M23 - ggplot2

LearningSpoonsR

2019-02-11

시작하기

시각화

- 간단한 그래프는 다른 어떤 장치보다 데이터 분석가의 마음에 더 많은 정보를 제공합니다. – John Tukey

데이터셋 불러오기 mpg

- **ggplot2** 패키지에 내장된 데이터셋
- 1999년 부터 2008년 까지의 38개 차종 연비 데이터
- 데이터셋 **diamond**, **iris**와 함께 R에서 가장 많이 예제로 쓰임

변수이름	설명
manufacturer	
model	차종
displ	엔진크기
year	
cyl	기통

변수이름	설명
drv	전륜(f)/후륜(r)/사륜(4)
cty	도심 마일리지
hwy	고속도로 마일리지
fl	연료종류 (fuel)
class	클래스 (중형, 트럭, SUV,…)

```
library(ggplot2)
? mpg # getting help from web
```

```
## starting httpd help server ... done
```

```
help(mpg) # getting help from help panel
```

```
class(mpg)
```

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

- mpg는 여러개의 class의 특징을 가지고 있습니다.
- tbl_df는 data.frame의 업그레이드 버전입니다.
  - data.frame의 모든 기능 사용
  - 몇 가지 기능 추가

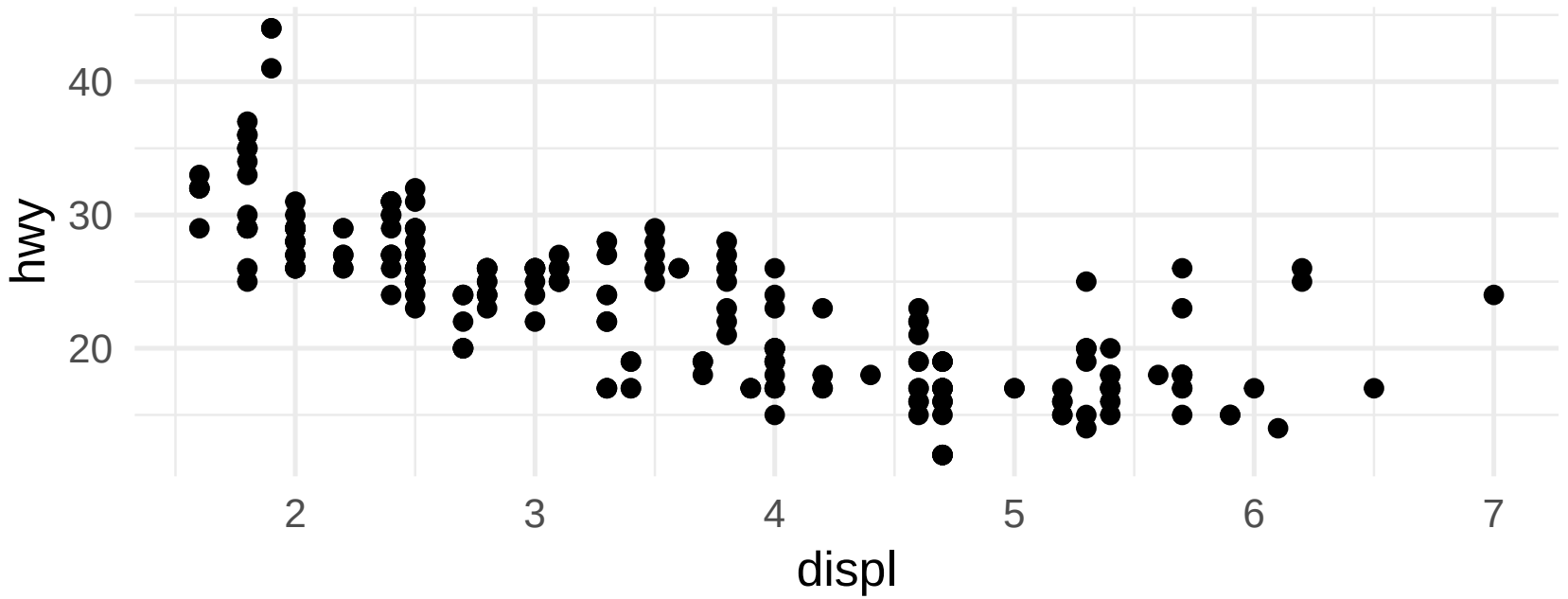
```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

I. Scatterplot (산점도)

- 엔진이 크면 연비가 안 좋을까요?
- 엔진 크기(**displ**)를 x축, 고속도로 마일리지(**hwy**)를 y축으로 하는 산점도(scatterplot) 그리기

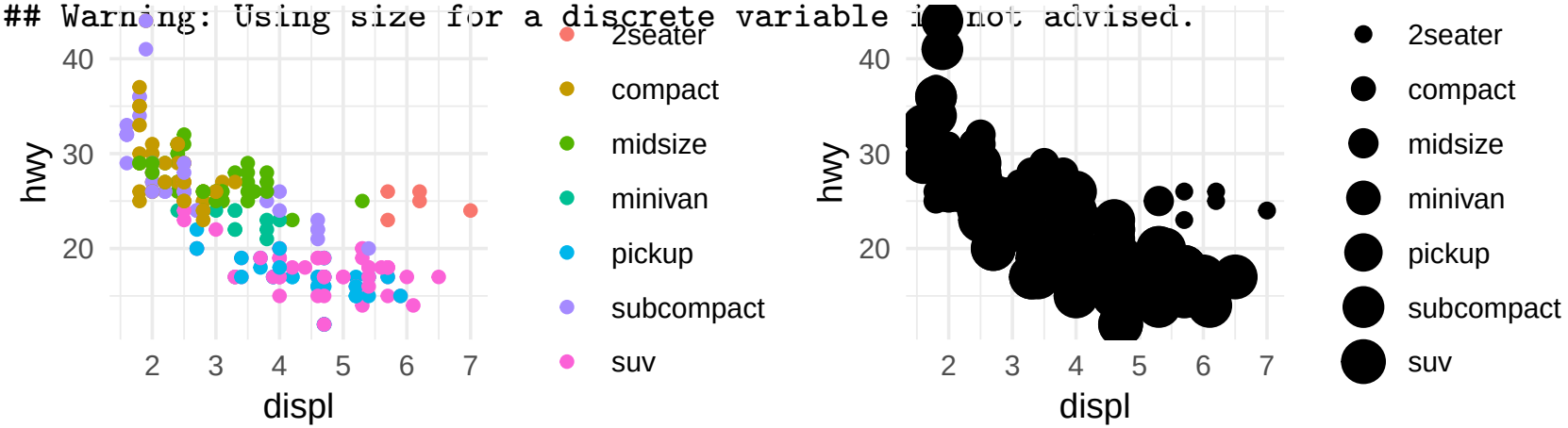
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```




```
# basic ggplot command
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPING>))
```

```
suppressMessages(library(gridExtra)) # grid.arrange()
a <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
b <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
grid.arrange(a, b, nrow=1, ncol=2)
```

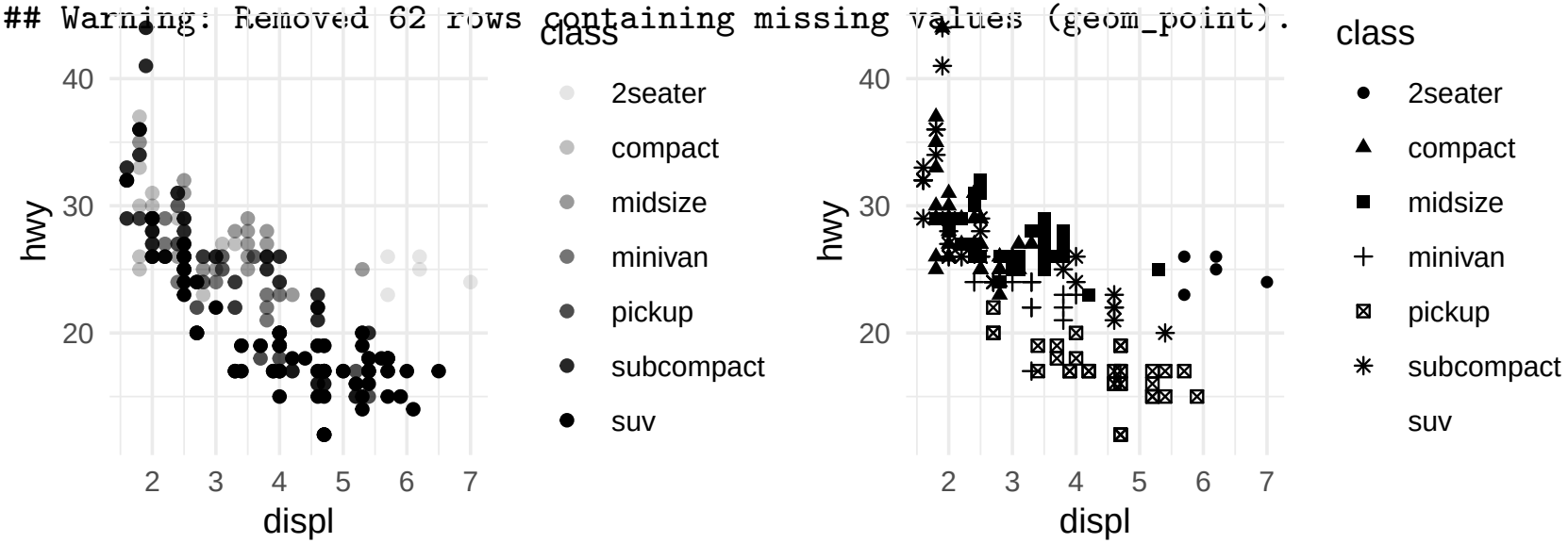
```
## Warning: Using size for a discrete variable is not advised.
```



- ggplot2
  - a와 b는 각각 graph 객체
  - color=class: class 변수의 값에 따라 color가 다르게
  - size=class: class변수의 값에 따라 size가 다르게
- others
  - suppressMessages()는 "로딩되었습니다."와 같은 메시지를 보기 싫을때 사용
  - gridExtra는 grid.arrange 함수를 사용하게 해 줌
  - grid.arrange(a, b, nrow=1, ncol=2): a와 b를 1개행, 2개열에 배치

```
c <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
d <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
grid.arrange(c, d, nrow=1, ncol=2)
```

