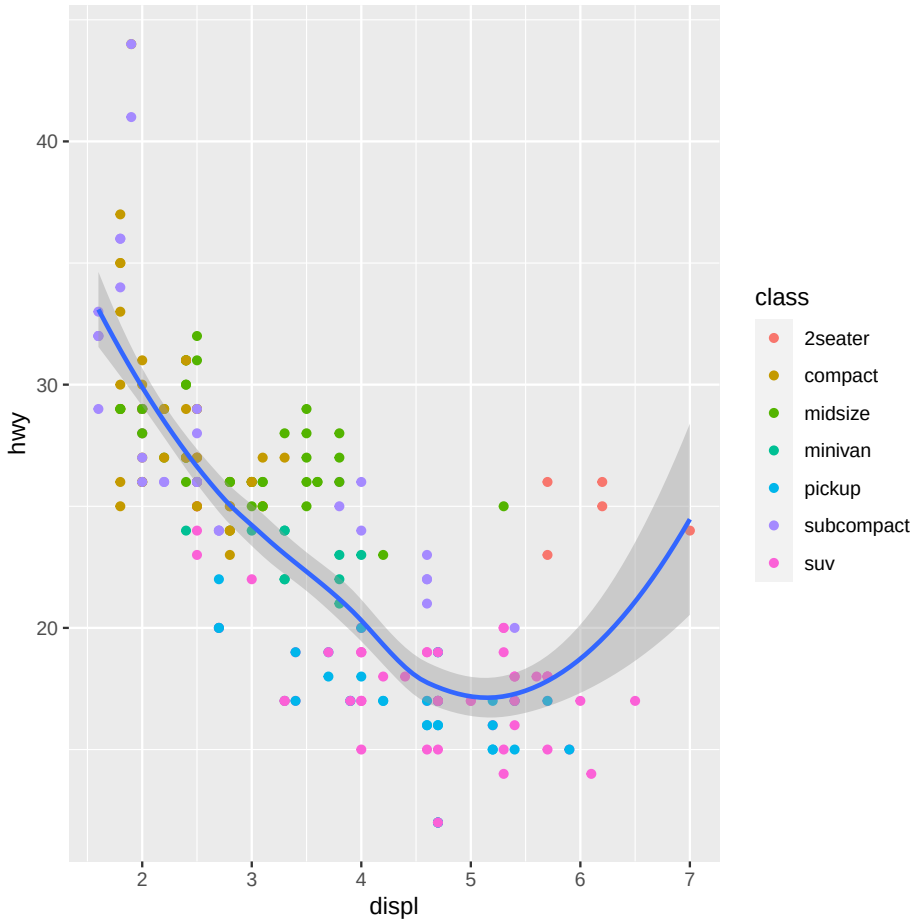



```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()
```

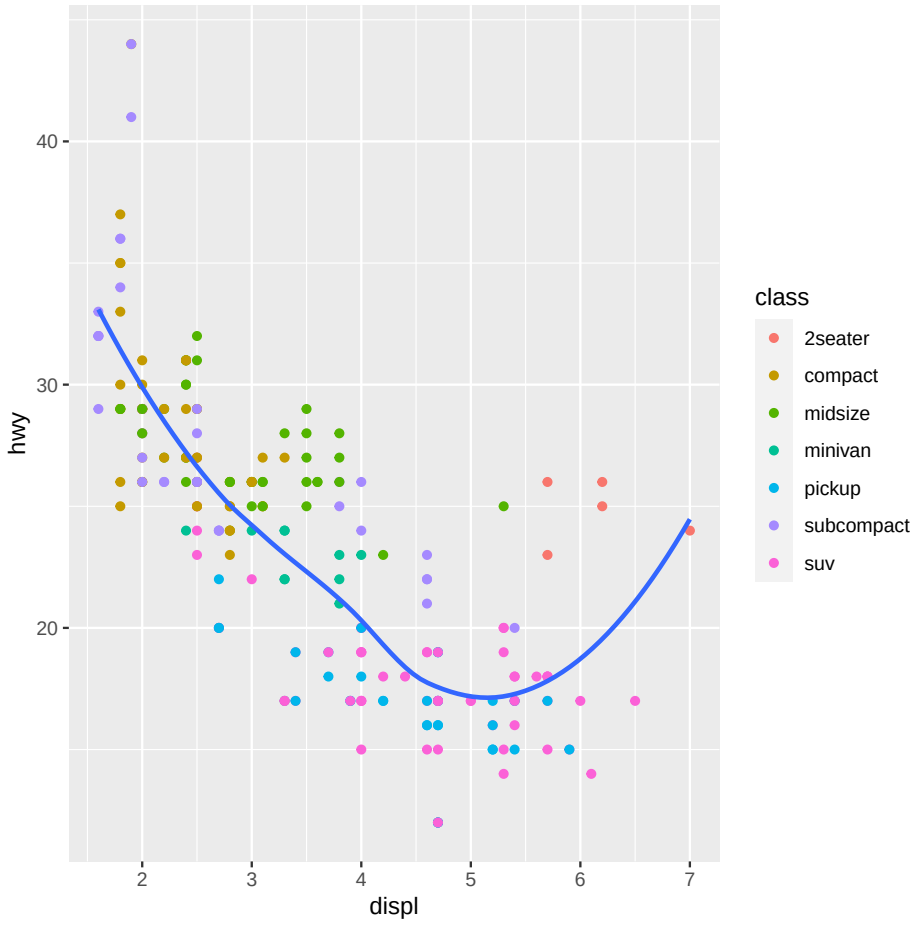


다양한 방식으로 응용한 예들을 살펴보자.

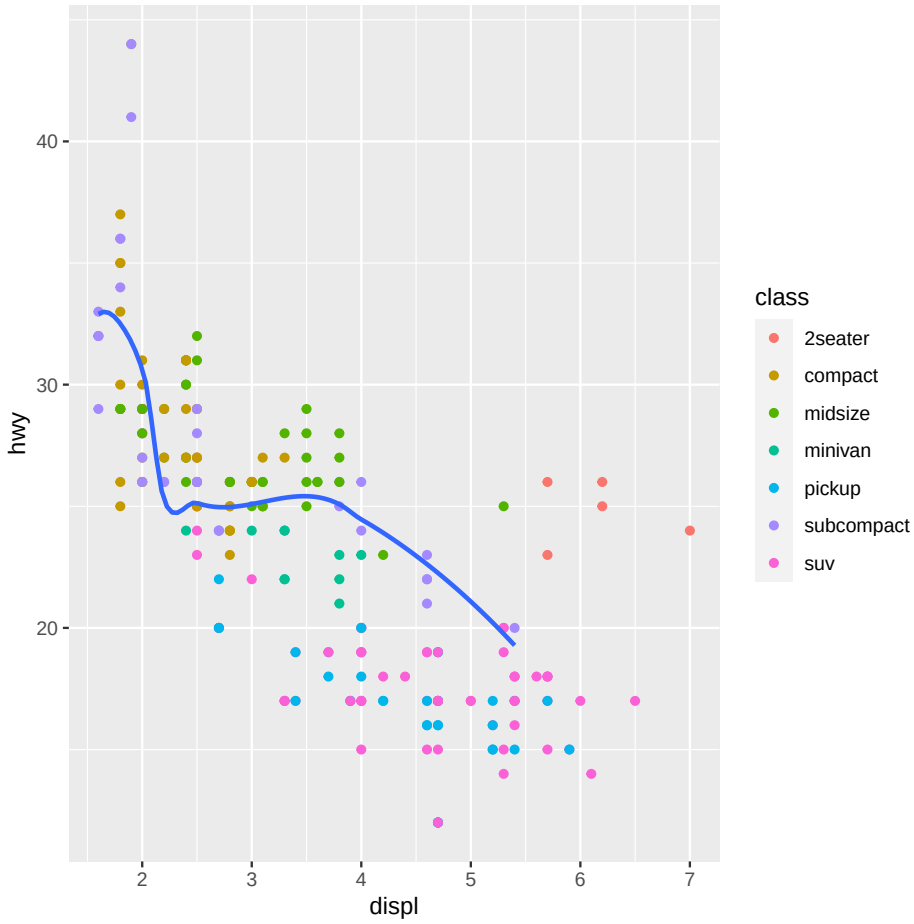
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
    geom_point(mapping = aes(color = class)) +
    geom_smooth()
```