```
## Warning: Using alpha for a discrete variable is not advised.
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 7.
## Consider specifying shapes manually if you must have them.
## Warning: Removed 62 rows containing missing values (geom_point).
```



- ggplot2
  - alpha=class: class 변수의 값에 따라 alpha(진하기)가 다르게
  - shape=class: class변수의 값에 따라 shape이 다르게

Discussion

a, b, c, d 중에 어느 그림이 가장 효과적인가요?

- **size**, **alpha**, **color**, **shape**의 사용 기준은 무엇인가요?
- 일반화 시키실 수 있나요?

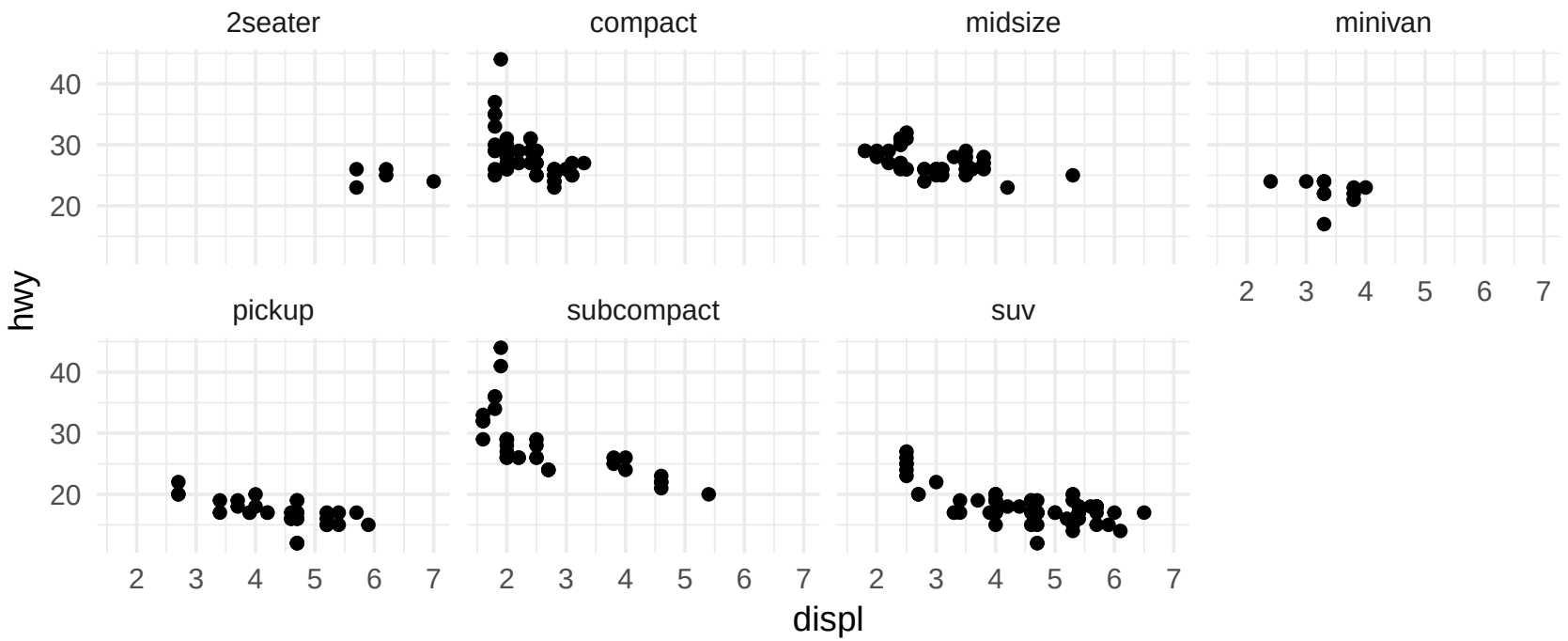
Aesthetics (**aes**) – size, alpha, color, shape

- Numeric 변수
  - 수치 변수, 정량적 변수, numeric, quantative
  - 크고 작음이 있음, 더할 수 있음
  - **size**
  - **alpha**: 상한/하한이 있는 경우에 더 적합
- Categorical 변수
  - 범주형 변수, 정성적 변수, categorical, factor, qualitative
  - 집합. A,B,C로 치환 가능, 더할 수 없음, 크고 작음이 없음
  - **color**: 10개 미만의 범주에서 사용
  - **shape**: 7개 미만의 범주에서 사용

II. Faceting

Facets (1 variable) – facet_wrap

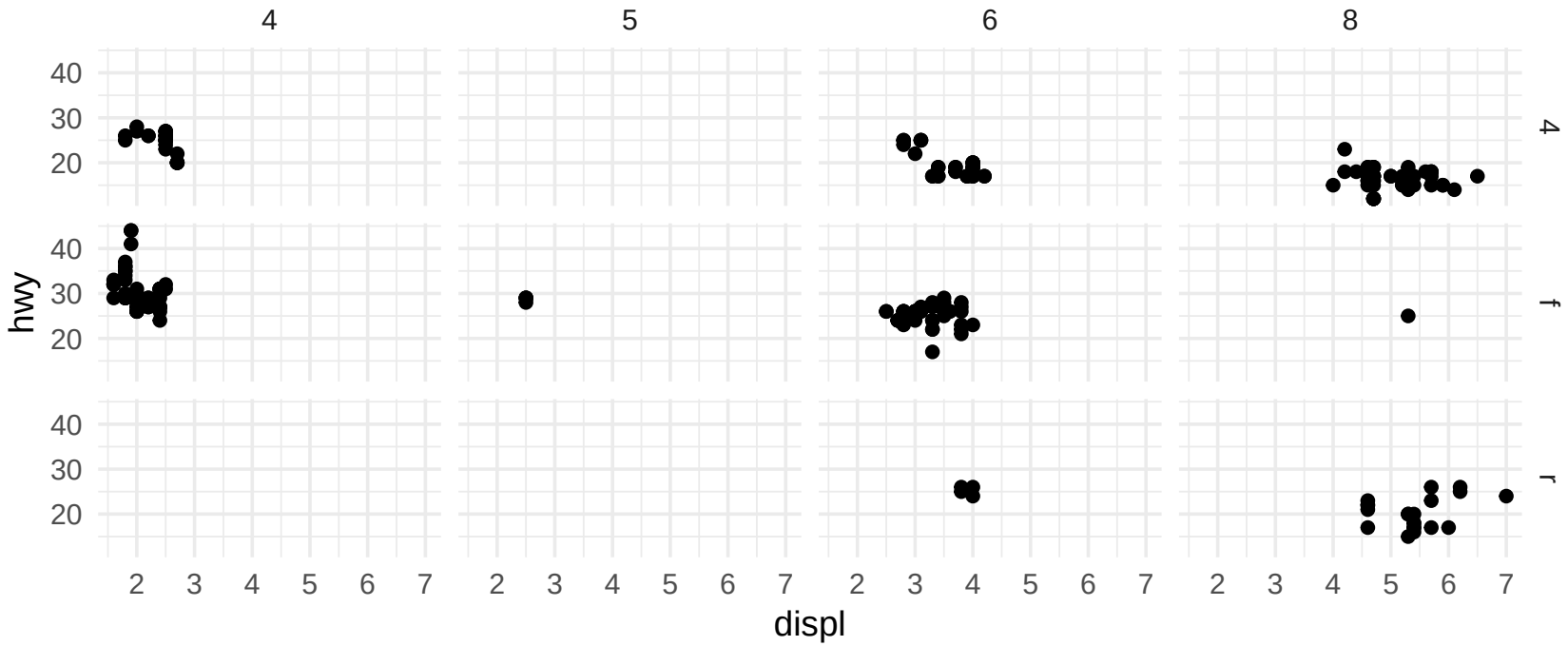
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



- `facet_wrap(~ class, nrow = 2)`
  - `class`를 x축으로 하여
  - 2행으로 배열

Facets (2 variables) – facet_grid

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



- facet_grid(drv ~ cyl)
  - cyl을 x축으로
  - drv를 y축으로
  - grid 모양으로 배열

Discussion

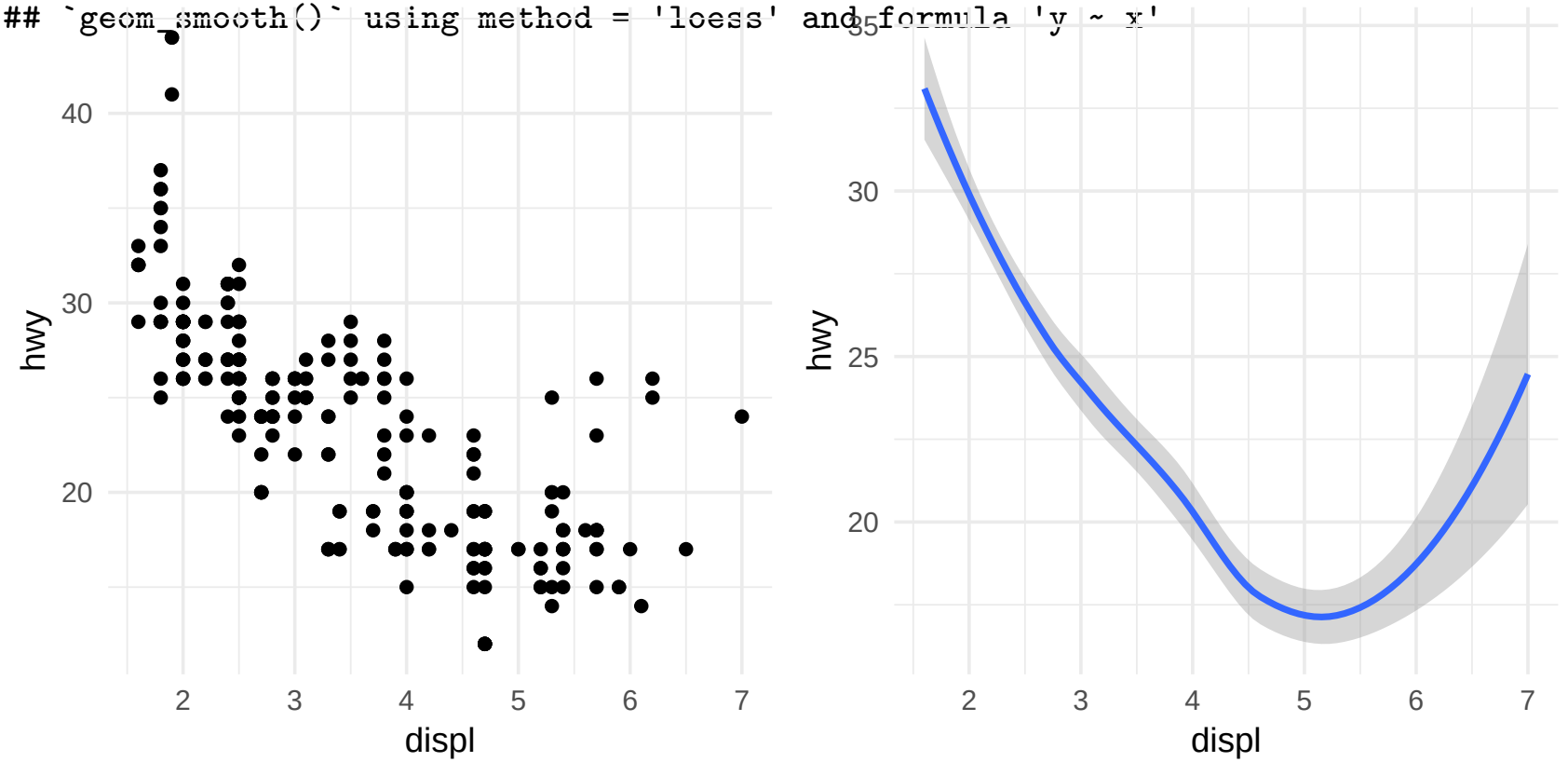
- Do `facet`
  1. Data가 충분히 많을 때
  2. Categorical 변수별로 각각의 distribution을 보고 싶을 때
- Don't `facet`
  1. 근접 비교를 하고 싶을 때
  2. Reader들의 사전 지식 수준이 높을때

III. 중첩

geom_point vs geom_smooth

```
e <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
f <- ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
grid.arrange(e, f, nrow = 1, ncol = 2)
```

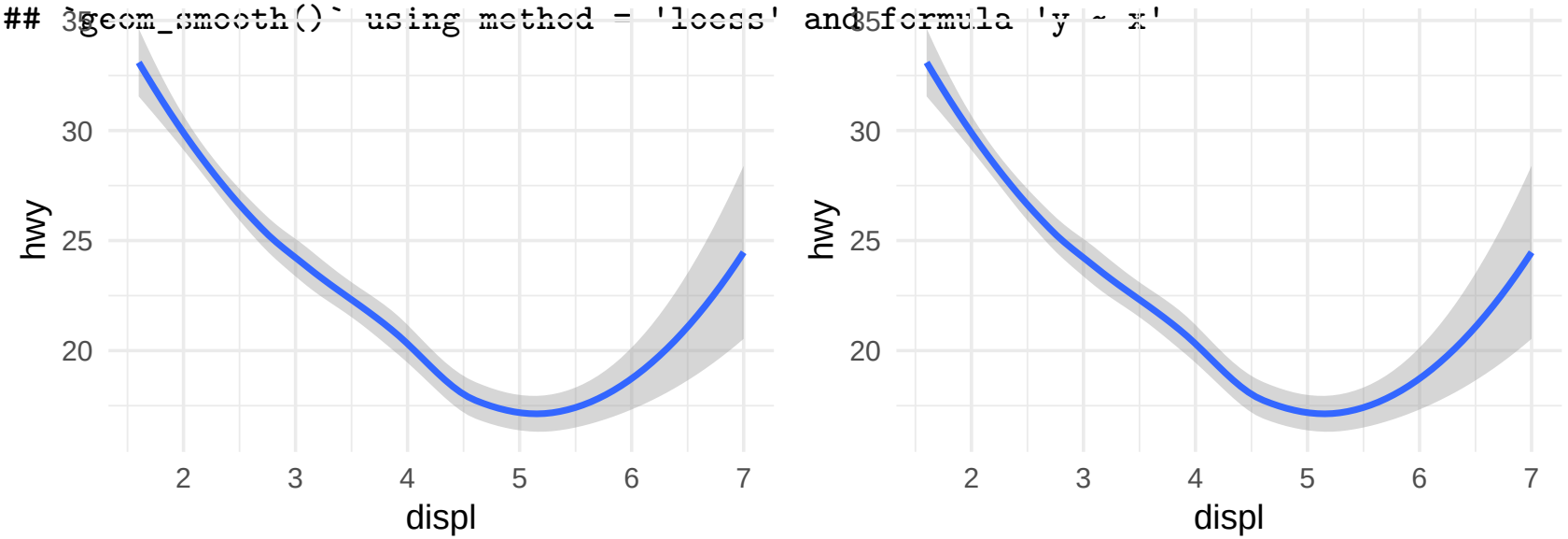
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



mapping의 상속

```
g <- ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
h <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth()
grid.arrange(g, h, nrow = 1, ncol = 2)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

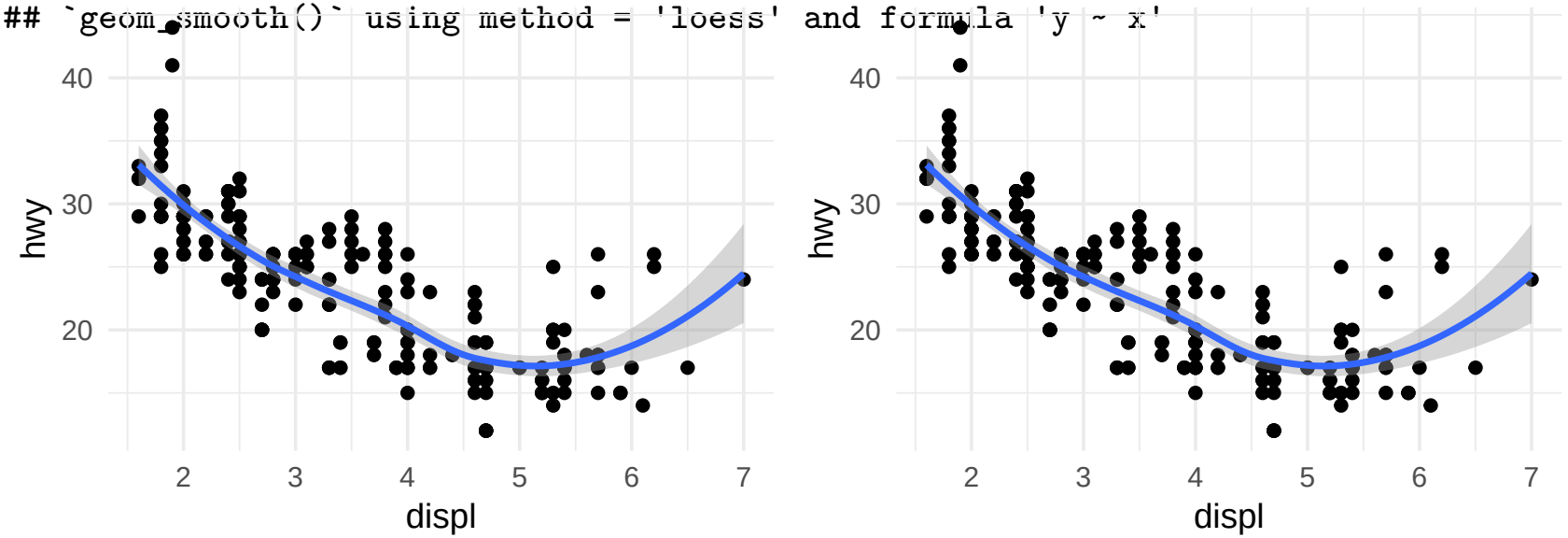


- g와 h의 명령어의 약간 variation이 있지만, 결과는 동일함
- h의 geom_smooth()에서 mapping이 없음 → 바로 앞의 ggplot()의 값을 사용

geom_point + geom_smooth

```
i <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
j <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() + geom_smooth()
grid.arrange(i, j, nrow = 1, ncol = 2)
```

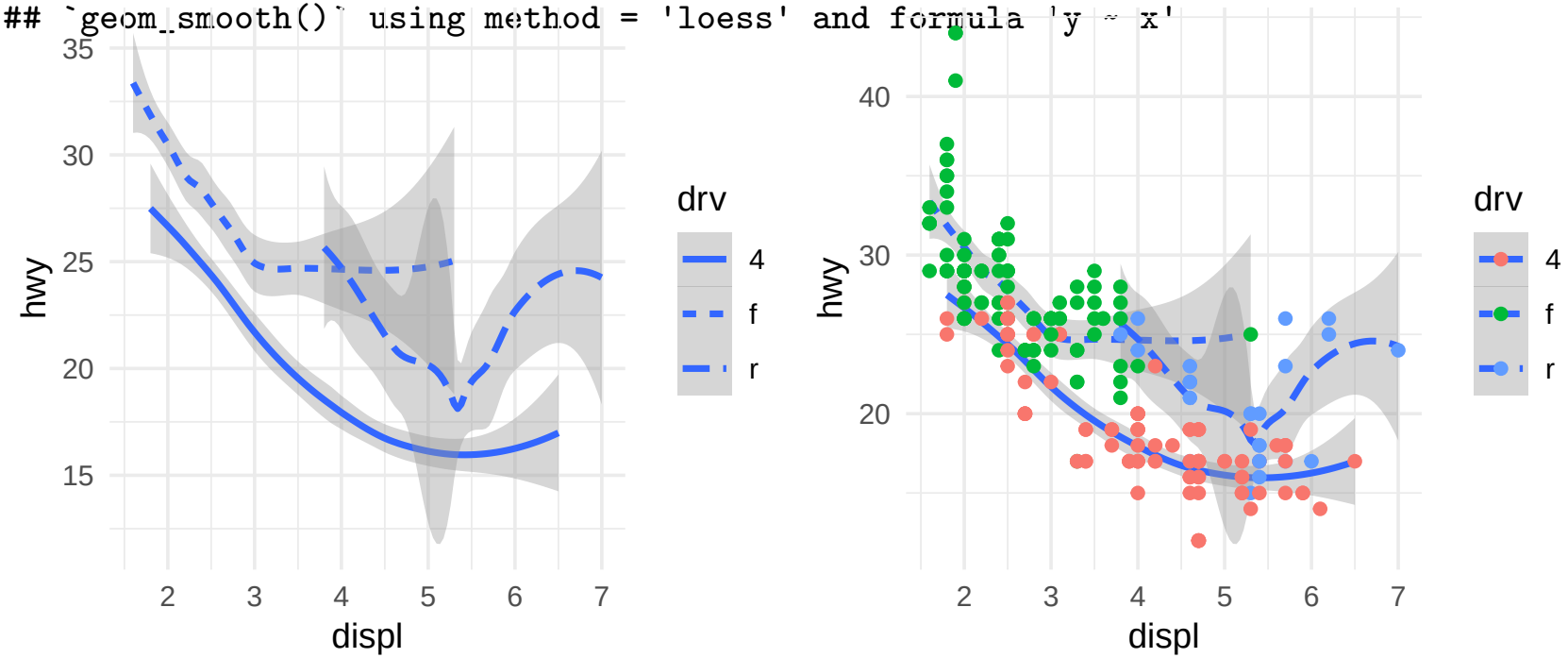
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- i와 j는 동일한 결과를 만듬
- j의 geom_point()와 geom_smooth()는 앞의 ggplot()에서 mapping을 상속받음
- 코드가 간결해짐

```
k <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth(aes(linetype = drv))
l <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth(aes(linetype = drv)) +
  geom_point(aes(color = drv))
grid.arrange(k, l, nrow = 1, ncol = 2)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- `linetype=drv`는 `drv`값에 따라서 각각 다른 curve를 그림
- `k`와 `l`의 `geom_smooth()`에는 `x`, `y`의 값이 없지만 이는 `ggplot()`에서 상속 받음
- `l`의 `geom_point()`도 마찬가지

IV. Discussion

Summary so far

- **Data**, **Aesthetic**, **Geometric Object**를 명시하여 그림을 rendering.
- 변수의 성격을 이해하여 **size, alpha, color, shape**와 **facet**등의 기능을 사용

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

1. 그림의 목적을 정하고
2. X (explanatory variable – 설명 변수)의 개수와 각 변수의 중요도와 순서
3. Y (dependent variable)를 명시
4. ggplot 객체를 rendering

Carseats와 additivity

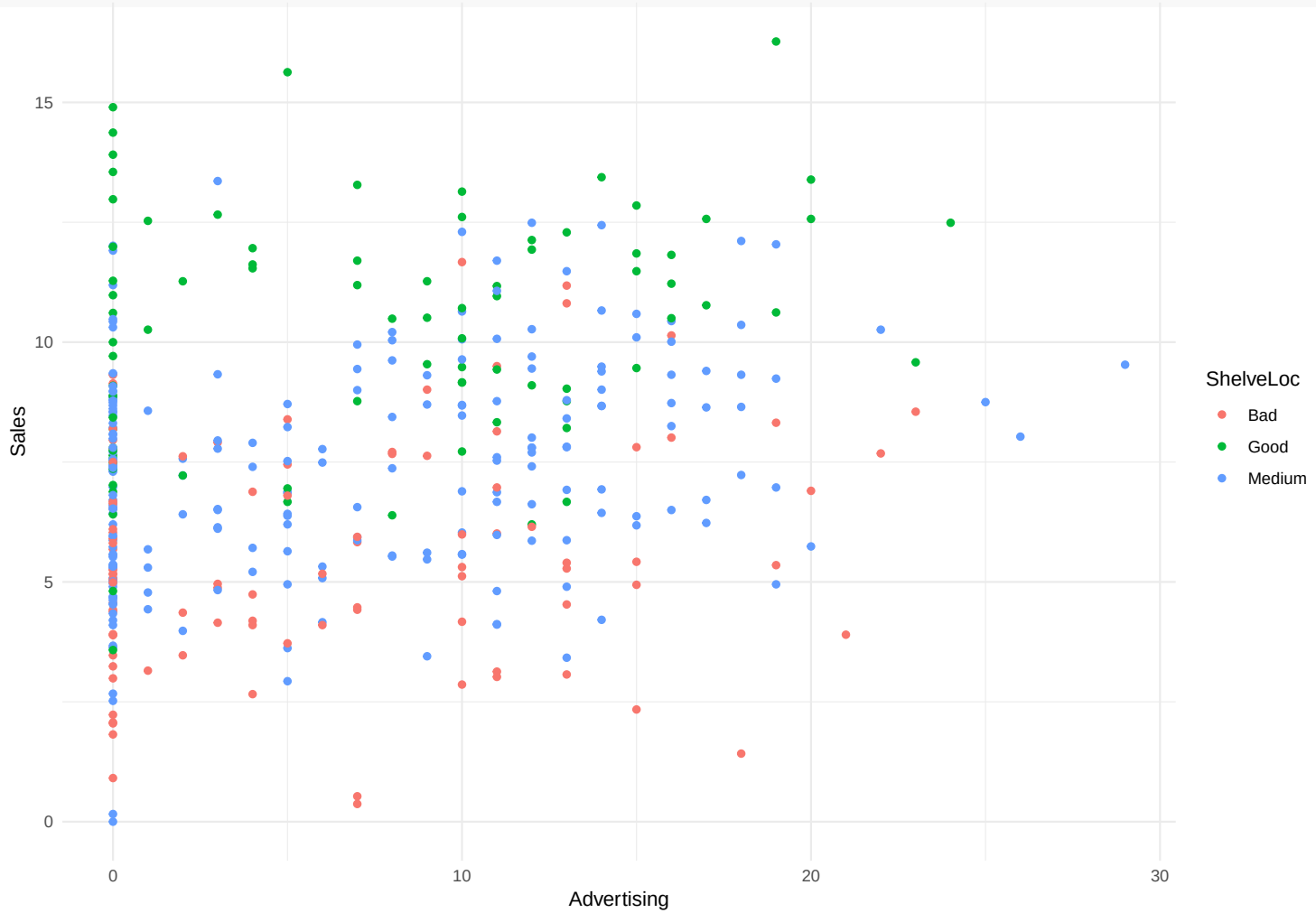
Dataset review