```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
    geom_point(mapping = aes(color = class)) +
    geom_smooth(se = FALSE)
```




```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
   geom_point(mapping = aes(color = class)) +
   geom_smooth(
     data = filter(mpg, class == "subcompact"),
     se = FALSE)
```



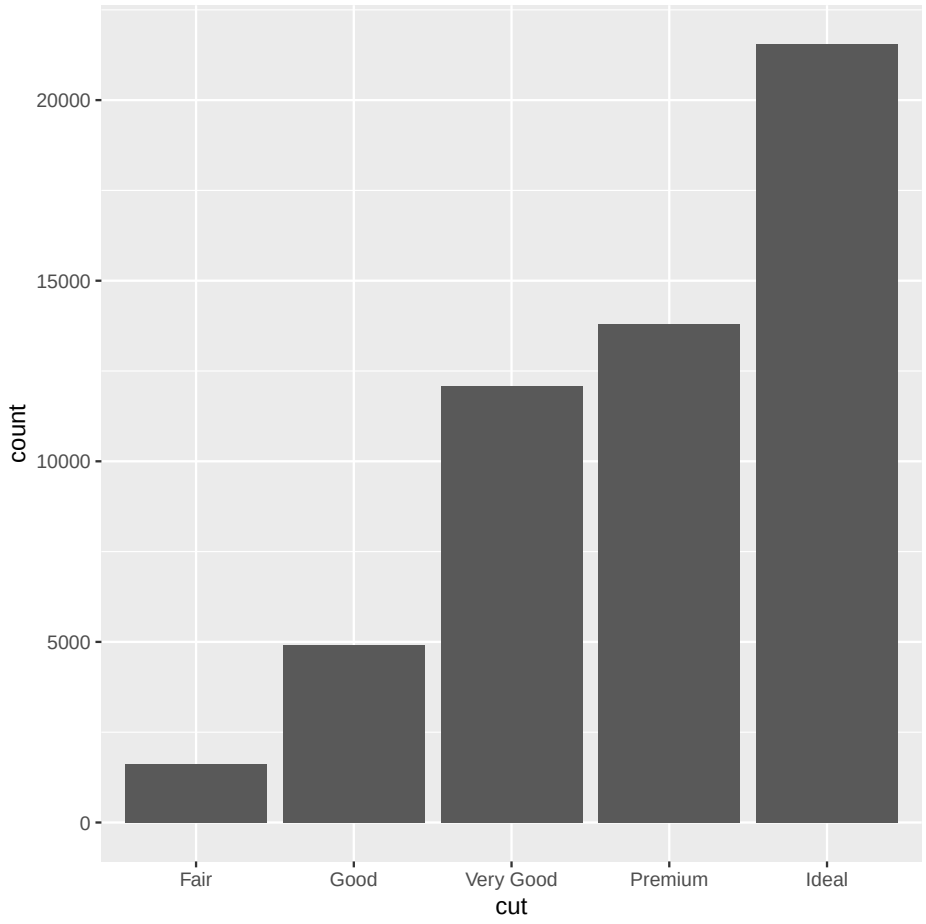
4.3 간단한 통계 분석 결과를 시각화

내장 데이터 diamonds를 로드하자.

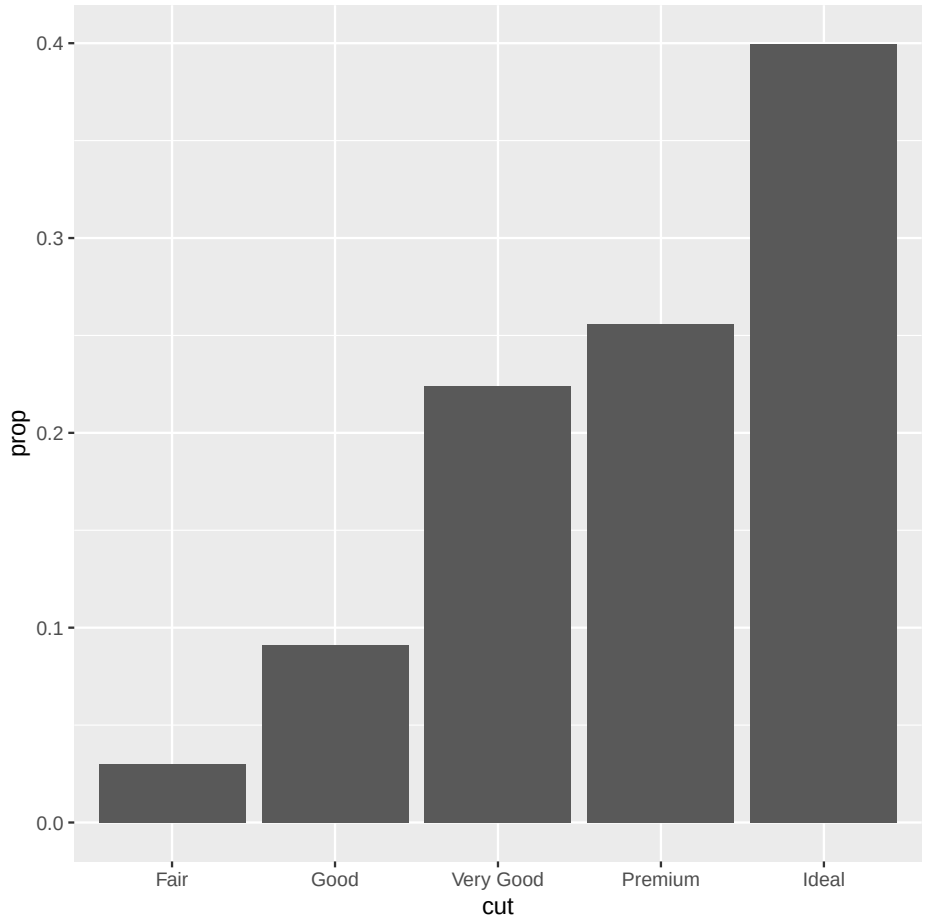
```
> data(diamonds)
```

다이아몬드 품질(cut)별 분포를 살펴보자.

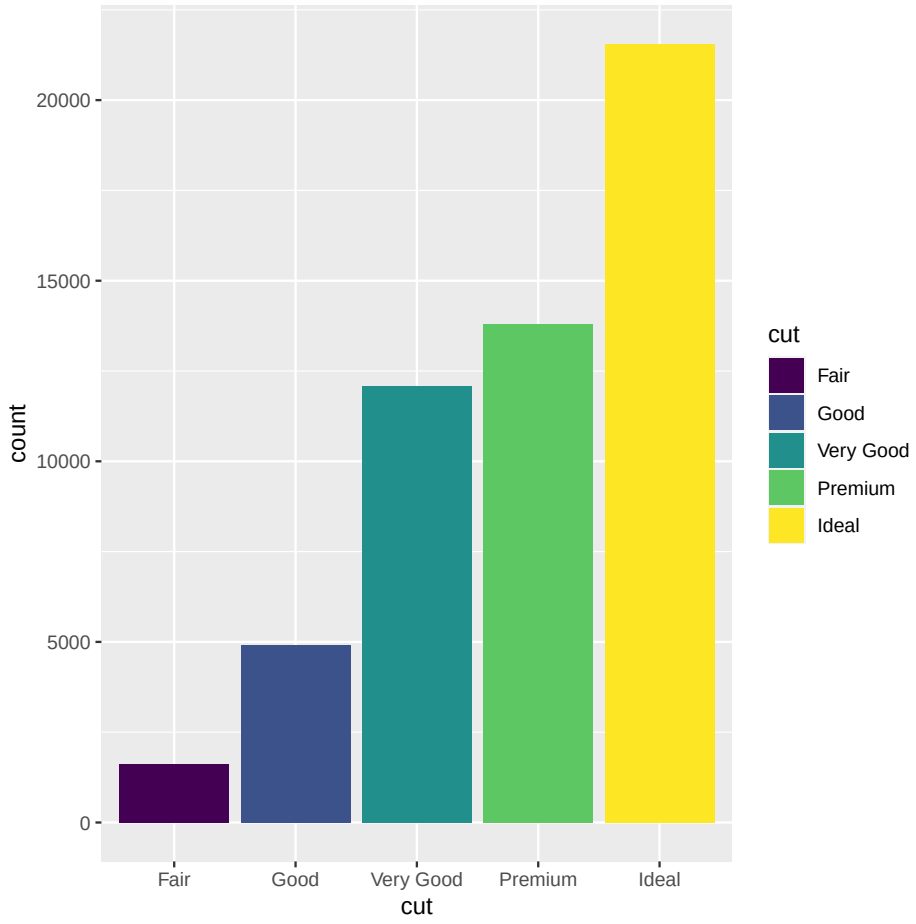
```
> ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut))
```



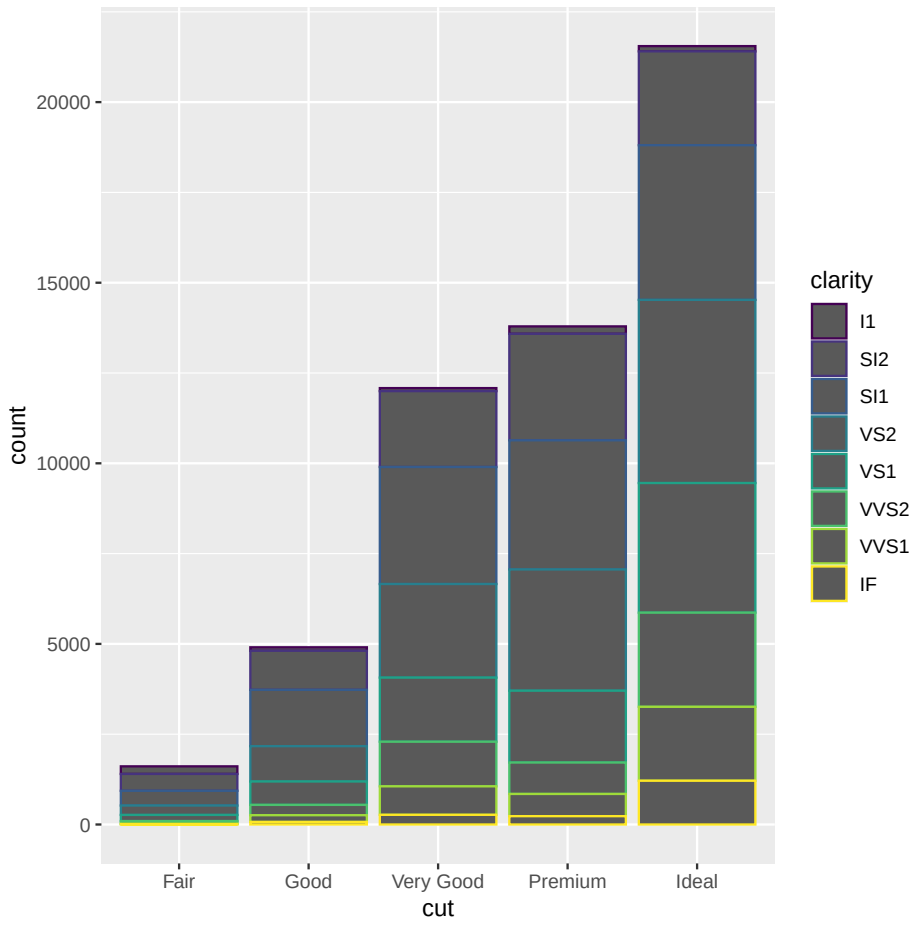