```
library(ISLR)
str(Carseats)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales       : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice   : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income      : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price       : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc   : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age         : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education   : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban       : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US          : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

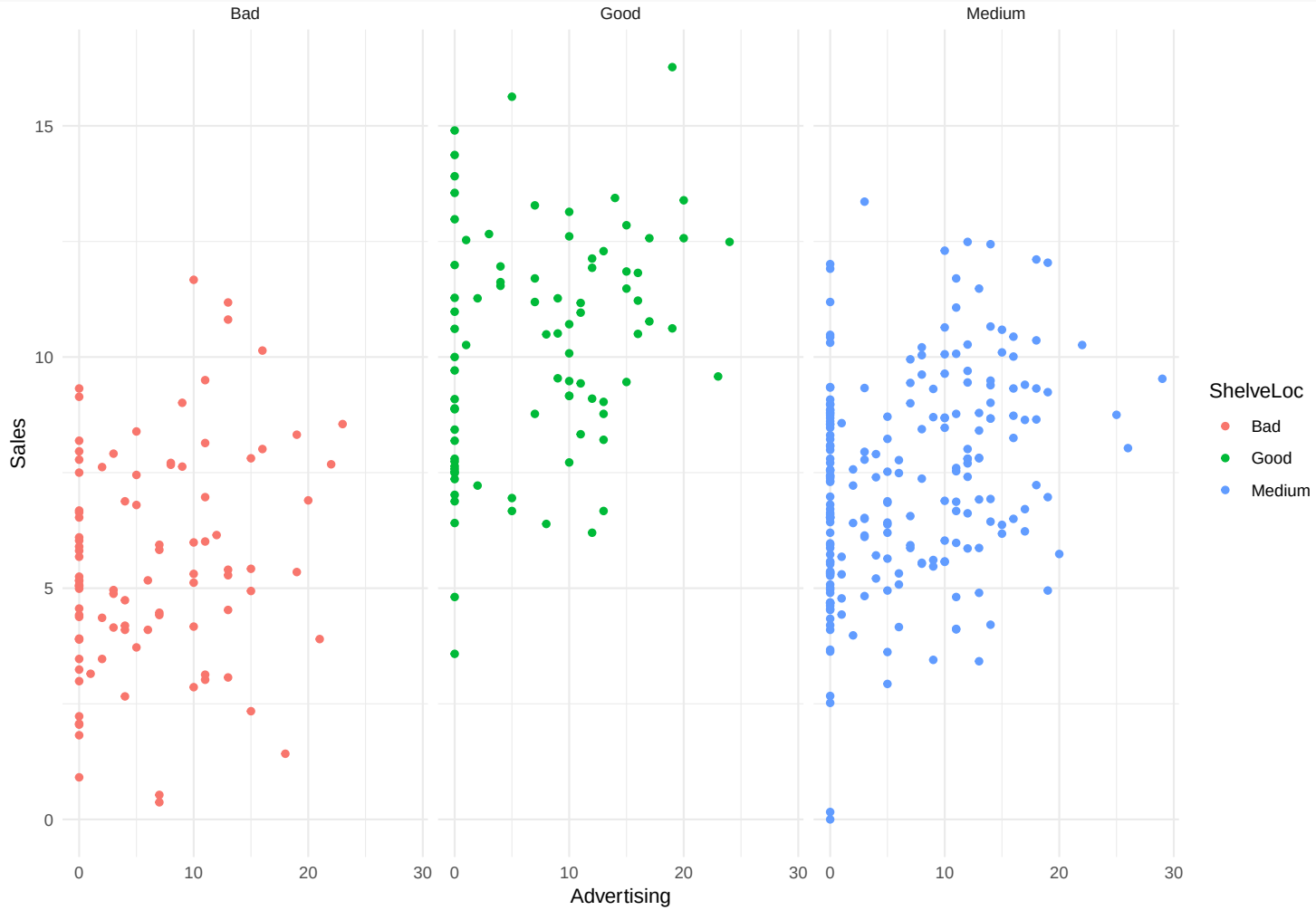
ggplot review

```
a <- ggplot(data = Carseats, aes(x = Advertising, y = Sales)) +
  geom_point(aes(color = ShelveLoc))
print(a)
```



Amazingly Additive!

```
# Add features to existing ggplot object by "Adding"
a <- a + facet_wrap(~ ShelveLoc)
print(a)
```



기능을 더하면 plot에 더해집니다!

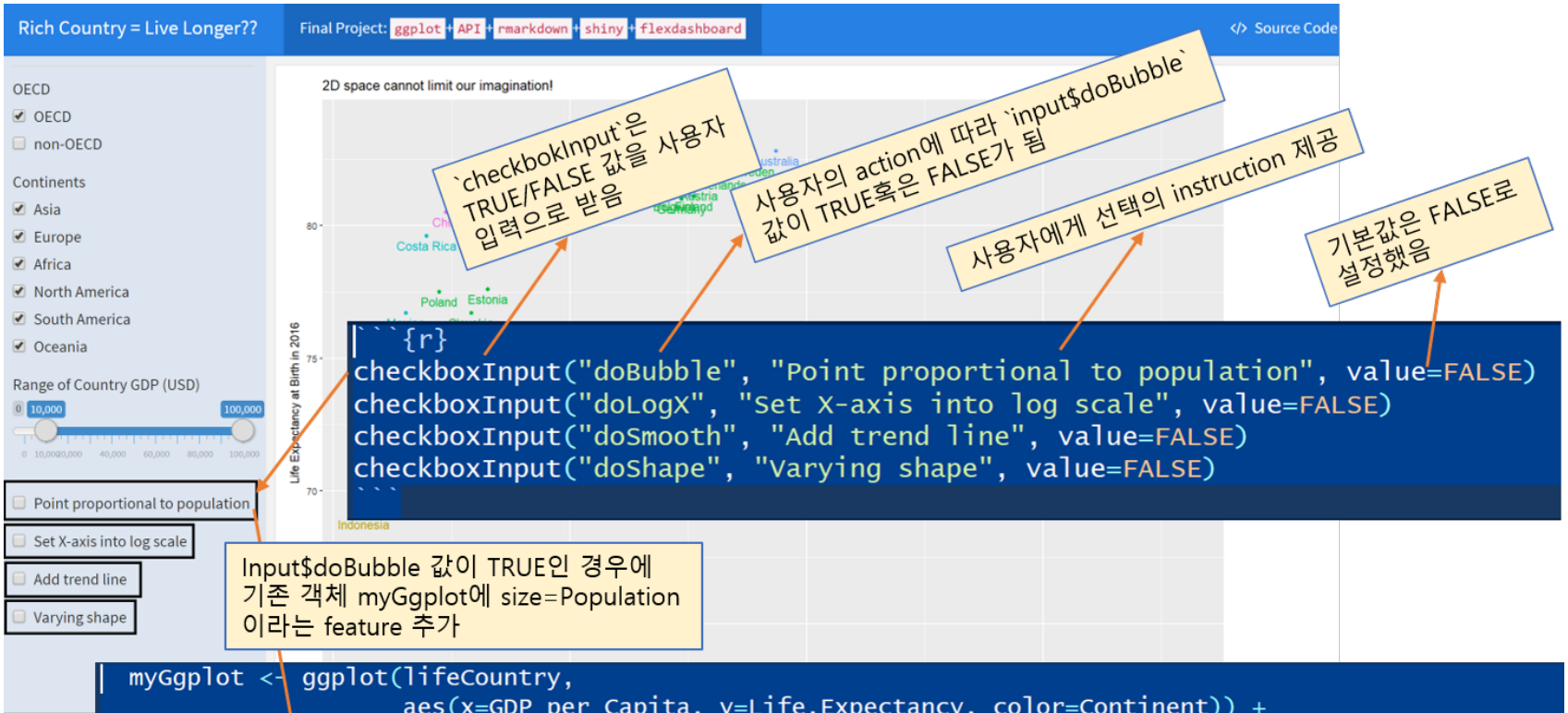
```
a <- ggplot(data = Carseats, aes(x = Advertising, y = Sales)) +
  geom_point(aes(color = Urban))
```

```
a <- a + facet_wrap(~ Urban)
```

rmarkdown과 함께 사용하면 매우 강력합니다.

- 스토리텔링 기반 발표 및 보고서 작성
  - 점점 깊이 들어가면서 발표
  - 좌우 혹은 상하로 배열하여 보고서 작성
  - 좌우 혹은 상하로 배열하여 대시보드 작성 (M32)
  - 사용자가 선택할 수 있게 대시보드 작성 (M33)

Example: M41-longevity




```r
```{r}
checkboxInput("doBubble", "Point proportional to population", value=FALSE)
checkboxInput("doLogX", "Set X-axis into log scale", value=FALSE)
checkboxInput("doSmooth", "Add trend line", value=FALSE)
checkboxInput("doShape", "Varying shape", value=FALSE)
```
```

```r
myGgplot <- ggplot(lifeCountry,
                   aes(x=GDP_per_Capita, y=Life.Expectancy, color=Continent)) +
  geom_text(aes(label=Country), size=4, vjust=1.5) +
  geom_point() +
  xlab("GDP per Capita in 2017 (in US Dollars)") +
  ylab("Life Expectancy at Birth in 2016") +
  ggtitle("2D space cannot limit our imagination!")
if (input$doBubble) { myGgplot <- myGgplot + geom_point(aes(size=Population)) }
if (input$doLogX)   { myGgplot <- myGgplot + scale_x_log10() + xlab("GDP per Capita (log-scale)")}
if (input$doSmooth) { myGgplot <- myGgplot + geom_smooth(method='lm', se=TRUE, size=1) }
if (input$doShape)  { myGgplot <- myGgplot + geom_point(aes(shape=Continent))}
myGgplot
```

실습과제 1

1. `C:/LS-DS/classProject`라는 폴더를 만드세요.
2. Rstudio를 열어서 rmarkdown의 html형식에 해당하는 .Rmd파일을 만드세요.
3. `classProject1.Rmd`로 위 폴더에 저장하세요.
4. Rstudio를 닫고 `classProject1.Rmd`를 더블클릭하여 실행하면 working directory가 자동으로 위의 폴더가 됩니다.
5. github `M41-longevity`에서 `lifeCountry.csv`파일을 다운받아서 위 폴더에 넣으세요.
6. 이제 분석을 위한 모든 준비가 끝났습니다.
7. Infile/Preprocessing을 해야합니다. `classProject1.Rmd`에서 `lifeCountry.csv`를 불러와서 `data.frame`으로 저장합니다.
8. 각 나라의 GDP와 기대수명에 대해서 산점도를 그리고 대륙에 따라서 점의 색깔이 달라지게 해보세요.
9. 자유롭게 분석을 시작해보세요.
10. html로 제작해 이메일을 보내주세요. `learningSpoonsR@gmail.com`

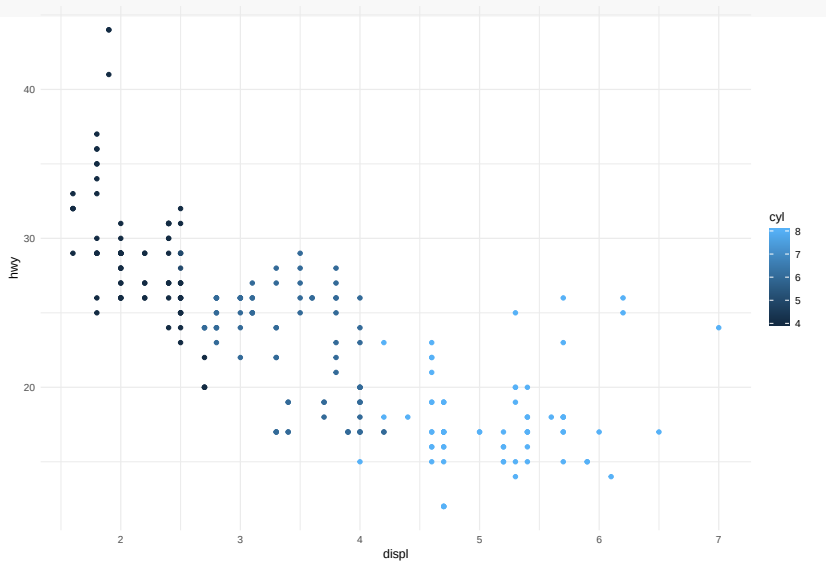
V. Grammar of Graphics

GG? – Grammar of Graphics

- Motivation
  1. 그래픽스에 대한 원리가 없다면, 그래픽 관련 패키지와 함수는 단지 특수 경우의 모음일 뿐
  2. 요리 백과사전을 다 읽는 것 vs. 물과 기름과 불의 작용에 대해서 익히고 백과사전을 **찾아가면서** 요리하는 것
- Advantage
  1. 새로운 package나 함수의 등장을 빠르게 반영
  2. 새로운 graphics를 만들어 내는 아이디어가 체계적이 됨
- Features
  1. 독립적이고 더할 수 있는 구성 요소들로 그래픽을 표현
  2. 개발과정에서 그래프의 특징을 한 가지 씩, 반복적으로 바꾸면서 그래프를 만들어 감
  3. 생각의 흐름, 스토리텔링의 흐름과 연계시킬 수 있기에 interactive graphics와 잘 조화됨

구성 요소

```
library(ggplot2)
ggplot(mpg) +
  aes(x = displ, y = hwy, color = cyl) +
  geom_point()
```



- Aesthetics **aes()**
  1. position
  2. size
  3. color
  4. shape
- Geometric Object (**geom_()**)
  1. Scatterplot – **point**
  2. Bubblechart – **point** (size)
  3. Barchart – **bar** (frequency)
  4. Box-and-whisker plot – **boxplot** (distribution)
  5. Line chart – **line**

Behind the scene

data

```
head(mpg[,c("displ", "hwy", "cyl")],10)

## # A tibble: 10 x 3
##    displ   hwy   cyl
##    <dbl> <int> <int>
##  1  1.80    29     4
##  2  1.80    29     4
##  3  2.00    31     4
##  4  2.00    30     4
##  5  2.80    26     6
##  6  2.80    26     6
##  7  3.10    27     6
##  8  1.80    26     4
##  9  1.80    25     4
## 10  2.00    28     4
```

aes

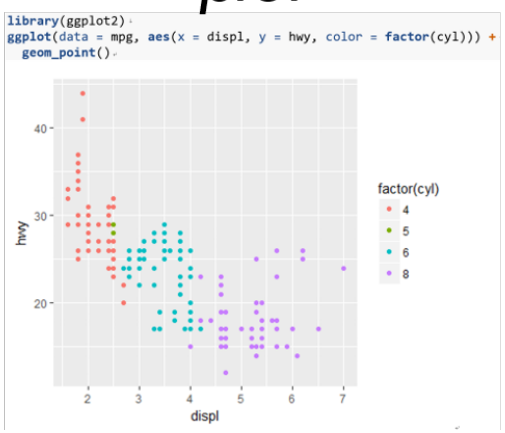
x	y	colour
1.8	29	4
1.8	29	4
2.0	31	4
2.0	30	4
2.8	26	6
2.8	26	6
3.1	27	6
1.8	26	4

geom

x	y	colour	size	shape
0.037	0.531	#F8766D	1	19
0.037	0.531	#F8766D	1	19
0.074	0.594	#F8766D	1	19
0.074	0.562	#F8766D	1	19
0.222	0.438	#00BFC4	1	19
0.222	0.438	#00BFC4	1	19
0.278	0.469	#00BFC4	1	19
0.037	0.438	#F8766D	1	19

plot

```
library(ggplot2)
ggplot(data = mpg, aes(x = displ, y = hwy, color = factor(cyl))) +
  geom_point()
```

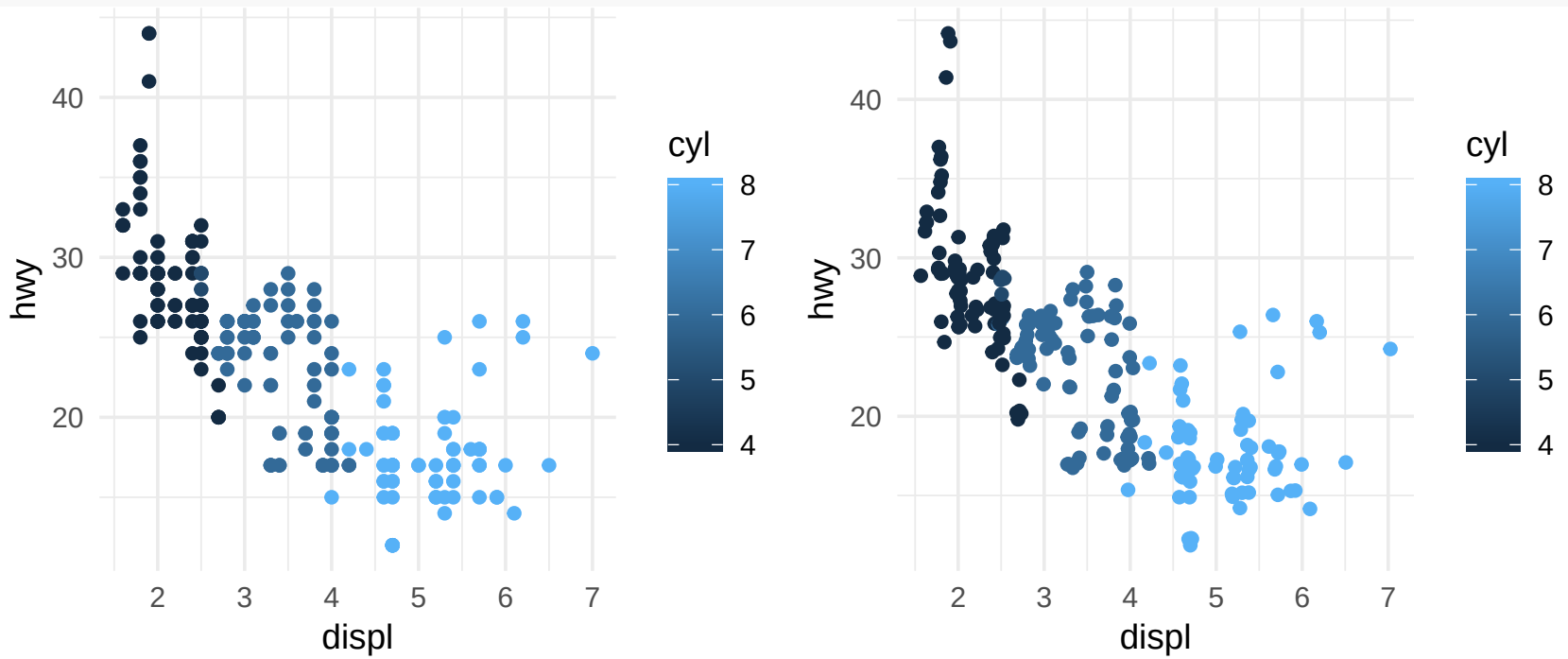


VI. More Plots

position = "jitter"