```
> ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1))
```



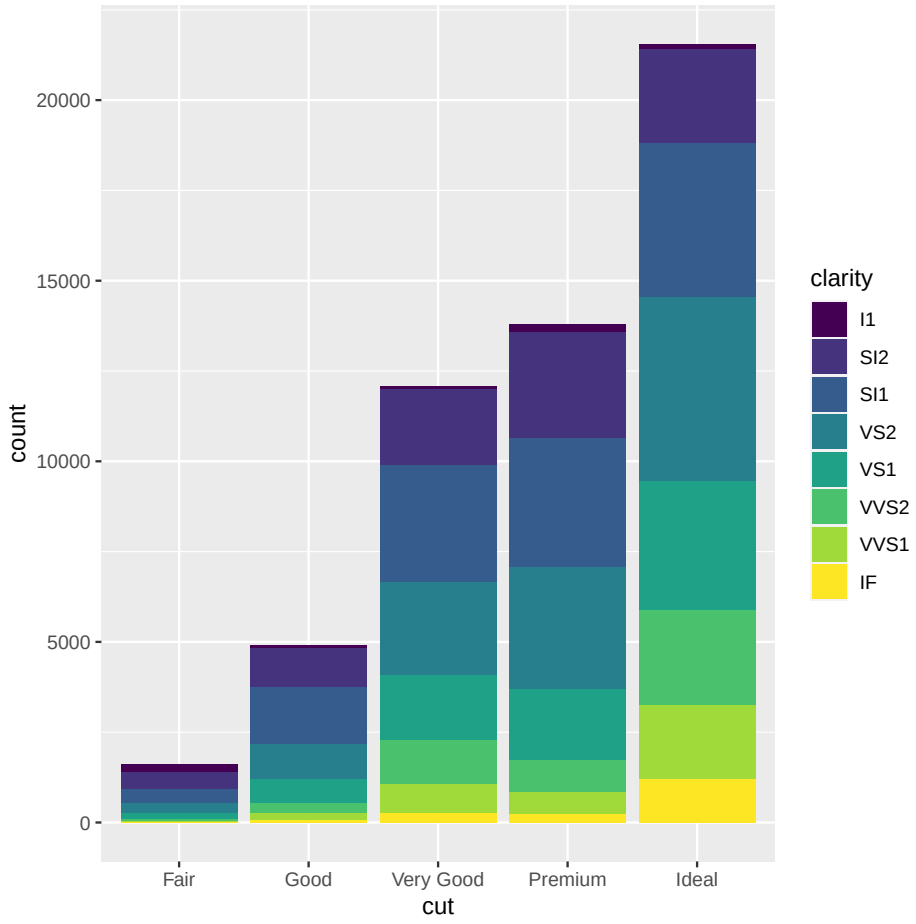