- 중첩된 관찰값에 노이즈를

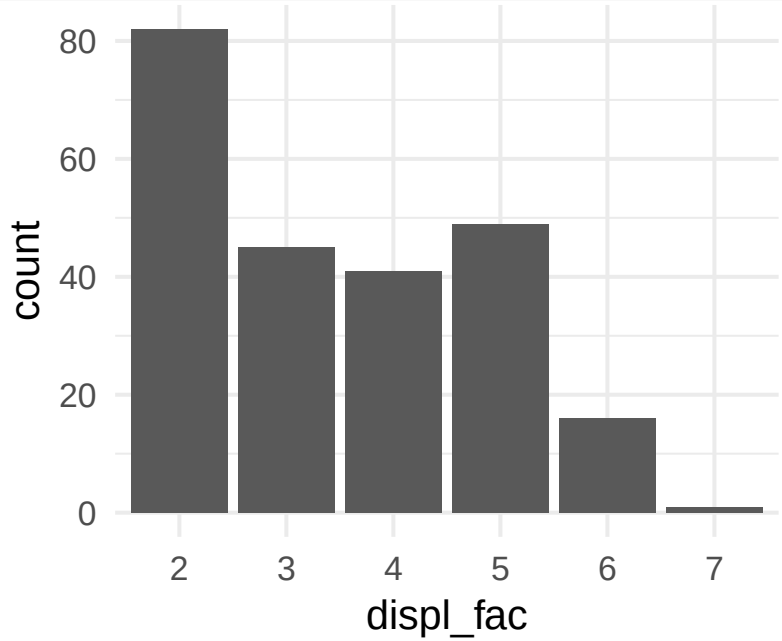
```
a <- ggplot(mpg) + geom_point(aes(displ, hwy, color = cyl))
b <- ggplot(mpg) + geom_point(aes(displ, hwy, color = cyl), position = "jitter")
grid.arrange(a, b, nrow = 1, ncol = 2)
```



Barchart

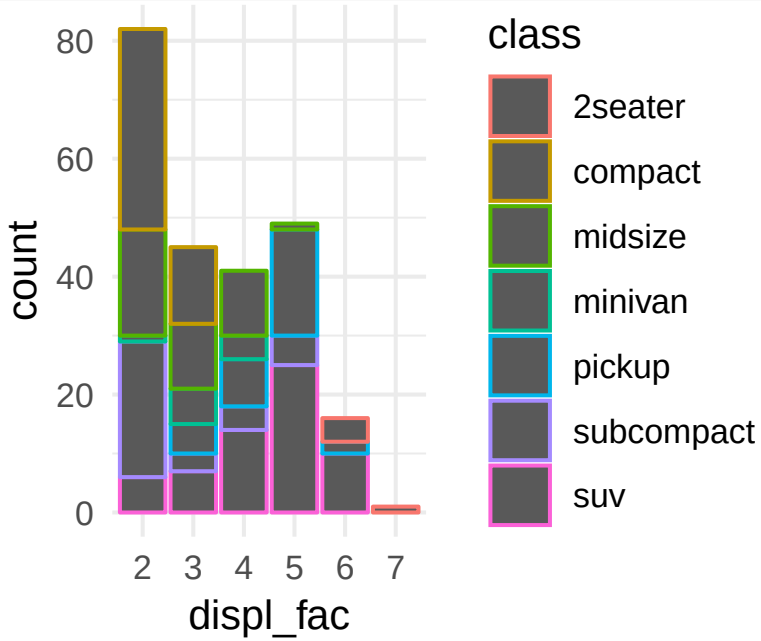
- x축에 Categorical 변수를 넣어서
- 각각의 값에 대해서 몇 개의 관찰값이 있는지를 보여줌.
- #count #density #distribution

```
mpg$displ_fac <-
  as.factor(round(mpg$displ,0))
ggplot(mpg) +
  geom_bar(aes(x = displ_fac))
```



- x변수 외에도 다른 Categorical 변수를 추가할 수 있음.

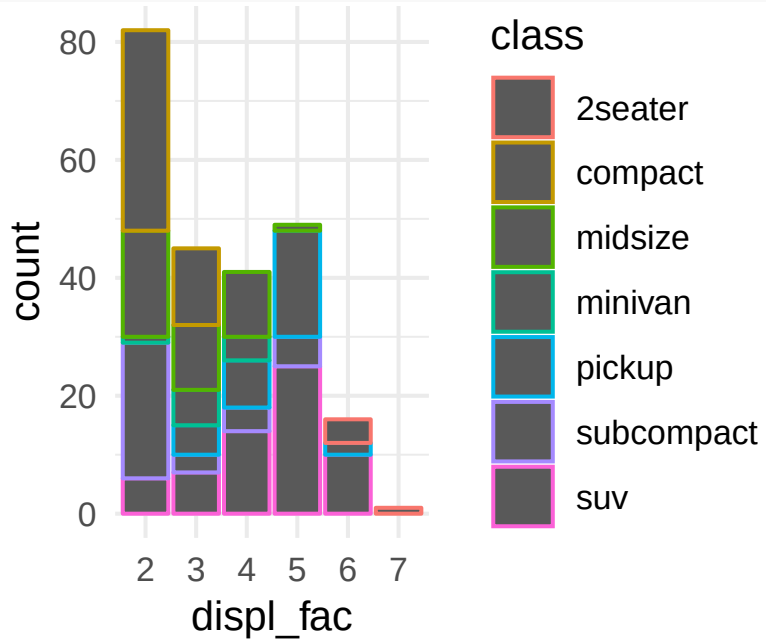
```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, color = class))
```



color and fill

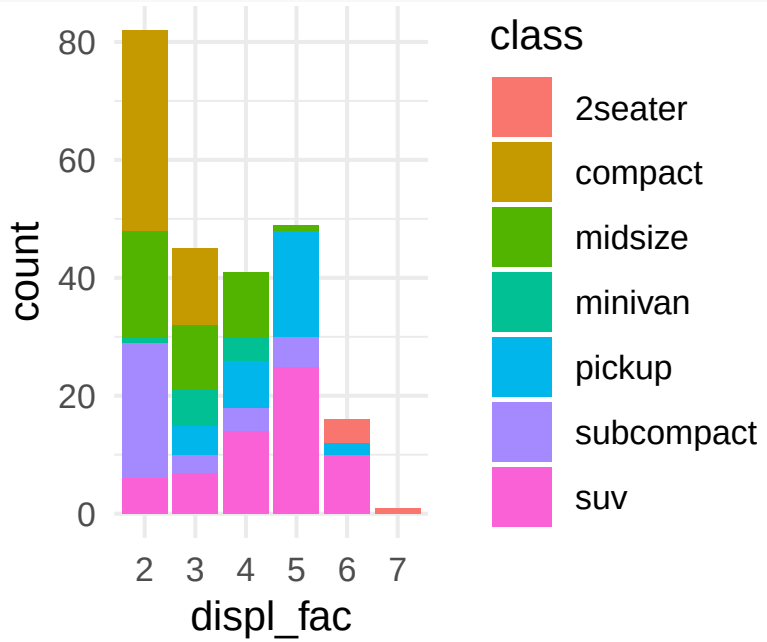
- **color** – 테두리만 됨

```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, color = class))
```



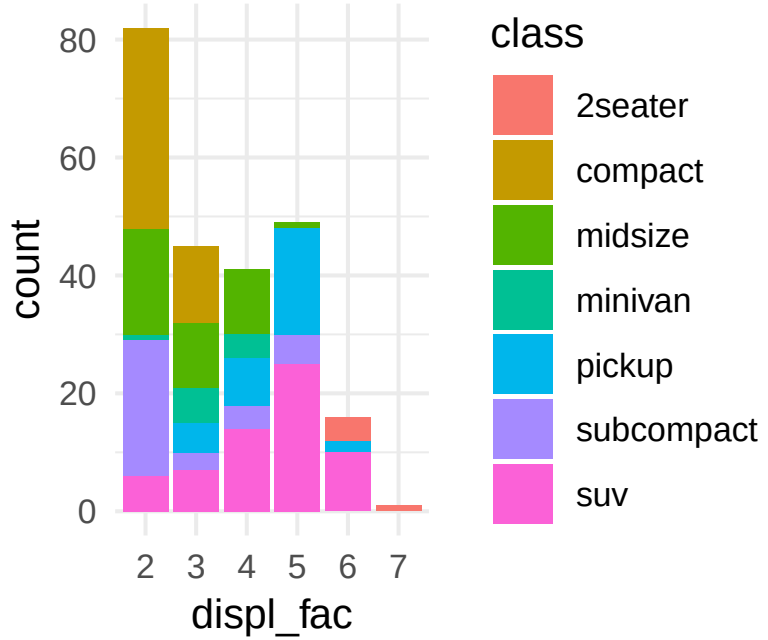
- **fill** – 채워짐

```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, fill = class))
```

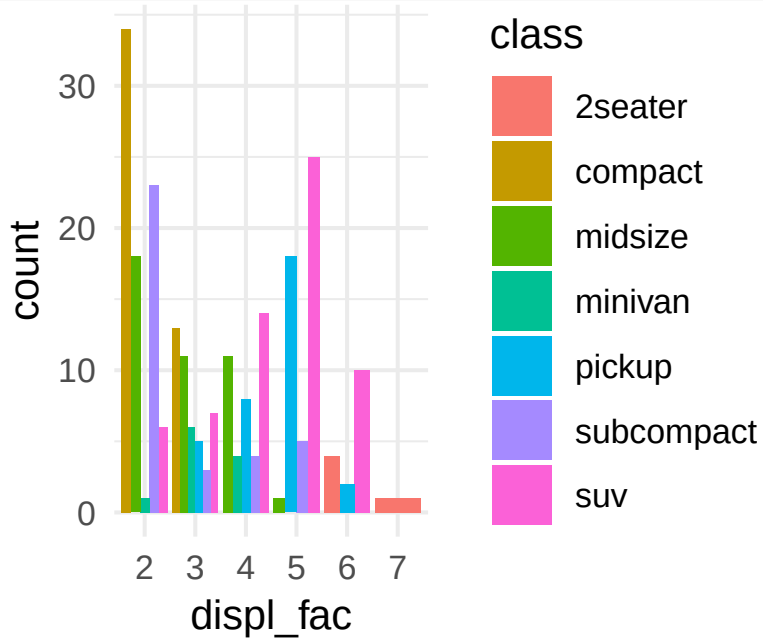


position="dodge"

```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, fill = class))
```



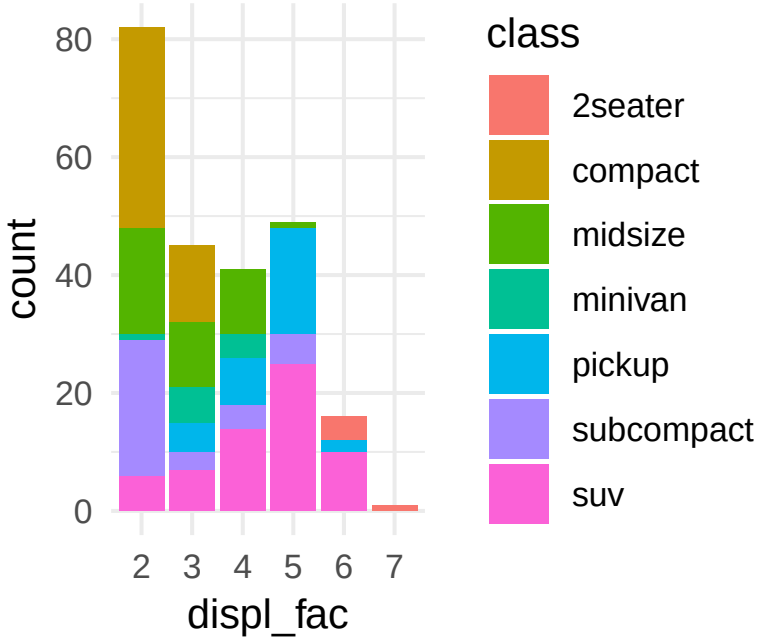

```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, fill = class),
           position = "dodge")
```



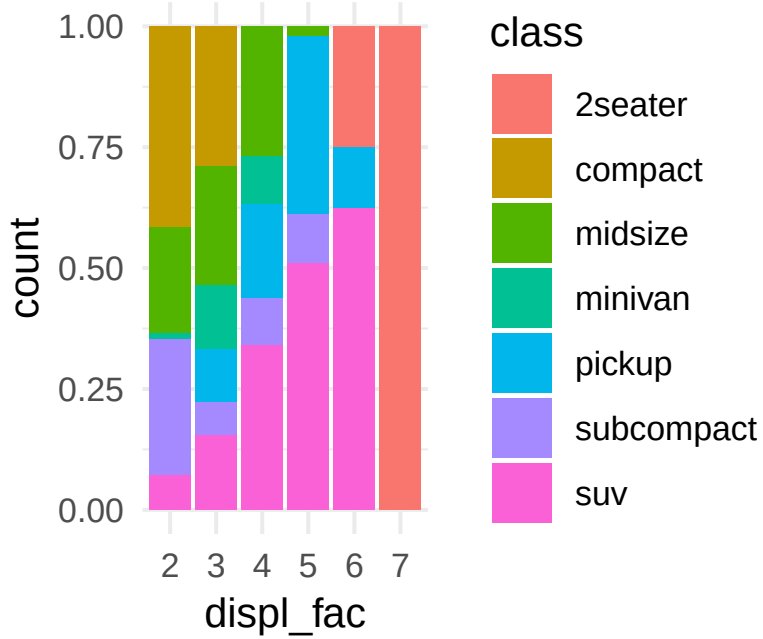
- 2개의 discrete 변수를 잘 처리하는 법?
  - 쌓아서 배치 vs 비껴서 배치

Barchart (position = "fill")

```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, fill = class))
```




```
ggplot(mpg) +
  geom_bar(aes(x = displ_fac, fill = class),
           position = "fill")
```

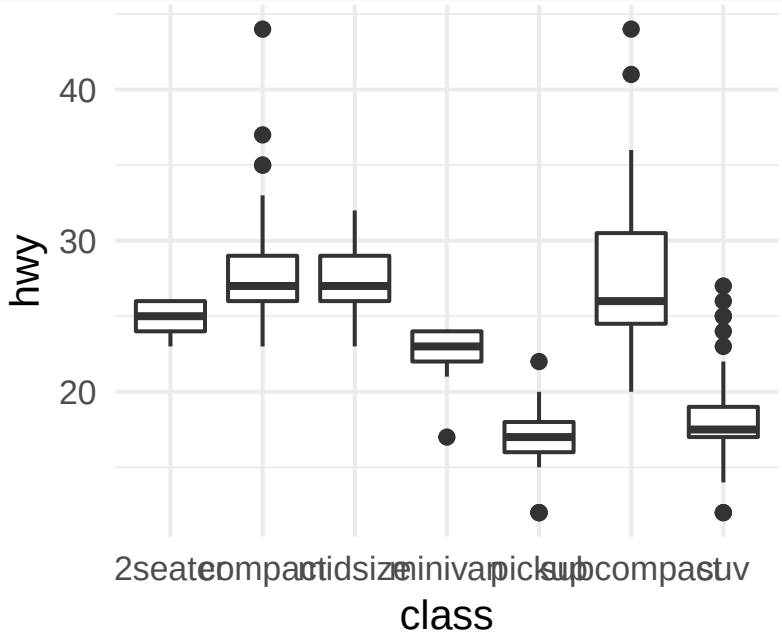


- `class`의 각각의 `displ_fac` 값에서의 분포?
- count vs proportion

Boxplot

- 그룹 변수인 x의 각각의 값에 대해서 연속 변수인 y의 분포를 보기 위함.

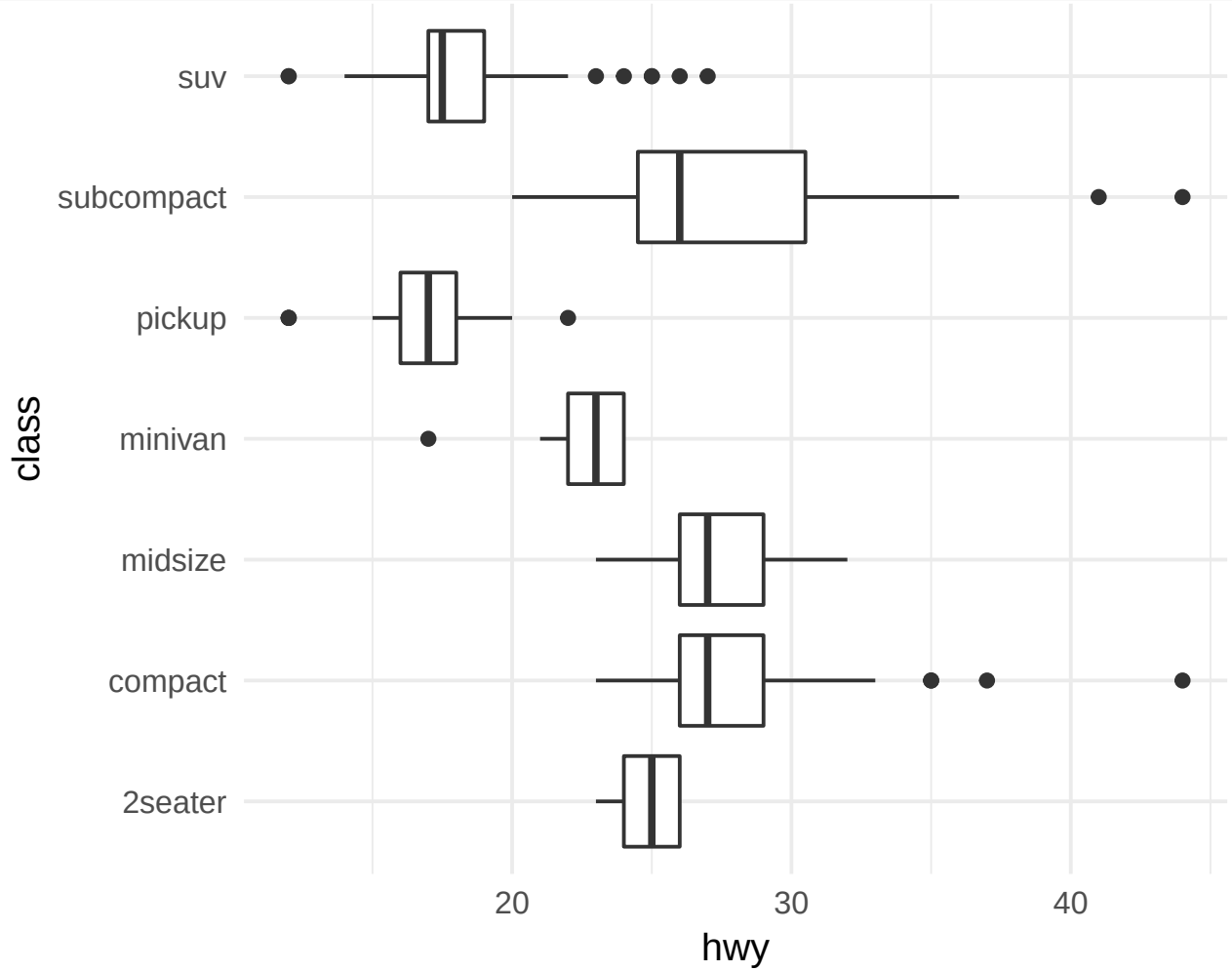
```
ggplot(mpg) +
  geom_boxplot(aes(x = class, y = hwy))
```



- 점으로 표현된 것은 이상치(outlier)로서 이상하게 높거나 낮은 값
- 박스의 상단은 상위 25%, 하단은 하위 25%
- 박스의 가운데 직선은 중간값
- x변수의 값이 `character`라서 display가 복잡하네요.
- 이럴때는 어떻게 할까요?

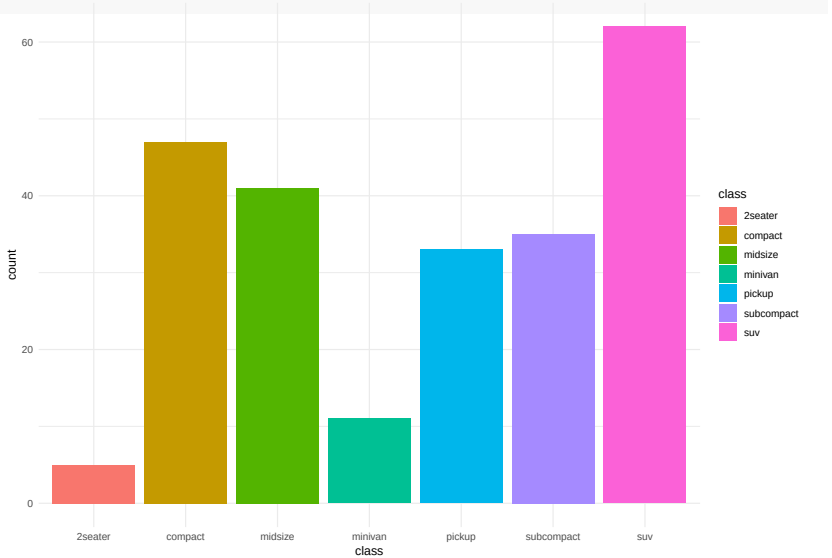
coord_flip()

```
ggplot(mpg) +
  geom_boxplot(aes(x = class, y = hwy)) +
  coord_flip()
```

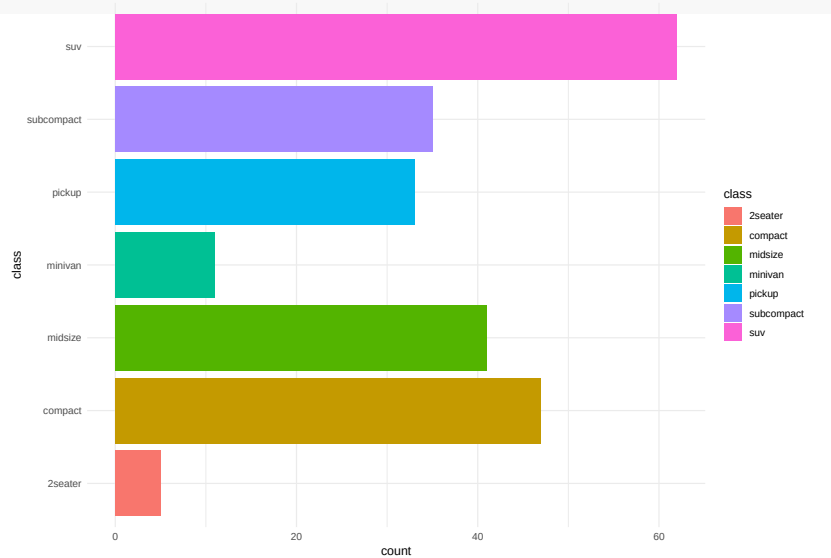


`coord_flip()` for Barplot = ???

```
ggplot(mpg, aes(x = class, fill=class)) +
  geom_bar()
```