```
> ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, fill = cut))
```




```
> ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, color = clarity)) # Not good
```

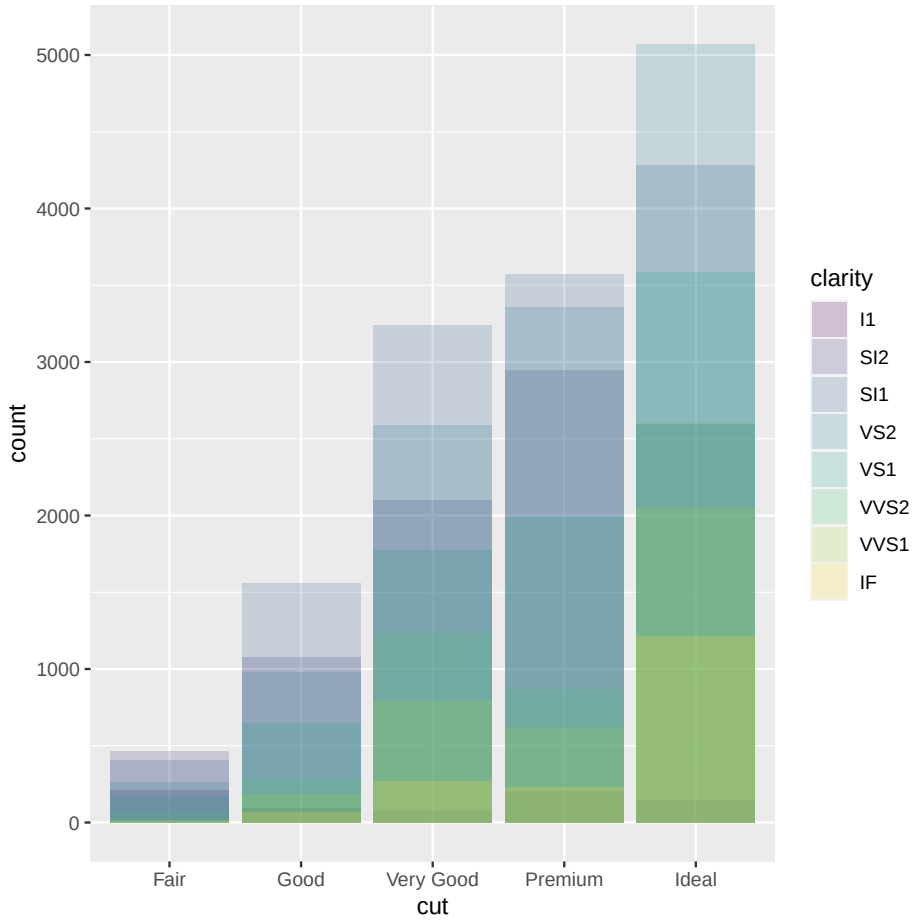



```
> ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, fill = clarity))
```

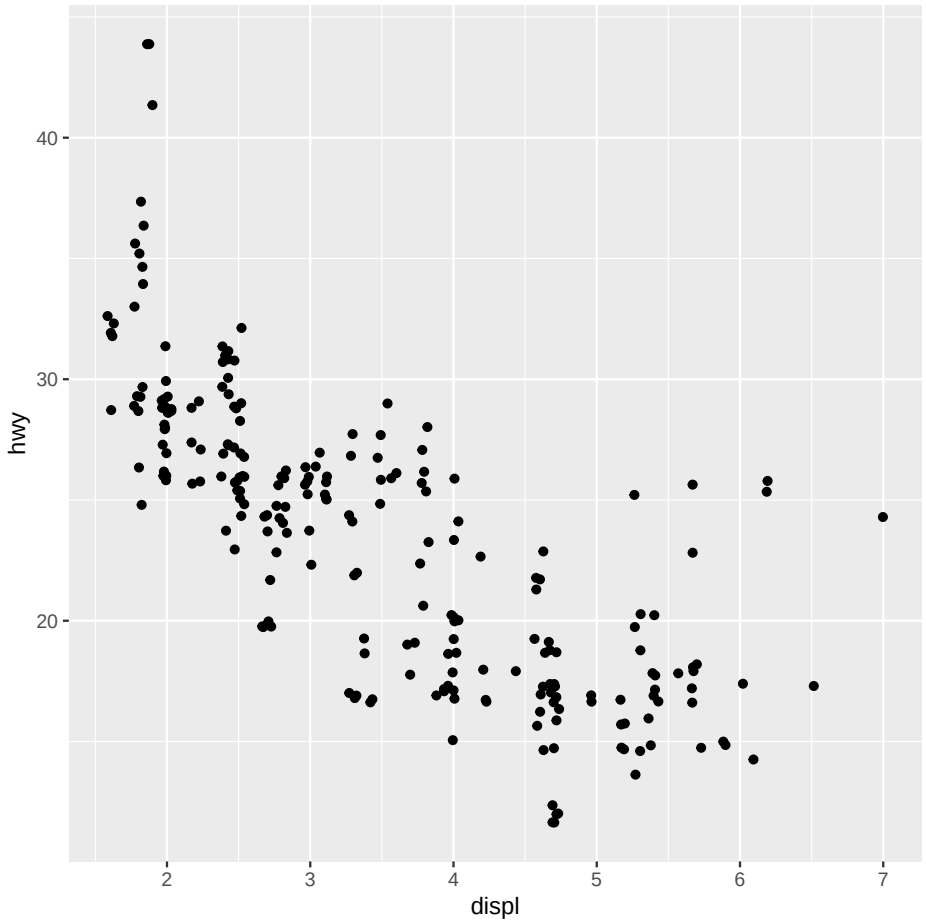


`position`는 막대를 표시하는 방식을 지정하는 인수이다.