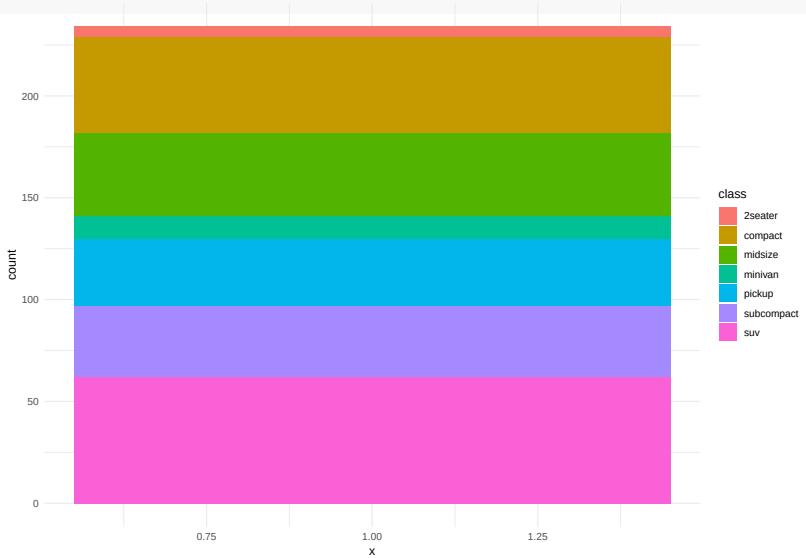



```
ggplot(mpg, aes(x = class, fill=class)) +
  geom_bar() + coord_flip()
```

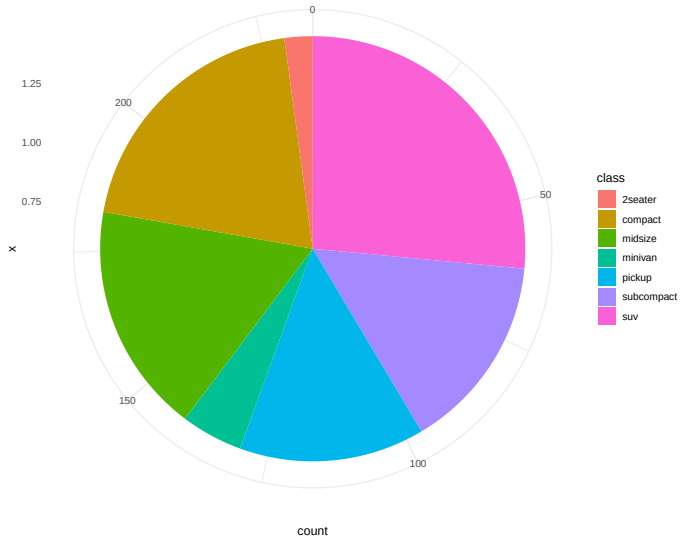


coord_polar() for Barplot

```
ggplot(mpg, aes(x = 1, fill = class)) +
  geom_bar()
```



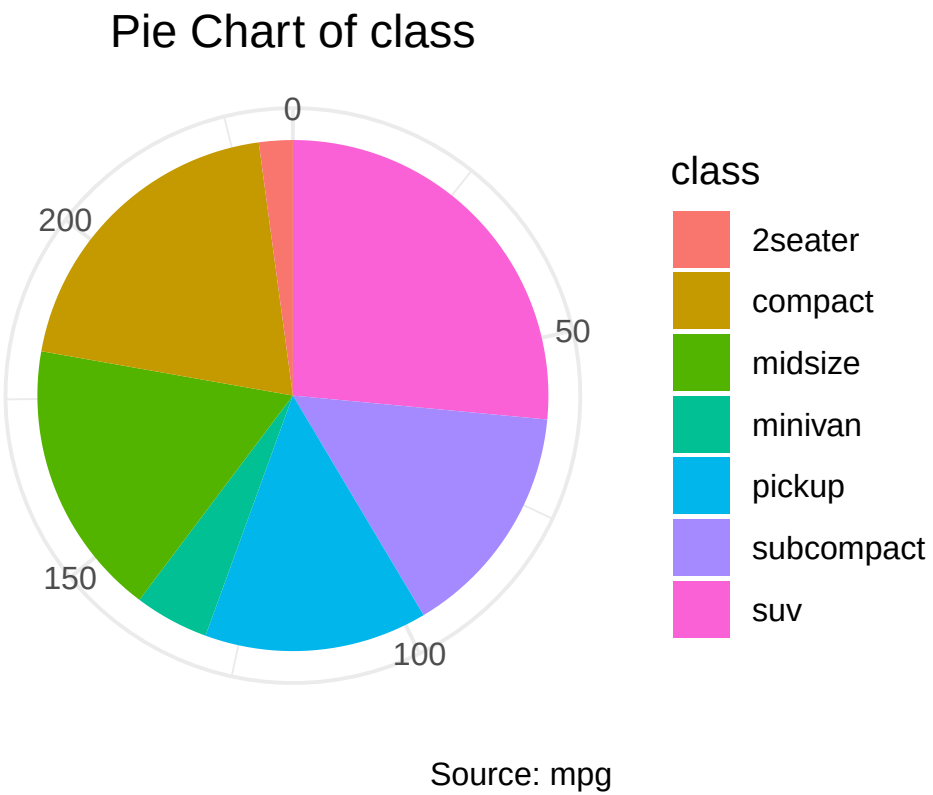

```
ggplot(mpg, aes(x = 1, fill = class)) +
  geom_bar() + coord_polar(theta = "y")
```



- Bar chart vs Pie chart
- 둘의 차이는 **ggplot2** 개발의 motivation이 되었다고 합니다.

Pie Chart (from M24-ggplot2 Gallery)

```
ggplot(mpg, aes(x = "", fill = factor(class))) +
  geom_bar(width = 1) +
  theme(axis.line = element_blank(), plot.title = element_text(hjust=0.5)) +
  labs(fill = "class", x = NULL, y = NULL,
       title = "Pie Chart of class", caption = "Source: mpg") +
  coord_polar(theta = "y", start=0)
```

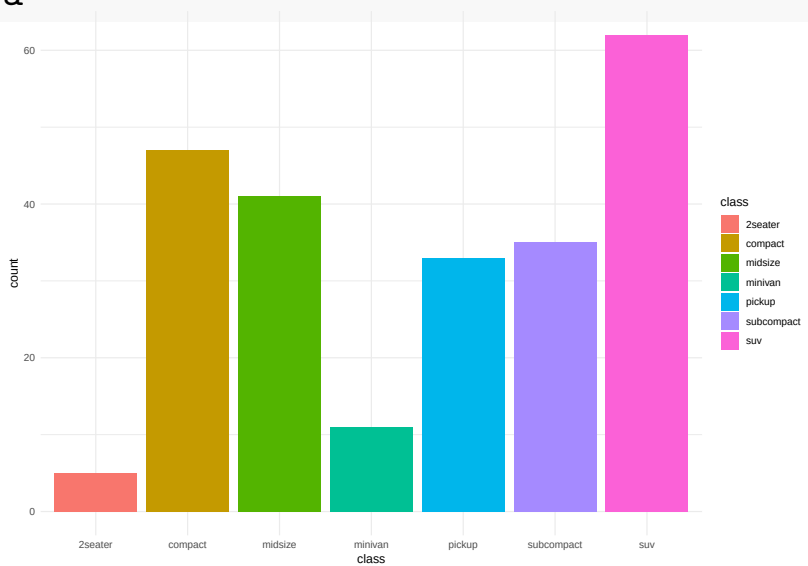


ggplot 기타 기능 (저장 & plotly)

- ggplot객체를 png 파일로 저장

```
png("out_file.png") # initiate
ggplot(mpg) + geom_bar(aes(x=class)) +
  coord_flip() # save
dev.off() # finish
```

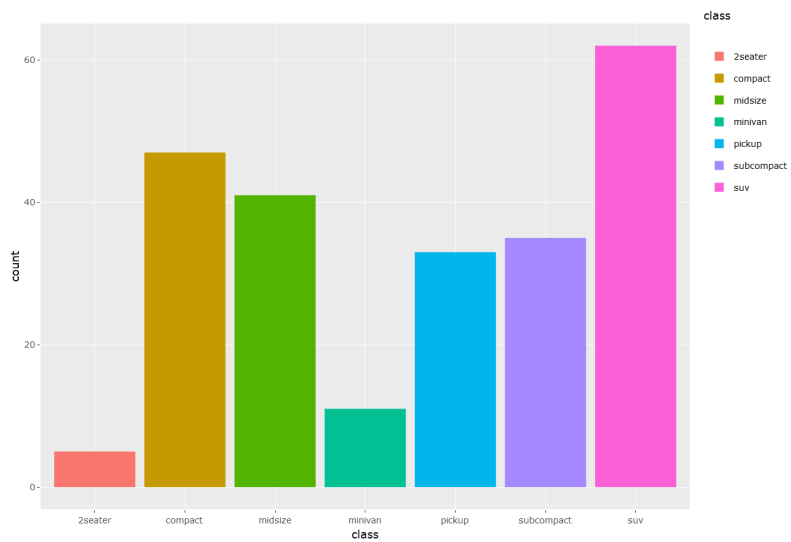
```
a
```



- plotly – html에서 각종 추가 기능 제공

```
library(plotly)
a <- ggplot(mpg) +
  geom_bar(aes(x=class, fill=class))
```

```
ggplotly(a) # better in html
```



VII. Summary

Summary

1 Variable

변수	이름	Name	`geom_()`
Continuous (연속 변수)	확률분포 히스토그램	density plot histogram	`geom_density()` `geom_histogram()`
Discrete (이산 변수)	막대차트 파이차트	bar plot pie chart	`geom_bar() ("dodge", "jitter")` `geom_bar() + coord_polar()`

2 variables

x	y	이름	Name	`geom_()`
Discrete	Continuous	박스플랏	box plot	`geom_boxplot()`
Continuous	Continuous	산점도 산점도	scatter plot scatter plot	`geom_point() "jitter"` `geom_smooth()`

More References

- `M24-ggplot2-50examples` – ggplot의 갤러리 (50개 예제)
- `M6X References/books/ggplot2.pdf`
  - 무료 베포된 ggplot관련 영문 서적
  - 번역본도 출간되어 있음
- google: “ggplot gallery”
  - https://www.r-graph-gallery.com/portfolio/ggplot2-package/

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                  required
<GEOM_FUNCTION> (mapping = aes( <MAPPINGS> ),
 stat = <STAT> , position = <POSITION> ) +                Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = “point”**)** Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z**))** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group**))**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1**))** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900**))** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1**))**
b + geom_hline(aes(yintercept = lat**))**
b + geom_vline(aes(xintercept = long**))**

b + geom_segment(aes(yend=lat+1, xend=long+1**))**
b + geom_spoke(aes(angle = 1:1155, radius = 1**))**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy**))** x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**, map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))

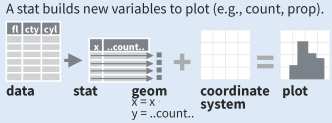
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z**)**, hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z**))**, x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal themes

r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)**))**

α^{c} α^{d} α^{e} α^{p} α^{r}

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) **+ ylim**(10, 20)
t + scale_x_continuous(limits = c(0, 100)) **+**
scale_y_continuous(limits = c(0, 100))