```
> d <- ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity))
> d + geom_bar(alpha = 1/5, position = "identity")
> d + geom_bar(position = "fill")
> d + geom_bar(position = "dodge")
```

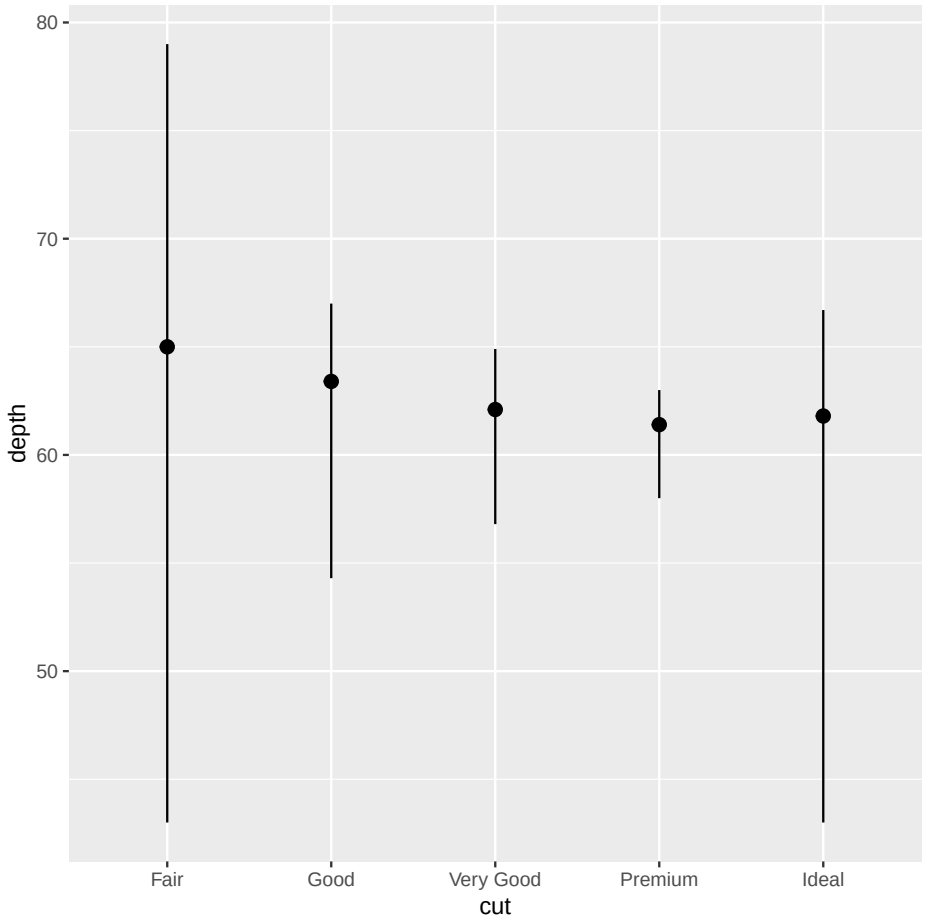



```
> ggplot(data = mpg) +
   geom_point(
     mapping = aes(x = displ, y = hwy),
     position = "jitter"
   )
```



이런 그림도 가능하다.

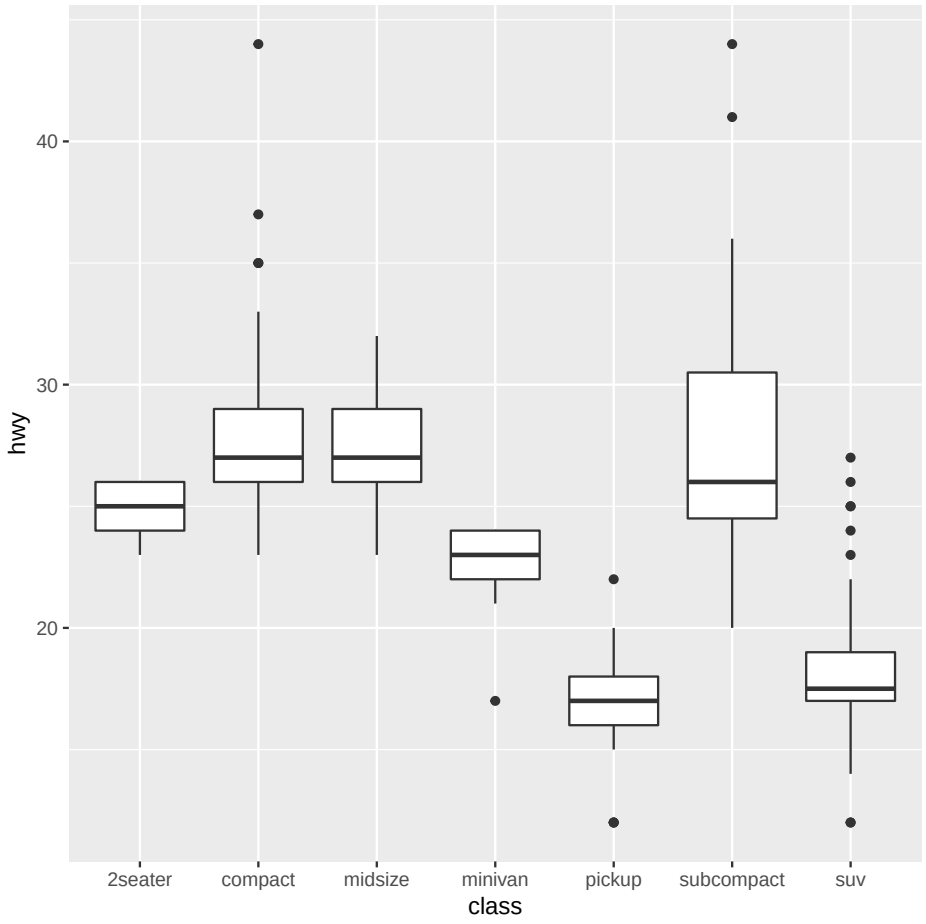
```
> ggplot(data = diamonds) +
   stat_summary(
     mapping = aes(x = cut, y = depth),
     fun.ymin = min,
     fun.ymax = max,
     fun.y = median
   )
```



4.4 좌표축 다루기

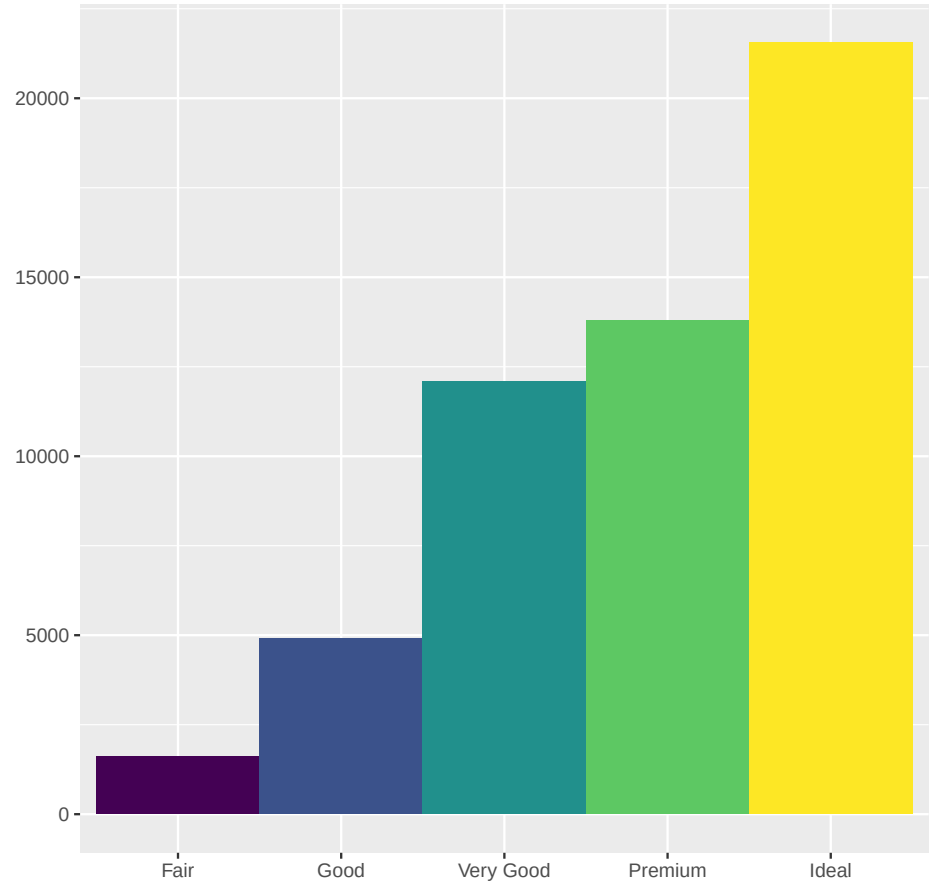
`coord_flip()` 함수는 가로축과 세로축을 바꿔준다.

```
> m <- ggplot(data = mpg, mapping = aes(x = class, y = hwy))
> m + geom_boxplot() # Not good
> m +
   geom_boxplot() +
   coord_flip()
```



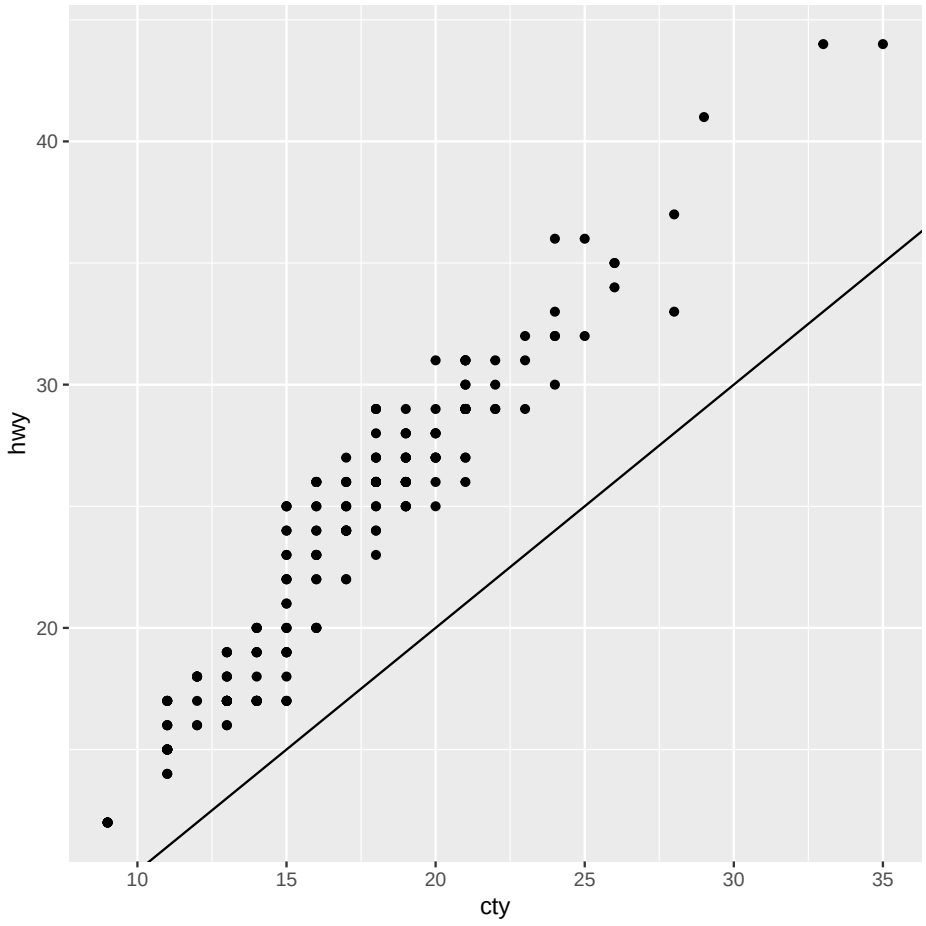
극좌표로 나타내고 싶으면 `coord_polar()` 함수를 사용한다.

```
> dbar <- ggplot(data = diamonds) +
   geom_bar(
     mapping = aes(x = cut, fill = cut),
     show.legend = FALSE,
     width = 1
   ) +
   theme(aspect.ratio = 1) +
   labs(x = NULL, y = NULL)
> dbar
> dbar + coord_flip()
> dbar + coord_polar() # Coxcomb chart
```

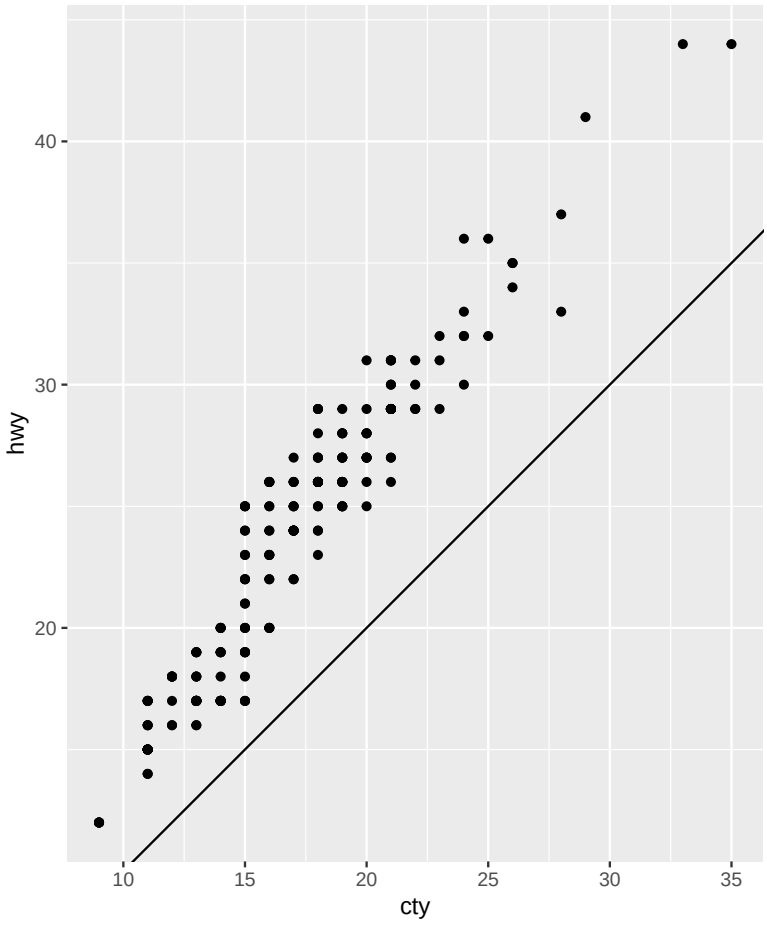


`coord_fixed()` 함수는 가로축과 세로축의 비율을 지정한 값으로 고정시킨다.

```
> ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
    geom_point() +
    geom_abline()
```




```
> ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed() # coord_fixed(ratio = 1): default
```



4.5 Layered grammar

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

제 5 절 dplyr package

5.1 dplyr is...

데이터 munging(또는 wrangling)은 데이터 전처리, 파싱, 필터링과 같이 분석가가 원하는 형태로 데이터를 변형하고 이리저리 핸들링하는 행위를 의미하는 신조어이다.

(Wikipedia: `https://en.wikipedia.org/wiki/Data_wrangling`)

`dplyr` 패키지는 하들리위컴(Hadley Wickham)이 최근 작성한 데이터 처리에 특화된 패키지이다. 하들리위컴이 같은 목적으로 작성한 패키지인 `plyr`는 모든 함수가 R로 작성되어 있어 속도가 느린 문제가 있었으나, `dplyr`은 C++로 작성되었기 때문에 매우 빠른 속도로 데이터 처리를 수행할 수 있게 되었다.

이 패키지를 이용하면 코드 작성이 직관적이고 가독성이 좋은 장점(특히 pipe 연산자를 이용한 chaining syntax를 이용 시)이 있다. 다만 `dplyr` 패키지 내에 몇몇 base 함수와 겹치는 이름의 함수가 있어 masking이 일어남에 주의해야 한다 (`filter()`, `lag()`, `intersect()`, `setdiff()`, `setequal()`, `union()` 등).

또한 `plyr` 패키지와 동시에 사용하는 경우 masking 문제에 더욱 유의할 필요가 있다. 굳이 동시에 사용하려면 `plyr`을 먼저 로드하기를 추천한다.

다음 절부터 사용할 예제 데이터인 `nycflights13` 패키지의 `flights` 데이터셋을 로드하자.

```
> #install.packages("nycflights13")
> library(nycflights13)
> data(flights)
```

5.2 기본 함수들

- `filter()`: 지정 조건에 맞는 관측치 추출
- `arrange()`: 지정 조건에 따라 관측치 정렬
- `select()`: 변수 추출
- `mutate()`: 기존 변수를 이용해 새로운 변수 생성
- `summarize()`: 요약 정보 추출
- `group_by()`: 지정 변수에 따라 정의된 그룹별로 적용 범위를 제한

5.2.1 filter() 함수

```
> filter(flights, month == 1, day == 1)

# A tibble: 842 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# ... with 832 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>

> filter(flights, month == 11 | month == 12)

# A tibble: 55,403 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    11     1        5           2359         6      352            345
 2  2013    11     1       35           2250       105      123           2356
 3  2013    11     1      455            500        -5      641            651
 4  2013    11     1      539            545        -6      856            827
 5  2013    11     1      542            545        -3      831            855
 6  2013    11     1      549            600       -11      912            923
 7  2013    11     1      550            600       -10      705            659
 8  2013    11     1      554            600        -6      659            701
 9  2013    11     1      554            600        -6      826            827
```

```
10  2013    11     1      554            600        -6      749            751
# ... with 55,393 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>

> filter(flights, month %in% c(11, 12))

# A tibble: 55,403 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    11     1        5           2359         6      352            345
 2  2013    11     1       35           2250       105      123           2356
 3  2013    11     1      455            500        -5      641            651
 4  2013    11     1      539            545        -6      856            827
 5  2013    11     1      542            545        -3      831            855
 6  2013    11     1      549            600       -11      912            923
 7  2013    11     1      550            600       -10      705            659
 8  2013    11     1      554            600        -6      659            701
 9  2013    11     1      554            600        -6      826            827
10  2013    11     1      554            600        -6      749            751
# ... with 55,393 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.2.2 arrange() 함수

```
> arrange(flights, year, month, day)

# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
```

```
 4  2013     1     1      544           545        -1     1004           1022
 5  2013     1     1      554           600        -6      812            837
 6  2013     1     1      554           558        -4      740            728
 7  2013     1     1      555           600        -5      913            854
 8  2013     1     1      557           600        -3      709            723
 9  2013     1     1      557           600        -3      838            846
10  2013     1     1      558           600        -2      753            745
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
> arrange(flights, desc(arr_delay)) # 내림차순 정렬
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     9      641            900      1301     1242           1530
 2  2013     6    15     1432           1935      1137     1607           2120
 3  2013     1    10     1121           1635      1126     1239           1810
 4  2013     9    20     1139           1845      1014     1457           2210
 5  2013     7    22      845           1600      1005     1044           1815
 6  2013     4    10     1100           1900       960     1342           2211
 7  2013     3    17     2321            810       911      135           1020
 8  2013     7    22     2257            759       898      121           1026
 9  2013    12     5      756           1700       896     1058           2020
10  2013     5     3     1133           2055       878     1250           2215
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.2.3 select() 함수

```
> select(flights, year, month, day)
```

```
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# ... with 336,766 more rows

> select(flights, year:day)

# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# ... with 336,766 more rows

> select(flights, -(year:day))

# A tibble: 336,776 × 16
```

```
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
      <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
 1      517            515         2      830            819        11 UA
 2      533            529         4      850            830        20 UA
 3      542            540         2      923            850        33 AA
 4      544            545        -1     1004           1022       -18 B6
 5      554            600        -6      812            837       -25 DL
 6      554            558        -4      740            728        12 UA
 7      555            600        -5      913            854        19 B6
 8      557            600        -3      709            723       -14 EV
 9      557            600        -3      838            846        -8 B6
10      558            600        -2      753            745         8 AA
# ... with 336,766 more rows, and 9 more variables: flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>

> select(flights, time_hour, air_time, everything())

# A tibble: 336,776 × 19
   time_hour           air_time  year month   day dep_time sched_dep_time
   <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
 1 2013-01-01 05:00:00      227  2013     1     1      517            515
 2 2013-01-01 05:00:00      227  2013     1     1      533            529
 3 2013-01-01 05:00:00      160  2013     1     1      542            540
 4 2013-01-01 05:00:00      183  2013     1     1      544            545
 5 2013-01-01 06:00:00      116  2013     1     1      554            600
 6 2013-01-01 05:00:00      150  2013     1     1      554            558
 7 2013-01-01 06:00:00      158  2013     1     1      555            600
 8 2013-01-01 06:00:00       53  2013     1     1      557            600
 9 2013-01-01 06:00:00      140  2013     1     1      557            600
10 2013-01-01 06:00:00      138  2013     1     1      558            600
# ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
```

```
#   hour <dbl>, minute <dbl>
```

```
> # everything() helper function moves two variables to the start of the data frame
```

everything() 이외의 유용한 Helper 함수들은 다음과 같은 것들이 있다.

- starts_with("abc"): abc로 시작되는 변수
- ends_with("xyz"): xyz로 끝나는 변수
- contains("ijk"): ijk를 포함한 변수
- num_range("x", 1:3): x1, x2, x3

5.2.4 rename() 함수

```
> rename(flights, tail_num = tailnum)
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.2.5 mutate() 함수

```
> flights_sml <- select(flights,
                        year:day,
                        ends_with("delay"),
                        distance,
                        air_time)
> mutate(flights_sml,
         gain = arr_delay - dep_delay,
         hours = air_time/60,
         gain_per_hour = gain/hours)

# A tibble: 336,776 × 10
    year month   day dep_delay arr_delay distance air_time  gain hours
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
 1  2013     1     1         2        11     1400      227     9 3.78
 2  2013     1     1         4        20     1416      227    16 3.78
 3  2013     1     1         2        33     1089      160    31 2.67
 4  2013     1     1        -1       -18     1576      183   -17 3.05
 5  2013     1     1        -6       -25      762      116   -19 1.93
 6  2013     1     1        -4        12      719      150    16 2.5
 7  2013     1     1        -5        19     1065      158    24 2.63
 8  2013     1     1        -3       -14      229       53   -11 0.883
 9  2013     1     1        -3        -8      944      140    -5 2.33
10  2013     1     1        -2         8      733      138    10 2.3
# ... with 336,766 more rows, and 1 more variable: gain_per_hour <dbl>
```

새로 만든 변수만 남겨놓고 싶으면 transmute() 함수를 이용한다.

```
> transmute(flights_sml,
            gain = arr_delay - dep_delay,
            hours = air_time/60,
            gain_per_hour = gain/hours)

# A tibble: 336,776 × 3
```

```
   gain hours gain_per_hour
  <dbl> <dbl>         <dbl>
1     9 3.78           2.38
2    16 3.78           4.23
3    31 2.67          11.6
4   -17 3.05          -5.57
5   -19 1.93          -9.83
6    16 2.5            6.4
7    24 2.63           9.11
8   -11 0.883        -12.5
9    -5 2.33          -2.14
10   10 2.3            4.35
# ... with 336,766 more rows
```

5.2.6 summarise()를 이용한 그룹별 요약

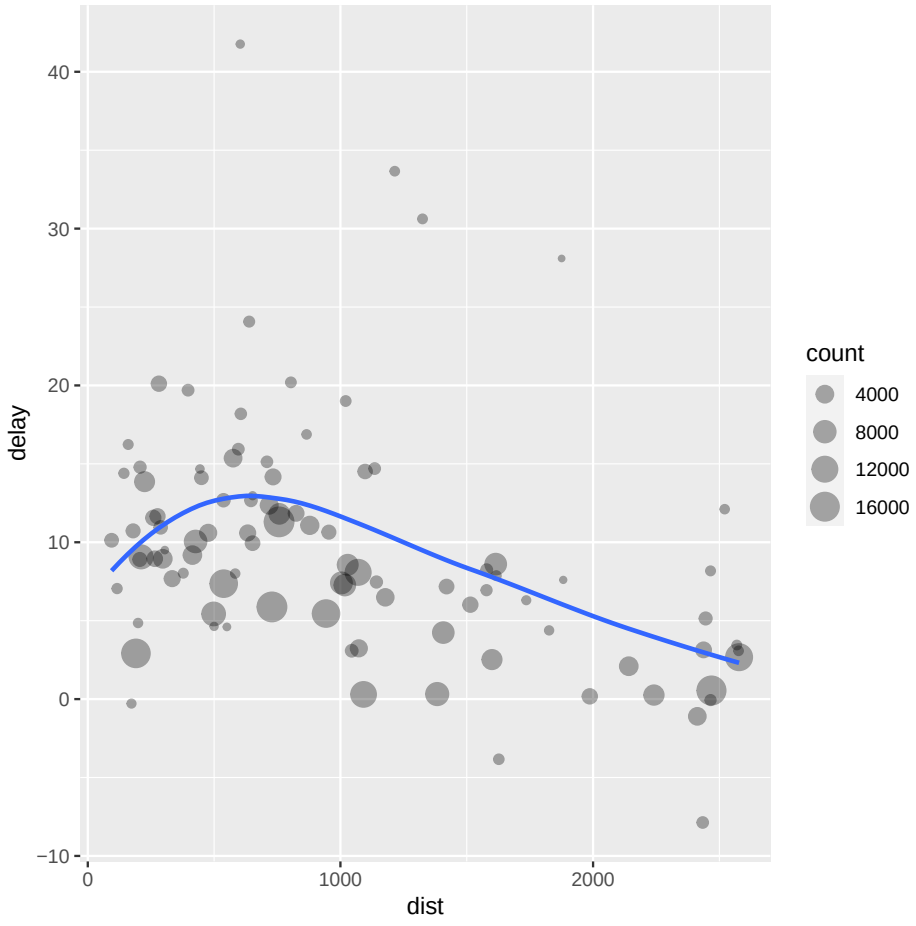
```
> summarise(flights, delay = mean(dep_delay, na.rm = TRUE))

# A tibble: 1 × 1
  delay
  <dbl>
1  12.6

> by_month <- group_by(flights, year, month)
> summarise(by_month, delay = mean(dep_delay, na.rm = TRUE))

# A tibble: 12 × 3
# Groups:   year [1]
    year month delay
   <int> <int> <dbl>
 1  2013     1 10.0
 2  2013     2 10.8
 3  2013     3 13.2
 4  2013     4 13.9
```

```
 5  2013     5 13.0
 6  2013     6 20.8
 7  2013     7 21.7
 8  2013     8 12.6
 9  2013     9  6.72
10  2013    10  6.24
11  2013    11  5.44
12  2013    12 16.6
```

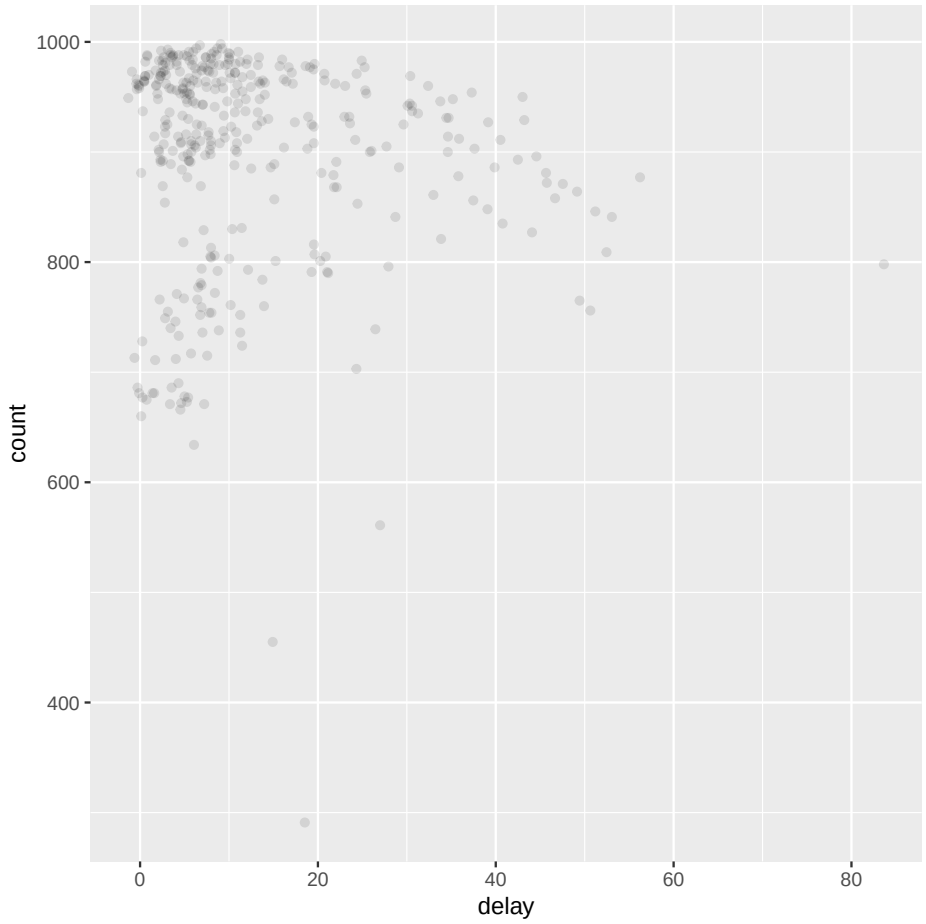
5.3 Pipe 연산자

파이프 연산자(%>%)는 magrittr 패키지에서 제공하는 기능으로 코드의 가독성 및 작성 시 편의성을 제고해 준다.

```
> delays <- flights %>%
   group_by(dest) %>%
   summarise(count = n(),
             dist = mean(distance, na.rm = TRUE),
             delay = mean(arr_delay, na.rm = TRUE)) %>%
   filter(count > 20, dest != "HNL")
> ggplot(data = delays, mapping = aes(x = dist, y = delay)) +
   geom_point(aes(size = count), alpha = 1/3) +
   geom_smooth(se = FALSE)
```




```
> flights %>%
    filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
    group_by(year, month, day) %>%
    summarise(count = n(),
              delay = mean(dep_delay)) %>%
    ggplot(mapping = aes(x = delay, y = count)) +
      geom_point(alpha = 1/10)
```


1000
800
600
400
count
0
20
40
60
80
delay